Exploring Computer Science with the Raspberry Pi

Derek C. Schuurman
Professor of Computer Science, Calvin University
version: 1.12 (2023-2025)

Contents

Compatibility Note

The Raspberry Pi project has periodic updates to hardware and software. This was written and tested using the Buster and Bullseye version of the Raspberry Pi OS and with various models up the Raspberry Pi up to the model 5. Edits are underway to update the contents to reflect the Bookworm version of the OS. This book is a "work in progress" and the contents are provided "as is," but the hope is to incrementally provide updates as time allows.

About the Author

Derek C. Schuurman worked as an electrical engineer for several years and later completed a PhD at McMaster University in the area of robotics and computer vision using machine learning. He is currently professor of computer science at Calvin University in Grand Rapids, Michigan. He is author of *Shaping a Digital World* and co-author of *PSpice Simulation of Power-Electronics Circuits*, and *A Christian Field Guide to Technology for Engineers and Designers*. He is a an enthusiastic user of the Raspberry Pi.

Dedication

This guide is dedicated to my late wife Carina Schuurman, who encouraged me in my vocation as a computer scientist, tolerated my frequent tinkering, brought art and beauty (and four children) into my technical life, and without whom I would not be where I am.

License Terms

1 Introduction

The respected computer scientists Edsgar Dijkstra once remarked that “computer science is no more about computers than astronomy is about telescopes.” Neither is it about nifty computing devices like the Raspberry Pi. Even so, having access to a good educational computer is certainly an asset in the study of computer science. As it turns out, the Raspberry Pi is a wonderful little computer for experimenting with many basic concepts in computer science. The body of knowledge in computer science includes a wide range of topics such as programming languages, data structures and algorithms, digital logic, computer organization, operating systems, networking and the web, the Internet of Things (IoT), artificial intelligence, and ethics. The chapters of this book are organized around many key topics which are explored with practical examples and exercises which can all be performed on a recent model of the Raspberry Pi.

The discipline of computer science is relatively young compared to other disciplines. It relies on many abstract and theoretical concepts from mathematics, including topics from discrete math, algebra, and calculus. While such theoretical foundations are essential for researchers, there are many concepts in computer science that are quite accessible to the keen student or hobbyist. This book is written for such an audience, containing many practical examples for those with only a modest mathematical background. Furthermore, this book also serves as a gentle introduction to using the Linux operating system.

1.1 The Raspberry Pi

The Raspberry Pi is a a nifty little computer about the size of a deck of cards and capable of running a full Linux desktop operating system while consuming only modest power. It includes USB ports for connecting a keyboard and mouse along with a variety of other peripherals, an ethernet adapter, and HDMI monitor connections. The Raspberry Pi was originally constructed for education, but has found fruitful uses for hobbyists, home automation, industrial applications, and as an appropriate technology for use in schools in the majority world. It is manufactured to comply with RoHS (Restriction of Hazardous Substances) directives and relies on a single microSD card for its storage. It runs a variant of Linux called the Raspberry Pi OS, and supports a wide variety of open source software, including a plethora of educational programs. Moreover, it can be purchased at a modest price.

This guide was written primarily with engineering and computer science students in mind, but it will be of interest to others who have a keen interest in learning more about programming and technical computing.

Figure 1: The author visiting the Raspberry Pi Store in Cambridge, England

1.2 Initial Setup of the Raspberry Pi

Insert an SD card with the Raspberry Pi OS in the Raspberry Pi and apply power. When you first boot the regular Raspberry Pi OS it will be running a graphical desktop environment with a friendly menu-driven interface. Upon the first boot, a dialogue box will appear guiding you through an initial setup. Follow the prompts to configure the keyboard and username. Provide a username and a password as prompted. Configure your Wi-Fi settings and select the option to "Update Software" (note that this may take a very long time when you first setup the Raspberry Pi).

1.3 Getting Started with the Command Line

There is also a command-line based version of the operating system called **Raspberry Pi OS Lite**. This version of the OS consumes less power than the regular Raspberry Pi OS running a desktop environment and can be used on older models of the Raspberry Pi with less RAM. When setting up the Raspberry Pi with the **Lite** OS, you will be prompted to configure the keyboard and provide a username and password. The Lite version of the OS is well suited for using the Raspberry Pi as a server, as an embedded system, or in an IoT (Internet of Things) application.

However, for regular desktop use, the regular version of Raspberry Pi OS is best. When using the Desk-

top OS, the command line can still be accessed using the *Terminal* app. Alternately, it can also be accessed using a *virtual console* by pressing CTRL+ALT+F1 which will enter a full screen terminal. If a login prompt appears, you can login using the username and password you configured during setup. One can return to the desktop from a virtual console by pressing CTRL+ALT+F7. Additional virtual consoles can be independently accessed using CTRL+ALT along with the keys F2 through F6.

1.3.1 The Shell

Once you enter the command line, you will be running in a Linux *shell*. In simple terms, a shell is a command interpreter which provides a rich set of commands which can be used to execute programs and interface with the operating system. The Raspberry Pi OS uses the Bash (Bourne Again Shell) by default, a shell based on an older shell called the *Bourne Shell*. BASH is popular among users of Linux and features automatic command line completion using the tab key and can be used to create programs called *shell scripts*.

There are a variety of different Linux shells that can be used. As indicated, the default Linux shell is the **Bash** shell, but other shells are also available. Each shell has its own features and options. For example, to switch the default shell from **Bash** to the **Z shell** (**zsh**), type the following

```
sudo apt install zsh -y
chsh -s /bin/zsh
```

After issuing these command, logout and then back in and you should now be running with **zsh**. Various configuration options can be set inside a file named `.zshrc` located within your home folder.

1.3.2 Shell commands

Some of the commands available in the shell are summarized below:

Command	**Description**
`cd` *directory*	Changes the current working directory to *directory*
`pwd`	Displays the name of the current working directory
`mkdir` *directory*	Create a new directory called *directory*
`rmdir` *directory*	Removes the directory called *directory*
`ls`	Display a list of files in the current directory
`cp` *f1 f2*	Copy a file from the source *f1* to the destination *f2*
`rm` *filename*	Remove a file *filename*

Command	Description
`mv` *f1 f2*	Move a file from *f1* to *f2*
`ftp` *host*	Transfer files to and from *host*
`diff` *f1 f2*	Display the difference between file *f1* and file *f2*
`cat` *filename*	Displays the contents of *filename*
`more` *filename*	Displays contents of *filename* pausing when screen is filled
`head` *filename*	Displays contents at the beginning of *filename*
`tail` *filename*	Displays contents at the end of *filename*
`wc` *filename*	Displays the number of newlines, words, and bytes in *filename*
`clear`	Clears the terminal display.
`date`	Display the current date and time.
`df`	Report the amount of free disk space available
`du`	Report the amount of disk usage
`find` *path conditions*	Utility for finding files
`locate` *search-string*	Utility for locating files
`free`	Report information about memory usage
`hostname`	Display the name of the current host system
`talk` *user*	Talk to user on your machine or on another host
`uptime`	Display the current time stats
`who`	Display the users currently logged-on the system
`whoami`	Display your User ID
`uname -a`	Display information about the operating system

These commands only represent a portion of the user commands available in a Linux shell. An on-line manual referred to as the **man** (manual) pages provides help on the many commands and programs that can be called from the shell. The syntax for invoking the `man` utility is as follows:

```
man command-name
```

The information regarding the specified `command-name` will then be displayed on the screen. To search for a keyword in the man pages, type:

```
man -K keyword
```

To find out more about the man utility simply type

```
man man
```

from the command prompt. There are several basic system commands which are commonly used. Many of the commands listed below have additional options which may be invoked. Consult the man pages for more information.

> Another fun way to learn the command line is to download and play **The Command Line Murders**, a game that requires you to access the command line to figure out whodunit. To install the game, visit https://github.com/veltman/clmystery.

1.4 Text Editors

Text editors are used to edit system configuration files as well as program source files. There are several simple text editors available with Linux distributions, including with the Raspberry Pi. These include a simple text editor named nano and some classic text editors like **Emacs** and **vim**. These editors can be used within a simple command line environment.

To get started, a very brief summary of the basic commands for these text editors are summarized below. In particular, the **Emacs** and **vim** editors possess more powerful and advanced features which the reader may wish to learn once the basics have been mastered. For more information, consult the man pages or the web.

1.4.1 Nano

The nano editor is a very basic command line editor. It is quite limited but also very easy to use. It is also included by default in all Raspberry Pi software distributions. To edit a file using nano, type:

```
nano filename.txt
```

where `filename.txt` is the name of the file you want to edit. If the file does not exist, it will be created. Other editors are available in the command line environment, like emacs and vi, which are simple but far more powerful alternatives to nano.

1.4.2 Emacs

One editor you may wish to consider using is Emacs, although Emacs is much more than just an editor. When running in a graphical environment, X-Emacs provides a toolbar menu of the various commands.

However, when running in a text terminal, the user must use various control keys to issue the editor commands.

The list of basic commands for Emacs is summarized in the table below:

Command	Description
Ctrl-x Ctrl-s	Save
Ctrl-x Ctrl-c	Exit
Ctrl-h	Help
Ctrl-h t	Emacs tutorial
Del	Delete preceding character
Ctrl-g	Cancel Command
Ctrl-x	Search (in the forward direction)
Ctrl-x	Move down one page
Ctrl-x u	Undo last change
Esc <	Move to the beginning of the file
Esc >	Move to the end of the file

Use the help command (`ctrl-h`) to list additional commands or for more information.

1.4.3 Vim

Another popular editor is the Vim editor. The name Vim refers to "vi improved" since it was inspired by the classic "vi" editor created for the Unix operating system by the computer engineer, Bill Joy. In 2018 it was found to be the most popular editor by readers of *Linux Journal*. Note that a playful rivalry between `vim` and `emacs` users sometimes referred to as the "editor wars".

To edit a file using `vim`, type:

```
vi filename
```

Vim has two basic modes of operation: *command mode* and *input mode*. Text may only be entered while in input mode. The user must hit the escape key to exit input mode and enter command mode before any editing commands are entered. The list of basic commands is summarized below.

Command	Description
↑↓→←	Move up, down, right, left (if terminal emulation is configured right)
0	Move to the start of the current line
$	Move to the end of the current line
escape key	Ends input mode and returns to command mode.
a	Appends text after current position and enters input mode
A	Appends text at the end of the current line and enter input mode
i	Inserts text at current position and enters input mode
o	Inserts a line below the current line and enters input mode
rc	Replaces the current character with c
R	Replaces the current text with the text you type after R
x	Deletes current character
dw	Deletes a word
dd	Deletes current line of text
:q	Quits and exits from vi (only if no changes were made)
:q!	Quit and exits without saving
:w	Saves the file you are editing
:wq	Saves the file you are editing and quits
/pattern	Searches the file for text matching pattern

For those who are looking for a more "suped up" version of `vim`, try installing neovim. This can be installed using `apt` as follows:

```
sudo apt install neovim
```

To launch the `neovim` editor, simply type:

```
nvim filename
```

where `filename` is the name of the file you wish to edit.

1.5 Configuring the Raspberry Pi OS

The Raspberry Pi OS comes with a utility called `raspi-config` that can be used to configure a wide variety of settings and services. To run this utility, type:

```
sudo raspi-config
```

This will launch the Raspberry Pi configuration utility within the terminal with a main menu like the one shown below.

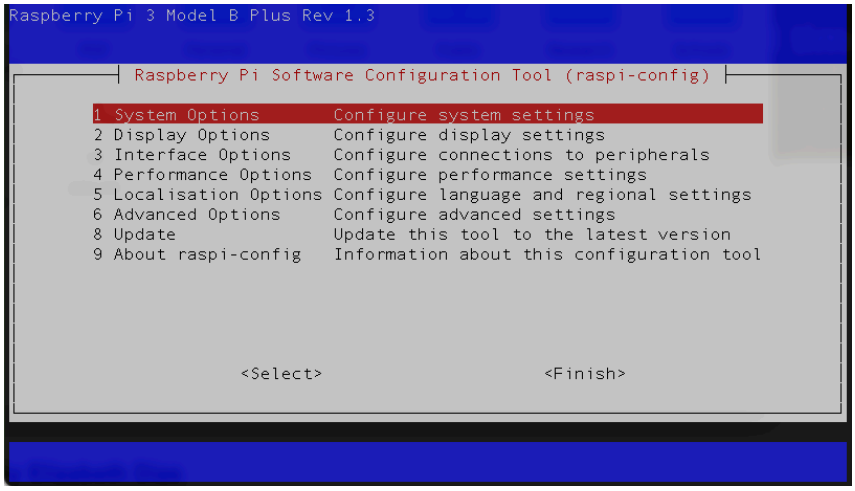


Figure 2: Raspberry Pi configuration utility

The Raspberry Pi configuration utility can be used to enable the camera interface as well as serial, I^2C, and SPI communications. It also includes an option to boot directly to the command line rather than the desktop.

1.6 Connecting Remotely to the Raspberry Pi

It is possible to run the Raspberry Pi *headless*, without a screen, keyboard, or mouse. This is often the case when using the Raspberry Pi in an embedded application or an IoT (Internet of Things) configuration. The following subsections describe how to connect to your Raspberry Pi *remotely* using one of the following options:

- using a USB-to-TTL serial cable
- using SSH over an Ethernet or Wi-Fi connection
- Raspberry Pi Connect service

Some models of the Raspberry Pi can be connected using a USB cable. This works by enabling *USB Gadget Mode*, which allows a USB port to present itself as a variety of different types of devices. This

is known to work with the Raspberry Pi Zero and not with most other models. The two approaches described below should work with all models of the Raspberry Pi.

1.6.1 Connecting using a USB-to-TTL serial cable

A USB-to-TTL serial console cable can allow you to log into your Raspberry Pi from another computer using a serial link. A serial port console must be first enabled on the Raspberry Pi using `raspi-config`. To enable the serial console, type:

```
sudo raspi-config
```

The console service can then be enabled in the `raspi-config` menu under *Interface Options* → *serial*.

Next, a suitable USB-to-TTL serial console cable must be connected to the proper pins on the GPIO port. With the power off, start connecting the cable wires to the GPIO pins on the Raspberry Pi as follows:

- black lead to GND on the GPIO port
- white lead to TXD on the GPIO port
- green lead to RXD on the GPIO port
- Leave the red lead disconnected!

A diagram showing the serial cable connections to the GPIO port is shown below.

> Note: Ensure that the Raspberry Pi is powered **off** before adding or removing the USB serial cable pins from the GPIO port. In general, always power down before connecting or disconnecting circuits to the GPIO port pins!

Next, connect the USB-to-TTL serial console cable to a USB port on a Linux workstation. Type the following command in a terminal window on the workstation:

```
screen /dev/ttyUSB0 115200
```

where `/dev/ttyyUSB0` is the serial port device and `115200` is the bit rate (or baud rate) of the serial connection. The `screen` program is a serial communications program that can use the USB port to communicate with the Pi. Power up the Pi and, after a few moments, hit enter a few times. It may take a few moments for the serial connection to "sync," and you may initially see "gibberish" on the screen. If the connection has trouble syncing, try re-inserting the USB cable or hitting return a few more times. A login prompt should eventually appear. Log into your Raspberry Pi and confirm that the serial connection is working.

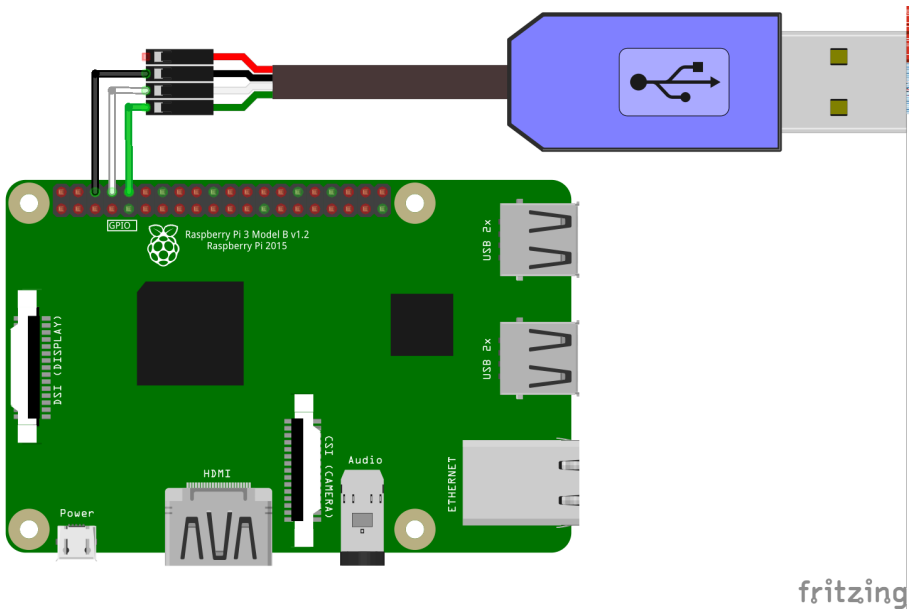


Figure 3: USB-to-TTL serial cable GPIO wiring details

> **Note:** Here the serial port device is assumed to be `/dev/ttyUSBB0`, however the USB device name may be different on your system. While Linux includes all necessary drivers to use the USB-TTL serial cable, OSX or Windows may require installing extra drivers. Moreover, OSX and Windows will require using a different serial communication program.

1.6.2 Connecting over Ethernet or Wi-Fi

The previous two approaches allow remote connections to the Raspberry Pi, but the remote distance is limited to the length of the serial or USB cables being used. It is also possible to use a network connection which enables secure, high speed remote connections over much greater distances.

The Raspberry Pi can be connected to a network via the Ethernet port or with Wi-Fi. To use the Ethernet connection, simply insert an Ethernet cable connected to a network using DHCP (Dynamic Host Configuration Protocol) and it should be automatically configured.

To use Wi-Fi, one can click the icon in the top right corner of the graphical desktop environment and select the access point and enter authentication details as required. If you are using the command-line, one can use the `raspi-config` tool to setup the Wi-Fi network by selecting *System Options* → *Wireless LAN*. From there, follow the prompts to setup the Wi-Fi interface.

Once a network interface is running, we can use SSH (secure shell) to connect to the Raspberry Pi over a network connection. Before you can use SSH, you need to enable the **secure shell server** in `raspi`

`-config`. The SSH service can be enabled in the `raspi-config` menu under *Interface Options* → *SSH*. Once the server is running, users can log in remotely using the **SSH** protocol.

Before we can connect, we need to know the IP address of the Raspberry Pi. The `ifconfig` program allows you to query the status of your WiFi or Ethernet network as follows:

```
ifconfig wlan0
```

If the Wi-Fi network is running, an IP address should be assigned to the `wlan0` adapter. For example, the first two lines should look like this:

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
       inet 10.1.2.3  netmask 255.255.192.0  broadcast 153.106.63.255
```

The decimal dotted number beside the `inet` label is your IP address. Make a note of the IP address (in the example above, the IP address is `10.1.2.3`). Note that if you were to use Ethernet, the IP address assigned can be queried by typing:

```
ifconfig eth0
```

> **Note:** The Wi-Fi and Ehternet IP address *may* change each time you boot if they are assigned by your router using DHCP leases.

Now that we have determined the Wi-Fi or Ethernet IP address of your Raspberry Pi we can connect to a shell from a remote computer using SSH. The SSH client is normally included with Linux and OSX operating systems. If you are using a recent version of Windows, you could use Putty (a common open source SSH client) or the Windows subsystem for Linux or WSL. WSL includes support for a basic shell (including ssh and many other commands) and can be launched by typing WSL in the start menu.

To make an ssh connection, open a terminal window on your remote computer and type:

```
ssh user@10.1.2.3
```

where `user` is the username and `10.1.2.3` is the IP address of the Raspberry Pi. If everything is working, you should be able to log in successfully using the username and password you configured.

You can also use `ssh` to transfer files between your computer and the Raspberry Pi using a tool related to SSH called *secure copy* (`scp`). To us `scp`, type:

```
scp filename user@10.1.2.3:
```

This will transfer a local file named `filename` to the home folder of `user` on the Raspberry Pi at IP address `10.1.2.3`. Note that you will be prompted for the password for `user` (unless you are using *ssh keys*).

Using `ssh` to connect to a Raspberry Pi has a few limitations. First, you can normally only access a Raspberry Pi on the same local network (if it is remote, you will need a "pinhole" opened on any firewalls to allow traffic to port 22, which presents possible security issues). Second, even if the Raspberry Pi is on the same local network, you still need to know its Wi-Fi or Ethernet IP address. In some situations, a static IP address can be configured to ensure the IP address remains fixed over time, but IP addresses are often "leased" from a DHCP server and can change over time. One could temporarily connect a monitor and keyboard to the Raspberry Pi to log in and determine the IP address, but that is cumbersome and impractical.

The next section discusses a tool that overcomes these challenges by providing a way to connect to the Raspberry Pi that can be done from anywhere in the world and requires no knowledge of its IP address.

1.6.3 Raspberry Pi Connect

Raspberry Pi Connect provides access to your Raspberry Pi from anywhere in the world. There is a "Lite" version that only supports remote shell access, and to install the Raspberry Pi Connect Lite software, type:

```
sudo apt install rpi-connect-lite
```

Next, use the `rpi-connect` command to start Connect for your current user as follows:

```
rpi-connect on
```

Similarly, when you want to stop Connect, run:

```
rpi-connect off
```

It is also recommended to enable "user-lingering" which allows you to log in remotely even when you are not logged in locally. Type:

```
loginctl enable-linger
```

Once Connect is running, use the following command to generate a link that will allow you to use the device:

```
rpi-connect signin
```

Point a browser to the link provided. If you don't already have a Raspberry Pi ID account, you will need to create one. Raspberry Pi ID also provides options for two-factor authentication (2FA) which you can enable for enhanced security.

Once you have a Raspberry Pi ID, you can follow the link and log in to your Raspberry Pi ID account. Choose a unique name to identify the device, click "Create device." You should receive an email notification that a new device is available. Once the device is setup, you will be able connect to a shell remotely from anywhere by pointing a browser to connect.raspberrypi.com and clicking on the Connect button next to the device name.

> If you are having problems connecting, try using a different browser, such as Google Chrome.

> **Warning**: If you receive an email reporting a strange sign-in on Connect, immediately change your Raspberry Pi ID password and remove the device from your account. Consider enabling two-factor authentication for greater security.

To learn more, visit Raspberry Pi Connect.

1.7 Remote Editing with Vscode

Using text editors like `nano` on the Raspberry Pi can be cumbersome. Another option is to make use of a more elaborate editor on a desktop or laptop and *remotely* edit files on the Raspberry Pi. One such program for doing this is Visual Studio Code, or simply *vscode*. Vscode is a general-purpose editor that provides a variety of useful features and plugins for editing programs.

While vscode can be installed locally on the Raspberry Pi, it can also be used for remote editing on desktop or laptop (running Windows, OSX, and Linux). After launching vscode, perform the following steps:

- type `ctrl-shift-x` to bring up the vscode extensions
- type "Remote SSH" in the search box
- select and install the *Remote-SSH extension*

After the installation is complete, you should see a new green "connect" icon appear in the bottom left corner of vscode. Click the green connect icon and select "Remote SSH: Connect current window to host". Next, enter the SSH connection details, for example, `user@10.1.2.3` where `user` is your username and `10.1.2.3` is IP address of your Raspberry Pi. Finally, you will be prompted to enter your Raspberry Pi password and then you will need to wait briefly as vscode initializes and establishes a connection.

Once the setup and initialization is complete, click on the link to "open folder" (or select File→Open Folder) and your home folder on the remote Raspberry Pi should appear! Select the folder you wish to work within and click "OK." This will become your current "working folder" on the Raspberry Pi. A list of files should appear on the left where you can select and open source files for editing. Vscode

provides a quick and convenient way to ssh into the Raspberry Pi by typing `ctrl+'` which opens a new remote terminal window to the Pi. In the terminal window you can compile and run the programs you edit.

Note how much more delightful it is to use an advanced editor over SSH rather than editing code locally with the `nano` editor.

1.8 Proper Shutdown

A final note regarding proper shutdown of the Raspberry Pi. One should never just remove power from the Raspberry Pi since this can lead to corruption of the SD card. In order to perform a proper shutdown of the Raspberry Pi, type the following command:

```
sudo halt
```

Once Linux has shutdown and the green activity LED has stopped flashing, you may unplug the power supply. Alternately, you may reboot the Raspberry Pi from the command line by typing:

```
sudo reboot
```

2 Introduction to Programming Languages

Grace Hopper is a famous pioneer in computer science who is often referred to as the person "who taught computers to talk." Her pioneering work led to the development of the *compiler*, a program that translates higher level instructions into the primitive machine code that computers can execute directly. Her work contributed to the development of the COBOL programming language and blazed a trail for the development of future programming languages. For this reason, Grace Hopper has sometimes been referred to as "Amazing Grace."

Figure 4: Grace Hopper, U.S. Navy, 1984 (public domain)

The Raspberry Pi repositories include support for COBOL, as well as a rich set of more modern programming languages such as Python, C, C++, and Java. It has support more niche and legacy programming languages.

Generally, the paradigms for programming languages fall into three general categories:

- procedural programming languages
- object oriented programming languages
- functional programming languages

2.0.1 The Python Programming Language

Python was invented in the early 1990's by Guido van Rossum. Python can be written using either a procedural or object oriented paradigm. It is an open source project that is widely available, uses simple syntax and includes rich libraries and offers a variety of programming tools, Python source code is run on a *virtual machine* which translates code into the specific machine-code executed by the processor. Because Python is interpreted by a virtual machine, it is *platform independent*.

The Raspberry Pi should have Python installed by default and can run Python programs directly from the command line. To enter a program, you first need a plain text editor. If you are using a desktop environment, there is a friendly, graphical, integrated development environment (IDE) for Python suitable for beginners called Thonny. To install Thonny, type:

```
sudo apt install thonny
```

For more advanced users in a graphical environment, the vscode program provides an excellent editor for coding in a variety of different languages, including Python. As described earlier, vscode can also be run to edit files remotely.

If you are using the command line, you can use any of the command line editors described in the preceding sections, including vi, emacs, or nano. For example, to edit a Python source file called hello.py, type the following:

```
nano hello.py
```

Next, enter the following code into the source file:

```
name = input('What is your name? ')
print('Hi', name,' welcome to the Raspberry Pi!')
print('Good Bye')
```

Next, save and exit nano and run the file by typing:

```
python3 hello.py
```

The program should run as expected.

2.0.1.1 Plotting in Python

Matplotlib is a nifty Python plotting library which produces publication quality figures including plots, histograms, power spectra, bar charts, error charts, scatter plots, and more. Of course, using Matplotlib presupposes a graphical desktop environment in order to display the plots. To install matplotlib, type:

```
sudo apt install python3-matplotlib
```

Once the library is installed, it can then be imported and used in a Python program. For example, the following code takes two lists and plots them in an x-y scatter chart shown in the figure below.

```
import matplotlib.pyplot as plt

xdata = [1, 2, 3,  4,  5, 6, 7, 8]
ydata = [1, 4, 9, 16, 25, 36, 49, 64]

plt.plot(xdata, ydata)
plt.xlabel('some numbers')
```

```
plt.ylabel('some squares')
plt.show()
```

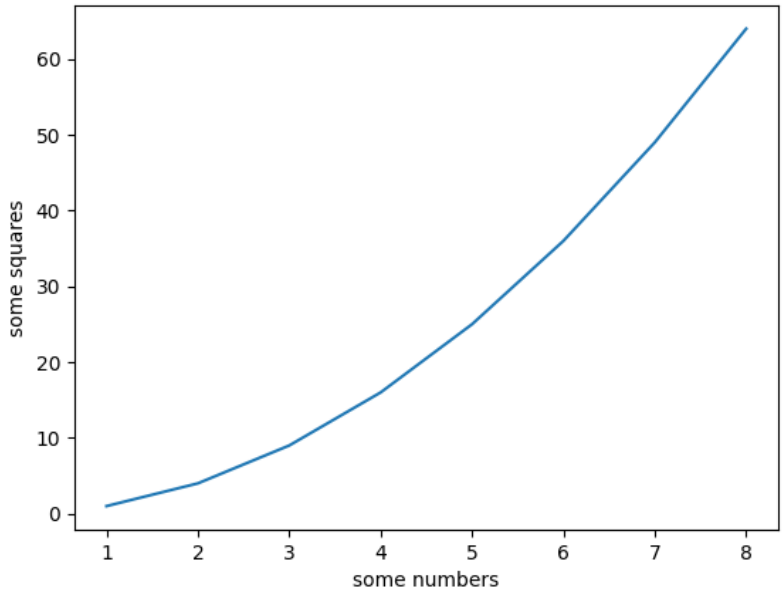


Figure 5: Example of a simple matplotlib plot

Further examples of matplotlib in action can be found on the matplotlib documentation pages.

2.0.2 Setting up a Python Virtual Environment

For most situation, one can install Python globally with all libraries and dependencies. However, there are times when tinkering with Python that you may want to isolate all the settings from your system-wide settings. Making changes to system-wide Python settings can lead to instabilities in the Python environment or cause conflicts with other system-level applications. A virtual environment uses its own isolated installation directories and Python packages. This is particularly useful if you are developing new libraries or experimenting with bleeding-edge Python packages and releases. In essence, a Python *virtual environment* gives you a "sandbox" in which you can adjust settings, libraries, and Python versions without impacting any of your other settings. According the the Python documentation, "Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages…"

To setup a Python virtual environment, we can use the built-in `venv` module as follows:

```
python -m venv env_name
```

where `env_name` is a name for the new virtual environment. To *activate* this new virtual environment, type:

```
source env_name/bin/activate
```

Once a virtual environment is activated, you can install any packages or libraries you wish, and these will be isolated within the virtual environment. To install a package in the new virtual environment, type:

```
pip3 install package_name
```

where `package_name` is the name of the package you want to install in the virtual environment. To list the packages and version currently installed, type:

```
pip3 list
```

To leave a virtual environment, simply type:

```
deactivate
```

Once a virtual environment is established, one can *freeze* the requirements into a file as follows:

```
pip3 freeze > requirements.txt
```

To rebuild a virtual environment from the requirements file, type:

```
pip3 install -r requirements.txt
```

An alternative to the `pip3` tools for virtual environments is the `uv` tool. To install `uv`, type:

```
sudo apt install uv
```

To begin a project, type:

```
uv init project_name
cd project_name
```

where `project_name` is the name of the project. Suppose you need a project that requires the `flask` library. In this you simply type:

```
uv add flask
```

Other dependencies can then be added as needed. Note that a project file named `pyproject.toml` will be created in the folder with information about the Python version and all the library dependencies. Finally, to run the project, simply type:

```
uv run myprogram.py
```

and the proper Python version and libraries will be made available to run the program. The `uv` tool can be used to query and install different versions of Python. To list the installed and available Python versions, type:

```
uv python list
```

To install another version of Python, type:

```
uv python install 3.13
```

2.0.3 Using JupyterLab Notebooks

A Jupyter notebook provides a sharable document that combines Python code, plain language documentation, data, along with charts and graphs.

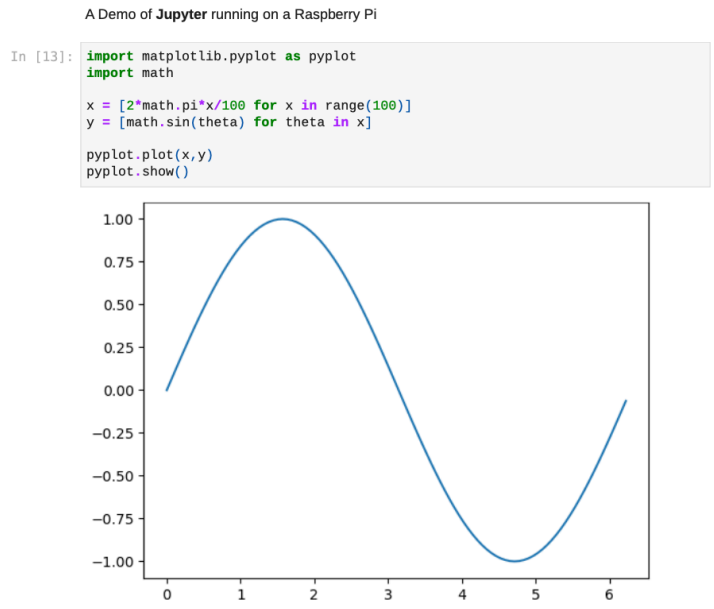


Figure 6: Sample Jupyter notebook running in a web browser.

To install JupyterLab, type:

```
sudo apt install jupyter
```

To start JupyterLab, open a terminal window and enter the directory in which you wish to work. Next, type:

```
jupyter notebook
```

This should will start the jupyterlab running on your Raspberry Pi. Leave the terminal window open and open a browser and point it to the address http://localhost:8888. You can begin a new Python notebook or open an existing notebook.

For more information on using Jupyter, visit https://jupyter.org.

2.0.4 Python Drill Exercises

1. Write a loop that counts from 100 down to 0 backwards by 2 using a for loop and then a while loop.
2. Write a function called `triangle` that takes one argument specifying the height of a triangle. Use a nested loop to print a triangle using the '*' character where the number of lines corresponds to the height parameter. For example, given an input of 8, the function should print:

```
*
**
***
****
*****
******
*******
********
```

3. Write a program that chooses a random integer between 1 and 10 and prompts the user to guess the number, indicating whether the guess is too high, too low, or correct.
4. Given the following block string:

```
rhyme = \
'''Twinkle, twinkle little star
how I wonder what you are
up above the stars so high
like a diamond in the sky'''
```

Write a code to print out each of the following information:

- the first 7 letters and also the 22nd to the 28th letters
- test if the word "hat" appears int the string
- a new string where all occurrences of "star" are replaced with "planet"
- the number of letters
- the number of words
- a sorted list of all the words

5. Write a function that prints the first 100 Fibonacci numbers in a neat left-justified and aligned table. The first column is an integer sequence signifying the index of the Fibonacci number (1,2,3 and so on) and the second column is the actual Fibonacci number.
6. Write a function called lastWord that takes a sentence string as an argument and returns the last word in the sentence.
7. Install a list of dictionary words as follows:

```
sudo apt install wamerican
```

This command installs a list of dictionary words for the Raspberry Pi OS in the file `/usr/share/dict/words` where each line in the file is a dictionary word.

Next, write a program to generate a list of 10,000 random 5 letter words using lower case letters only. To select random letters use the choice function from the random library as follows:

```
random.choice("abcdefghijklmnopqrstuvwxyz")
```

Finally, write a program that takes each random "word" and searches the dictionary and print all the random words which are real words found in the dictionary. What percentage of the random words are real words?

8. Create a dictionary of names and ages as follows:

```
info = { 'Bob': 23, 'Larry': 16, 'George': 37, 'Kim': 19, 'Lisa': 45 }
```

Using the dictionary above, do the following:

- Write an expression to give the length of the dictionary
- Write an expression to print Kim's age
- Remove Larry from the dictionary
- List all the keys and then all the values in the dictionary

9. The following steps revolve around a Python class.

a) Write a Python class called `Book` that represents a library book. The book class should define the following object variables with appropriate accessors and mutators:

- author
- title
- call number
- publisher
- publication date

b) Next, create another class called Catalog that holds a dictionary of all the books in the library (with the key being the call number of the book and the value being a book object).
c) Create a class called "Borrower" that represents a library client with a library card. The class should include a name, a card number and a list of books that are currently signed out along with appropriate accessor and mutator methods.
d) Create a list of "borrowers" and have them take out several books.

10. Create a class named `Fraction` to represent a fraction and add the following methods:

- add a constructor that sets the numerator and a denominator
- add a `__str__` method
- add accessor and mutator methods for the numerator and denominator
- create method to overload the multiplication operator so that it works with fractions
- write code to test your class

2.1 Compiling and Running a C/C++ Program

Linux has a variety of tools to support software development in both C and C++. In fact, the Linux operating system itself is written in C. The C programming language is a procedural language, and C++ builds on C to provide support for object oriented programming. The compilers which we will use under Linux are the GNU C Compiler (`gcc`) and the GNU C++ compiler (`g++`). To ensure the GNU C/C++ compiler tools are installed, type:

```
sudo apt install gcc g++ gdb build-essential
```

For example, to enter a simple C program named `hello.c` using the `nano` editor, type the following:

```
nano hello.c
```

Using the editor, enter the following code into the source file:

```
/* A Raspberry Pi C program */
#include <stdio.h>

int main(void)
{
   printf("Hello world.\n");
   printf("Compiled and run on a Raspberry Pi.\n");
   return 0;
}
```

Save and exit the editor. To compile your source code type the following at the prompt in a terminal window. For example, to compile the `hello.c` program above, you can enter:

```
gcc -Wall -o hello hello.c
```

The gcc compiler has numerous other command line options which you may use. For more information consult the man pages. Note that the C++ compiler can be invoked by using g++ in place of gcc.

To run the compiled program in the current working directory, do not forget to specify a ./ in front of the program name to specify the path as the current directory. For example, to run the hello.c program after compiling it as ini, type:

```
.\hello
```

If no output filename was provided to the compiler the default output filename will be a.out.

2.1.1 Using a C Debugger

Debugging is frequently part of the process of programming. There are several techniques to debug code. Sometimes debugging may be accomplished by sprinkling printf statements throughout your code to display the state of your program and variables as it executes. Another way to debug code is to use a special debugging tool that allows you to see what is going on inside your program while it executes. The gcc compiler has a powerful debugger called gdb (the GNU debugger). Most integrated development environments (IDEs) provide a friendly interface for using the debugger. In this tutorial we will look at using the debugger from the command line.

To demonstrate gdb in action, create the sample program as shown below to perform simple calculations based on user input.

```
/* Sample C program
An Introductory Guide to Using Linux with the Raspberry Pi
This program multiplies 3 numbers provided by the
user and prints the result to the display  */
#include <stdio.h>

int main(void)
{
   float x,y,z;

   printf("\nWhat is the first number? ");
   scanf("%f",&x);
   printf("\nWhat is the second number? ");
   scanf("%f",&y);
   printf("\nWhat is the third number? ");
   scanf("%f",&z);
   printf("\n%f * %f * %f is equal to %f\n",x,y,z,(x*y*z));
   return 0;
```

```
}
```

Practice using the debugger with the code you wrote in the previous step. Load the source file and then re-compile it using the following command:

```
gcc -Wall -g multiply.c -o multiply
```

Note that the extra `-g` option tells the gcc compiler to include information in the output file for use by the debugger. To start the GNU debugger, type:

```
gdb multiply
```

At the prompt in the debugging window you can now enter various commands. For example, to set a breakpoint for the start of the `main()` function by type:

```
break main
```

A breakpoint stops program execution allowing you trace your program step-by-step. To begin running your program type:

```
run
```

The program will halt at the first line in your program which will be high-lighted in the source file. To display the contents of the x, y and z variables type:

```
display x
display y
display z
```

Note that before these variables are initialized they contain "garbage" values. This will show why it is always a good idea to initialize variables to a known value. You may begin single-stepping through each line of your code by typing "s" (for "step") in the debugging window.

When your program ends, you may quit the debugger by typing:

```
quit
```

The debugger includes several additional commands that are useful when debugging code. Some of the gdb commands which can be entered after the (gdb) prompt are listed in the table below:

gdb Command	Description
`run`	Start running a program
`break` *function*	Set a breakpoint at the start of a function
`break` *line number*	Set a breakpoint at a line number

gdb Command	Description
`info break`	list all breakpoints
`disable` *breakpoint*	disables a breakpoint
`enable` *breakpoint*	enables a breakpoint
`continue`	resume running the program
`list`	list the next source lines
`next`	execute the next statement
`step`	same as next, but step into a function
`print` *variable*	print the value of a variable
`display` *variable*	display the value of a variable after each step
`set` *variable=value*	assign a new value to a variable
`help`	display a list of gdb commands
`help command`	display help on a specific gdb command
`quit`	quit gdb

The debugger can also be used to examine core dumps when segmentation faults occur. For more information on using the debugger, type help at the debug prompt or type `man gdb` for more information.

2.1.1.1 Strace Another debugging tool is the `strace` program which can use used to trace systems calls made during the execution of a compiled program. For example, to trace the execution of the `hello` program in the previous section, type:

```
strace ./hello
```

2.1.2 Library Documentation

The `man` pages include documentation on most of the functions found in the various C Libraries. To display information on a C function type the following:

```
man -S3 function_name
```

where `function_name` is the name of the function you want documentation for. Documentation also includes the names of the `#include` header files that must be included for a given function.

2.1.3 Other Tools for C and C++

The `make` utility can be used to automatically determine which pieces of a large program need to be recompiled. Type `man make` for more information.

Another program to assist C programmers is `splint`, a tool for statically checking C programs for security vulnerabilities and coding mistakes. To use `splint`, it should be first installed as follows:

```
sudo apt install splint
```

Once it is successfully installed, it can be invoked as follows:

```
splint source.c
```

where `source.c` is the name of your C source file.

2.2 Compiling and Running Java Programs

It is also possible to develop and run Java programs using Linux on the Raspberry Pi. There are two different packages which can be installed: on provides the Java Runtime Environment (JRE) and another provides the Java Development Kit (JDK). The JRE just allows you to run Java programs, but the JDK enables one to compile and run Java programs.

To install the OpenJDK Java development kit, type the following:

```
sudo apt install default-jdk
```

Once this is installed you can compile a Java program. For example, enter the following simple Java program named `hello.java` using a plain text editor:

```
/* A Java program */
class Main {
public static void main(String args[]) {
      System.out.println("Hello world.\n");
   }
}
```

Compile the program by typing the following at the prompt:

```
javac hello.java
```

where `myprogram.java` is the name of a Java source file. To run a Java program, type the following at the prompt:

```
java Main
```

where `Main` is the name of the class where the program begins.

2.3 Other Programming Languages

The Linux platform provides a plethora of open source developments tools and programming languages. The options range from fashionable programming languages to more niche languages as well as many historical programming languages. The following table lists some additional programming languages that are available as well as the name of the corresponding package in the `apt` repository.

Programming Language	Package Name(s)	Description
Fortran	`gfortran`	GNU Fortran project implementation of FORTRAN 2018
PHP	`php`	scripting language for web development
Nodejs	`nodjs`,`npm`	JavaScript runtime environment
Perl	`perl`	classic scripting language
C/C++	`gcc`,`g++`	GNU C/C++ compiler
Python	`python3`	Python version 3
Ruby	`ruby`	high-level, interpreted programming language
Java	**`default`**`-jdk`	OpenJDK Java compiler and virtual machine
Gambas	`gambas3`	object-oriented version of BASIC programming language
Mono	`mono-complete`	open-source .NET compatible software framework
Scheme	`mit-scheme`	dialect of the Lisp family of programming languages
Prolog	`swi-prolog`	classic logic programming language
Tcl	`tcl`,`tk`	Tool Command Language (pronounced “tickle”)

2.4 Comparing Runtime Efficiency of Different Programming Languages

One advantage of C is that it is *compiled* to machine code, a process by which the code is converted to native machine language (in the case of the Raspberry Pi, the native code for the ARM processor).

In contrast, Python runs using an *interpreter* which translates the Python code, step-by-step, into the local machine code. As such, Python has additional overhead at run-time.

The difference in execution time can be demonstrated by running the same algorithm implemented in three different programming languages: C, Python, and Java.

2.4.1 Measuring execution time

The runtime of a program can then be determined using the special `time` utility. For example, the following command:

```
time sleep 3
```

returns the time taken for the command to execute. In this example, it will return something like the following:

```
sleep 3  0.00s user 0.00s system 0% cpu 3.002 total
```

This shows the `user` time (the number of CPU seconds spent in user mode), the `system` time (the number of CPU seconds spent in kernel mode), and the `cpu` time (the elapsed "wall clock" time).

The `time` utility can resolve runtimes on the order of a millisecond or so. Hence, comparing execution times requires running programs that require a non-trivial execution time, ie. much greater than a millisecond. For our runtime comparison, we will implement a classic numerical integration algorithm. The approach to numerical integration we will use is sometimes referred to as a *Riemann summation*. In this approach, the area under a function $f(x)$ over the interval $[a, b]$ can be approximated by summing a series of n rectangles. The amount of computation time (and the accuracy of the numerical integration) is directly related to the number of steps, n. Thus, an integral can be approximated by the following:

$$\int_b^a f(x)\,dx \approx \Delta x[f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \cdots + f(a + (n-1)\Delta x)] = \sum_{i=0}^{n-1} f(x_i)\Delta x$$

where $\Delta x = \dfrac{b-a}{n}$. A plot illustrating this expression for $f(x) = \sqrt{x}$, $a = 1$, and $b = 5$ with $n = 10$ steps is shown below.

The Riemann summation for the integral $\int_1^5 \sqrt{x}\,dx$ using $n = 1000000$ can be computed in Python as follows:

```
import math

NUM_STEPS = 1000000 # number of integration steps
a = 1  # lower limit of integration
```

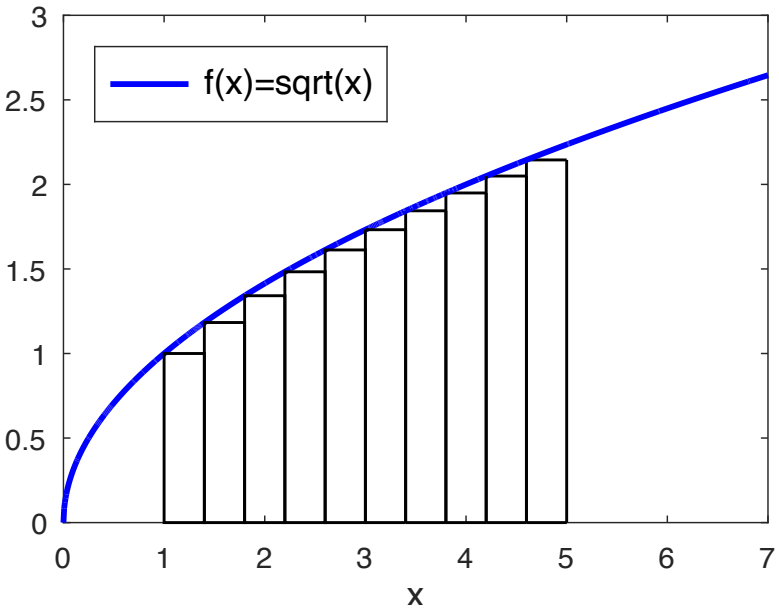


Figure 7: Illustration of integration using a Riemann sum of rectangles

```
b = 5  # upper limit of integration
DELTA_X = (b-a)/NUM_STEPS
sum = 0.0
x = a
for step in range(NUM_STEPS):
    y = math.sqrt(x)
    sum += y
    x += DELTA_X
integral = sum * DELTA_X
print(integral)
```

Thus, the runtime of the Python program can be determined as follows:

```
time python3 integral.py
```

where `integral.py` is the name of the Python program. The same program can be implemented in C as follows:

```
#include <math.h>
#include <stdio.h>
#define NUM_STEPS 1000000  // number of integration steps
#define a 1  // lower limit of integration
#define b 5  // upper limit of integration
#define DELTA_X  (double)(b-a)/NUM_STEPS

int main()
{
```

```
    double sum = 0.0;
    double y, integral;
    double x = a;

    for (int step = 0; step<NUM_STEPS; step++)  {
        y = sqrt(x);
        sum += y;
        x += DELTA_X;
    }
    integral = sum * DELTA_X;
    printf("%f\n", integral);
}
```

This program can be compiled and the runtime determined as follows:

```
gcc -Wall integral.c -lm
time ./a.out
```

The runtime performance of a C program can sometimes be further tweaked by making suggestions to the optimizer in the compiler. This can be done from the command line as follows:

```
gcc -Wall -Ofast integral.c -lm -o integral
```

The `-Ofast` command lime parameter instructs the `gcc` compiler to optimize for speed. Running the program again with this compiler option should generally provide some slight improvements in speed.

Finally, the same program can be implemented in the Java programming language as follows:

```
import java.lang.Math;

class Integrate {
    public static void main(String args[])
    {
        final int NUM_STEPS = 1000000; // number of integration steps
        final int a = 1;  // lower limit of integration
        final int b = 5;  // upper limit of integration
        final double DELTA_X = (double)(b-a)/NUM_STEPS;

        double sum = 0.0;
        double y, integral;
        double x = a;
        for (int step = 0; step<NUM_STEPS; step++) {
            y = Math.sqrt(x);
            sum += y;
            x += DELTA_X;
        }
        integral = sum * DELTA_X;
        System.out.println(integral);
    }
```

```
}
```

This program can be compiled and the runtime determined as follows:

```
javac integrate.java
time java Integrate
```

The algorithm used in this numerical integration is the same for all three of these programs. However, the C program runs many times faster than Python due to the fact that it is compiled and runs using native ARM instructions. Java provides “just in time compilation” and so adds overhead but runs efficiently once the compilation is complete. As one can observe, runtime efficiency can sometimes vary by orders of magnitude for different programming languages.

However, efficiency is just one consideration when selecting a programming language. Other considerations include maintainability, cost, interoperability, availability of tools, and community support.

3 Computer Organization and Assembly Language

3.1 Early Computers

One of the first concepts of a computer was developed by Charles Babbage in the mid 19th century. Babbage created the concept of an "Analytical Engine" that could perform basic computations using mechanical parts like gears, camshafts, and pinions. Babbage never completed the Analytical Engine, but it's design included many of concepts that were eventually incorporated into modern computers like a store for intermediate results and a unit for performing arithmetic processing. Ada Lovelace (daughter of Lord Byron) met Babbage at a party and was intrigued by the idea of his Analytical Engine. She later published an account of how it might be used to manipulate not just numbers, but symbols of letters and musical notes. For this reason, Babbage is considered by some to be the "father of the computer" and Ada Lovelace is often recognized as the first computer programmer. Moreover, the official Raspberry Pi mascot is a bear named Babbage.

Figure 8: The author next to a portion of Charles Babbage's Analytical Engine (on display at the British Science Museum, London, England)

3.1.1 The First Electronic Computers

The first general-purpose programmable electronic digital computer was completed in 1946 and called the ENIAC (Electronic Numerical Integrator And Computer). It was built by John Mauchly and J. Presper Eckert at the University of Pennsylvania for the U.S. military during World War II. It weighed 30 tons, occupied a 30-by-50 foot space, and was powered by thousands of vacuum tubes. The ENIAC was programmed by physically wiring interconnections between various components.

Figure 9: The ENIAC in a US Army Photo, 1946 (public domain) and a photo of the author posing next to a section of the original ENIAC on display at the University of Michigan Computer Science Department

Later, innovations in computer design led to the *Von Neumann architecture*, which enabled programs to the be stored in memory. Computers could then be programmed without altering the physical wiring of a computer by using software loaded into memory. This architecture was named after the mathematician and computer pioneer, John von Neumann, who formalized this structure in the 1950s.

3.2 Modern Computer Organization

The organization of modern digital computer typically includes the following elements:

- a processing unit

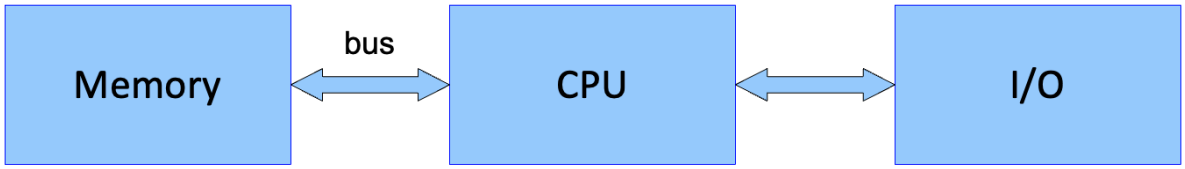


Figure 10: The von Neumann architecture.

- a control unit
- memory to store both data and instructions
- external storage devices
- input and output devices (I/O)

This basic architecture was first described by the computing pioneer John Von Neumann, and has since come to be referred to as the *Von Neumann architecture*. This architecture uses a shared *bus* for accessing both program instructions and data. A *bus* is a set of common (typically electrical) pathways for sending data. A more complex architecture includes a separate bus for instructions and data and is referred to as the *Harvard Architecture*.

In rest of this chapter takes a closer look at each of the components of the von Nwuemann architecture: the processing unit in a modern computer, the memory, and the input and output (I/O).

3.3 The Processor

The processor, or Central Processing Unit (CPU), represents the heart of the organization of a computer. The processor carries fetches and carries out instructions and performs calculations.

The Raspberry Pi is a single-board computer (SBC) built around an **ARM** processor (an acronym for "Advanced RISC Machines"). The Raspberry Pi combines an ARM processor with memory and various inputs and outputs as described by the von Neumann architecture. The inputs and outputs on the Raspberry Pi include USB ports, an Ethernet port, WiFi, HDMI video output, and a general purpose I/O (GPIO) port. Secondary storage is accomplished using a micro-SSD card. The result is a small package that fits in your hand and can run a modern Linux operating system.

A table summarizing the hardware specifications of some recent Raspberry Pi models is summarized in the table below.

Model	Processor	CPU Clock	RAM
Raspberry Pi 5	QUAD Core BCM2712	2.4 GHz	4GB or 8GB

Model	Processor	CPU Clock	RAM
Raspberry Pi 4	QUAD Core BCM2711	1.5 GHz	1GB, 2GB, or 4GB
Raspberry Pi 3B+	QUAD Core BCM2837B0	1.4 GHz	1 GB
Raspberry Pi Zero 2W	QUAD Core BCM2710A1	1 GHz	512MB

To find information about the CPU on your Raspberry Pi, type:

```
more /proc/cpuinfo
```

This will display detailed information about each processor core. Another nifty utility to display a useful summary of CPU information can be run by typing:

```
lscpu
```

Yet another utility is named `hardinfo` and can be installed as follows:

```
sudo apt install hardinfo
```

Once installed, running this utlity provides various hardware information along with a variety of benchmarks for the CPU.

Drill Exercise: Use the `lscpu` utility to answer the following questions:

- what kind of CPU do you have?
- how many cores does it have?
- what is the clock frequency?
- does the CPU support 64-bit instructions?

The architecture of a processor can be understood as having various levels. The bottom level, or **Digital Logic Level**, is comprised of the transistors that make up the logic gates used to implement the processor. The layer above is the **Microarchitecture Level**, is built on the digital logic level and implements the instruction set. The microarchitecture level can be implemented directly in hardware or it can be controlled by a *microprogram*. Above that is the **Instruction Set Architecture (ISA)** level which describes the set of instructions, registers, and operations a processor can perform which are carried out by the microarchitecture. The **Assembly Language Level** provides a symbolic form for the underlying level can be used to write programs.

In the following sections, we will take a lood at each of these levels in turn.

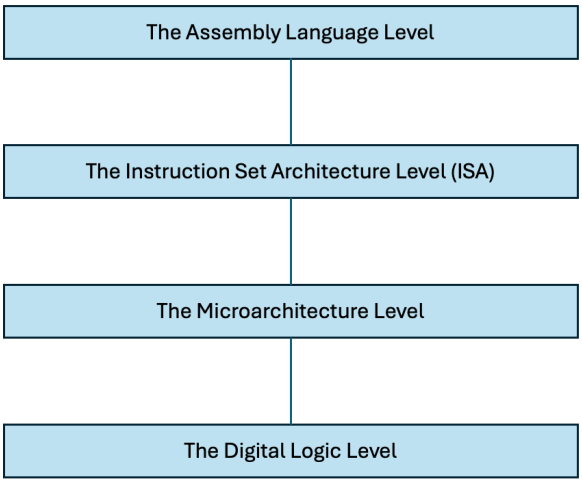


Figure 11: Processor architecture levels.

3.3.1 The Digital Logic Level

Modern digital computers are now constructed using solid state electronics constructed using transistors. These transistors can be placed on integrated circuits, enabling billions of transistors to exist on a single silicon chip.

Transistors can be used to create the basic building blocks of a computer: digital logic gates. The Raspberry Pi repositories includes a program for simulating digital logic called **logisim**. It can be installed from the command line as follows:

```
sudo apt install logisim
```

The program generally functions well but but the project is no longer being actively developed. Two alternative programs include logisim-evolution and digital. Any one of these programs can be used for experimenting and learning about digital logic. The interested read can begin by experimenting with basic logic gates and building up to more complex digital circuits like adders, flip-flops, and decoders.

For an example of simulating an entire simple CPU architecture, the author has a paper titled Step-by-step design and simulation of a simple CPU architecture. This simulation can be performed entirely on a Raspberry Pi using `logisim`.

3.3.2 The Microarchitecture Level

A processor's microarchitecture refers to its internal structure and how it fetches, decodes, and executes instruction. A processor typically includes a control unit, registers, and an Arithmetic Logic

Unit (ALU). The microarchitecture also defines the pathways between registers, the control unit, and the ALU. The pathway where data flows between registers and the ALU in a CPU is referred to as the *datapath*. The microarchitecture includes any other functional units like pipelines, caches, and other enhancements.

The ARM processor used in the Raspberry Pi is actually a *modified Harvard Architecture* since it includes separate instruction and data caches, allowing the processor to behave like a Harvard architecture when accessing the cache. However, behind the caches are shared buses for instructions and data.

The ARM processor is an example of a **RISC** (Reduced Instruction Set Computing) architecture, an approach to processor design that emerged in the early 1980s as the result of research performed at the University of California at Berkeley. The RISC architecture simplified the instruction set of a processor so that the instructions could run faster and consume less power and space on the chip. The RISC architecture stands in contrast to a **CISC** (Complex Instruction Set Computing) architecture, which generally requires more power due to their more complex architectures.

The microarchitecture implements the Instrcution Set Architecture (ISA) in the level above, which defines oeprations which are visible to an assembly language prograamer. This level will be described next.

3.3.3 The Instruction Set Architecture (ISA)

The Instruction Set Architecture (ISA) describes operations a processor can perform which are carried out by the microarchitecture. The Raspberry Pi uses an ARM processor, one that has its own Instruction Set Architecture (ISA). The ISA is a model that defines the set of instructions, registers, and operations that a processor can perform. The ARM processor has two different Instruction Set Architectures: a 32-bit architecture referred to as ARM32, and a 64-bit architecture referred to as ARM64. The 64-bit ISA is sometimes referred to as **AArch64**, and the 32-bit ISA is sometimes referred to as **AArch32**. Hence the Raspberry Pi OS has two different versions, a 32-bit version and a 64-bit version. All recent versions of the Raspberry Pi (since 2016) support the ARM64 architecture, but are backwards compatible with ARM32, allowing you to to execute 32-bit applications on ARM64 processors.

The ISA defines the instructions that form the assembly language level which is described in the next section.

3.3.4 The Assembly Language Level

Assembly language is a *machine-specific* programming language with a one-to-one correspondence between its statements and the computer's native machine language. Programs written in Assembly Language are translated into the native *Machine Language* using an *Assembler*. Assembly language

programming is difficult since it takes longer to write, is more difficult to code, debug, and maintain. Assembly Language is helpful when learning about computer organization and architecture. In the case of the Raspberry Pi, ARM assembly language requires understanding some of the basics of the ARM architecture. Furthermore, assembly language can be useful to bypasses restrictions and limitations of higher level languages and for high-speed or time-critical applications that require direct communication with hardware.

To begin editing an assembly language program, type the following:

```
nano program1.s
```

The assembly code you enter will vary depending on whether you are running a 64-bit or 32-bit OS. To determine which version of the Raspberry Pi OS you are running, simply type:

```
uname -m
```

`aarch32` indicates a 32-bit OS and `aarch64` indicates a 64-bit OS. Assuming a 64-bit OS, enter the assembly language program as follows:

```
// ARM64 Assembler program to print a greeting
.data
message: .ascii  "Hello World!\nAssembled for a 64-bit Raspberry Pi.\n"
.text
.global _start
_start:
// Linux system call to print message
    mov     w0, #1          // File descriptor 1 (stdout)
    ldr     x1, =message    // Load address of message
    mov     w2, #57         // Message length
    mov     w8, #64         // system call for write operation
    svc     #0              // perform system call

// Linux system call to terminate program
    mov     w0, #0          // Return code 0 (success)
    mov     w8, #93         // exit system call
    svc     #0              // terminate program
```

An ARM32 version of the same program would appear as follows:

```
@ ARM32 Assembler program to print a greeting
.data
message: .ascii  "Hello World!\nAssembled for a 32-bit Raspberry Pi.\n"
.text
.global _start
_start:
@ Make a Linux system call to print the message
   mov r0, #1        @ File descriptor 1 (stdout)
   ldr r1, =message  @ Load message address into r1
   mov r2, #57       @ Length of message in bytes
```

```
    mov r7, #4          @ System call number 4 (write)
    svc 0               @ perform system call

@ Make Linux system call to terminate program
    mov     r0, #0      @ Return code 0 (success)
    mov     r7, #1      @ System call number 1 (exit)
    svc     0           @ perform system call
```

One key difference between these two listings is the symbols used to denote comments, with the 32-bit code using an `@` symbol and the 64-bit code using the more traditional double slashes (`//`). I find this difference in assembler syntax puzzling, but the more significant differences are related to the difference in the ARM32 and ARM64 architectures. One of the key differences in the instruction set architecture is apparent by comparing the register names in these two programs. For the ARM32 ISA, there are thirteen 32-bit general purpose registers named `r0-r12` and the ARM64 ISA has 31 general purpose registers named `x0-x30` whose bottom 32-bits are named `w0-w30`. Another key difference between 32-bit and 64-bit Linux is the system call numbers. As an example, the `exit` system call for ARM32 is performed by placing a `1` in `r7` and for ARM64 a `93` is placed in `x8`. In addition, there are various other differences to both registers and instructions.

At this point we can assemble, link, and run the program by typing the following sequence of commands:

```
as -o program1.o program1.s
ld -o program1 program1.o
./program1
```

If everything was entered correctly, your program should run and you should see a message displayed. Next, type the following command to view your resulting object file:

```
objdump -S program1.o
```

The second part of the output will display a disassembly of the program, showing the machine code (in hexadecimal) alongside the corresponding assembly instructions.

Note how many primitive instructions are required to display a simple message in assembly language! To read more about ARM assembly language, see the Introduction to Assembly Language on the developer webpages for ARM.

3.4 Memory

There are basically two large categories of memory: - **Volatile memory:** storage that only maintains its data while the device is powered - **Non-volatile memory:** storage that preserves contents when power is off

3.4.1 Volatile Memory

There are two general categories of volatile memory:

- **SRAM:** static random-access memory
  - fast access time
  - typically more power hungry
  - typically used for caches and small embedded memories
- **DRAM:** dynamic random-access memory
  - slower than SRAM
  - denser (less costly) than SRAM
  - requires periodic refreshes (typically every 64msec)
  - often used for main memory

The Raspberry Pi utilizes SRAM for cache memory and its main memory is composed of DRAM.

3.4.2 Non-Volatile Memory

There are a variety of types of non-volatile memory:

- **EPROM:** erasable programmable read only memory
  - erased by exposing the chip to strong UV light, making field updates difficult
  - older technology
- **EEPROM:** electrically erasable programmable read-only memory
- **Flash memory and SSD**
  - Erased a “block” at a time
  - Limited number of program/erase cycles
- **Magnetic Disk drives**
  - Not well-suited for embedded systems
- **Magnetic Tape storage**
  - used for backup of large amounts of data

The Raspberry Pi has an SD card which is made from flash memory which comprises its secondary storage. Many mobile and embedded systems rely on flash memory. Flash memory devices have unique characteristics for writing: memory blocks have to be explicitly erased before they can be written to.

Unlike magnetic drives that have seek latencies due to the delays in the rotation of the disk to bring the required disk sector under the read-write head. In contrast, flash memory has no seek latency and have the advantage of random access. However, flash memories have a finite number of erase cycles, so they can wear out when a single block is repeatedly overwritten. To address this, flash file systems can implement "wear leveling" to spread out writes evenly among the memory blocks. An example of a file system that deals with this is F2FS (Flash-Friendly File System), which is designed specially for the characteristics of flash memory

> Flash memories can be corrupted if power if removed during a write operation. The green LED on the Raspberry Pi indicates when the SD card is being accessed, so *do not remove power until the green LED stops flashing*. Once Linux has completely shut down and the green LED has stopped flashing, you may unplug the power supply. Always perform a `sudo halt` before removing power from your Raspberry Pi.

To display details about the Raspberry Pi's memory, type:

```
more /proc/meminfo
```

Drill Exercise: Use the `meminfo` utility to determine:

- what is the total memory?
- how much free memory is available?
- how much swap space is allocated?

3.5 Inputs and Outputs (I/O)

The Raspberry Pi has general purpose I/O pins, referred to as **GPIO** pins, which can be configured as digital inputs or outputs. To display the pinouts of the GPIO port type the following from the command line:

```
pinout
```

The output should show various pinouts on the Raspberry Pi, including the GPIO headers, and should appear something like the following:

```
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
```

```
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21
```

Some of these GPIO pins can also be reconfigured for special purposes, such as for Pulse Width Modulation (PWM) or serial communications. For a more elaborate display of the Raspberry Pi GPIO pinouts and their special functions, visit the website pinout.xyz.

Note that extreme care should be taken when connecting to GPIO ports. Interfacing the pins improperly, exceeding voltage input ratings, or shorting out certain pins can damage or destroy the Raspberry Pi!

The GPIO pins can be controlled as simple inputs or outputs using a command line tool. They can also be controlled from within a program (which will be explored further in chapter 7). As an example, we can set GPIO12 to act as an input from the command line as follows:

```
raspi-gpio set 12 ip
```

We can also turn on a weak internal "pull up" resistor using the pu parameter. This ensures that when the pin is unconnected the input value "floats" high.

```
raspi-gpio set 12 pu
```

Alternately, we can turn on a weak internal "pull down" resistor using the pd parameter. This ensures that when the pin is unconnected the input value "floats" low.

```
raspi-gpio set 12 pd
```

We can read the input state using the gpio command on GPIO12 as follows:

```
raspi-gpio get 12
```

We can repeatedly call this command to monitor any changes to the input and observe the state of the input pin. To learn more about the gpio utility, type:

```
raspi-gpio help
```

In a similar manner, we can use the command line to control digital outputs. For example to set GPIO16 as an output, we can enter the following command:

```
raspi-gpio set 16 op
```

Next, if we want to set the output of GPIO16 high, we issue the following command:

```
raspi-gpio set 16 dh
```

Likewise, the command to set GPIO16 low is:

```
raspi-gpio set 16 dl
```

4 The Linux Operating System

An operating system is software to control the hardware of a computer. The Raspberry Pi runs with the Raspberry Pi OS, a variant of the free *Linux* operating system originally written by Linus Torvalds when he was a graduate student at the University of Helsinki in Finland. Linux is inspired by the Unix operating system, first developed in 1969 at AT&T's Bell Labs, and is part of the long history of Unix-like operating systems (see the figure below). But unlike Unix, Linux is completely free and distributed under the GNU general public license (GPL). Linux aims to be POSIX (Portable Operating System Interface) compliant, a standard for Unix-like operating systems. Linux has grown over time with support from developers around the world and continues to grow increasingly popular, being found not only in desktops and servers, but smartphones and tablets, embedded systems, and in factory automation systems.

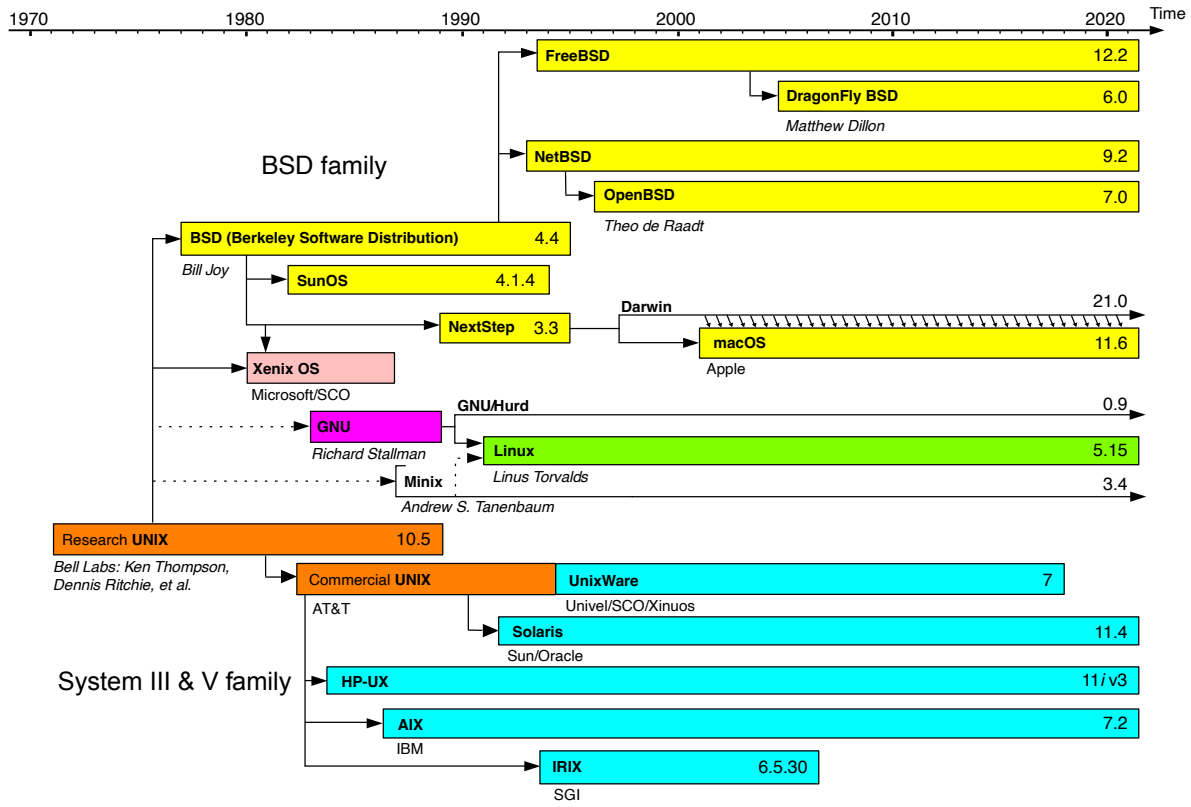


Figure 12: History of Unix-like operating systems and Linux (public domain image)

4.1 Introduction

In simple terms, an Operating System (or **OS**) makes computing power available to users by controlling the hardware resources. In general, an OS is responsible for the following functions:

- Process Management: process creation, scheduling, switching, and synchronization
- Memory Management: allocation, swapping, page management
- File Management: accessing and organizing files and directories on a secondary storage device
- Input/Output (I/O) Management
- Other Support functions: accounting, monitoring, updates

4.2 Process Management

A *process* is a program in execution which is managed by the OS. Modern operating systems like Linux are *multitasking* which allows for the interleaved execution of two or more processes on a single processor.

The execution of different processes is controlled by the *Dispatcher*. The Dispatcher is a program that switches execution from one process to another. The main job of the Dispatcher is to ensure that the processor time is allocated appropriately to all the processes that are active.

A process is normally represented by a *Process Control Block* (PCB), a data structure within the OS that contains information such as the process ID, its state, and other information required by the OS.

Linux is not only a multitasking OS, it is also a *multiprocessor* OS with support for systems with multiple processors or cores.

4.2.1 Tools for Managing Processes

What is a process? The following table lists some terminal commands related to processes:

Command	Description
`ps`	Display a list of processes
`kill <pid>`	Terminate the process with process ID `<pid>`
`htop`	Observe all processes running on the system

4.2.2 Example Program to fork a new process

A process which spawns another process is called a *parent* and the new process is called a *child*. In Linux, new processes may be started using one of several C library functions as summarized in the table below.

System call	Description
`system()`	executes a shell command string and waits for it to be completed
`exec()`	replaces the current process with another
`fork()`	duplicates the current process

A sample C programming demonstrating creating a child process using a call to `fork` is shown below:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
   pid_t pid;      /* Process ID */
   pid = fork();  /* Call to fork a child */

   switch (pid) { /* check for parent or child process*/
   case -1:
      printf("Unable to create process");
      exit(1);
      break;
   case 0:
      printf("Child Process started...\n");
      break;
   default:
      printf("This is the Parent Process...\n");
      break;
   }
   return 0;
}
```

4.3 Parallel Computation

There are different forms of parallel computing and these can be explored with the Raspberry Pi. Programs can run concurrently on a single CPU as it switches between tasks, or it can make use of mul-

tiple CPU cores. A computation can be also distributed across multiple computers, as is the case in computing clusters. The following sections explore different types of parallel computing.

4.3.1 Multithreaded programming

Multithreading refers to the ability of an OS to support multiple *threads* of execution within a single process. A *thread* is like a process that is independently scheduled but it shares the address space with other threads in a process.

To run a thread in Java, implement a class that extends the `Thread` class as illustrated in the following example code.

```
public class Main
{
  public static void main (String[]args) {
     HelloThread t1 = new HelloThread ("Bob");
     HelloThread t2 = new HelloThread ("Larry");
     t1.start ();
     t2.start ();
  }
}

class HelloThread extends Thread
{
   private String name;

   public HelloThread (String aName) {
      name = aName;
   }

   public void run () {
      for (int i = 0; i < 10; i++) {
         try
         {
            System.out.println ("Hello " + name);
            sleep (1000);   // Wait one second
         }
         catch (InterruptedException exception)
         { }
      }
   }
}
```

Python also includes support for multithreading. However, Python uses something called a GIL (Global Interpreter Lock) which requires that only one thread can run at a time. Hence, Python multithreading will *interleave* different threads of execution but cannot be used to perform computations in *parallel*.

4.3.2 Multiprocessing

Another approach is to use *multiprocessing*, a feature of the Linux operating system which can exploit the multiple *cores* typically found in modern processors like the ARM processor. A *core* is an independent processing unit that can be scheduled by the operating system. By running on multiple cores, it is possible to increase the speed of execution by looking for opportunities to parallelize execution. The Raspberry Pi 3, 4, and 5 include four ARM cores that we can be used to parallelize execution. In general, the *speedup*, S, that can be achieved from running a task in parallel is given by **Amdahl's law** as follows:

$$S = \frac{1}{(1-p) + \frac{p}{N}}$$

where p represents the proportion of execution time that can be parallelized and N represents the number of processors. Hence, with a four core processor ($N = 4$) and a program which can be fully parallelized ($s = 1$), the theoretical *speedup* should be four times. In reality, the theoretical speedup is never achieved due to the overhead of managing parallel processes.

Related to Amdahl's law is **Gustafson's law**, which can be described as:

$$S = s + pN = s + (1-s)N = N + (1-N)s$$

where S is the theoretical speedup of the program, N is the number processors, and s and p are the fractions of time spent on the serial and parallel portions of a task respectively.

Recall the example of numerical integration illustrated in Figure 7. As it turns out, numerical integration is an example of an *embarrassingly parallel* algorithm — one that is trivial to parallelize because the area can be broken down into separate chunks that are computed independently. Recall that numerical integration can be approximated using the Riemann summation follows:

$$\int_b^a f(x)\,dx \approx \sum_{i=0}^{n-1} f(x_i)\Delta x$$

where $\Delta x = \dfrac{b-a}{n}$. We could easily parallelize this integration over the range $[a, b]$ by subdividing it into four separate summations as follows:

$$\sum_{i=0}^{n-1} f(x_i)\Delta x = \sum_{i=0}^{x} f(x_i)\Delta x + \sum_{i=x+1}^{y} f(x_i)\Delta x + \sum_{i=y+1}^{z} f(x_i)\Delta x + \sum_{i=z+1}^{n-1} f(x_i)\Delta x$$

such that $0 < x < y < z < n-1$. Each of the above summations can run independently as parallel processes and the final answer can be obtained by summing the results of each.

4.3.2.1 Parallel Execution in Python Next, we will implement a program that computes these summations in parallel using each of the four cores on a Raspberry Pi. This can be implemented in Python using the multiprocessing module as follows:

```
from multiprocessing import Pool
import math

PROCESSES = 4  # number of processes
NUM_STEPS = 1000000 # total number of integration steps
a = 1  # lower limit of integration
b = 5  # upper limit of integration

def integrate_sqrt(a,b,n):
    ''' integrate sqrt(x) over[a,b] in n steps
    '''
    sum = 0.0
    x = a
    delta_x = (b-a)/n
    for step in range(n):
        y = math.sqrt(x)
        sum += y
        x += delta_x
    return sum * delta_x

if __name__ == '__main__':
    # start a pool of worker processes split over [a,b]
    with Pool(processes=PROCESSES) as pool:
        subrange = (b-a)/PROCESSES
        integrals=pool.starmap(integrate_sqrt,
                  zip([a+x*subrange for x in range(PROCESSES)],
                      [a+x*subrange for x in range(1,PROCESSES+1)],
                      [int(NUM_STEPS/PROCESSES)]*PROCESSES))
        print(sum(integrals))
```

Using the above code running on four processing cores available on our Raspberry Pi, we run and time the program as follows:

```
time python3 parallel_integral.py
```

We should observe a *speedup* of roughly four times over the sequential implementation given in section 2.4.1. Note that the speedup will not be exactly four times faster due to the finite overhead in creating and managing four new processes.

4.3.2.2 Parallel Execution in C There are a variety of libraries that can be used to perform multiprocessing in Linux using the C programming language. In this example we will use the OpenMP library. Some sample code is shown below:

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

#define NUM_STEPS 10000000  // number of integration steps
#define a 1  // lower limit of integration
#define b 5  // upper limit of integration
#define DELTA_X  (double)(b-a)/NUM_STEPS

int main()
{
   int step;
   double total_integral = 0.0;
   double sum;
   double y, integral;
   double x;

   printf("Number of cores: %d\n", omp_get_num_procs());
   #pragma omp parallel private(sum,x,y,integral)
   {
      sum = 0.0;

      // parallelize this chunk of code
      #pragma omp for
      for (step = 0; step<NUM_STEPS; step++)  {
         x = (double)a + (double)(b-a)*(double)(step)/NUM_STEPS;
         y = sqrt(x);
         sum += y;
      }
      integral = sum * DELTA_X;

      #pragma omp critical
      {
         total_integral += integral;
      }
   }
   printf("%f\n", total_integral);
   return 0;
}
```

This program can be compiled and the runtime determined as follows:

```
gcc -Wall -fopenmp integral.c -lm
time ./a.out
```

Once again, we can observe that there is a speedup over the sequential C code that was given in section 2.4.1.

4.3.2.3 Parallel Execution on a Graphics Processing Unit (GPU) Additional speedup can also be realized by utiliziing a GPU (Graphical Processing Unit). The Raspberry Pi's Broadcom ARM processor includes an on-chip graphics processing unit (GPU). There are standard libraries for taking advantage of GPUs, such as OpenCL and CUDA, but, as of the time of writing, programming support for the Raspberry Pi GPU is limited. Search the Raspberry Pi forums for more up-to-date information about programming with the GPU.

4.3.3 Parallel and Distributed Computing with the Raspberry Pi

The Raspberry Pi can also be used to teach and explore parallel and distributed computing (PDC). These topics can be explored using the Raspberry Pi with free, interactive, web-based PDC teaching modules.

For more infomation, consult the following links:

- Hands-on parallel & distributed computing with Raspberry Pi devices and clusters
- Distributed Computing Using Python and the Raspberry Pi
- Curriculum Initiative on Parallel and Distributed Computing – Core Topics for Undergraduates
- CSinParallel

4.4 File and Memory Management

4.4.1 Memory Management

Memory management is the task of dynamically subdividing memory to accommodate multiple processes. To view the state of the memory, type:

```
free -h
```

The output of this command includes the size of the *swap* memory, which is part of the *virtual memory* system. Virtual memory uses secondary storage to extend the main memory by temporarily swapping pages of memory in and out. Many virtual memory management settings are available via the `/proc/sys/vm/` folder. To see the settings that are available, consult the documentation for /proc/sys/vm/.

4.4.2 File Management

File management in Linux supports accessing, manipulating, and protecting files and directories on a secondary storage device. Linux uses a hierarchical file system structure which is organized into

files and directories within the root file system. Some of the shell commands for manipulating and accessing files are summarized in the table below.

Command	Description
`cd` *directory*	Changes the current working directory to *directory*
`pwd`	Displays the name of the current working directory
`mkdir` *directory*	Create a new directory called *directory*
`rmdir` *directory*	Removes the directory called *directory*
`ls`	Display a list of files in the current directory
`cp` *f1 f2*	Copy a file from the source *f1* to the destination *f2*
`rm` *filename*	Remove a file *filename*
`mv` *f1 f2*	Move a file from *f1* to *f2*
`df`	Report the amount of free disk space available
`du`	Report the amount of disk usage
`find` *path conditions*	Utility for finding files
`locate` *search-string*	Utility for locating files
`chmod`	Utility for modifying file permissions
`chown`	Utility for modifying file ownership

Linux has support for magnetic hard drives, solid-state drives (SSDs), as well as USB drives. While Linux supports a variety of different file system types, the most common type is the **Ext4** file system, which replaced the older **Ext3** file system. **Ext4** is a high performance *journaling* file system, meaning that it keeps a "journal" to keep track of changes *before* those changes are made to prevent data loss if a crash occurs during a data write.

To create a new **Ext4** file partition, type:

```
sudo mkfs.ext4 /dev/partition
```

where `/dev/partition` is the partition you want to create. The features of the **Ext4** file system can be viewed by looking at the `/etc/mke2fs.conf` configuration file. This file indicates which features are enabled along with settings like the default `blocksize`, `inode_size`, and `inode_ratio`. An *inode* represents an *index node*, which is the data structure that is used to store information about a file or directory.

> **Note:** Using the `mkfs.ext4` command needs to be done with caution since it will wipe any existing data on the partition!

4.4.2.1 Overlay File Systems An **OverlayFS** is a feature in the Linux kernel which allows a *union filesystem* in which one filesystem is *overlaid* on top of another. For example, this allows an SD card to be mounted read-only and overlaid with a *ramdisk* that can be used to writing (although data written to a ramdisk is lost when device is turned of or reboots). This is used in containerization technologies like Docker.

4.5 Controlling Inputs and Outputs

4.5.1 Software I/O Strategies

There are basically three strategies generally used for servicing I/O:

1. **Programmed I/O**: the CPU polls for completion for each I/O operation. This is a time-consuming task for the CPU. An example of programmed I/O for reading a switch connected to a GPIO input is given below. Note the continuous loop that constantly monitors the input state.

```
from gpiozero import Button

# Create a Button object with pull_up=True
button = Button(12, pull_up=True)
previous_state = False  # Keeps track of the last button state

try:
    while True:
        # is_active will be True when button is pressed
        if button.is_active and previous_state == False:
            print('Button press')
            previous_state = True
        # Check if button is released
        if not button.is_active and previous_state == True:
            print('Button release')
            previous_state = False

except KeyboardInterrupt:
    pass
```

2. **Event-driven I/O**: CPU can execute code during I/O operation but gets notified when I/O operation is done using *events* and *callback* functions. An example of a program using an event-driven input.

```
from gpiozero import Button
from signal import pause

button = Button(12, pull_up=True)
count = 0

def count_press():
    global count
    count += 1
    print(count)

# Trigger a callback whenever the button is pressed
button.when_pressed = count_press

try:
    pause()
except KeyboardInterrupt:
    pass
```

3. **Direct Memory Access (DMA)**: allow the peripherals to directly communicate using the memory bus, removing the intervention of the CPU

4.5.2 The General Purpose Input and Output (GPIO) Pins

The Raspberry Pi includes a variety of inputs and outputs that are controlled by the operating system. These incude a camera input, USB inputs, HDMI, and a bank of pins named the GPIO (General Purpose I/O) pins.

GPIO pins can be configured as digital inputs or digital outputs. Furthermore, digital inputs can be configured with a weak "pull up" resistor or a weak "pull down" resistor. Likewise, digital inputs can be further configured to detect rising edges, falling edges, or both, and fire an event.

Some digital outputs can be optionally configured as PWM (pulse width modulation) outputs. Note that there are no analog-to-digital inputs on the GPIO port.

4.5.3 Reading and Setting GPIO Pins

The Raspberry Pi is equipped with a variety of GPIO (General Purpose I/O) pins that can be used for controlling or sensing signals to or from the outside world. It also has a dedicated camera port.

You can view a diagram of the GPIO port and each the pins from the command line by typing the following command:

```
pinout
```

The output from this command will show all 40 GPIO pins with the signal names next to each as follows:

```
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21
```

Before wiring the GPIO ports, remember to turn off power to the Raspberry Pi. It is always prudent to check your wiring to ensure you are accessing the correct pin and that there are no exposed leads touching each other as this could cause short circuits. It is important to make sure you are properly interfacing with external sensors and actuators to ensure you comply with the electrical limits of the GPIO ports.

You can test your GPIO connections using the `raspi-gpio` command line utility. To read all the GPIO pins, type the following:

```
raspi-gpio readall
```

This should display the state of all the GPIO pins. Note that there are several different pin numbering schemes that can be used with the Raspberry Pi, which can lead to some confusion. Note that we will be using the BCM numbering scheme. BCM represents the Broadcom SOC channel and reflects the numbering scheme used by the Broadcom ARM processor and the `raspi-gpio` utility.

For information on contolling GPIO pins, see section 7.2.

4.6 Other OS Support Functions

4.6.1 Logfiles

A variety of log files are kept in the folder `/var/log` which keep track of the state of the system. Use the `tail` command to read the latest log entries as follows:

```
tail syslog
```

Viewing logfile is useful for troubleshooting.

4.6.2 Updating the Operating System

Part of keeping an operating system secure is to ensure that it is kept up to date with the latest updates and patches. Un-patched security issues can become attack vectors for malicious hackers. To view some current security vulnerabilities, visit the NIST National Vulnerability Database. This is even more critical for a system that is connected to a network such as an IoT device. In fact, IoT devices must remain up-to-date after they are deployed in the field for the entire life-time of the product! To update the Raspberry Pi from the command line, use the following commands:

```
sudo apt update
sudo apt full-upgrade
```

After performing an update, you may want to clear space by removing any packages that are obsolete or are no longer required:

```
sudo apt autoremove
```

Furthermore, you can clean any files left behind during the update process by using the following command:

```
sudo apt clean
```

4.6.2.1 Enabling Automatic Updates Rather than typing this command each day, one approach is to use a program to automate updates. There are several ways to accomplish this in Linux. One approach is to use a special package for automatic updates named `unattended-upgrades`. To install this package, type:

```
sudo apt install unattended-upgrades
```

Next, edit the configuration file as follows:

```
sudo nano /etc/apt/apt.conf.d/50unattended-upgrades
```

This will open a configuration file to setup unattended updates. Remove the double slashes (//) in front of the following lines in the configuration file to enable various automatic updates:

```
"origin=Debian,codename=${distro_codename}-updates";
"origin=Debian,codename=${distro_codename},label=Debian";
"origin=Debian,codename=${distro_codename},label=Debian-Security";
"origin=Debian,codename=${distro_codename}-security,label=Debian-Security"
    ;
```

To save the file, type ctrl-x and select "yes" to save and exit. Finally, ensure automatic updates are enabled by running:

```
sudo dpkg-reconfigure unattended-upgrades
```

At this point, automatic updates should be enabled to run daily.

4.6.3 Securing your Raspberry Pi

The Raspberry Pi is like any other networked computer — with all the wonderful possibilities as well as the threats. While physical security of a computing device is one consideration, once it is connected to a network a whole new set of vulnerabilities arise. For this reason, it is prudent to follow basic procedure for *system hardening* to reduce the vulnerability of your Raspberry Pi.

The first step is to disable all unnecessary services that are running. Each service expands the possible **attack surface** and exposes new vulnerabilities. This is particularly true of programs that open networking ports that are accessible by other computers. To obtain a list of all the services running on a Raspberry Pi, type:

```
systemctl --type=service --state=running
```

If any of these services are not required, it can be stopped and disabled from starting up on subsequent reboots as follows:

```
systemctl disable --now service-name
```

where `service-name` is the name of the service you wish to stop.

In addition, the Raspberry Pi OS comes with a whole set of tools and utilities to help secure your device through intrusion detection and prevention.

4.6.3.1 Virus Scanning There is an open source virus scanning program called **ClamAV** that can be used to scan your Raspberry Pi for viruses. To install `clamav`, type:

```
sudo apt-get install clamav
```

Before you can start scanning for viruses, an updated ClamAV virus signature database must be installed. A tool called `freshclam` is used to download and update this database. First, the `freshclam` configuration file must be setup properly by editing:

```
sudo nano /etc/clamav/freshclam.conf
```

Once the configuration file is in place, `frashclam` can be run by typing:

```
sudo freshclam
```

Once the virus signature database is update, the whole hard drive can be scanned as follows:

```
clamscan -r -i --bell /
```

4.6.3.2 Scanning for Root-kits Another utility provides the ability to scan for known *root-kits*. A *root-kit* is a program that enables an unauthorized user to gain control of a computer system. To install the `chkrootkit`, type:

```
sudo apt-get install chkrootkit
```

Next, we can scan for root-kits as follows:

```
sudo chkrootkit
```

To display only warnings and suspected infected files, type: `bash sudo chkrootkit -q`

4.6.3.3 fail2ban If you are connecting to your Raspberry Pi using SSH, you are susceptible to both *brute force attacks* and *dictionary attacks*. A *brute force attack* connects the the SSH port and tries cracking a password using different combination of characters. A *dictionary attack* connects to an SSH port and attempts to log in using a list of common passwords.

The **fail2ban** utility is an intrusion prevention system that monitors the SSH port (as well as other ports) for repeated unsuccessful login attempts. If a particular client exceeds a certain threshold of failed login attempts, its IP address is blocked.

To install **fail2ban**, type:

```
sudo apt install fail2ban
```

To protect the Raspberry Pi from SSH attacks, edit the configuration file located at `/etc/fail2ban/jail.d/defaults-debian.conf`. This file should contain at least the following settings for SSH:

```
[sshd]
```

```
enabled = true
```

The configuration file can be used to fine-tune others settings. For example, you can configure the maximum allowable login retries and the time that an IP address is banned as follows:

```
[sshd]
enabled = true
filter = sshd
maxretry = 5
bantime = 600
```

4.6.4 Setting up a Print Server

Printing in Linux is typically done using CUPS (Common Unix Printing System). CUPS enables you to use your Raspberry Pi as a print server which can allow you to share a printing device with others on your local network. To begin, install CUPS on the Raspberry Pi, type:

```
sudo apt install cups
```

Next, add the default user, `pi`, to the group of users that can access the printer:

```
sudo usermod -a -G lpadmin pi
```

A variety of CUPS settings can be modified in the file `/etc/cupsd/cupsd.conf` to enable operation of a CUPS server (and to adjust access control). The server incudes a web interface to view print jobs and administer the printers. Once running, the web interface can be accessed by pointing a browser to the IP address of the Raspberry Pi followed by `:631`. For example, if the Raspberry Pi is located at IP address `192.168.1.1`, you should point your browser to `192.168.1.1:631`. Consult the online CUPS documentation for more information.

4.7 Compiling the Linux Kernel

Using the `gcc` tool chain, it is also possible to customize and recompile the Linux kernel itself! Before beginning, install the git tool along with other build dependencies:

```
sudo apt install git bc bison flex libssl-dev make libncurses5-dev
```

Next, use git to download the kernel source code as follows:

```
git clone --depth=1 https://github.com/raspberrypi/linux
```

Note that omitting the `depth` parameter will download the entire repository history and will take a *very* long time! The next step is to prepare the default configuration of the kernel. For a 64-bit kernel

on a Raspberry Pi 4 this is done as follows:

```
cd linux
KERNEL=kernel8
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- mrproper
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcm2711_defconfig
```

Setting `KERNEL=kernel8` selects the 64-bit kernel for the ARM processor. In the commands above, `kernel8` and `bcm2711_defconfig` refer to the kernel filename and build target respectively for the Raspberry Pi 4. If you are using a different model of the Raspberry Pi, you will need modify these commands with a different a different kernel filename and build target as summarized below:

Model	Default kernel filename	Config build target
Raspberry Pi Zero	kernel.img	bcmrpi_defconfig
Raspberry Pi 2, 3, 3+	kernel7.img	bcm2709_defconfig
Raspberry Pi 4	kernel8.img b	cm2711_defconfig

If you are not sure which model you are using, it should be printed on the Raspberry Pi circuit board, otherwise you can query your device model as follows:

```
cat /proc/device-tree/model
```

After running the appropriate make command, a `.config` file will be generated with various settings for your particular Raspberry Pi model. These kernel settings can be customized according to your wishes and requirements. To make changes to these kernel setting we can use the `makemenu` utility as follows:

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

This utility essentially provides a friendly interface to modify numerous configuration settings stored in the `.config` file. These settings will later be used to guide the compilation process to produce a customized kernel image.

For example, one simple customization that might be made is to give the new kernel a custom name to distinguish it from other kernels. You can change the kernel name under *General setup -> Local Version*. Once all the customizations are set, save the configuration. After making changes to the configuration, a 64-bit kernel can then be compiled by typing the following:

```
make -j4 Image.gz modules dtbs
```

Note that the `-j4` flag will allow the compilation workload to be spread across four cores in the ARM processor thus speeding up the compilation. Even so, the compilation process will take a very *long*

time on a Raspberry Pi since the Linux kernel is a large program with millions of lines of code. When compilation ends successfully, you may install the new kernel as follows:

```
sudo make -j4 modules_install
sudo cp /boot/firmware/$KERNEL.img /boot/firmware/$KERNEL-backup.img
sudo cp arch/arm64/boot/Image.gz /boot/firmware/$KERNEL.img
sudo cp arch/arm64/boot/dts/broadcom/*.dtb /boot/firmware/
sudo cp arch/arm64/boot/dts/overlays/*.dtb* /boot/firmware/overlays/
sudo cp arch/arm64/boot/dts/overlays/README /boot/firmware/overlays/
```

Note that it is possible use a kernel with a different filename by setting a kernel parameter in `/boot/config.txt`. After these steps are successfully completed, reboot the Raspberry Pi to make the new kernel active. Log in and verify that the new kernel is running using the following command:

```
uname -a
```

The new name you configured for the kernel should now be reported along with the Linux kernel version. You will need to reboot each time you change your kernel configuration for the new kernel to become active.

To reinstall a default kernel from the Raspberry Pi repository, type:

```
sudo apt --reinstall install raspberrypi-kernel
```

Note that to speed up kernel compilation it is possible to cross-compile a modified ARM kernel on a desktop workstation and then transfer the new kernel to the Raspberry Pi.

5 Networking

5.1 Networking Utilities

The Linux shell supports a variety of tools and utilities for networking. What follows are some helpful practical tools related to networking.

5.1.1 ping

Ping (Packet Inter-Newtwork Groper) is helpful utility for testing end-to-end connectivity and transport delays in a TCP/IP network. Ping works by sending an Internet Control Message Protocol (ICMP) packet with an "echo request" to the specified machine. When a machine receives an echo request, it normally replies with an echo reply packet (note that network interfaces can also be configured to ignore ping requests). To `ping` a remote host, simply type:

```
ping hostname
```

where `hostname` represents the host name (or a "dotted" IP address) of the host you wish to ping. By default, the "pings" with continue each second, and the results along with the delay time will be displayed for each ping.

5.1.2 ifconfig

The `ifconfig` utility can be used to get or set the status of a networking port. To use `ifconfig`, type:

```
sudo ifconfig
```

For example, on a Raspberry Pi connected to an Ethernet port, one can get the status of the port as follows:

```
sudo ifconfig etho
```

where `eth0` is the name of the first Ethernet port. The Wi-Fi port is typicallly named `wlan0`. The above command will return something like the following:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet a.b.c.d  netmask 255.255.255.0  broadcast a.b.c.255
        inet6 aaaa::bbbb:cccc:dddd:eeee  prefixlen 64  scopeid 0x20<link>
        ether aa:bb:cc:dd:ee:ff  txqueuelen 1000  (Ethernet)
        RX packets 10000  bytes 100000
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
        TX packets 100000  bytes 100000
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note that this output shows the IPv4 and IPv6 addresses. The address next to the `ether` label is the **MAC**, or *Media Access Control* address, a unique hardware address assigned at the factory. The first few bytes (or "octets", as they are often referred to in networking) of the MAC address are an *Organizationally Unique Identifier*, or **OUI**, assigned by the IEEE. The OUI identifies the manufacturer of the equipment and can be looked up at standards-oui.ieee.org.

Note that `ifconfig` also indicates the number of received and transmitted packets along with the counts of any communication errors, which can soemtimes be helpful when debugging networking errors.

5.1.3 traceroute

Traceroute uses similar ICMP packets that are used by `ping` but with the time-to-live (TTL) packet field set so that it can get replies from each of the hops along the way to a destination machine. To install `traceroute`, type:

```
sudo apt install traceroute
```

To run `traceroute`, simply type the following:

```
traceroute hostname
```

where `hostname` is the host name to which you want to trace the route. Typically, a list of hosts and routers that packets take on there way to `hostname` should be listed along with their corresponding delays. `traceroute` can be handy for identifying the number of hops and sources of congestion in a route between two machines on the internet.

5.1.4 mtr

Closely related to `traceroute` is `mtr`, which is another network diagnostic tool. `mtr` conntinually sends packets along the route to a destination and keeps track of the statistics of the response times at each hop in the route. A summary diagnostic is displayed showing the average, best, and worst times along with the percentage of packet losses detected. To install `traceroute`, type:

```
sudo apt install mtr
```

To use `mtr`, type:

```
mtr hostname
```

where `hostname` indicates the destination and route that you want to analyze.

5.1.5 dig

Not so long ago, when the when the Internet was comprised of a limited number of hosts, there was a file called `/etc/hosts` contained information about every hosts on the network. This file was maintained and distributed across the Internet. This approach became impractical as the number of hosts grew large. A hierarchical, domain-based, naming scheme called DNS (the Domain Name System) was invented and is defined in RFC 1034 and 1035. The DNS system uses UDP packets (Request/Reply) and a resolver is used to map names to an IP address by sending a request to a local DNS server. The `/etc/hosts` file is still found on some hosts, but is often used to store local addresses such as the address of the local machine.

The Internet is divided into various top level domains which are approved by ICANN. There are about a dozen root servers in the world which know the addresses of the top-level domain servers. These top-level domains are further subdivided into subdomains, which can be further partitioned, and so on. These domains on the Internet can be represented by a tree with the leaves of the tree representing hosts or groups of hosts.

Every domain has a set of resource records associated with it, and DNS servers use these resource records when replying to queries. When a host has a name query, it passes the query to one of the local name servers. If the name falls under the jurisdiction of the local name server, it returns the authoritative resource records. If the requested name cannot be satisfied locally and is part of a remote domain, then a recursive query can be made to a remote name server.

Dig is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers to queries returned from your name server(s). The `dig` utility is often used to troubleshoot DNS problems. To install the `dig` utility (alongside other DNS utilities), type:

```
sudo apt-get install dnsutils
```

Once installed, a basic `dig` hostname query is made as follows:

```
dig @server hostname
```

where `server` is the DNS server, and `hostname` is the name to query. The server parameter is optional; if it is not given the default name server will be used. To perform a *reverse look-up* (i.e. determine the hostname given an IP address), do the following:

```
dig -x 1.2.3.4
```

where `1.2.3.4` is the IP address of the host to query. Type `man dig` from the command line for a detailed description of how it can be used.

5.1.6 wget

Wget is a handy utility for downloading web files from the command line. The basic syntax for `wget` is as follows:

```
wget URL
```

where `URL` is the *uniform resource locator* of the file you wish to download. For example, to download this guide, use:

```
wget https://sites.calvin.edu/derek/tutorials/rpi-guide.pdf
```

`wget` includes a rich set of command line options; for more information, type:

```
wget --help
```

5.1.7 curl

Curl is a tool to transfer data from or to a server. It supports many different protocols, including HHTP, HTTPS, FTP, and FTPS. The basic syntax for `wget` is as follows:

```
wget URL
```

where `URL` is the *uniform resource locator*, including its protocol, of the file you wish to download. By default, `curl` writes the received data to the screen but can be instructed to save the data to a local file using the `-o` option. For example, to download this guide to a local file named `guide.pdf`, use:

```
curl https://sites.calvin.edu/derek/tutorials/rpi-guide.pdf -o guide.pdf
```

`curl` includes a rich set of command line options so that it can operate without any user interaction. For more information, type:

```
curl --help
```

5.1.8 telnet

Telnet is a client for connecting to remote systems using the Transmission Control Protocol (TCP). For example, to use telnet to connect to a web server, type:

```
telnet hostname 80
```

This will connect to port 80 on server `hostname`. If it is running a web server you can type `GET` and afer hitting enter, you should see HTML code send in reply.

> Note: `telnet` can connect to any port, but note that all text is sent *in the clear*, ie. insecurely and without encryption. Anyone observing the traffic on the network will be able to read any text that is sent.

5.1.9 nmap

The `nmap` utility, or "network mapper", is a tool for network exploration and security scanning. It can rapidly scan large networks to find hosts, identify open ports, operating systems, and other network characteristics.

> NOTE: Because the `nmap` utility can be used to detect network security issues, it should only be used with permission on a network. Using `nmap` on a network without permission is like checking the windows and doors to someone's home and is not considered polite, or worse, may be interpreted as an aggressive act or a precursor to further intrusions. Recall the sage movie line from Spiderman's uncle: *with great power comes great responsibility*.

5.1.10 tcpdump

The `tcpdump` tool enables one to "snoop" network packets. To install the tool, type:

```
sudo apt install tcpdump
```

Once the tool is installed, it can be used to monitor network traffic. For instance, to monitor traffic from

monitor your MQTT traffic (once again, modify the hostname of your MQTT broker if you are using a different one): sudo tcpdump -XA host mqtt.eclipseprojects.io Note that we are using command line options to display the packet contents in ASCII and to limit our monitoring only to traffic exchanged with the MQTT broker. Each time an MQTT packet is sent, you should see a network packet "dumped" to the console.

> NOTE: Running `tcpdump` or similar tools needs to be used ethically and judiciously. Once again: *with great power comes great responsibility*.

5.1.11 Wireshark

Wireshark is a popular network protocol analyzer which captures packet data and display it in detail. It can be used to troubleshoot network problems, chase down security issues, and debug or verify

networking applications. To install the tool, type:

```
sudo apt install wireshark
```

To run Wireshark, type:

```
sudo wireshark eth0
```

where `eth0` is the port name you want to monitor. Wireshark can perform live capture of network data, but this can be like "drinking from a firehose." Thus, Wireshark can use filters to limit the capture of packets to those which are of interest.

> NOTE: Running Wirshark or similar tools on servers is normally not "polite" since it allows you to view network traffic, some of which is "sent in the clear." If you are monitoring your own network or Raspberry Pi, and the traffic is your traffic, you may give yourself permission to do this. However, in general, Wireshark is among a set of tools that need to be used ethically and judiciously. Once again, recall the sage movie line from Spiderman's uncle: *with great power comes great responsibility*.

5.1.12 Drill Exercises

1. What is the IP address of your local name server?
2. What is the IP address of `www.calvin.edu`?
3. What is a DNS MX record? What is the MX record for `calvin.edu`?
4. Do DNS requests use the TCP or UDP transport protocol? Why does DNS use the transport protocol that it does?
5. Use `wget` to download this book.
6. Use Wireshark to monitor the packets leaving your networking port. What kind of packets are coming and going? Use a filter to view *only* web traffic. Are the packets enrypted? How can you tell?

5.2 The Web

It is possible to configure the Raspberry Pi to run a *web server*. A web server is a program that accepts requests using **HTTP** (Hypertext transfer Protocol) or **HTTPS** (secure HTTP). Typically, web server listed for incoming HTTP requests on port 80 and on port 443 for HTTPS. Various open source web servers are available, including the Apache web server, Nginx, and Lighttpd. In the following sections we will explore both Lighttpd and Nginx.

5.2.1 Lighttpd

Lighttpd is a simple, lightweight web server which can be installed from the command line as follows:

```
sudo apt update
sudo apt -y install lighttpd
```

It is recommended that you consult the online manuals for Lighttpd to ensure the server is configured and setup securely.

Test the webserver by creating a simple web file as follows:

```
sudo nano /var/www/html/index.html
```

Using an editor of your choice, enter a simple HTML file as follows:

```
<html>
<head></head>
<body>
   Hello world!
</body>
</html>
```

Point a browser to `http://a.b.c.d` where `a.b.c.d` is the local IP address of your Raspberry Pi. The message, "hello world" should appear in a web page.

You can see the status of the server by using the following command:

```
sudo service lighttpd status
```

To disable the lighttpd web service from starting up on the next reboot boot, type:

```
sudo systemctl disable lighttpd
```

5.2.2 Nginx

Nginx is a full-featured open source web server which can be installed from the command line as follows:

```
sudo apt update
sudo apt -y install nginx
```

If the web server needs to support PHP, type the following:

```
apt install php php-fpm php-cli
```

Test the webserver by creating a simple web file in the root web folder as follows:

```
sudo nano /var/www/html/index.html
```

Using an editor of your choice, enter a simple HTML file as follows:

```
<html>
<head></head>
<body>
   Hello world!
</body>
</html>
```

Point a browser to `http://a.b.c.d` where `a.b.c.d` is the local IP address of your Raspberry Pi. The message, "hello world" should appear in a web page.

You can view the status of the server by using the following command:

```
sudo service nginx status
```

To disable the nginx web service, type:

```
sudo systemctl disable nginx
```

5.3 Java Network Programming

The Berkeley *sockets* interface was originally developed at the University of California at Berkeley as a tool to for network programming. A **Socket** is a handle to a communications link over a network with another application. A socket *connection* includes a local **IP address** and **port number** and a remote **IP address** and remote **port number**.

Java is one of the first languages designed with networking in mind. Java applications can conveniently send and receive data across the Internet. The `java.net` package provides the classes for implementing networking applications. Several of the key classes for Java networking are summarized in the table below.

Class	Description
`ServerSocket`	implements server sockets
`Socket`	implements client sockets
`InetAddress`	represents an IP Address
`DatagramPacket`	represents a UDP datagram packet

Class	Description
DatagramSocket	socket for sending and receiving UDP datagram packets
MulticastSocket	used for sending and receiving IP multicast packets
URL	represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web

5.3.0.1 ServerSocket Class The ServerSocket class is used by servers to listen for client connections. The ServerSocket class specifies a *port* number to listen on for connections. This constructor method blocks until a connection is made and then returns a socket for communicating with the client. Once the communication is complete, a close method can be called to close the socket connection.

5.3.0.2 Sample Java Network Code What follows is a sample java program for establishing a Server Socket on port 7777, waiting for a connection, and performing a simple exchange of text messages with a remote client. Use a text editor to enter the program and save it as Server.java.

```
import java.net.*;
import java.util.*;
import java.io.*;

class Server {
    public static void main(String args[]) throws IOException {
        System.out.println("Creating server socket on port 7777 and
           waiting for connection:");
        ServerSocket server = new ServerSocket(7777);
        Socket socket = server.accept();

        // Read text from the socket connection and display it
        Scanner in = new Scanner(socket.getInputStream());
        String msg = in.nextLine();
        System.out.println("Received message: "+msg);

        // Send a simple message back to the client
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        out.println("Hello from the server side!");
        out.flush(); // empty the buffer

        // Close socket and server
        socket.close();
        server.close();
    }
}
```

To compile this program to java *bytecode* and then run it, type:

```
javac Server.java
java Server
```

This program will create a server socket and wait for a client connection. You can test the program by connecting to the server socket using the `telnet` utility, a program that can connect to a remote server and send and receive text characters. To install telnet on your Raspberry Pi, type:

```
sudo apt install telnet
```

You can connect to the Java server program remotely using the IP address of the Raspberry Pi or you can connect locally using the *local loopback interface* which uses a reserved IP address of `127.0.0.1` and a special hostname of `localhost`. Open another terminal session and use a `telnet` connection to the local loopback interface on port 7777 as follows:

```
telnet 127.0.0.1 7777
```

Type a short message and you should it displayed by the server, and then the client should display the server message: `Hello from the server side!`.

Rather than using `telnet`, you can also create your own Java program for making a client connection to the server. What follows is a sample java program for communicating with our simple server on the local loopback interface. Use a text editor to enter the program and save it as `Client.java`.

```
import java.net.*;
import java.util.*;
import java.io.*;

class Client {
    public static void main(String args[]) throws IOException {
        // Create a socket connection to a server port 7777
        Socket socket = new Socket("localhost", 7777);

        // Send some data to the server
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        out.println("Hello from the client!");
        out.flush();

        // Read reply from server and display it
        Scanner in = new Scanner(socket.getInputStream());
        String msg = in.nextLine();
        System.out.println("Received message: "+msg);

        // Close socket connection
        socket.close();
    }
}
```

Note that if you want to run the client program remotely, replace the `localhost` hostname with the hostname or IP address of the remote server. To compile the client program, type:

```
javac Client.java
```

Next, start the server in one terminal window. It will create a server socket and wait for a connection. On another terminal window, run the client java program as follows:

```
java Client
```

Observe both terminal sessions and note the messages that are displayed as the client connects to the server, exchanges a pair of messages, and then exits.

For more information, see the OpenJDK Java documentation.

6 Databases

A database is a structured collection of logically related data. Normally, one interacts with a database using software referred to as a database management system (DBMS). The database itself along with the DBMS and associated software is referred to as a *database system*.

6.1 Introduction to SQL Databases and the Raspberry Pi

One common type of database is a relational DBMS, a term that was originally defined and coined by Edgar Codd in 1970. In a relational database the data is stored in 2-dimensional tables of rows (called tuples or records) and columns (called attributes or fields). A table or relation is defined as a collection of records or tuples that have the same fields or attributes.

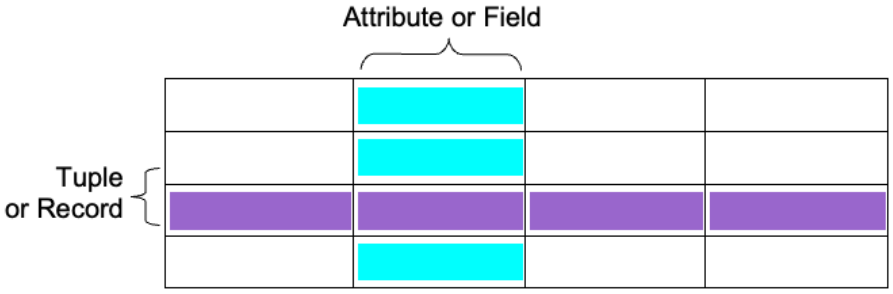


Figure 13: Data is stored in 2-dimensional tables of rows

Structured Query Language or SQL is a standard language used to interact with relational database systems and is maintained as an ISO standard. SQL provides a convenient level of abstraction to interact with a database to define, manage, and query data. SQL is a declarative programming language as opposed to an imperative programming language such as the C programming language in which all the computations are explicitly stated step-by-step. A declarative language is one that defines what the program should accomplish, rather than describing *how* to go about accomplishing it. An SQL statement is processed by the database system which determines the best way to return the requested data. Although there are small differences in SQL syntax between relational database systems, the syntax is largely the same across many systems.

There are numerous commercial and open source SQL database systems available. Some examples of open source projects include SQLite, MySQL, and PostgreSQL, all of which are included in the standard Raspberry Pi software repositories.

6.1.1 Using SQLite

SQLite is a simple, a friendly, lightweight database program which supports most SQL commands. SQLite is also convenient since it does not run as a server and stores data in a single file which can be placed anywhere.

SQLite can be installed as follows:

```
sudo apt install sqlite3
```

To setup an example sqlite database, type the following:

```
sqlite3 temperature.db
```

Once SQLite launches, create a table to store temperature data by typing the following SQL command:

```
CREATE TABLE TemperatureData
(datetime TEXT NOT NULL, temperature double NOT NULL);
```

Make sure you include a semicolon at the end of the command to indicate that your SQL command is complete. Note that SQLite does not have a storage class for dates and/or times so we will store the date and time as a text field. Next, check that the table was successfully created by typing:

```
sqlite> .tables
```

Type the following sqlite3 command to show the structure of the table you just created:

```
pragma table_info('TemperatureData');
```

Once the table is defined you can insert, delete, edit, and query data in the table. For example, to insert data into the table, type the following:

```
INSERT INTO TemperatureData VALUES (datetime('now','localtime'), 23.0);
```

To show the table data, use a query like the following:

```
SELECT * FROM TemperatureData;
```

Finally, to quit SQLite, type the following at the prompt:

```
sqlite> .quit
```

6.1.2 Using MySQL

Although there are some graphical user interfaces available for MySQL, this tutorial will focus on the command line interface for MySQL. MySQL can be installed as follows:

```
sudo apt install mariadb-server
```

Run the following command and follow the prompts to setup and secure the MySQL server:

```
sudo mysql_secure_installation
```

To access your MySQL server with root access, type:

```
sudo mysql -u root -p
```

Once other usernames and passwords have been established, one can access MySQL using them as follows:

```
mysql -u username -p
```

Where `username` is the username, and after pressing enter a prompt will appear for a password. In order to use a database or create new ones, you must have sufficient privileges assigned to your username.

To list all the databases on the MySQL server, type:

```
SHOW DATABASES;
```

SQL statements include one or more SQL keywords that are often written in uppercase as a matter of style and which end with a semi-colon. To create new database in SQL from the command line, use the CREATE DATABASE statement as follows:

```
CREATE DATABASE school;
```

After executing this statement, MySQL should return a message indicating whether the command was successful or not. The SHOW DATABASES statement will show all the databases on the database server. After creating a new database, the SHOW DATABASES statement can be used to verify that the new database has been created. To show all the databases on the server, type the following:

```
SHOW DATABASES;
```

MySQL should return with a list of the current databases. The USE command is issued to select and use a specific database. For example, to use the database we just created, type:

```
USE school;
```

A database is a collection of tables. One the database selected, you can query and access the tables in the database. To create a table within the school database, you use the CREATE TABLE statement. For example, to create a table of students names type the following:

```
CREATE TABLE students (
       studentNumber int NOT NULL,
       lastName varchar(50) NOT NULL,
       firstName varchar(50) NOT NULL,
       PRIMARY KEY  (studentNumber)
       );
```

The table name is specified after CREATE TABLE statement and then the columns names are given followed by data type, size, NOT NULL or not. A field which is specified as NOT NULL must contain a value. It is also possible to specify the primary key of the table. The primary key is used to uniquely identify each row in the table. Since student numbers are supposed to be unique, the primary key in this example is set to studentNumber. If the table has more than one primary key, you can separate them by a comma. In order to view details about a table, including information about fields and data types, use the DESCRIBE statement. For example, type:

```
DESCRIBE students;
```

This will return information about the table we just created and information about the fields that comprise it. The output from MySQL should resemble the following:

```
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| studentNumber | int(10)     | NO   | PRI | NULL    |       |
| lastName      | varchar(50) | NO   |     | NULL    |       |
| firstName     | varchar(50) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

To modify the structure or type of data in existing tables, MySQL provides an ALTER command. Although the SQL keywords themselves are not case sensitive, the database and tables may be case sensitive depending on the underlying operating system being used. The data type of each of MySQL fields or attributes must also be specified. MySQL supports a variety of numeric, character and binary data types. Some of the common data types supported in MySQL are summarized in the following table:

Data Type	Description
INT	A signed integer (4 bytes)
FlOAT	A floating-point number (4 bytes)

Data Type	Description
DOUBLE	Double precision floating-point number (8 bytes)
DATE	Date in the format YYYY-MM-DD
DATETIME	Date and time
CHAR(M)	A fixed-length string with a length of M characters (between 1 to 255 character)
VARCHAR(M)	A variable-length string with a length of up to M characters (between 1 to 255)
TEXT	A text field with a maximum length of 65535 characters
BLOB	A binary large object (used to store binary data such as images)

In order to show all the tables in a database, you can use the SHOW TABLES statements as follows:

```
SHOW TABLES;
```

To add records to the students database, use the INSERT statement. For example, to add "John Calvin" with a student ID number of 12345 into the students table, do the following:

```
INSERT INTO students
(studentNumber,firstName,lastName) VALUES
(12345, "John", "Calvin");
```

6.1.2.1 MySQL Queries You can also ask MySQL to search for data by submitting a query asking for all the records or rows that match a specific criteria. The SQL SELECT statement is used to perform queries on an SQL table. For example, to list all the students in the table, type:

```
SELECT * FROM students;
```

The * indicates that all columns should be returned. The SELECT query returns the requested data as text in a tabular format like follows:

```
+---------------+----------+-----------+
| studentNumber | lastName | firstName |
+---------------+----------+-----------+
|         12345 | Calvin   | John      |
+---------------+----------+-----------+
```

To display specific columns, replace the * with a comma-separated list of columns that you would like to see displayed. For example, to display just the lastName and firstName columns, type:

```
SELECT lastName, firstName FROM students;
```

To list the students in alphabetical order using the ORDER BY clause as follows:

```
SELECT * FROM students ORDER BY lastName, firstName;
```

The order can be explicitly set to be ascending or descending by placing the ASC or DESC keywords at the end of the query. To prevent the SELECT statement from returning duplicate values in the results, the DISTINCT keyword can be used. For example, to list all the distinct first names in the students table, type:

```
SELECT DISTINCT firstName FROM students;
```

It is also possible to restrict the search to meet specific criteria using the WHERE keyword. For example, to list all the students who have the first name of "John", type:

```
SELECT * FROM students WHERE firstName = 'John';
```

The conditions to restrict search results can be further combined with boolean operators such as AND and OR operators to express search based on different conditions. In addition to WHERE, there are other keywords such as LIKE and BETWEEN which can be used to restrict the results of a search. For example, to list all the students whose first name starts with a "J", type:

```
SELECT * FROM students WHERE firstName LIKE 'J%';
```

where % is a wildcard which matches any character or sequence of characters. The BETWEEN keyword will restrict a search to values that fall in a range between some minimum and maximum value. For example, to return all the last names that lie alphabetically between "Calvin" and "Luther", type:

```
SELECT * FROM students WHERE lastName
       BETWEEN 'Calvin' AND 'Luther';
```

One can also query the number of rows that match a certain condition using the COUNT keyword:

```
SELECT COUNT(*) FROM students WHERE firstName LIKE 'J%';
```

To delete data from a table, use the DELETE statement. For example, to delete all the students with the last name of "Calvin", do the following:

```
DELETE FROM students WHERE lastName = 'Calvin';
```

To make another table called courses that stores a course code and student number for each course a student is enrolled in, use the CREATE statement again as follows:

```
CREATE TABLE courses (
       studentNumber int NOT NULL,
       courseCode varchar(7) NOT NULL );
```

In this table the studentNumber will not necessarily be unique since it will be appear once for each course in which a student is enrolled. The courseCode will also not necessarily be unique since it will be repeated for each student in the course. Note that the students names do not need to be stored again; they can be retrieved if required by looking up the studentNumber in the students table. To add some records to the courses database, use the INSERT statement once again:

```
INSERT INTO courses (studentNumber,courseCode)  VALUES (12345, "CSC101A");
```

To change or modify data in a table, use the UPDATE keyword as follows:

```
UPDATE students SET studentNumber = 123
WHERE firstName = 'John' AND lastName = 'Calvin';
```

This statement will modify the students table and replace the studentNumber for the student with the name John Calvin. It is possible to modify multiple field values with an UPDATE statement using a comma-separated list of assignments. The WHERE clause in this case uses a boolean AND operator to make a more complex condition. You can also ask MySQL to search data from multiple tables by using a JOIN operation. The JOIN keyword relates two or more tables, typically by using values that are common between them. The students and courses database have a common value of studentNumber that can be used to join them. The ON keyword can be used to specify a condition with which to join tables. For example, to list all the first names and last names of students enrolled in CSC101A, you can use a join operation based on the condition of matching a studentNumber in a query as follows:

```
SELECT firstName, lastName
FROM students
JOIN courses
ON students.studentNumber = courses.studentNumber
WHERE courseCode = 'CSC101A';
```

To close a database, type the following:

```
CLOSE DATABASE school;
```

It is also possible to delete a table and all it contents using the DROP command. This command should be used with care since it permanently deletes your table and cannot be undone.

```
DROP TABLE students;
```

Finally, you can quit MySQL at any time by typing the QUIT command.

```
quit
```

6.1.2.2 Backing Up MySQL Data You can use the `mysqldump` utility to create a simple backup of your database to a file using the following syntax:

```
mysqldump -u username -ppassword databasename > backup.sql
```

where `username` and `password` are your MySQL username and password and `databasename` is the name of the database you want to backup. The resultant file called backup.sql will contain all the SQL statements needed to create the table and populate the table in a new database server. If you examine the file backup.sql in a text editor you will observe the necessary SQL commands to create the database, its tables, and all the data contents within the tables. The data can be restored by typing the following:

```
mysql -u username -p password databasename < backup.sql
```

6.1.2.3 Using PHP and MySQL Once you are familiar with MySQL syntax in the command-line environment, you can begin to write PHP code which can connect to a MySQL database and query it using SQL statements. PHP includes several functions to connect to a MySQL server and perform various queries. Some of the many PHP MySQL functions are shown in the following table:

Function	Description
mysql_connect	Opens a connection to a MySQL database server
mysql_select_db	Opens a database on the MySQL server
mysql_query	Performs a MySQL query on the currently selected database
mysql_close	Closes a MySQL database connection
mysql_fetch_array	Returns an associative array with the next row from a MySQL query
mysql_error	Returns the text of the error message from a previous MySQL operation
mysql_create_db	Create a MySQL database
mysql_drop_db	Drop a MySQL database
mysql_real_escape_string	makes data safer before sending query to MySQL

The SELECT syntax used on the MySQL command line is the same syntax used to query data using the PHP `mysql_query` function (the query is passed as a string argument). However, before data can be queried, the proper PHP functions must be called in order to connect to the MySQL server and to select the appropriate database. For example, the following PHP code connects to the school database and retrieves a list of students and displays it as a list within a webpage:

```
<ul>
<?php
```

```
        $db = mysql_connect("localhost", "username", "password");
        mysql_select_db("school");
        $results = mysql_query("SELECT * FROM students");
        while ($row = mysql_fetch_array($results)) {  ?>
        <li> <?= $row["firstName"]." ".$row["lastName"] ?> </li>
        <?php
        }
        mysql_close($db);
?>
</ul>
```

6.1.2.4 Using MySQL with Java MySQL can also be used with the Java Programming language. The JDBC (Java Database Connectivity) API is provides DBMS connectivity to a wide range of SQL databases including MySQL as well as access to other tabular data sources such as spreadsheets. The JDBC API includes several classes all found in the java.sql package.

6.1.3 Using PostgreSQL

PostgreSQL can be installed as follows:

```
sudo apt install postgresql
```

This package includes a command line client utility called `psql` which can be used to interact with the PostgreSQL server. After the installation you can verify that the postgresql service is active as follows:

```
sudo service postgresql status
```

It is recommended that you consult the online manuals for PostgreSQL to ensure the server is setup securely.

When you first run PostgreSQL a default administrator named “postgres” is created. This username can then be used with `psql` to connect to the PosgreSQL service as follows:

```
sudo -u postgres psql
```

Since the new “postgres” user has no password, your first action should be to set the password as follows:

```
\password postgres
```

You can now create a new database as follows:

```
CREATE DATABASE school;
```

This command creates a new database name `school`. Within this database we can define tables, such as tables to store students and courses.

Rather than using the administrator account, we can create a new PostgreSQL user for this database as follows:

```
create user pi with encrypted password 'raspberry';
grant all privileges on database school to pi;
```

These commands create a new user name `pi` with password `raspberry` and privileges to use the new `school` database. We can then type `\q` to quit psql and connect to the school database using this new username and password as follows:

```
psql school pi
```

An interactive prompt should appear with full access to the school database. To access a different database, you must initiate a new connection as follows:

```
psql database username
```

where `dbname` and `username` are the database name and username you wish to use.

6.1.3.1 Creating Tables To create a table within the `school` database, you use the CREATE TABLE statement. For example, to create a table of student names, type the following:

```
CREATE TABLE students (
       studentnumber int NOT NULL,
       lastname varchar(50) NOT NULL,
       firstname varchar(50) NOT NULL
       );
```

The table name is specified after CREATE TABLE statement and then the columns names are given followed by data type, size, NOT NULL or not. A field which is specified as NOT NULL must contain a value. Identifiers such as column names are converted to lowercase in PostgreSQL unless they are enclosed in double quotes. In order to view details about a table, including information about fields and data types, use the DESCRIBE statement. For example, type:

```
\d students;
```

This will return information about the table we just created and information about the fields that comprise it. The output from PostgreSQL should resemble the following:

```
+---------------+-----------------------+-----------+----------+
|    Column     |         Type          | Collation | Nullable |
+---------------+-----------------------+-----------+----------+
| studentnumber | integer               |           | not null |
```

```
| lastname      | character varying(50) |           | not null |
| firstname     | character varying(50) |           | not null |
+---------------+-----------------------+-----------+----------+
```

To modify the structure or type of data in existing tables, PostgreSQL provides an ALTER command. The data type of each of PostgreSQL fields or attributes must also be specified. PostgreSQL supports a variety of numeric, character and binary data types. Some of the common data types supported in PostgreSQL are summarized in the following table:

Data Type	Description
integer	A signed integer (4 bytes)
double precision	Double precision floating-point number (8 bytes)
date	Date in the format YYYY-MM-DD
timestamp	Date and time
char [(n)]	A fixed-length string with a length of n characters
varchar [(n)]	A variable-length string with a length of up to n characters
text	variable-length character string

In order to show all the tables in a database, you can use the SHOW TABLES statements as follows:

```
\dt
```

To add records to the students database, use the INSERT statement. For example, to add "John Calvin" with a student ID number of 12345 into the students table, do the following:

```
INSERT INTO students
(studentnumber,firstname,lastname) VALUES
(12345, 'John', 'Calvin');
```

6.1.3.2 PostgreSQL Queries You can also ask PostgreSQL to search for data by submitting a query asking for all the records or rows that match a specific criteria. The SQL SELECT statement is used to perform queries on an SQL table. For example, to list all the students in the table, type:

```
SELECT * FROM students;
```

The * indicates that all columns should be returned. The SELECT query returns the requested data as text in a tabular format like the following:

```
  studentnumber | lastname | firstname |
```

```
+---------------+----------+-----------+
          12345 | Calvin   | John
```

To display specific columns, replace the * with a comma-separated list of columns that you would like to see displayed. For example, to display just the lastname and firstname columns, type:

```
SELECT lastname, firstname FROM students;
```

To list the students in alphabetical order using the ORDER BY clause as follows:

```
SELECT * FROM students ORDER BY lastname, firstname;
```

The order can be explicitly set to be ascending or descending by placing the ASC or DESC keywords at the end of the query. To prevent the SELECT statement from returning duplicate values in the results, the DISTINCT keyword can be used. For example, to list all the distinct first names in the students table, type:

```
SELECT DISTINCT firstname FROM students;
```

It is also possible to restrict the search to meet specific criteria using the WHERE keyword. For example, to list all the students who have the first name of "John", type:

```
SELECT * FROM students WHERE firstname = 'John';
```

The conditions to restrict search results can be further combined with boolean operators such as AND and OR operators to express search based on different conditions. In addition to WHERE, there are other keywords such as LIKE and BETWEEN which can be used to restrict the results of a search. For example, to list all the students whose first name starts with a "J", type:

```
SELECT * FROM students WHERE firstname LIKE 'J%';
```

where % is a wildcard which matches any character or sequence of characters. The BETWEEN keyword will restrict a search to values that fall in a range between some minimum and maximum value. For example, to return all the last names that lie alphabetically between "Calvin" and "Luther", type:

```
SELECT * FROM students WHERE lastname
BETWEEN 'Calvin' AND 'Luther';
```

One can also query the number of rows that match a certain condition using the COUNT keyword:

```
SELECT COUNT(*) FROM students WHERE firstname LIKE 'J%';
```

To delete data from a table, use the DELETE statement. For example, to delete all the students with the last name of "Calvin", do the following:

```
DELETE FROM students WHERE lastname = 'Calvin';
```

To make another table called courses that stores a course code and student number for each course a student is enrolled in, use the CREATE statement again as follows:

```
CREATE TABLE courses (
       studentnumber int NOT NULL,
       coursecode varchar(7) NOT NULL );
```

In this table the studentnumber field will not necessarily be unique since it will appear once for each course in which a student is enrolled. The `coursecode` will also not necessarily be unique since it will be repeated for each student in the course. Note that the student names do not need to be stored again; they can be retrieved if required by looking up the `studentNumber` in the students table. To add some records to the courses database, use the INSERT statement once again:

```
INSERT INTO courses (studentnumber,coursecode)
VALUES (12345, 'CSC101A');
```

To change or modify data in a table, use the UPDATE keyword as follows:

```
UPDATE students SET lastname = 'knox'
WHERE studentnumber = 12345;
```

This statement will modify the students table and replace the `studentnumber` for the student with the `lastname` knox. It is possible to modify multiple field values with an UPDATE statement using a comma-separated list of assignments. The WHERE clause in this case uses a boolean AND operator to make a more complex condition. You can also ask PostgreSQL to search data from multiple tables by using a JOIN operation. The JOIN keyword relates two or more tables, typically by using values that are common between them. The students and courses database have a common value of `studentnumber` that can be used to join them. The ON keyword can be used to specify a condition with which to join tables. For example, to list all the first names and last names of students enrolled in CSC101A, you can use a join operation based on the condition of matching a `studentnumber` in a query as follows:

```
SELECT firstname, lastname
FROM students
   JOIN courses
   ON students.studentnumber = courses.studentnumber
WHERE coursecode = 'CS101A';
```

It is also possible to delete a table and all it contents using the DROP command. This command should be used with care since it permanently deletes your table and cannot be undone.

```
DROP TABLE students;
```

To close a database connection and quit `psql`, type the following:

```
\q
```

Drill Exercise

Add some two more students to the `students` table. Add two more entries to the `courses` table using the `studentnumber` of the two students you added for a course named `CS326`. Use a `SELECT` with a `JOIN` to show all the students in `CS326`.

6.2 Cloud Databases

It has become quite common to store data in a cloud database. There are a variety of cloud services available, including services that support MySQL and PostgreSQL. These services require a subscription (and some offer free trial plans you can try out).

Once you have registered for a cloud service, you will be given a **URI** (Uniform Resource Identifier) which will have a format something like the following:

```
postgres://username:password@hostname/database
```

The URI is an unique string which includes information about the `username`, `password`, and `hostname` of your cloud service. This will be used to connect to the cloud database from your Raspberry Pi.

Next, `ssh` into the Raspberry Pi. Assuming you are using a PostgreSQL cloud service, you will need to install the PostgreSQL interactive client as follows:

```
sudo apt install -y postgresql-client
```

This command installs `psql`, a terminal-based front-end for PostgreSQL. Copy the URI given by your cloud service provider and enter the following on the command line of your Raspberry Pi:

```
psql postgres://username:password@hostname/database
```

where `postgres://username:password@hostname/database` is the URI provided by your cloud service provider. An interactive prompt should appear when you press enter and a successful connection is made.

> **Note**: The structure of the URI includes sections made up of the `username`, `password`, `hostname`, and `database` name. Consequently, your URI should always be kept confidential.

Next, create a table for the data you wish to store in the cloud. For instance, to create a new SQL table to store temperature data type the following SQL command at the `psql` prompt:

```
CREATE TABLE temperaturedata (datetime timestamp DEFAULT
CURRENT_TIMESTAMP, temperature FLOAT NOT NULL);
```

The DEFAULT keyword allows the datetime field to be automatically set to the current date and time when adding a new temperature value. Don't forget the semicolon at the end of the command to indicate that your SQL command is complete! Check that the table was successfully created by typing:

```
\d+
```

The output should list the specifications and fields for your new table. To quit the PostgreSQL client, type \q at the psql prompt.

Your database should now be set for recording a date and time and temperature in the cloud! To use Python with a cloud database, you will need to install the psycopg2 library as follows:

```
sudo apt install python3-psycopg2
```

Now you are ready to use a cloud database in a Python program. As an example, enter the following sample code and be sure to change the URI constant so that it contains your personal URI for your cloud database.

```
'''
This program stores temperature to a PostgreSQL cloud database.
'''
import sys
import signal
import urllib.parse as up
import psycopg2

# Constants
TIMEZONE = "America/Detroit"
URI = 'postgres://username:password@hostname/database'  # cloud database
    URI
TABLE = 'temperaturedata'

# Connect to the SQL cloud database
up.uses_netloc.append("postgres")
uri = up.urlparse(URI)
conn = psycopg2.connect(database=uri.path[1:], user=uri.username, password
    =uri.password, host=uri.hostname, port=uri.port )

# Open a cursor to perform database operations
cursor = conn.cursor()

# Set the local timezone for this session
cursor.execute(f'SET TIME ZONE "{TIMEZONE}"')
conn.commit()

# Continuously loop prompting for temperature data
while True:
    temp = input("Enter a temperature reading in Degrees Celsius. Type '
```

```
        quit' to exit: ")
    if temp == 'quit':
        break
    sqlcmd = f"INSERT INTO {TABLE} (temperature) VALUES ({float(temp)})"
    cursor.execute(sqlcmd)
    conn.commit()

conn.close()
print('Done')
```

Note that this program continuously prompts for a new temperature until the user enters `quit`. Each new temperature entered is inserted into the cloud database. Test that values are being entered into your database by typing the following in your `psql` terminal:

```
SELECT * FROM temperaturedata;
```

You should see a list of rows for each temperature sample that has been inserted in the table and the times for each.

6.3 Vector Databases

Vector databases are a speical form of database for storing vectors. Vectors can be used to store a fixed-length lists of numbers, which could be data representing high-dimensional data such as images, audio, and text.

Vectors can also be used to represent text data in applicatios such as Large Language Models (LLMs) by converting text into numerical data in a way that captures the structure of the original text. One technique for converting text data to vector representatins is Word2vec, a technique used in natural language processing (NLP).

Vector databases typically include a *similarity search* feature to find vectors with the closest match to a query vector. One common technique for checking similarity is something called *euclidean distance*, which computes the length of a line separating two data points. Vector databases can also be indexed to support searches with low latency. An example of an open source vector database is the Milvis database. Milvus includes libraries for popular languages like Python. For more information, see the Milvus tutorials.

7 Embedded Systems and the Internet of Things

The Raspberry Pi is a single board computer that can be used as an *embedded system*. An embedded system is one that combines computer hardware and software with various sensors and actuators within another device to perform a specific function. Examples of embedded systems include robots, automobiles, factory automation, and home appliances.

The Raspberry Pi includes a GPIO port that can serve as a means of interfacing inputs and outputs with the real world. Controlling the GPIO port is relatively simple.

7.1 Reading GPIO inputs

GPIO pins can also be configured as a digital input, which allows the Raspberry Pi to read the input state: either a logic high or low. Note that an input pin that is not left unconnected will "float" and will not necessarily read a logical zero. A floating input is unpredictable: it may be high, may be low, or somewhere in between. For this reason, a *pull-up resistor* or *pull-down resistor* can be used to give an input pin a defined default state, even if there is nothing is connected to it.

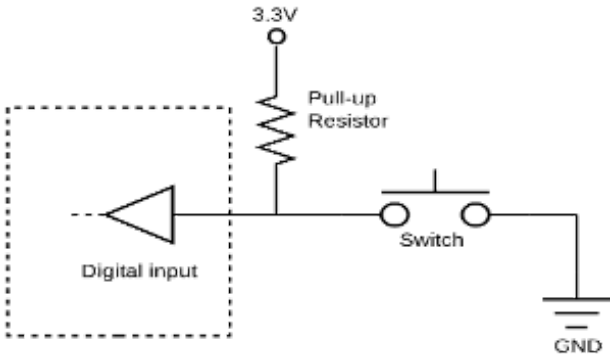


Figure 14: Input circuit with pull-up resistor

Conveniently, the Raspberry Pi includes internal pull-up and pull-down resistors that can be enabled if need be. These can be configured at the time that the GPIO pin is being configured.

7.1.1 GPIO Input Events

The Python `gpiozero` library includes detection of different input events that can be used to call a *callback* function. Examples of these events and the Python code to call a callback function named `my_callback` are listed below: detection for the following: 1. Rising edges

```
from gpiozero import Button
```

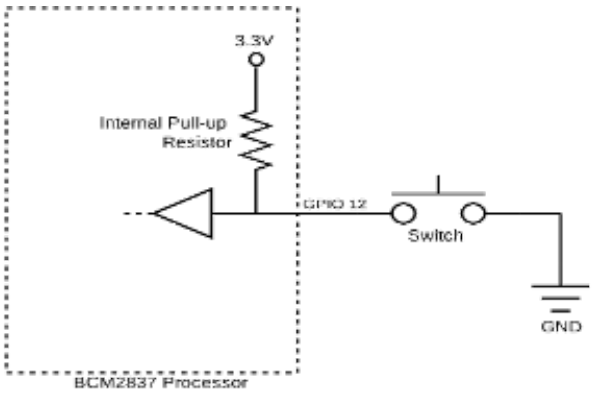


Figure 15: Input circuit with *internal* pull-up resistor

```
from signal import pause

button = Button(18, edge="rising")

def button_pressed(button):
    print”(Rising edge detected”!)

button.when_activated = button_pressed
pause()
```

2. Falling edges

```
from gpiozero import Button
from signal import pause

button = Button(18, edge="falling")

def button_pressed(button):
    print(Falling edge detected”!)

button.when_activated = button_pressed
pause()
```

Note that when a mechanical switch closes, the metal contacts may mechanically “bounce” for a short time before it settles. The result will produce a “chatter” of on/off spikes which may appear like multiple on/off transitions at a digital input. Therefore the above code includes a setting for `debounce_time` to deal with switch bouncing. The `debounce_time` paramter ensures that the callback function (`my_callback`) is triggered only once within the specified period (100ms in this example) after a button press or release event. Debouncing is accomplished by adding a delay that is *longer* than the mechanical bounce time of the switch.

Some sample Python code that sets up an event to detect falling edges with software debouncing is

given below:

```
from gpiozero import Button
from signal import pause
import time

# Use GPIO 21 as button input with pull-up and debounce enabled
button = Button(21, pull_up=True, edge="falling", bounce_time=0.1)

def my_callback():
    print("Falling edge detected!")
    print(time.time())

# Attach event handler for falling edge
button.when_activated = my_callback

print("Waiting for events. Press Ctrl+C to exit.")
pause()  # Keep the program running
```

7.2 Setting GPIO outputs

GPIO pins can also be configured as digital outputs, which allows the Raspberry Pi to set the state or a pin to either a logic high or low. These outputs set the state of a pin either to 0 volts (low) or 3.3 volts (high). These outputs have relatively weak current drive, but can be interfaced to power switches or relays to control larger loads.

The digital outputs on the Raspberry Pi do have enough drive to turn on a standard LED, if they are connected through a suitable current-limiting resistor. A typical setup is indicated in the schematic diagram below.

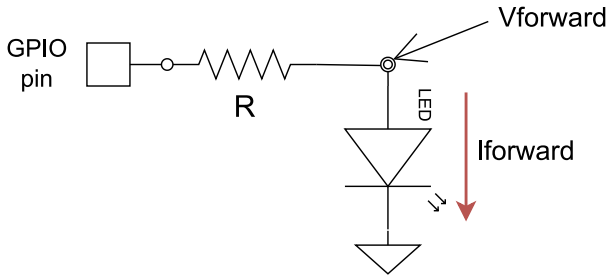


Figure 16: LED current limiting resistor circuit

Selecting a suitable resistance, R, will depend on the forward current required to light up the LED, as well as the the forward voltage of the LED. The required resistance, R, to achieve the appropriate LED

current is determined by solving the following equation (based on Ohm's Law):

$$R = (V_{GPIO} – V_{forward})/I_{forward}$$

where:
R = the value of series resistor required
V_{GPIO} = the voltage of the GPIO port when turned on (approximately 3.3V)
$V_{forward}$ = the "forward voltage" of the LED when it is on (approximately 1.8V)
$I_{forward}$ = the forward current required to illuminate the LED (roughly 1.5mA)

According the datasheet for a particular LED, $V_{forward} \approx 1.8V$ and $I_{forward} \approx 1.5mA$. Using these values, we get a resistance of roughly $1000ohms$ or $1kohm$. A suitable wiring diagram connecting the LED to the Raspberry Pi GPIO port pin is shown below (note the resistor color codes displayed in the diagram reflect a different resistance).

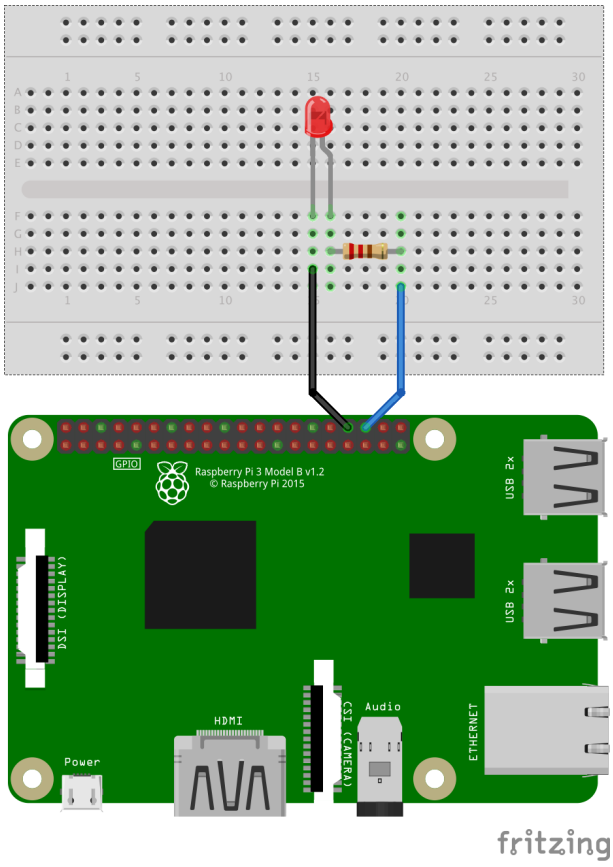


Figure 17: BCM 16 connected via resistor to an LED

> Caution should always be observed when connecting GPIO pins to the outside world. Ensure that the power remains **off** while you are performing wiring and always check your wiring before applying power! For some general guidelines to help ensure you do not damage your Raspberry Pi , refer to the section on Troubleshooting Tips.

7.2.1 Controlling GPIO outputs in a Program

The following is an equivalent Python source file that uses the `gpiozero` library to flash an LED and then read a digital input. Note that an internal pull-up is configured on the digital input.

```
from gpiozero import LED, Button
import time

# Use GPIO 16 as output and GPIO 12 as input with pull-up enabled
led = LED(16)
button = Button(12, pull_up=True)

# Turn output on and off
led.on()
time.sleep(1.0)
led.off()

# Read input
if button.is_pressed:
    print('BCM 12 is LOW')
else:
    print('BCM 12 is HIGH')

print("Done")
```

Save the file as `testgpio.py` and then run it as follows:

```
python3 testgpio.py
```

To discover more, visit the gpiozero library documentation.

Next is a sample program that will flash the LED on and off with a delay of 1 second coded in the C programming language using a GPIO library called `pigpio`. Edit the program using a standard text editor and name the source file `blink.c`.

```
#include <stdio.h>
#include <pigpio.h>
#include <unistd.h>

#define LED 16
#define DELAY 1
```

```
int main (int argc, char *argv[])
{
   if (gpioInitialise()<0) return 1;

   gpioSetMode(LED, PI_OUTPUT);
   while (1)
   {
      gpioWrite(LED, PI_ON);
      printf("LED ON\n");
      sleep(DELAY);
      gpioWrite(LED, PI_OFF);
      printf("LED OFF\n");
      sleep(DELAY);
   }
   gpioTerminate();
   return 0;
}
```

Before you can run you must compile the program from the command line as follows:

```
gcc -Wall blink.c -o blink -lpigpio
```

Finally, run the program in the terminal by typing:

```
sudo ./blink
```

Note that this code requires `sudo` privileges. Once started, the LED should begin blinking. Type `ctrl-C` to exit.

7.2.2 Pulse Width Modulation (PWM) Outputs

A PWM (Pulse Width Modulation) signal can be used to control certain kinds of output devices. A PWM signal can vary the amount of power delivered to an output device by changing the duty cycle of the signal. It is also commonly used with microservo motors to vary the width of a pulse to set the position of a microservo motor. The duty cycle of a PWM signal is illustrated below:

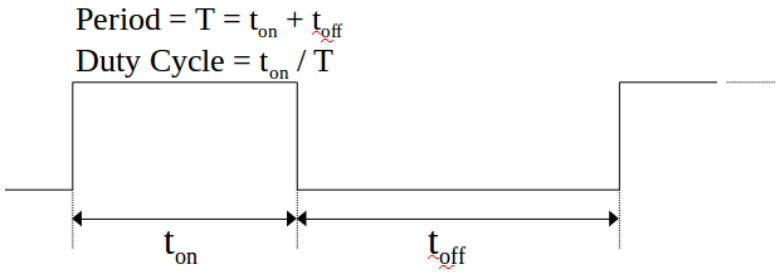


Figure 18: Diagram showing a PWM duty cycle

The `gpiozero` library has the capabilty of generating PWM outputs on specified GPIO pins. Using the circuit shown in 17, we can progran a PWM output so that the duty cycle will vary the brightness of an LED Enter the following Python program which allows the user to set the duty cycle for a software PWM signal to the LED:

```
from gpiozero import PWMLED

# Use GPIO 16 as PWM output
led = PWMLED(16)

duty_cycle = 0
led.value = duty_cycle / 100  # Start with 0% duty cycle

while True:
    try:
        duty_cycle = int(input('Enter a PWM duty cycle (0-100, enter -1 to
            end): '))
        if duty_cycle == -1:
            break
        if 0 <= duty_cycle <= 100:
            led.value = duty_cycle / 100
            print(f'Duty cycle={duty_cycle}%')
        else:
            print("Please enter a value between 0 and 100.")
    except ValueError:
        print("Invalid input. Please enter a number.")

led.off()
print("Done")
```

Note how the LED brightness varies with the duty cycle.

7.2.2.1 Using PWM to Control a Microservo Servo motors are actuators that allow you to add motion to a system. They're useful because you can specify an angle to turn and the micro servo will automatically adjust the position for you. An ordinary motor will simply turn when power is applied, but the micro servo includes electronics, gears, and a feedback sensor to control the position of the output. Servos typically come with multiple attachments, such as wheels or levers (known as "horns") that attach to the shaft and can be coupled to whatever mechanical device they are operating.

Servo motors are typically controlled using PWM signals. The angular position of the servo motor is typically set by the length of a control pulse, one that is typically sent roughly every 20 milliseconds. By precisely setting the width of the pulse, one can adjust the position of the servo.

It is important to have a stable PWM signal to avoid jitter in the servo motor, hence it's recommended to use *hardware PWM* signals rather than *software PWM* signals. Hardware PWM signals use dedicated hardware to generate a PWM signal directly whereas software PWM relies on software timers to turn a

pulse on and off, and is therefore subject to various software latencies. The Raspberry Pi has support for both hardware and software PWMs.

7.3 GPIO Serial Communications

The Raspberry Pi has three types of serial interface on the GPIO header:

1. I^{2}C (Inter-Integrated-Circuit bus)
2. SPI (Serial Peripheral Interface)
3. serial UART (Universal Asynchronous Receiver/Transmitter)

The serial UART was mentioned briefly in section 1.6.1, and the two other serial interfaces are explored further in the following two sections.

7.3.1 Using I^{2}C

Certain GPIO pins also be used to perform I^{2}C communications. I^{2}C (Inter-Integrated Circuit) is a synchronous serial bus used to connect peripheral chips to processors and microcontrollers. I^{2}C requires only two bidirectional wires: the serial data line (SDA) and serial clock line (SCL).

A set of I^{2}C command line tools can be installed as follows:

```
sudo apt-get install i2c-tools
```

In order for these to work, the I^{2}C kernel module should be enabled using the `raspi-config` utility. Next, ensure your username is in the group permissions for talking to I^{2}C devices as follows:

```
sudo usermod -a -G i2c user
```

where `user` is set to your username. You can then scan the I^{2}C bus for devices as follows:

```
i2cdetect 1
```

If an I^{2}C device is present and wired properly, you should see a device reported along with its I^{2}C address. To query a specific device, issue the following command:

```
i2cget -y 1 address 0 b
```

where `address` is the I^{2}C address of the device you are interested in querying. If everything is wired correctly and functioning, you should a see value returned from this query.

Question: What value is returned from the `i2cget` command? Which base is this number displayed in and what are the units?

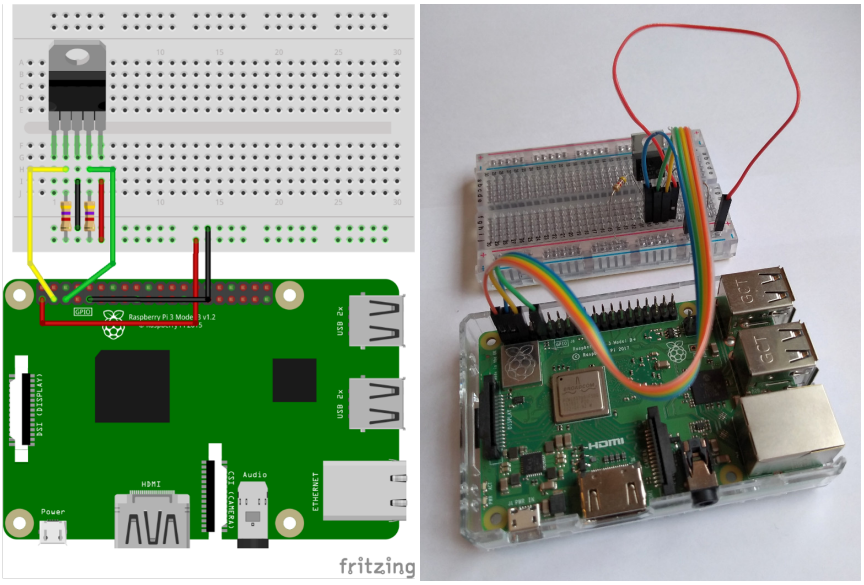
7.3.1.1 I²C Temperature Sensor Example An example of a device that uses an I²C interface is the TC74 temperature sensor. The TC74 temperature sensor comes in a TO-220 package with five legs. Each of the five leads on the device are summarized in the table below (and described in more detail in the datasheet):

Pin number	Description
1	No connect
2	I²C Serial Data (SDA)
3	System Ground (GND)
4	I²C Serial Clock (SCLK)
5	Supply Voltage (VDD)

The TC74 communicates using an I²C serial connection and requires two pullup resistors on the SDA a SCLK lines. These pullup resistors should be placed from the SDA and SCLK lines to the VDD supply and should have a value of around 4.7kohms. For the TC74A0-3.3VAT part number, the supply voltage should be set to 3.3volts.

> Note that there are variants of the TC74 where the nominal VDD is 5volts. See the datasheet for more details.

To connect the temperature sensor to a Raspberry Pi using the I²C interface, one can use a breadboard as shown in the diagram and photo below. Note that the leads of the TC74 device will need to be spread slightly so that it can be inserted into a breadboard. Consult a GPIO pinout diagram to ensure your wiring is correct (in particular, do not confuse the 3.3V and 5V pins on the GPIO since they are next to each other). An image of the complete wiring is shown below.



Once the wiring is complete, power up your Raspberry Pi. Ensure the I^{2}C command line tools are installed, the I^{2}C kernel module enabled, and the group permissions set as described in the preceding section. Next, scan the I^{2}C bus for the TC74 temperature sensor to ensure it can be found:

```
i2cdetect 1
```

If the TC74 is present and wired properly, you should see a device reported at address 48hex. Next, test if it is possible to read the temperature from that address as follows:

```
i2cget -y 1 0x48 0 b
```

If everything is wired correctly and functioning, you should see a value returned from this query.

Next, install the system management bus library for accessing the I^{2}C bus in Python as follows:

```
sudo apt-get install python3-smbus
```

The SMBus enables I^{2}C for communications and can be used to communicate with simple devices. We can test the sensor by creating and running the program below:

```
# This program periodically reads an I2C TC74 temperature sensor
# and prints the reading
import smbus
import time

# constants
BUS = 1             # I2C bus number
ADDRESS = 0x48      # TC74 I2C bus address
DELAY = 0.5         # delay between reads

# Connect to I2C bus
```

```
bus = smbus.SMBus(BUS)

try:
    while True:
        temp = bus.read_byte(ADDRESS)
        print(f'{temp} degrees C')
        time.sleep(DELAY)
except KeyboardInterrupt:
    bus.close()
    print('Done')
```

Run the program and verify that the temperature sensor is being read from the I^2C bus.

7.3.2 The SPI Interface

Another standard interface is SPI (Serial Peripheral Interface). SPI is used for communications between integrated circuits.

Before using SPI, make sure it is enabled using the `raspi-config` tool first. The presence of the SPI kernel modules can be verified by typing:

```
lsmod | grep spi
```

Next, run the following command:

```
ls -l /dev/spi*
```

This command lists device files that indicate the presence of SPI interfaces, and two should be present. An example of a SPI device is the MCP3008, an analog-to-digital converter (or A/D converter). This device could communicate with the Raspberry Pi by directly wiring it to special purpose SPI pins in the GPIO.

7.4 Introduction to MQTT for IoT

A Raspberry Pi equipped with sensors and actuators can be controlled and monitored using machine-to-machine (M2M) communications. M2M provides an important enabling technology for applications like **IoT** (the Internet of Things). One type of M2M is a protocol called *Message Queuing Telemetry Transport Protocol* or MQTT. MQTT is a bandwidth-efficient, lightweight protocol allowing clients to publish and subscribe data to a special "broker" server.

Data sensors (like the temperature sensor shown in figure 19) use MQTT to publish data to a broker and clients can monitor that data by *subscribing* to a broker. Publishers and subscribers don't need to know each other, only one connection to the broker is required. An advantage of this is approach is

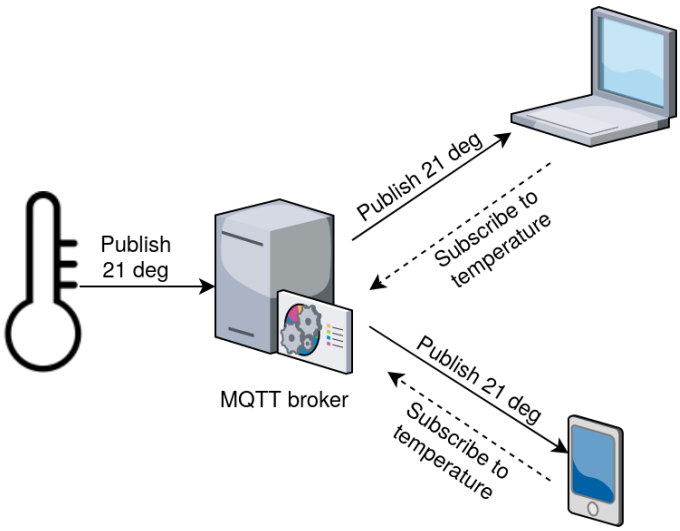


Figure 19: A diagram illustrating a temperature device using MQTT to publish temperature data to clients.

that IoT devices can remain behind a firewall since clients do not need to directly connect to the device. Moreover, subscribers can porentially receive data from many different publishers, and publishers can send data to many different subscribers.

An MQTT packet must include an MQTT *topic*, which is a string that describes the data and is used by the broker to filter messages for each client. A topic consists of one or more topic *levels* separated by a forward slash. For example, `Calvin/North Hall/NH296/temperature` is a topic string with a top level of `Calvin` with various sub-topic levels. The rules for topic strings include:

- a topic must have at least 1 character
- a topic may contain spaces
- topics are case-sensitive

Clients can use *wildcards* to subscribe to multiple topics at once. For example, the topic string `Calvin/NorthHall/+/temperature` will subscribe to all the temperature topics in Calvin's North Hall. Multi-level wildcards can also be used, for example, `Calvin/NorthHall/#` subscribes to all topic strings in Calvin's North Hall, Topics beginning with `$` are used for internal statistics, for example `$SYS/broker/uptime` shows the time that the broker has been up-and-running.

The payload of an MQTT packet is data-agnostic, so a programmer can choose how the payload is structured. An MQTT payload is made up of *bytes* and may contain:

- Strings
- Binary data
- JSON (JavaScript Object Notation)

7.4.1 Sending MQTT messages from the command line

The Eclipse Mosquitto project provides an open source MQTT message broker and tools. To install the mosquitto client tools, enter the following:

```
sudo apt install mosquitto-clients
```

These client tools will allow you to publish and subscribe messages to a broker. For this guide, we will make use of a public MQTT broker. Two options are `mqtt.eclipse.org` or `test.mosquitto.org`. These public MQTT brokers are open and do not require usernames or passwords (which comes with some security implications).

To send an MQTT message to a broker using the command line, type:

```
mosquitto_sub -h test.mosquitto.org -t raspberry/test
```

This command uses the MQTT protocol to *subscribe* with the broker server `test.mosquitto.org` to the topic `raspberry/test`.

Next, switch to another terminal on your local Raspberry Pi (or use another Raspberry Pi) and *publish* a message to the same topic and broker server by entering the following:

```
mosquitto_pub -t raspberry/test -m "Hello World" -h test.mosquitto.org
```

Note that after you enter the above command you should see the “Hello World” message appear where you subscribed to the broker. You have now successfully transfered a message using MQTT. Note that the message was transferred without needing to know the IP address of the publisher or subscriber since messages are nicely handled through a broker server.

7.4.2 Controlling an LED using Python and MQTT

Python programs can be written to publish and subscribe to MQTT messages. To begin, install the Python MQTT library on your Raspberry Pi using the command line as follows:

```
sudo pip3 install paho-mqtt
```

Next, consider the GPIO controlled LED as shown in Figure 17. MQTT can be used to control this LED for a remote Raspberry Pi using the following code:

```
from gpiozero import LED
import paho.mqtt.client as mqtt

# Constants
TOPIC = 'raspberry/LED'
PORT = 1883
```

```
QOS = 0
KEEPALIVE = 60

# Set hostname for MQTT broker
BROKER = 'test.mosquitto.org'

# Configure GPIO for LED output
led = LED(16)

# Callback when a connection has been established with the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print(f'Connected to {BROKER} successful.')
    else:
        print(f'Connection to {BROKER} failed. Return code={rc}')

# Callback when client receives a message from the broker to toggle LED
   state
def on_message(client, data, msg):
    print(f'MQTT message received -> topic:{msg.topic}, message:{msg.
       payload}')
    if msg.topic == TOPIC:
       if led.value == 1:
          led.off()
          print("LED off")
       else:
          led.on()
          print("LED on")

# Setup MQTT client and callbacks
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# Connect to MQTT broker and subscribe to the topic
client.connect(BROKER, PORT, KEEPALIVE)
client.subscribe(TOPIC, qos=QOS)

try:
    client.loop_forever()
except KeyboardInterrupt:
    client.disconnect()
    led.close()
    print('Done')
```

This program subscribes to a broker and waits for certain MQTT messages to remotely control the LED. For example, the following command issued from a remote computer will toggle the state of the LED on the Raspberry Pi:

```
mosquitto_pub -t raspberry/LED -m "test" -h test.mosquitto.org
```

Each time an MQTT message is received with the topic `raspberry/LED`, the state of the LED will be toggled on or off.

> Note there are security implications since this code uses a public MQTT broker and these MQTT messages are sent in the clear. Anyone could potentially publish messages to the public broker with the same topic and therefore control the LED. There are ways to encrypt and authenticate MQTT messages that are beyond the scope of this guide.

For more information about MQTT, see the MQTT webpage and the Mosquitto project webpage.

7.4.3 Using MQTT to control Zigbee Devices

It is also possible to use MQTT on the Raspberry Pi to control a network of smart devices and sensors connected over a Zigbee wireless network. **Zigbee** is built on the IEEE 802.15.4 specification to enable a low-cost, low-power wireless network of smart devices for home automation. There are a plethora of Zigbee devices available for sensing things like temperature, pressure, and humidity and to control devices such as lights and switches. Some example of Zigbee devices that have been used in the author's home are illustrated in the figure below.

Figure 20: Various Zigbee devices that can be used for home automation.

Zigbee2MQTT is a program that provides a wireless *bridge* between Zigbee and MQTT and thus allows you to control Zigbee devices through MQTT messages.

The first step is to setup a local MQTT broker using `mosquitto`. This can be installed alongside the `mosquitto` client utilities on the command-line as follows:

```
sudo apt install -y mosquitto mosquitto-clients
```

We can configure `mosquitto` to act as a local MQTT broker and listen *only* on the local loopback interface by adding the following lines in `/etc/mosquitto/conf.d/local.conf`:

```
listener 1883 127.0.0.1
allow_anonymous true
```

Next, enable the `mosquitto` broker service as follows:

```
sudo systemctl enable mosquitto.service
```

Ensure the mosquitto service is now running by typing:

```
sudo service mosquitto status
```

> Note that any service started on the Raspberry Pi that opens a port exposes you to potential security issues.

Next, there are several dependencies for **Zigbee2MTT** that need to be installed from the command-line as follows:

```
sudo apt-get install -y npm git make g++ gcc
```

Unfortunately, the Raspberry Pi repos may have an older version of the **nodejs** package, and Zigbee2MQTT requires a recent version of **nodejs**. You can add the repository and install a recent version of **nodejs** as follows:

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -
sudo apt install nodejs
```

Once the dependencies are installed, Zigbee2MQTT can be installed from github by typing the following commands:

```
sudo mkdir /opt/zigbee2mqtt
sudo chown -R ${USER}: /opt/zigbee2mqtt
git clone --depth 1 https://github.com/Koenkk/zigbee2mqtt.git /opt/
    zigbee2mqtt
cd /opt/zigbee2mqtt
npm ci
```

Note that the `npm ci` may produce some warnings which can be ignored. Zigbee2MQTT requires a YAML configuration file which may be edited by typing:

```
sudo nano /opt/zigbee2mqtt/data/configuration.yaml
```

Edit the configuration file so that it includes the following settings:

```
homeassistant: false
```

```
permit_join: true

# MQTT settings
mqtt:
  base_topic: zigbee2mqtt
  server: 'mqtt://127.0.0.1'

# Location of Zigbee USB adapter
serial:
  port: /dev/ttyACM0

# use a custom network key
advanced:
    network_key: GENERATE

# Start web frontend
frontend:
  port: 8081

# Enable over-the-air (OTA) updates for devices
ota:
    update_check_interval: 1440
    disable_automatic_update_check: false
```

Next, insert a **Zigbee adapter** into a USB port on the Raspberry Pi. The configuration above assumes the Zigbee USB adapter appears as `/dev/ttyACM0`. You can use the `dmesg` command to find the device file associated with your particular Zigbee USB adapter and then update the configuration file accordingly. Rather than hard-coding a unique network key, the `network_key` setting used above generates a new random key when Zigbee2MQTT is first run.

> ***Security Notes***
>
> It's recommended to disable `permit_join` after all the Zigbee devices have been paired with your Zigbee adapter to prevent further devices from attempting to join and possibly exposing the network key.
>
> Note that the `frontend` setting provides a web frontend at a specified port for viewing the Zigbee network. While this can be useful for setup and debugging, you may wish to disable it later.
>
> It is recommended that over-the-air (OTA) updates be enabled for all devices to keep them up-to-date.

Once the setup and configuration are complete, ensure the Zigbee USB adapter is inserted in the Raspberry Pi and start Zigbee2MQTT as follows:

```
cd /opt/zigbee2mqtt
npm start
```

This will build and launch `zigbee2mqtt` from the command-line. Once the it builds and launches successfully, you can exit the program by hitting `ctrl-c`. To launch automatically on boot under Linux, setup Zigbee2MQTT to run using systemctl. For more detailed informatoin about installing Zigbee2MQTT, refer to the official Zigbee2MQTT installation instructions.

Next, we need to establish a network of Zigbee devices by pairing each new device with the Zigbee hub on the Raspberry Pi. Zigbee2MQTT supports a plethora of Zigbee devices and a friendly device webpage includes notes on compatibility, pairing, and details on what values are exposed.

7.4.3.1 Pairing Zigbee devices Pairing can be easily accomplished using the web frontend to Zigbee2MQTT. The web frontend can be found by pointing a web browser to the IP address of the Raspberry Pi and the port number specified in the `configuration.yaml` file (port `8081` in the example file above). In the web frontend, click the `Devices` tab and then the button labelled `Permit join (All)`. Once this button is clicked a countdown will proceed during which time new devices can be paired to the Zigbee network (typically the countdown lasts for 255 seconds).

Typically a new device is paired by performing a factory reset of the device. The way to perform a factory reset varies by device type and manufacturer. For example, some bulbs can be factory reset by toggling the power a certain number of times and other devices can be factory reset using a reset button in a small pinhole. A few moments after resetting a device, the web page should report the pairing of the device. Clicking on the devices tab on the web page should display a list of paired devices along with each manufacturer, model, and IEEE address. The web frontend provides many nifty features like displaying a network map and the ability to perform updates on connected devices.

In addition to the IEEE address each Zigbee device may be configured with a "friendly name." By default, the "friendly name" is initialized to the IEEE address, but it is recommended that you assign a more meaningful "friendly name". For example, a bulb could be named "bulb1" or "porch light". This allows devices to be controlled and referenced using a *name* rather than relying on a cumbersome IEEE address. Keep a list of the "friendly names" since these will be needed later to control the devices.

7.4.3.2 Controlling Zigbee devices over MQTT Once devices have been paired, they can be controlled simply by sending specially crafted MQTT messages to the local broker. These messages must be published to the topic `zigbee2mqtt/FRIENDLY_NAME/set` where `FRIENDLY_NAME` is the friendly name for a device. In the case of a bulb or smartplug, sending a message of "ON" or "OFF" to the appropriate topic for the device will turn the device on or off.

MQTT messages can be sent from the command line on the Raspberry Pi using tools included with with

the `mosquitto-clients` package. For example, to turn on a light bulb with the friendly name of "bulb1" using the mostquitto client tool, type:

```
mosquitto_pub -h 127.0.0.1 -t zigbee2mqtt/bulb1/set -m "ON"
```

where `127.0.0.1` is the local loopback address to connect to the local MQTT broker and `zigbee2mqtt/bulb1/set` is the MQTT topic to control the settings for the device with the friendly name `bulb1`.

By subscribing the MQTT topic for a sensor you can receive updates from a sensor. Consult the online Zigbee2MQTT documentation for a complete list of MQTT topics and messages. For an example of using a Raspberry Pi with MQTT, Zigbee, and Python in a smart home application for controlling bulbs and monitoring sensors, visit the pi-home project.

7.5 Camera Sensors

The Raspberry Pi includes a dedicated camera port on the circuit board located between the Ethernet port and the HDMI port. This port can be used to install the standard Raspberry Pi Camera or the NoIR Camera which can be used in low light applications. Remove power before installing the camera cable. Be sure to consult the camera manual to ensure it is inserted properly and is snuggly inserted with the right cable orientation.

Once it is installed, you can test your camera by entering the following command:

```
libcamera-still -o output.jpg
```

If the command executes without error, a new image file named `output.jpg` should be created.

7.5.1 OpenCV

OpenCV is a comprehensive computer vision library. It contains over 2500 algorithms and is operated by the non-profit [Open Source Vision Foundation]. It supports a wide variety of platforms and computer languages. For example, to install the OpenCV library for Python, type: To install these libraries, enter:

```
pip3 install opencv-python
```

The following is a sample Python program that uses OpenCV to capture an image and store it to a file.

```
import sys
import cv2

```

```
# Initialize camera
print("Initializing camera...")
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print('Cannot open camera...')
    sys.exit(1)

# Capture a frame
ret, frame = cap.read()
if not ret:
    print('Frame capture failed...')
    sys.exit(1)

# Save frame
cv2.imwrite("image.jpg", frame)
cap.release()
```

7.5.2 AprilTags

AprilTags are a system of visual tags developed by researchers at the University of Michigan for use in robotics and other computer vision applications. Like QR codes, AprilTags are conceptually two-dimensional bar codes like the ones shown in Figure 21. However, AprilTags have simpler data payloads than QR codes (typically between 4 and 12 bits). In fact, AprilTags were specifically designed to work well with embedded devices, providing very good performance even with modest hardware (like the Raspberry Pi).

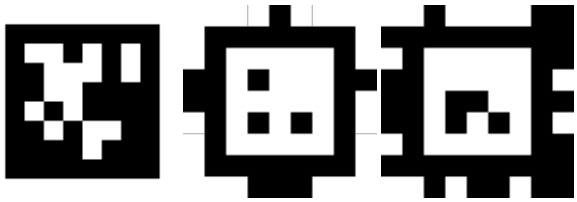


Figure 21: Examples of different AprilTags.

A simple python program can be made to demonstrate the use of AprilTags as follows. Declare a new Python virtual environment named `apriltags` and activate it as follows:

```
python3 -m venv --system-site-packages apriltags
source apriltags/bin/activate
```

Your prompt should change indicating that you are now in the `apriltags` virtual environment. Next, install the required Python libraries. We will use the Picamera2 library for image capture, the OpenCV library for image processing, and the pupil-apriltags library for AprilTags. To install these libraries, enter:

```
pip3 install opencv-python pupil-apriltags picamera2
```

Next enter the program as follows:

```
import cv2
from picamera2 import Picamera2
from pupil_apriltags import Detector

# Initialize camera
print("Initializing camera...")
picam2 = Picamera2()
config = picam2.create_still_configuration( )
picam2.configure(config)
picam2.start()

# initialize AprilTag detector
detector = Detector()

# Continuously capture frames from the camera
try:
    while True:
        # grab a frame and convert to grayscale
        frame = picam2.capture_array()
        img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect any AprilTags
        results = detector.detect(img)
        # Print detection details for all tags
        for r in results:
            print(f"AprilTag detected! ID: {r.tag_id}, Family: {r.
                tag_family}," f"Decision Margin: {r.decision_margin:.2f}")

except KeyboardInterrupt:
    print('Done')
    picam2.stop()
```

This program will print details about any AprilTags detected by the camera. To test the detection program, you will need to print out an AprilTag image from the AprilTag image repository. Note that there are many unique AprilTags available (different *families* and *IDs* of tags), but for now select any one of them. Note also that the image dimensions are small and will need to be rescaled before they are printed.

Test the code and ensure that you can detect an AprilTag when it is placed in front of the camera. Try moving the page closer, further back, and rotating the AprilTag. Note the robustness of the detector.

In the chapter that follows on artifical intelligence, we provide additional examples of image classification and recognition.

7.5.3 Computer Vision at the Edge

Computer vision at the edge refers to using edge devices to process visual data instead of sending the data to the cloud. Computer vision at the edge is a kind of "fog computing" which uses edge devices (or near-edge devices) to carry out processing rather than sending raw data to the cloud. One example of computer vision at the edge would be the recognition and classification of AprilTags, which could be computed locally on the Raspberry Pi and the results could then be sent to the cloud. Moreover, computer vision at the edge could use MQTT to send results to a broker in the cloud. Recall that MQTT is designed for *small* payloads, so it should not be used to send images or video frames. Rather, MQTT could be used to send the results of image computations, such as classification or measurement results (such as AprilTag detection data). See the previous section for a detailed description of MQTT.

The Raspberry Pi is well suited to be an edge device for computer vision. It supports a variety of different types of cameras, and there is now even a Raspberry Pi AI Camera which can be used to deploy neural network models directly in the camera module! The next chapter will go into depth about AI and how to work with it on the Raspberry Pi.

8 Exploring Artificial Intelligence

8.1 Introduction

In the early 2000's, I pursued a PhD in the field of robotics and computer vision. At the time, the field of AI was climbing out of an "AI winter," and I found myself attracted to newer machine learning methods that were being used for image recognition. I recall being astounded at the profound elegance of "training" a computer with a set of example images and then observing how well it could identify new images that were not part of the original training set. Even those early machine-learning techniques seemed magical.

In the years since I completed my graduate work, a myriad of helpful machine learning tools and libraries have emerged. What follows are some examples of machine learning tools that can be used with the Raspberry Pi.

8.2 Hardware and Software Support for AI

As discussed in a previous chapter, recent models of the Raspberry Pi processor includes multiple ARM cores. Despite its respectable hardware capabilities, the Raspberry Pi may struggle with the computational demands of more demanding artificial intelligence (AI) applications. For more demanding applications, there is a Raspberry Pi add-on board — referred to as a HAT (Hardware Attached on Top) — that provides an NPU (Neural Processing Unit) to accelerate machine learning computations. The Raspberry Pi AI HAT+ provides a high-performance, power-efficient AI processor for the Raspberry Pi 5.

Besides hardware support, there are a wide variety of powerful machine learning libraries which can be used with the Raspberry Pi, including scikit-learn and LiteRT. Python includes a number of powerful supporting libraries, such as Matplotlib and plotly for plotting, NumPy for providing fast operations on arrays, and Pandas for data analysis and manipulation. The following sections provide examples of some of these libraries in action.

8.3 SciKit Learn

SciKit Learn is an open source machine learning library that works with Python. SciKit Learn provides many different features, including regression and clustering algorithms, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Support Vector Machines (SVMs).

All forms of machine learning require a dataset which is used for training. SciKit includes a selection of toy datasets that can be used to experiment with the machine learning libraries. The following sections demonstrate PCA and SVMs using the classic iris flower dataset which is part of the collection of

toy datasets. This iris dataset was originally compiled by biologist Ronald Fisher in a well-known 1936 paper. This dataset comprises samples of three species of the Iris flower (*Iris setosa*, *Iris virginica*, and *Iris versicolor*) and measurements of the length and the width of both the *sepals*[1] and the *petals*. The **SciKit Learn** library includes this iris dataset for testing and demonstration purposes. The followig Python code uses Matplotlib to plot the sepal width versus the sepal length for each of the three iris classes in the dataset.

```
from sklearn import datasets
import matplotlib.pyplot as plt

# Load the classic iris dataset
iris = datasets.load_iris()

# Plot sepal lengths vs. widths for irises in dataset
fig, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set_title("Plot of Iris features: sepal lengths and widths")
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
fig = ax.legend(scatter.legend_elements()[0], iris.target_names, loc="best
    ", title="Classes of Irises")
plt.show()
```

Running this code results in the following plot of the iris dataset:

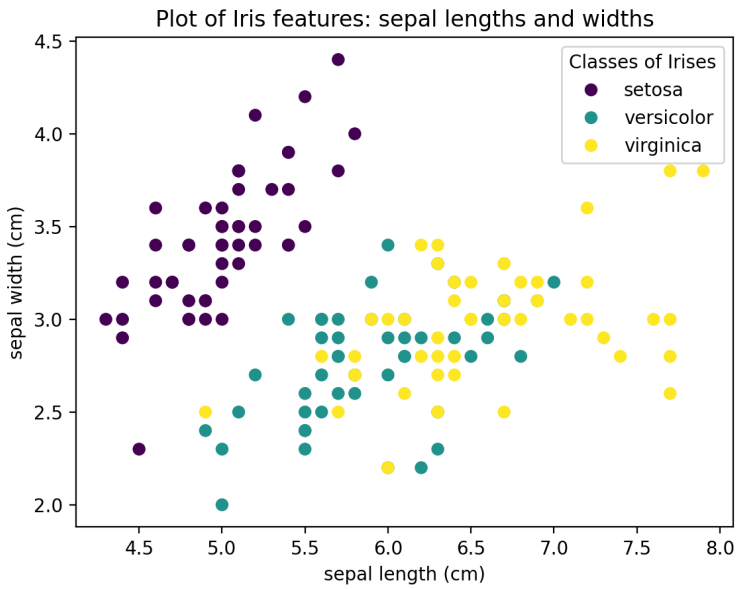


Figure 22: Iris flower dataset plotted with sepal length and width features

[1] A *sepal* is a green leaf-like structure at the base of a flower.

8.3.1 Linear Discriminate Analysis (LDA)

Linear Discriminate Analysis (LDA) is a technique that finds a linear combination of features to best separate the classes in a dataset. A program that performs an LDA transformation on the iris dataset is shown in the program below.

```
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Load the classic iris dataset
iris = datasets.load_iris()

# transform iris dataset using LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X = lda.fit_transform(iris.data, iris.target)

# Show plot with title and legend
fig, ax = plt.subplots()
scatter = ax.scatter(X[:, 0], X[:, 1], c=iris.target)
ax.set_title("Plot of LDA features of IRIS dataset")
fig = ax.legend(scatter.legend_elements()[0],
      iris.target_names, loc="best", title="Classes of irises")
plt.show()
```

The plot that is produced by this code is shown below.

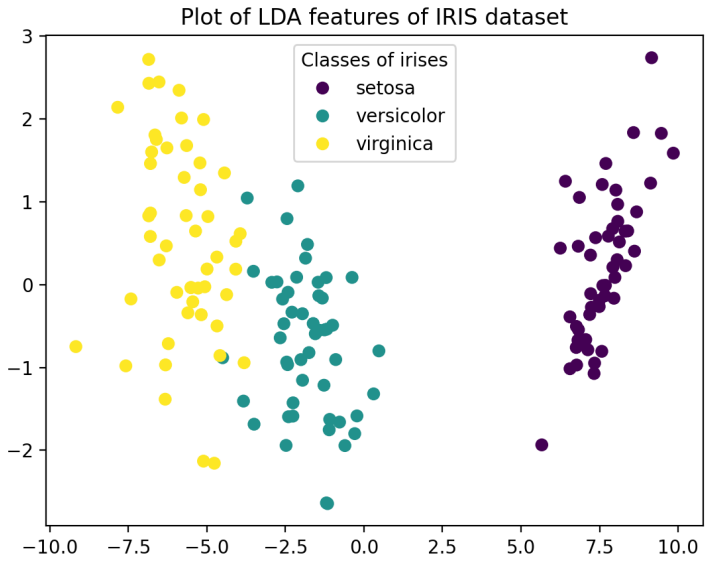


Figure 23: Iris flower dataset transformed with LDA

Comparing this plot to the one shown in Figure 22 *clearly* shows that the LDA transformation seperates the different classes of irises much better than just using sepal length and width as features.

8.3.2 Principal Component Analysis (PCA)

Next, we will use **Principal Component Analysis** (PCA) to transform the iris sepal lengths and widths to another set of features based on eigenvectors computed from the dataset. These eigenvectors provide an orthonormal axis that captures the statistically most significant directions in the data. To perform a PCA transformation on `iris.data`, one can use SciKit Learn library and the transformed data can be plotted on a new set of axes defined by the first two eigenvectors as follows:

```
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Load the classic iris dataset
iris = datasets.load_iris()

#legends, ax = plt.subplots()
fig, ax = plt.subplots()

# transform sepal length into eigenspace using PCA
pca = PCA(n_components=2).fit_transform(iris.data)

# Show plot with titles and axis names
scatter = ax.scatter(pca[:, 0], pca[:, 1], c=iris.target)
ax.set_title("Plot of first two PCA dimensions")
ax.set_xlabel("First Eigenvector")
ax.set_ylabel("Second Eigenvector")
fig = ax.legend(scatter.legend_elements()[0], iris.target_names,
                loc="lower right", title="Classes of irises")
plt.show()
```

This produces a plot using the eigenvectors as the axes as shown below:

Note that the plot of the iris data using the Eigenvector basis functions also appears to separate the classes more clearly. This is due to the fact that PCA produces eigenvector basis functions that capture the statistically most significant features in the data.

8.3.3 Support Vector Machines (SVM)

SciKit Learn can also be used with a machine learning approach called Support Vector Machines (SVMs). The goal of an SVM is to determine a *hyperplane* which defines the boundary between

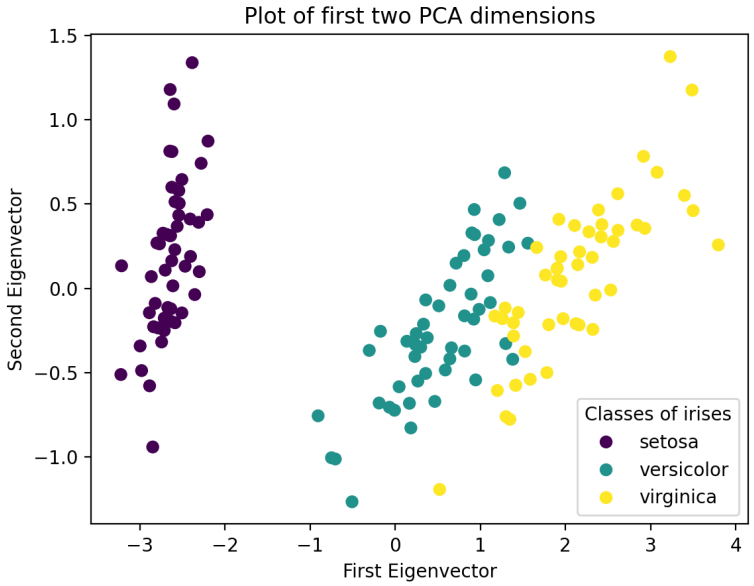


Figure 24: Iris flower dataset plotted using Eigenvectors

different data points in order to classify them appropriately. **SciKit Learn** includes a library for machine learning using an SVM.

The following example uses the iris flower dataset and uses SVM to computer a hyperplane between the different iris classes. This example is derived from the SciKit Learn SVM examples found on the SciKit documentation pages.

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm
from sklearn.inspection import DecisionBoundaryDisplay

# import classic iris dataset
iris = datasets.load_iris()

# Take the first two features from the iris dataset (sepal width and
   length)
X = iris.data[:, :2]
y = iris.target

# create an SVM to fit the data
clf = svm.SVC(kernel="linear", C=1.0)
clf.fit(X, y)

# Plot the data and support vectors
fig, ax = plt.subplots()
X0, X1 = X[:, 0], X[:, 1]
```

```
disp = DecisionBoundaryDisplay.from_estimator(clf, X,
       response_method="predict", cmap=plt.cm.coolwarm, alpha=0.8,
       ax=ax,xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_title('Support Vector Machine with Linear Kernel')
plt.show()
```

Running the above program displays the following plot, showing the sepal lengths and widths of the iris dataset along with the hyperplane boundaries which can be used for classification. Note that SVM algorithm was not able to compute a perfect linear decision boundary that perfectly separates the classes. Consequently, when used to identify irises this will likely lead to occasional classification errors.

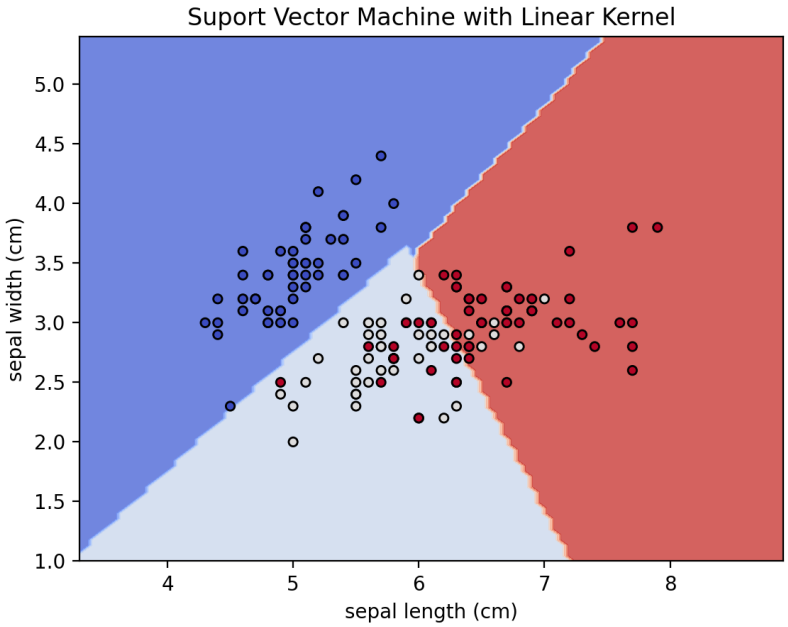


Figure 25: Iris dataset with three classes plotted with SVM boundaries

Finally, we can project the iris dataset features into eigenspace using PCA and then dertermine the support vectors for the new eigenspace as follows:

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.decomposition import PCA

# import classic iris dataset
iris = datasets.load_iris()

# transform the iris dataset features into eigenspace using PCA
```

```
pca = PCA(n_components=2).fit_transform(iris.data)

# create an SVM to fit the 2D eigenspace
clf = svm.SVC(kernel="linear", C=1.0)
clf.fit(pca, iris.target )

# Plot the PCA features and support vectors
fig, ax = plt.subplots()
disp = DecisionBoundaryDisplay.from_estimator( clf, pca,
       response_method="predict", cmap=plt.cm.coolwarm, alpha=0.8, ax=ax)
ax.scatter(pca[:, 0], pca[:, 1], c=iris.target,
           cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_title('Support Vector Machine for PCA features')
ax.set_xlabel('Eigenvector #1')
ax.set_ylabel('Eigenvector #2')
plt.show()
```

The resulting plot below shows the eigenvector features and the support vector boundaries. Note that the boundaries between iris classes is more sharply separated using Eigenspace features as compared to usng just the sepal length and width as shown in Figure 25. A similar approach could be taken by performing an LDA transformation to better separate the classes prior to computing the SVM.

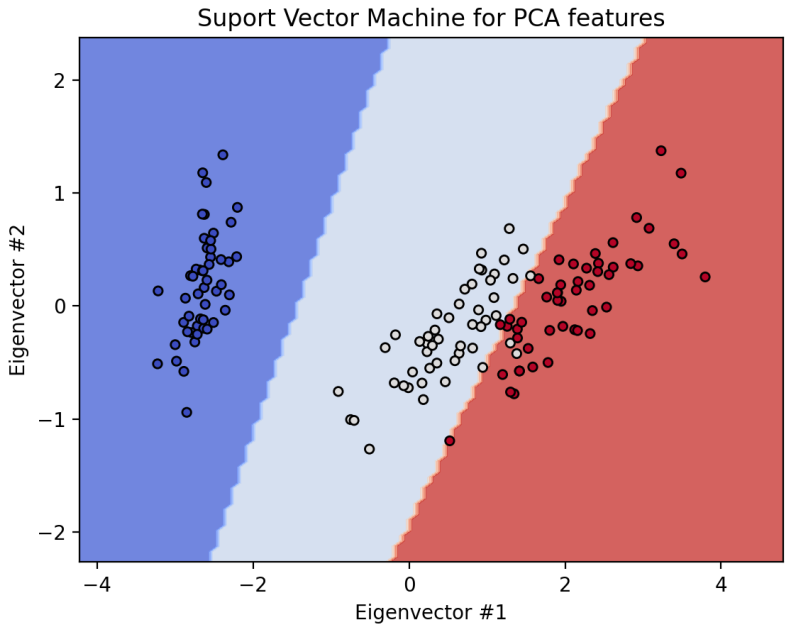


Figure 26: Support Vector Machine boundaries of features in Eigenspace

8.3.4 SVM Image Classification

A support vector machine can also be used to perform image classification. The following example is inspired by the SciKit example on recognizing hand-written digits and uses the hand-written digits dataset from the UC Irvine machine learning repository. This dataset has 1797 image samples comprised of an $8x8$ array of pixels with 10 classes where each class refers to one digit (0-9).

A short Python program to load the hand-written digits dataset and display them is shown below:

```
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

# load hand-written digits dataset
digits = datasets.load_digits()

# display a sample of ten hand-written digits in the dataset
fig, axes = plt.subplots(nrows=2, ncols=5)
for ax, digit in zip(axes.flatten(), digits.images):
    ax.set_axis_off()
    ax.imshow(digit,cmap='gray')
fig.tight_layout()
plt.show()
```

The plot showing ten hand-written digits in the dataset is shown below.

Figure 27: Ten 8x8 hand-written digits from the UC Irvine digits dataset

We can split the hand-written digits dataset into two sets: a training set and a set for testing. SciKit has a function for taking the larger digits dataset and splitting it in half into training and test sets as follows:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

where `X` is an array of of the features and `y` is a vector of labels that classify the data.

Hence we can proceed to train an SVM using half the dataset for training and then use the remaining half of the dataset for testing. The accuracy of the SVM can be determined by comparing the digits

predicted by the SVM with the actual labels for the hand-written digits and the *recognition rate* can be reported. The *recognition rate* is the total number of correctly identified digit images divided by the total number of test images. The following code defines an SVM classifier based on half of the dataset and then determines the recognition rate using the other half of the dataset as test images.

```
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

# Read digits and reshape digits as a vector of images
digits = datasets.load_digits()
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a support vector machine classifier
svm = svm.SVC()

# Split the dataset: half for training and half for testing
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)

# Train on hand-written digits in the training set
svm.fit(X_train, y_train)

# Predict the value of the digit using the testing set
predicted = svm.predict(X_test)

# compute the recognition rate
matches = 0
for x in range(len(y_test)):
    if y_test[x] == predicted[x]:
        matches += 1
recognition_rate = (matches/len(y_test)) * 100;
print(f'Recognition rate: {recognition_rate:.2f}%')
```

The output of this code shows the following:

```
Recognition rate: 96.11%
```

This indicates a respectable recognition rate using an SVM for handwritten digit classification.

For more details, we can plot a *confusion matrix* to visualize the accuracy of a machine learning algorithm. The *confusion matrix* is organized into rows and columns: each row represents each of the classes in a dataset and each column represents the predictions that were made. *Ideally*, the actual classes and the predictions will perfectly align such that the diagonal of the *confusion matrix* is populated with 100's and with 0's everywhere else. The martix diagonal corresponds to true recognition rates whereas all other cells represent the occurences of false classifications. SciKit includes a nifty library that can create a beautifully formatted *confusion matrix* as follows:

```
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split

# Read digits and reshape digits as a vector of images
digits = datasets.load_digits()
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a support vector machine classifier
svm = svm.SVC()

# Split the dataset: half for training and half for testing
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)

# Train on hand-written digits in the training set
svm.fit(X_train, y_train)

# Predict the value of the digit using the testing set
predicted = svm.predict(X_test)

# display the resulting Confusion Matrix
disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, predicted)
disp.figure_.suptitle("Confusion Matrix")
plt.show()
```

The resulting confusion matrix is shown below.

More information about SciKit Learn with many nifty examples can be found on the SciKit Learn webpages.

8.4 LiteRT

LiteRT provides an open source library that can be used to develop and train machine learning models. LiteRT was developed by the folks at Google and released under an open source license. It includes libraries that can be used with Python, C++, or Java. While the standard TensorFlow package has large footprint, a "lite" version is also available that is suitable for running on the Raspberry Pi. To install a "lite" version of TensorFlow for Python on the Raspberry Pi, type:

```
python3 -m pip install ai-edge-litert
```

We can now use LiteRT to perform tasks like classifying objects in images. While it is possible to collect your own image set and train your own models, there are a variety of pre-trained models available. To run a demo using a pre-trained model, follow the instructions on the Python image classification demo page.

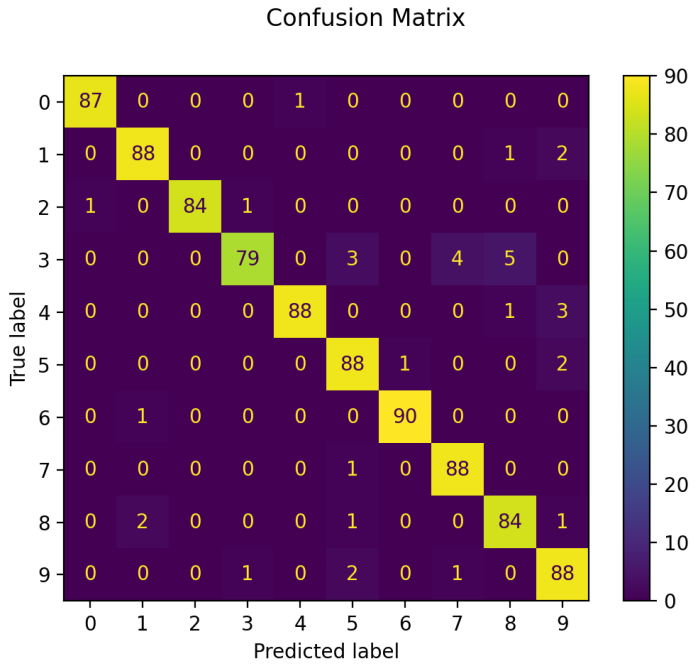


Figure 28: Confusion Matrix for SVM classification of hand-written digits dataset

For more information, consult the LiteRT documentation.

8.5 Large Language Models (LLMs)

A large language model (LLM) is an AI model designed for tasks such as language generation. These models are built by learning statistics on very large data sets of text resulting in models with a vast number of parameters. LLMs require substantial computing resources, but a modest model can be run on the Raspberry Pi 5 using an open source tool called ollama.

To begin, download the ollama model as follows:

```
curl -fsSL https://ollama.com/install.sh | sh
```

The download will take several minutes, but you should receive a message when it completes. Sadly, at the time of writing the model could not take advantage of the GPU in the Raspberry Pi and so it was set to run in "CPU-only mode" which results in slower performance. You can can try running with a small model called tinydolphin which has *only* 1.1 billion paramters. To install and run the `tinydolphin` model, type:

```
ollama run tinydolphin
```

After a few minutes of downloads, a prompt should appear. Try testing the LLM by typing a prompt, such as the following:

```
>>> Give 3 reasons why the Rasperry Pi is cool.
```

While this model is modest and running with "CPU-only" is slow, this platform should allow for some fun experimentation with LLMs. To explore other models, consult the ollama library. Note than many of the models will be too large to run on current versions of the Raspberry Pi.

9 Other Tools for Engineers and Computer Scientists

The Raspberry Pi provides a variety of tools for computer scientists, ranging from document preparation to software version control to a variety of mathematical and simulation tools.

9.1 Document Preparation

9.1.1 LaTeX

LaTeX is a document preparation system for high-quality typesetting and for creating "beautiful documents." It is often used for making mathematical, technical or scientific documents, but it can be used for almost any type of document. There are numerous resources available on the web describing how to edit a LaTeX document file. To install LaTeX and its supporting files and utilities, type the following:

```
sudo apt-get install texlive
```

You can edit your LaTeX document using any plan text editor and saving the file with a `.tex` file extension. Once you have prepared your LaTeX file, you can create a DVI (Device Independent file format) file. To do this, run the following from the command line:

```
latex file.tex
```

If your document contains errors, LaTeX prints a message, but if LaTeX was successful it should generate a DVI (Device Independent) file along with some other files. The DVI file can then be converted to a PDF file as follows:

```
dvipdf file.dvi
```

The PDF file is a platform independent file format that can then be shared widely.

9.1.2 pandoc

Pandoc is the "swiss army knife" of document conversion—a utility that can convert documents between a multitude of different formats. As such, `pandoc` is a convenient utility for the creation of PDF or HTML files from markdown (in fact, this book is written in markdown using pandoc!). To convert a markdown file to a PDF, type:

```
pandoc -o output.pdf input.md
```

Likewise, to convert from markdown to HTML, type:

```
pandoc -s -o output.html input.md
```

where the `-s` parameter indicates that the output should produce a standalone document (ie. a complete HTML document with <head> and <body> sections).

For more details, consult the pandoc documentation.

9.1.3 PDF Utilities

The PDF Toolkit (pdftk) is a utility for PDF manipulations in Linux. To install pdftk, type:

```
sudo apt install pdftk
```

Once installed, `pdftk` can be used to perform many different PDF operations from the command line. For example, to join two files, `in1.pdf` and `in2.pdf` into a new PDF named `output.pdf`, type:

```
pdftk in1.pdf in2.pdf cat output.pdf
```

To remove page 13 from `in.pdf` and create a new file called `out.pdf`:

```
pdftk in.pdf cat 1-12 14-end output out.pdf
```

To find out more features in `pdftk`, type:

```
pdftk --help
```

9.2 File Utilities

In addition to text editors, Linux includes a wide variety of nifty command line tools for working with files, including source code files. Here are descriptions of a few of those utilities.

9.2.1 `diff`

A utility for comparing text files (including source code files) line-by-line is the `diff` utility. To compare two files, type:

```
diff -color file1.py file2.py
```

This will show all the differences (the "diff") between the two files, `file1.py` and `file2.py`.

9.2.2 grep

Grep is a classic command to search files for the occurrence of a string of characters that matches a specified pattern. For example, to search a file named `file.txt` for the word "raspberry", type:

```
grep raspberry file.txt
```

This will return all line numbers that include the string "raspberry". `grep` can also be used with *regular expressions* to perform more complex search. For example:

```
grep "^[A-Z]" file.txt
```

will search for all lines that begin with a capital letter. To learn more, consult online documents about regular expressions.

9.2.3 hexdump

Hexdump is a file viewing utility that displays a file's contents in hexadecimal, decimal, octal, and ascii format. This can be particularly helpful when viewing binary files. To use hexdump, type:

```
hexdump file.dat
```

where `file.dat` is a binary file that you want to view. For example, to view the binary file with the `ls` program, type:

```
hexdump /usr/bin/ls
```

In order to display the output one screen at a time, the output from `hexdump` can be piped through the `more` utility as follows:

```
hexdump /usr/bin/ls | more
```

9.2.4 readelf

Readelf is a program to display information about executable files in Linux in ELF format (Executable and Linkable File Format). For example, to view information about the `ls` command, type:

```
readelf -hA /usr/bin/ls
```

9.3 Software Version Control Systems

Source code management systems are used to manage the revision history of a software project. A source code management system has a databases that stores the changes that are made to files in a project. It can be used as a tool to allow different programmers to collaborate on a software project by allowing them to blend and merge the various changes that are made.

9.3.1 Using Git and GitHub

Git is a free and open source distributed version control system. Git enables you track changes to your code and push and pull changes to remote repositories. Git is one of the most popular systems in use today. To install git, type:

```
sudo apt install git
```

After installing git, you will need to setup your local `git` *username* and *email* as follows:

```
git config --global user.name "user_name"
git config --global user.email "email_id"
```

It is possible to setup your own private *git* server (on a Raspberry Pi or other computer). This can be accomplished by setting up an SSH server and creating a `git` user and uploading public SSH keys to allow remote users to access the server. With proper SSH keys installed, remote users can create and push changes to a repository located on the server.

Another option is to use a publicly available cloud-based *git* server, such as GitHub, GitLab, or BitBucket. Many services include a basic free tier with limited storage or free options for students and educators.

9.3.1.1 Example git project

To initialize a new `git` project, type:

```
git init new-project
```

where **`new`**`-project` is the name of the project you wish to create. A new folder will with the project name will be created. To enter the new project folder, type:

```
cd new-project
```

At this point, you can create and edit your project files. One project file that is recommended to be created is one called `README.md`, a markdown file that contains information about the project. If we create a new file called `README.md`, we can add it to the repository as follows:

```
git add README.md
```

Likewise, we can add other files that belong to the project using the `git add` command. Once all the files are added, we can *commit* it to our repository. Use the command:

```
git commit -m "some_message"
```

The project is now ready to be uploaded to a remote repository in the cloud. For example, if you are using GitHub, go to GitHub.com and create a new repository. Get the repository **URL** and return to your local terminal and type:

```
git remote add origin REMOTE-URL
```

where `user_name` is your user name on your remote server and `REMOTE-URL` is the project URL on the server. To verify that you have set the remote **URL** correctly, type:

```
git remote -v
```

Once the remote **URL** is properly set, you can "push" all the local changes to the remote repository by typing:

```
git push -u origin main
```

Each time you reach a point where you want to check in the changes to your project, you can now `commit` and `push` the project and it will be saved in the remote **git** repository.

Alternately, you can *clone* an existing `git` project from the cloud using:

```
git clone REMOTE-URL
```

where `REMOTE-URL` is the project URL on the remote server.

9.3.1.2 Git Integration with VScode VScode extensions for GitHub, GitLab, and various other popular git servers exist. Install the appropriate extension, and you should be able to push, pull, and commit changes to the remote git server from within the VScode editor.

9.3.1.3 Download an existing git project The "clone" command downloads an existing `git` repository to your local computer. To clone an existing `git` project, you will need the project **URL**. Using the **URL**, you can clone a remote project as follows:

```
git clone https://github.com/gittower/git-crash-course.git
```

For a public repo:

```
$ cd folder/to/clone-into/
$ git clone https://github.com/gittower/git-crash-course.git
```

9.3.2 Mercurial Version Control

Mercurial is friendly and relatively easy to use for those who are new to software repositories. To install with Mercurial on the Raspberry Pi, type:

```
sudo apt install mercurial
```

Once installed, you can test your installation of Mercurial by typing:

```
hg
```

If this returns a list of `hg` commands that are available, then the program is installed. To get a complete list of commands, type:

```
hg help
```

Note that mercurial is invoked using the `hg` command, which also happens to be the chemical symbol for mercury on the periodic chart.

Step 2: Setup a mercurial configuration file

Before you start using mercurial, you should setup a local configuration file to tell it who you are. In local home folder, create a file called `.hgrc` using your favorite text editor with the following contents:

```
[ui]
username=Firstname Lastname username@youraddress.com
merge=internal:merge
```

In the `username` line, substitute your own name and email address. You are now ready to make your own software repository!

9.3.2.1 Example of creating a Mercurial software repository

A repository is a database that contains the files in a project. But we must first indicate the files that we want in the repository and those that we are not interested in saving. Typically, we are interested in saving all the source code files. However, we are not usually interested in saving all the intermediate object files and binary files since these can always be recreated from the source files.

Within your project folder, create a new text file called `.hgignore` which will contain information about the files that should be *excluded* from the repository. A sample `.hgignore` file is show below:

```
syntax: glob
\*.class
\*.o
```

This file indicates that `*.class` files (for Java) and `*.o` object files (for C/C++) should be excluded from the repository. There is no point storing files that can be easily recreated from the source files.

To initialize the repository, issue the following command from within the project folder:

```
hg init
```

This starts a new repository for a project and creates a new `.hg` folder in the project folder. You can now start to add and edit files within the project folder. As you do, you can check the status of this files with respect to the repository as follows:

```
hg status
```

This shows the current status of the repository. Initially, all the files in the project folder will be listed with a ? in front of them indicating that they are *unmanaged*. To add them to your mercurial repository, type the following:

```
hg add
```

This adds all the files in the current folder except the ones indicated by the `.hgignore` file. Running the status command again will show that files are now part of the repository by placing an `A` in front of each of the filenames. To remove a file from the repository, use the following command:

```
hg remove filename
```

where `filename` is the name of the file you want to remove from the repository

Once you are done editing your changes, you can decide to *commit* the current changes to the repository. This can be accomplished by typing:

```
hg commit -m "message"
```

where you can replace the `message` string with a meaningful summary of the changes that were made. Now run the status command again:

```
hg status
```

This time, no file status are displayed because all the files have been committed to the repository. We could now delete a file as follows:

```
rm filename
```

If this is a file that was part of the repository, it can now be easily restored as follows:

```
hg revert filename
```

You can also edit `filename` and make changes. To see the differences between a file and its copy in the repository, issue the following command:

```
hg diff
```

Subsequent updates can be committed to the repository as time goes on. You can get a complete history of past commits that were performed by typing:

```
hg log
```

This command prints a brief summary of each change that was committed including the date, time and name of the user who performed the commit along with any message that was saved with the commit.

9.3.2.2 Example of using a remote repository If you setup a mercurial repository on a remote server, you can copy and update your repository over the network using SSH. For example, to get a copy of a project from the remote server copied over to your local machine, type:

```
hg clone ssh://username@server.domain/project
```

where `username` is your user name on the server, and the `project` is the folder name of the project you want to copy. You will be prompted for your password on the server. The step of entering a password can be eliminated if you setup SSH keys.

To bring in changes from a remote repository to a local repository, use the pull command as follows:

```
hg pull ssh://username@server.domain/path/project_folder
```

Note that this updates the repository database, but not does not (by default) touch the files in the working project directory. Instead, use the `update` command to do this as follows:

```
hg update
```

To *push* your changes to a remote repository, use the `push` command:

```
hg push ssh://username@server.domain/path/project_folder
```

Note that the `push` command updates the remote repository database, but does not update the actual working files in the project directory since other people may be working on them while the update is in progress. To update the working files in the remote project directory, issue an update on the remote server after completing the push operation.

This simple introduction should help you get started with small personal and team projects. Once you are more comfortable with the basics, check out the Mercurial webpage to learn more.

9.4 Mathematical Tools

9.4.1 SageMath

SageMath is a free open-source mathematics software system with many mathematical capabilities including algebra, graph theory, numerical analysis, number theory, calculus, and statistics. Recent `apt` repositories include a package for SageMath, but due to the computational demands of `sage` it is recommended you use a more recent model of the Raspberry Pi with as much memory as possible.

To install `sage` from the apt repositories, type the following:

```
sudo apt install sagemath
```

The install process can take a long time, so feel free to walk away from your Raspberry Pi while it is installing. Once the install is complete, you can launch the program as follows:

```
sage
```

Once sage is loaded, give it a try. For example, to add two numbers type the following at the `sage` prompt:

```
sage: 2 + 3
```

To plot a graph of $y = x^2$ in sage, type:

```
sage: plot(x^2)
```

The following shows a prompt to determine the symbolic integration of $\int \sin(x)dx$ along with the output from sage:

```
sage: integral(sin(x),x)
-cos(x)
```

For more information about using other powerful features found in SageMath, visit the Sage Documentation pages.

9.4.2 Octave

GNU Octave is an open source mathematics program that has some similarities to Matlab. It has both a GUI mode as well as a command line interface. To install Octave, type:

```
sudo apt install octave
```

To start octave in command line mode, type:

```
octave-cli
```

Octave is particularly good at performing linear algebra operations. For example, to define a matrix like the following:

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 5 & 6 & 7 \end{bmatrix}$$

simply type:

```
x = [1 2 3; 2 1 4; 5 6 7]
```

To determine $\mathbf{x}^T$, the transpose of $\mathbf{x}$, type:

```
x'
ans =
   1   2   5
   2   1   6
   3   4   7
```

To find the eigenvalues for the matrix, simply type:

```
eig(x)
ans =
   11.5453
   -0.7892
   -1.7561
```

To find the determinant for the matrix, type:

```
det(x)
ans = 16
```

To determine the inverse matrix, type:

```
inv(x)
ans =
  -1.0625   0.2500   0.3125
   0.3750  -0.5000   0.1250
   0.4375   0.2500  -0.1875
```

Matrix multiplication can easily perform using the $\star$ operator as follows. For example, multiplying a matrix by its inverse should produce the identity matrix:

```
x * inv(x)
```

```
ans =
   1.0000  -0.0000        0
        0   1.0000   0.0000
  -0.0000   0.0000   1.0000
```

Octave can also generate large row vectors of values. For instance, the command:

```
x = linspace(a ,b)
```

generates a row vector x of 100 points linearly spaced between and including a and b. For example, to create a vector with 100 time steps for $0 <= t <= 6\pi$, type:

```
t = linspace(0, 6*pi)
```

In graphical environments, `octave` can also be used to produce plots of vectors. To plot a sinusoid for the range of time steps `t`, type:

```
plot(t, sin(t))
```

A new plot window will appear like the one shown below.

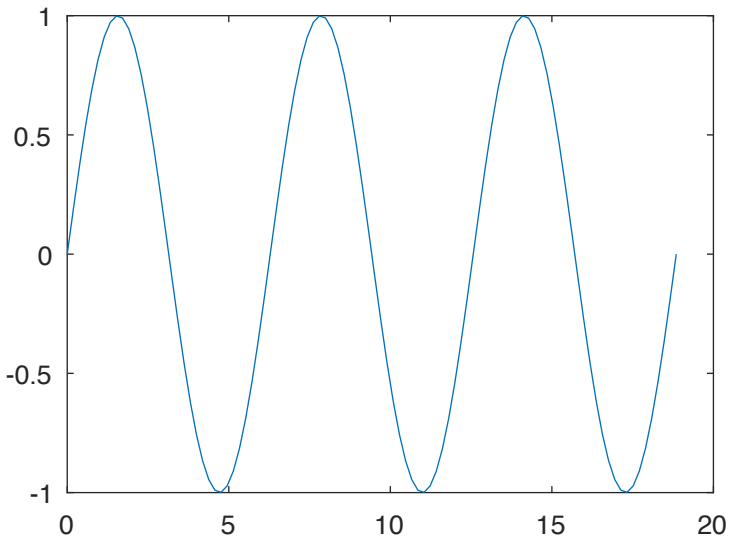


Figure 29: An octave plot of $sin(t)$ for $0 <= t <= 6\pi$

For more information about `octave`, visit the Octave Wiki.

9.4.3 gnuplot

Gnuplot is a free portable command-line driven graphing utility that works on many platforms. To install `gnuplot`, type:

```
sudo apt install gnuplot
```

To start `gnuplot`, simply type: `gnuplot`. You can then command `gnuplot` to plot various mathematical functions. While plots can be saved to image files, they can also be displayed in a graphical window (which presupposes a graphical desktop environmet). For example, to display a plot of a sine and cosine waveform, type:

```
plot [-10:10] sin(x),cos(x)
```

A $sin(x)$ and $cos(x)$ plot will appear for $-10 <= x <= 10$. For more information and demos of the features of `gnuplot`, visit the official gnuplot documentation page.

9.5 Circuit Simulation with NGSpice

NGSpice is a general-purpose, open-source circuit simulation program which is descended from SPICE, a program that began development in 1973 at the University of California at Berkeley. The name SPICE is short for "Simulation Program with Integrated Circuit Emphasis". SPICE is an industry-standard tool capable of linear AC analysis and non-linear transient analysis of electronic circuit by solving device model equations for each of the circuit elements based on Kirkhoff's current and voltage laws at each node.

SPICE was designed for integrated circuit simulation, but has also been used to simulate board-level circuit design. It has also been used for the simulation of power electronics circuits.

To install NGSpice on the Raspberry Pi, type:

```
sudo apt install ngspice
```

9.5.1 Defining a circuit file for simulation

To begin using NGSpice, you must first define a *circuit file*, a text file that describes a *netlist* of the electronic components and the nodes to which they are connected. Traditionally, a circuit file uses the file suffix `.cir` and contains various "dot commands" to specify the type of simulation to be performed.

In general, simple circuit elements like resistors, capacitors, and inductors are defined as using the following syntax:

Element	Definition
Resistor	R[name] [node 1] [node 2] [resistance value (Ohms)]
Capacitor	C[name] [node 1] [node 2] [capacitance value (Farads)]
Inductor	L[name] [node 1] [node 2] [inductance value (Henries)]

Note each element must have a unique name and **[node 1]** and **[node 2]** represent the node names or numbers that the element is connected to. For example, a series RC circuit, with corresponding element values 1kohm and 10μF, could be represented as follows:

```
R1 1 2 1.0k
C1 2 0 10uF
```

The special node numbered "0" indicates a ground connection. NGSpice includes many other circuit elements as well, including semiconductor models for JFET, MOS, and bipolar transistors and diodes.

Circuit files can be entered using your favorite text editor. For example, consider the an RC circuit with nodes labelled 1 and 2 fed by a sinusoidal voltage source in the diagram shown below:

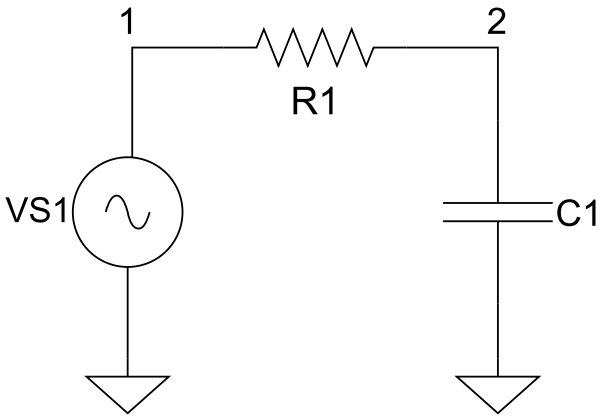


Figure 30: An RC circuit fed by a sinusoidal voltage source

Below is the circuit file corresponding to the schematic diagram shown above. This circuit file defines values to the circuit elements and initiates a transient analysis capturing the voltage at nodes 1 and 2.

```
TRANSIENT RESPONSE IN AN AC CIRCUIT WITH RC ELEMENTS
* Sinusiodal Source
VS1 1 0 SIN(0 10V 60Hz 0)
```

```
* Circuit elements
R1 1 2 1.0k
C1 2 0 10uF

.control
TRAN 100us 60ms
PRINT V(1) V(2)  >  output.txt
.endc
.END
```

Note the format of of the circuit file includes a title on the first line and comments are included by putting an asterisk at the beginning of the line. This circuit file defines a sinusoidal voltage source (VS1) connected to two series connected circuit elements (R1 and L1). This circuit file also includes a few "dot" commands–special commands that begin with a dot and are used to adjust settings and define the type of analysis to be performed. In particular, the `.control` and `.endc` lines surround the analysis and output to be captured in the simulation. In this case, a transient analysis is performed over a time scale of $60ms$ with a suggested time step of $100\mu s$. The voltages of nodes 1 and 2 at each time step will be sent to a file named `output.txt`. All circuit files must concludes with a special dot command, `.END`.

9.5.2 Example Circuit Simulation

To simulate this circuit, invoke the simulator as follows:

```
ngspice -b example.cir
```

where `example.cir` is the name of the circuit file to be simulated and `-b` indicates that the simulation is to be run in *batch mode* (rather than in an interactive mode). A summary of the simulation will be printed and, as indicated above, the voltages at nodes 1 and 2 at various timesteps will be printed to `output.txt`. This output file is arranged into rows of timestamps and voltages that can be imported into a spreadsheet or a plotting package to plot the transient response.

If you are running in a graphical environment, the simulation output can also be directly plotted using `gnuplot` (described in a previous section). To add a control command to plot the transient output, add the following command in place of the `PRINT` command in the circuit file above:

```
gnuplot plot.data V(1) V(2)
```

Run the simulation again as follows:

```
ngspice -b example.cir
```

A graphical window should appear like the one below showing a plot of the voltages at nodes 1 and 2.

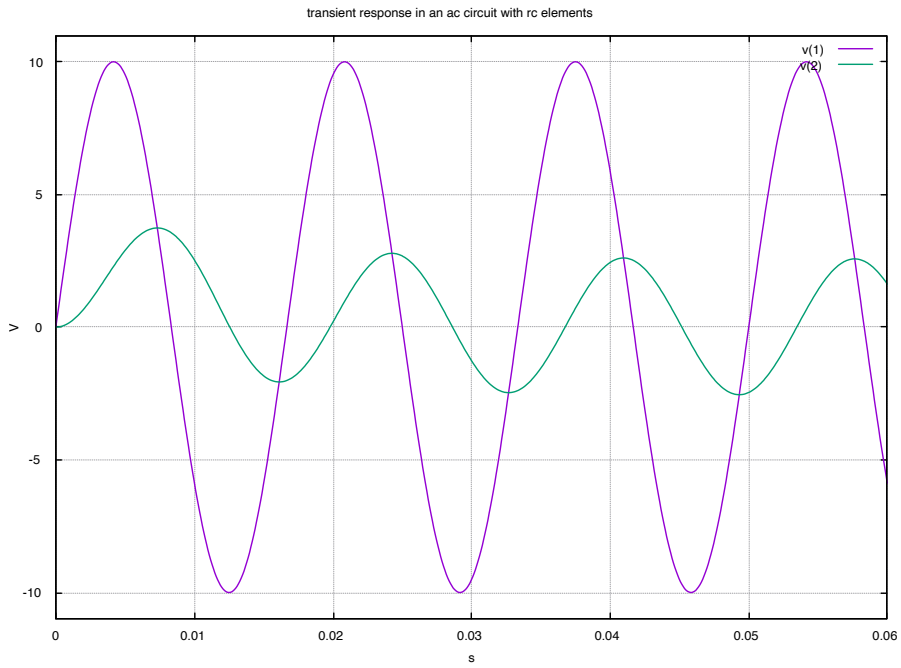


Figure 31: NGSpice voltage transient plot

For more information and to view the official `ngspice` manual, consult the ngspice home page.

9.5.3 Power Electronics Circuit Simulation

Although SPICE simulation was originally designed for integrated circuits, it can also be used for power electronics simiulations. What follows is a circuit simulation of a *phase angle control* circuit that can be used to vary the AC volage across a load.

A circuit is shown in Figure 32 wherein the power electronic device is represented by a simple voltage-controlled switch. Normally, such a power switch would be implemented using a power device such as a thyristor or IGBT. `VS` represents a sinusoidal voltage source, and the switch is fired at a given phase angle α, relative to the source voltage. The phase angle adjusts the average AC voltage seen across the load resistor `RLOAD`.

The corresponding circuit file is given below. This file defines a variety of circuit parameters which represent frequency, period, voltage, load resistance and other parameters. One key parameter is `ALPHA`, which represents the angle α in radians, the phase angle at which the switch is fired. For this simulation, we will set $\alpha = 1 radian/sec$.

```
AC VOLTAGE REGULATOR
```

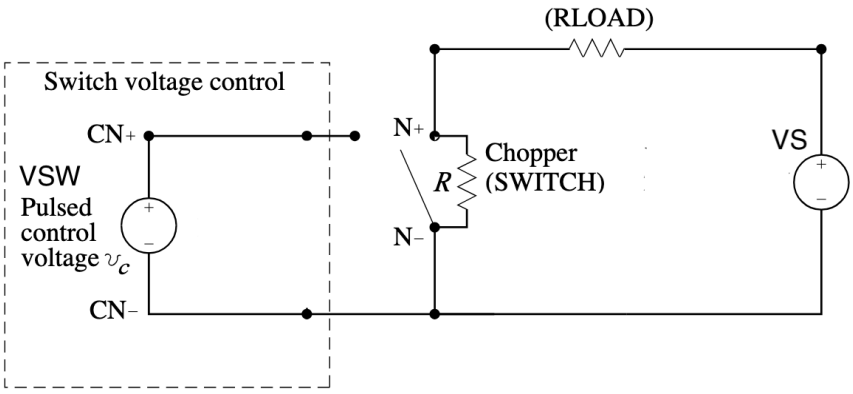


Figure 32: AC chopper circuit

```
.model switch1 sw vt=0.5 vh=0.2 ron=.001 roff=1MEG
.PARAM FREQ=50, RLOAD=10, PI=3.14159, TWOPI={2*PI}, ALPHA=1
.PARAM DELAY={ALPHA/(TWOPI*FREQ)}, VRMS=2V, VMAX={SQRT(2)*VRMS}
.PARAM PERIOD={1/FREQ}

VS 1 0 SIN(0 {VMAX} {FREQ})
VSW 100 0 PULSE(0 1 {DELAY} 1ns 1ns {PERIOD/2-DELAY} {PERIOD/2})

RLOAD 1 2 {RLOAD}
SW1 2 0 100 0 switch1

RLOAD 1 2 {RLOAD}
SW1 2 0 100 0 switch1

.control
TRAN 5us 40ms
gnuplot plot.data V(1)-V(2) xlabel 'time' ylabel 'load voltage'
.endc
.END
```

This simulation can be run using the command:

```
ngspice -b ac-voltage-regulator.cir
```

The resulting output plots the voltage across the load resistor as shown below:

Note that the average voltage across the load resistor is decreased depending on α, the angle of the firing control. Using the circuit parameter `ALPHA`, we can adjust the average AC voltage output across the load. The output waveform is periodic but is distorted from an ideal sinusoid. NGspice can help us determine how much harmonic distortion is present across the load using the following lines in the control block:

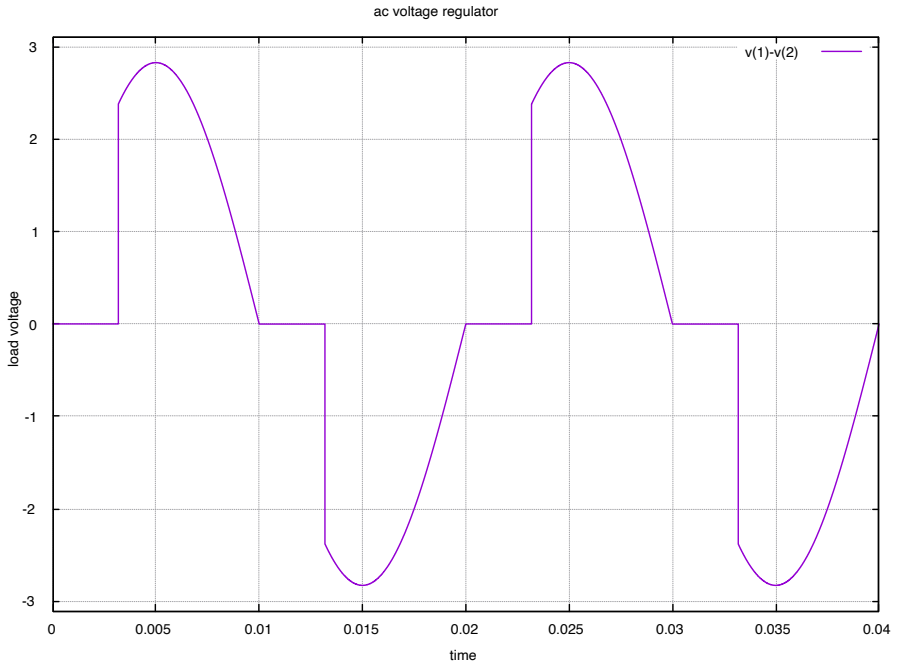


Figure 33: Phase angle controller transient plot

```
.control
TRAN 5us 400ms
fourier 50 V(1)-V(2)
gnuplot plot.data fourier11[1] vs fourier11[0] xlabel 'frequency' ylabel '
    Load Voltage Magnitude'
.endc
```

The output from running the simulation reports the total harmonic distortion (THD) as follows:

```
Fourier analysis for v(1)-v(2):
  No. Harmonics: 10, THD: 32.1307 %, Gridsize: 200, Interpolation Degree:
      1
```

The resulting frequency plot of harmonic magnitudes is plotted in Figure 34.

Note this circuit is based on an example the author has co-athored in another book on the topic titled *PSpice Simulation of Power Electronics Circuits*. Further examples of power electronic circuit simulations using SPICE can be found on the book's companion website.

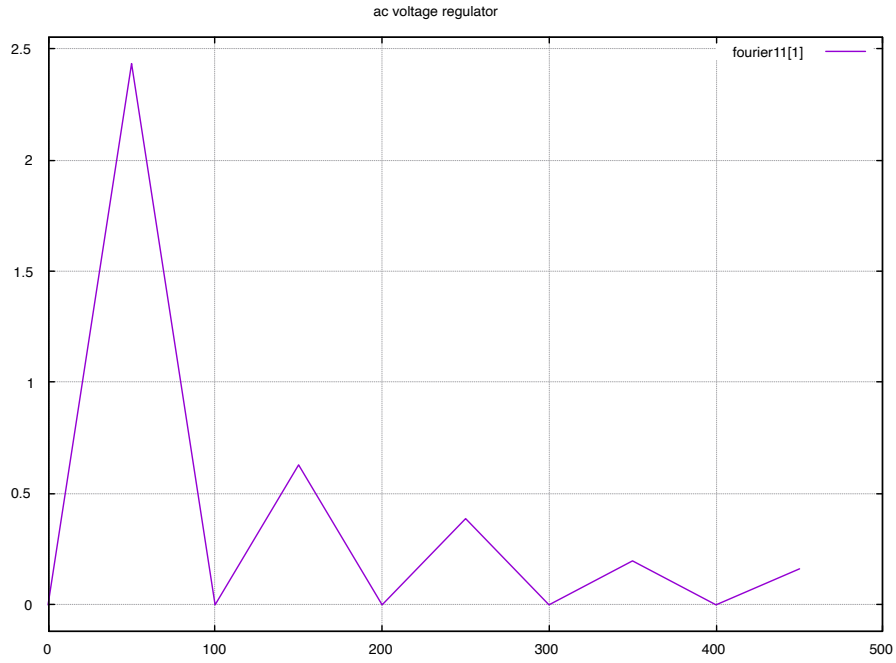


Figure 34: Frequency Harmonics in Load

9.6 Ham Radio Applications for the Raspberry Pi

The Raspberry Pi repositories include a variety of Ham Radio programs, some of which are described in the following sections. The author uses a Raspberry Pi for his ham radio station computer and has found it more than capable of running a variety of key radio programs, including the software listed below.

9.6.1 WSJT

For licensed amateur radio operators, WSJT can be used to exchange packets on various amateur bands using digital modes such as FT4 and FT8. A version of WSJT for the Raspberry Pi can be downloaded from the WSJT home page.

9.6.2 fldigi and flrig

The fldigi program is a "software modem" for amateur radio use. It can be used to communicate using digital modes like PSK31, QPSK, MFSK, RTTY, and Olivia. The flrig program can co-operate with *fldigi* to provide software control of your ham radio transciever (your "rig"). A screenshot of *fldigi* operating with a PSK31 transmission from W1AW is shown in Figure 36.

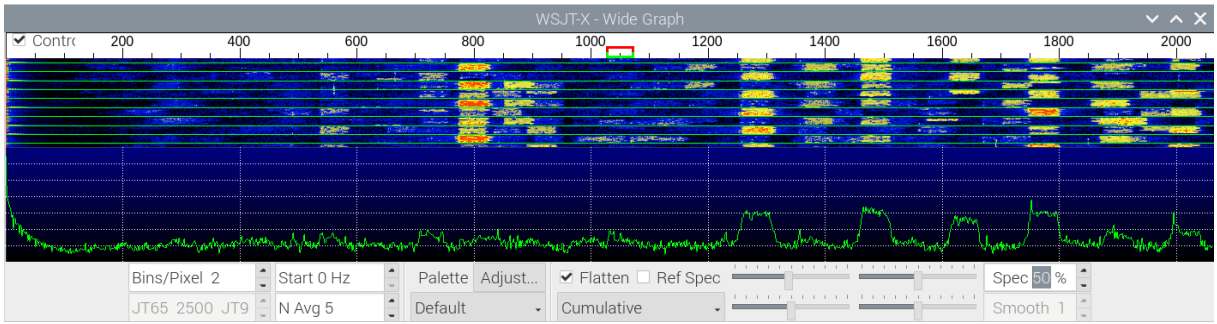


Figure 35: FT8 Spectrum waterfall window on WSJT program

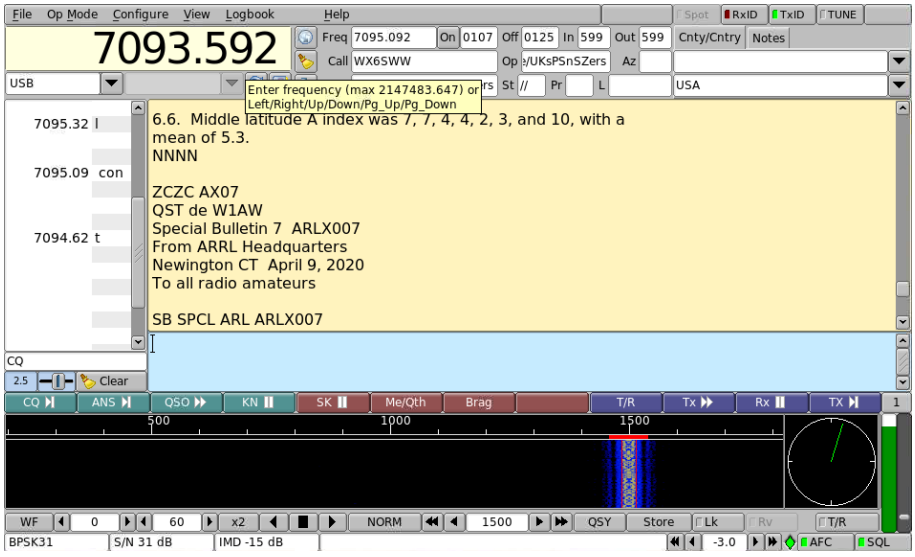


Figure 36: fldigi program window showing PSK31 transmission from station W1AW

9.6.3 TQSL

Trusted QSL (TQSL) is freely available software used in conjunction with the ARRL's Logbook of the World (LoTW). LoTW is a web-based database for submitting electronic logs of radio contacts ("QSOs") and for confirmations ("QSLs"). The Trusted QSL software allows a ham radio operator to digitally sign his contacts and submit them to using the ADIF (Amateur Data Interchange Format) to the LoTW. Digital signing requires a digital call sign certificate obtained from the ARRL.

10 Ethics and Computer Technology

Technology amplifies opportunities to do good as well as to do harm. While various examples displayed in the preceding chapters appear neutral, a variety of issues emerge when digital technology is applied to real-world applications. Engineers and computer scientist need to cultivate a posture of responsibility alongside their technical competencies. For this reason, engineers and compuer scientists must strive *to take more into account* than just technical considerations.

10.1 Design norms

A stream of philosophical thought emerged in the mid-twentieth century developed by Herman Dooyeweerd and Dirk Vollenhoven at the Free University in Amsterdam. One key insight of this philosophy was the concept of fifteen "modal aspects" of reality that include various *laws* and *norms*. A set of "design norms" can be derived from the normative "modal aspects" to provide a framework for designers to take more factors into account. Technology is not neutral, but rather has substantial cultural, social, lingual, economic, aesthetic, and justice issues. A collection of seven *design norms* are listed below along with descriptions and example questions associated with each norm.

1. **Cultural Appropriateness**
    - *Description*: Technology products should consider the culture into which they are embedded, cultivating improvement without disrespectful or unnecessary disruption.
    - *Questions*: Does the technology relieve burdens while preserving what is good in a cultural context? Is the design appropriate to its context, including questions of centralization vs. decentralization, large scale vs. small scale, and continuity vs. discontinuity.
2. **Transparency**
    - *Description*: Documentationx and user interface ought to be clearly understandable by the user without being overwhelming. Users should be informed about potential dangers and guided to diagnose failures. For example, software and dashboards that clearly communicate status and errors.
    - *Questions*: Is the documentation clear and unambiguous? Is the layout, color scheme, and icons of the interface helpful? Does the product perform as advertised or does it bear false witness or exaggerate its claims? Are potential dangers clearly indicated to users?
3. **Stewardship**
    - *Description*: Use of creational resources should be respectful, frugal, and caring, Design should reflect concern for sustainability and the environment. This norm also includes economic considerations.

- *Questions*: Does the design consider the entire life cycle of the product? Is it repairable and recyclable? Is it efficient, using energy and other resources wisely? Are there waste products result from its use? Is respect paid to all creatures?

4. **Aesthetics**
    - *Description*: This norm deals with delightful harmony: the form of the technological device should suggest its function and be pleasing and satisfying to use. For example, a hammer's form implies its function (of pounding).
    - *Questions*: Can new users easily intuit the function of this design? Is the user interface clear and pleasing to use? Is it delightful and beautiful?
5. **Justice**
    - *Description*: Technology should correct (not cause) injustice and should encourage justice, i.e., equity and fairness. The design should help give each person their due and facilitate the opportunity for all creatures to be the creature that God intends them to be.
    - *Questions*: Does this device promote fairness? Are copyrights and intellectual property respected? Could this design be easily used for unjust purposes? Does it respect intellectual property and privacy?
6. **Caring**
    - *Description*: Our tools should help us serve one another, promote wellness, contribute to healing, show love to our neighbor, and enable fellow creatures to flourish. Design should show loving concern for the welfare of all involved.
    - *Questions*: In what ways does this design show care for others? How does it show love for neighbor? Who might be harmed if this device is used?
7. **Trust**
    - *Description*: Technological devices ought to be reliable, especially in situations where safety is a crucial factor. Design should be a response to God and promote faith in him rather than faith in technology of any created thing.
    - *Questions*: Can the user depend on the design for its intended purpose. Is the design safe and secure? What habits and practices are associated with the device and how might that shape the user?

This paper presents Herman Dooyeweerd's philosophy as one such framework and provides a formulation of well-being informed by his Theory of Aspects and his "simultaneous realization of norms" principle.

Rather than focusing on one or just a subset of norms, designers should strive for what Dooyeweerd referred to as the *simultaneous realization of norms*. It is crucial that the consideration of design norms

should not be left as an afterthought, but rather it should be part of the design process — right from the beginning.

10.2 A Brief Normative Analysis of the Raspberry Pi

I think the Raspberry Pi project illustrates some aspects of these design norms. For instance, the Raspberry Pi facilitates **cultural appropriateness** by using open source software that enables developers from different regions to customize software for their particular context. The official power supplies also accommodate a wide range of voltages with adapters for different international power sockets. The Raspberry Pi OS can be configured to support a wide range of languages and regions and provides an appropriate platform for teaching computing in the majority world. The cultural norm is also refected in new models maintaining backwards compatibility with a consistent form factor and price point over many generations of the Raspberry Pi. The Raspberry Pi also exhibits **transparency** in terms of using open source software (however, some of the hardware, including the GPU, have not been as well-documented). The Raspberry Pi reflects **stewardship** by being compliant with RoHS (Restriction of Hazardous Substances) directives. Furthermore, the Raspberry Pi contributes to green computing by providing a plafform with very low power requirements. The Raspberry Pi reflects the **aesthetic** norm through beautiful board layouts and in the design of attractive cases and packaging. The Raspberry Pi has aspects that reflect the **justice** norm by providing a low-cost platform that helps reduce the "digital divide." Some electronics manufacturers further address the problematic issue of "conflict minerals" through efforts like the Responsible Minerals Initiative. The **caring** norm is reflected in the Raspberry Pi Foundation by fostering a community with free resources to help people understand computing better. Finally, the **trust** norm is reflected in providing a reliable computer (although some varieties of SD cards can be unreliable).

To learn more about design norms in computer science, see chapter 4 of *Shaping a Digital World* and for engineering, see chapters 4 and 5 of *A Christian Field Guide to Technology for Engineers and Designers*.

> **Discussion Question**
>
> Select a technological artifact that you use regularly in your life and perform a "design norm audit" by thinking through each of the norms with respect to the device. How could the device be made more "normative"? Can you think of ways the Raspberry Pi could further address some of these design norms?

11 A Collection of Lab Exercises

The following are a collection of miscellaneous labs and hands-on activities that can be performed with the Raspberry Pi.

Some Lab Safety Guidelines

Some of the following labs involve wiring from the Raspberry Pi board to a breadbaord. It is important to realize that caution should be exercised when working with electronic components. Improper handling or wiring can damage your circuit board and components.

Improper circuit wiring can also cause injuries. For example, an improperly wired integrated circuit or an electrolytic capacitor or tantalum capacitor with its polarity reversed can explode. In these labs, we will not be using any electrolytic or tantalum capacitors, but caution should always be exercised when wiring circuits. Out of an abundance of caution, when you first power up a circuit, do not position your eyes near the circuit. If you smell smoke, power down the circuit immediately. Some parts, especially resistors and integrated circuits, can get thermally hot. Assume parts to be hot unless you know otherwise. In a defective circuit, parts that are normally cool enough to touch may burn you. You should always remove power from your circuit before handling it.

It is recommended that when selecting electronic circuit boards and components you use parts that are RoHS (Restriction of Hazardous Substances Directive) compliant.

Keep your work area neat and organized. Messy work areas are more conducive to accidents.

Here are some general guidelines to ensure you do not damage your Raspberry Pi or other electronic components:

- Do not remove the power from the pi until after you first perform a software shutdown of the Linux operating system. Removing the power before performing a proper shutdown could corrupt your SD card.
- Do not run your Raspberry Pi circuit board outside the case on a metal surface! The metal can short out your board and destroy it. It's best to run your Raspberry Pi in its case.
- Do not drop metal wire parts or wire clippings onto your Rasbperry Pi. These could short out the circuit board causing damage.
- The GPIO pins on the Raspberry Pi use 3.3V logic. Do not connect them to a circuit powered at higher voltages (like a 5V logic circuit) which can destroy the board.
- Carefully check the GPIO pin numbers before connecting them. There are 40 pins on the header, and it is easy to make mistakes. Double check your wiring before powering up your circuit!

- Do not perform any wiring while the Raspberry Pi is powered on!
- Don't use any other power supply except the one supplied with the kit. Regular phone chargers may not have enough current to meet peak power demands.
- Ensure your SD card has good reliability ratings and consider a class A2 speed rating or better.
- Be aware that ESD (Electrostatic Discharge) can damage electronic components. Ideally, one should typically use an antistatic mat or wrist strap when handling circuit boards and components.

This list of guidelines, possible accidents, and procedures could not possibly be comprehensive. Use your good judgment!

Lab #1: Getting Started with the Raspberry Pi

The goals of this lab include:

- Setup and configuration of your Raspberry Pi
- Configuring wired and wireless networking
- Connecting and editing a file using secure shell (SSH)

Introduction

Once you obtain your Raspberry Pi kit from the lab instructor, carefully unpack the following:

- case
- Raspberry Pi board
- microSD card
- power supply

Set the microSD card to one side and carefully place the Raspberry Pi in its case following the instructions from the "Hardware Setup" in the "Quick Start" guide. Be gentle, insert the side with the HDMI connector first and then press down on the other side. The board should sit snugly in the case.

> Do not insert the SD card until *after* the board is properly mounted inside the case to prevent damage!

Leave the lid off the case since we will be accessing some of header pins in later labs. Once the Raspberry Pi is properly seated in its case, you may insert the microSD card.

The Raspberry Pi 4 has 2 micro HDMI mini ports, so your kit include a micro HDMI adapter HDMI cable. Use the adapter cable to connect the HDMI port nearest the USB C power input to one of the lab monitors (leave the other monitor connected to the workstation). Plug in a keyboard and a mouse and insert the power adapter. Next, apply power, but make sure the monitor is connected *before* applying power so that the Raspberry Pi can detect the presence of the monitor during power up.

If you left the lid off the case, you should observe a red LED illuminated on the Raspberry Pi board indicating power. A flashing green LED indicates activity on the SD card.

The first time the Raspberry Pi is powered on, it will boot into a setup and installation script. Follow any instructions on the screen to setup the Raspberry Pi OS and follow any prompts to setup country, language, time zone, and keyboard. Select a username and password as prompted (and don't forget your new password!). Likewise, if you reach the "Update Software" step, be sure to click "Skip" (this can take a *very* long time and will be done later).

The installation should take only a few minutes. Once the OS is installed, the Raspberry Pi should be restarted and it should boot directly into a graphical desktop environment.

> **Note**: Using default or weak passwords on IoT devices are a source of many security issues and exploits! Don't proceed to enable networking below without first establishing a secure password!

Setup wireless networking on the Raspberry Pi using Eduroam

You should notice a network connection icon in the task bar at the top right corner of your desktop. Click the icon and a list of available Wi-Fi connections should appear. Click on the network and enter the settings as prompted. After a few moments, the Wi-Fi network should become active.

Note that the network manager program has a command line configuration tool that can also be used by typing:

```
nmtui
```

> **Security Note:** WiFi passwords are stored in special configuration files on your computer. While file permissions prevent casual viewing of these files, they can be read by those with root privileges or if your SD card is mounted on another computer.

Perform software updates

Now that your device is connected to a network, it is even more crucial to keep your device updated with the latest security patches. To perform updates from the command line, open a terminal window and type the following commands:

```
sudo apt update
```

```
sudo apt upgrade
sudo apt autoremove
```

These updates may take several minutes. When the updates are complete, we can configure *automatic* updates by installing the `unattended-upgrades` package as follows:

```
sudo apt install unattended-upgrades
```

Next, edit the configuration to enable unattended upgrades as follows:

```
sudo nano /etc/apt/apt.conf.d/50unattended-upgrades
```

This command will open a configuration file to fine-tune the unattended updates. Remove the double slashes (//) in front of the following line in the configuration file so to enable automatic updates:

```
"origin=Debian,codename=\${distro_codename}-updates\";
```

Finally, ensure automatic updates are enabled by running:

```
sudo dpkg-reconfigure unattended-upgrades
```

At this point, automatic updates should be enabled.

Complete setup and configuration

Next configure more settings by typing the following at the command line:

```
sudo raspi-config
```

A text window should appear. Using the `raspi-config` program, perform the following configurations. Note that since this is a text application you will need to use the keyboard to navigate the menus (using the arrow and tab keys).

- select Interface Options→ SSH (enable SSH server)

Next, set your locale and give your Raspberry Pi a unique hostname (something other than the default hostname of *raspberrypi*). To do this, select System Options→Hostname (enter a new hostname of your choosing).

> **Note:** A hostname may contain only letters, digits, and the hyphen.

Most embedded systems run "headless," i.e. with no screen, keyboard, or mouse. In this course we will normally run our Raspberry Pi in a headless state so it makes sense to disable the graphical desktop environment and use a console interface instead. Doing this frees extra memory and CPU resources that a graphical desktop requires. (However, if you ever want to start the graphical desktop environment while using the console, just enter `startx`). Moreover, we want to disable auto-login for the

Raspberry Pi (you want to protect your Raspberry Pi from unauthorized access by those with physical access to the device). To run the Raspberry Pi with a console that requires a login on startup, do this in `raspi-config`:

- select System Options→Boot/Auto Login
- select Console - Text console, requiring user to login

Once all these settings have been made, exit `raspi-config` and select the option to reboot. Alternatively, you can initiate a reboot on the command line by typing:

```
sudo reboot
```

The Raspberry Pi will now reboot into a console login screen. Continue by logging into your Raspberry Pi using the username and password that you set earlier. After logging in successfully, you will be presented with a terminal interface.

To learn more about using terminal commands in Linux, refer to the Exploring Computer Science with the Raspberry Pi.

Remotely Connecting to your Raspberry Pi

Once your Raspberry Pi has rebooted and you have logged in, type the following command to check the status of your network connections:

```
ifconfig
```

The output of this command will show the displays the status of the currently active interfaces: `eth0` represent the Ethernet port, `wlan0` represents the Wi-Fi interface, and `lo` represents the local loopback interface. Looking next to the `wlan0` interface you should see something like the following:

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.1.2.15 netmask 255.255.255.0 broadcast 10.0.0.255
ether b8:27:eb:c2:7d:da txqueuelen 1000
:
```

The decimal dotted number beside the `inet` label is your **IP address**. Make a note of the IP address (in the example above, the IP address is `10.1.2.15`). The number beside the `ether` label is your *hardware address* (or **MAC** address) that is unique to your Raspberry Pi's interface. Note that if you have an Ethernet cable attached, you should see a separate IP address (and MAC address) assigned to it.

> **Note:** The Wi-Fi IP address may change each time you boot based on how the DHCP leases are set.

Log into a Linux desktop workstation using your Calvin username and password. Connect to the Raspberry Pi using a "secure shell" (`ssh`) connection over the network. In a terminal window, type:

```
ssh user@10.1.2.3
```

where `user` is your username and `10.1.2.3` is the Wi-Fi IP address of your Raspberry Pi as determined earlier. Note that running SSH from the command line should also work in OSX.

If you are using a recent version of Windows, please follow the tutorial for using the OpenSSH client. For Windows, you could also use Putty (a common open source SSH client) or the Windows subsystem for Linux (WSL). WSL includes support for a basic shell (including `ssh` and many other commands) and can be launched by typing `WSL` or `bash` in the start menu.

If everything is working, you should be able to log in successfully using your username and password you selected during installation. When your Ethernet cable is connected and has an IP address assigned you may also use that address to connect to the Raspberry Pi.

What's my IP Address?

Using `ssh` to connect to a Raspberry Pi has a few limitations. First, you can normally only access a Raspberry Pi on the same local network (if it is remote, you will need a "pinhole" opened on any firewalls to allow traffic to port 22 which presents possible security issues). Second, even if the Raspberry Pi is on the same local network, you still need to know its Wi-Fi or Ethernet IP address. In some situations, a static IP address can be configured to ensure the IP address remains fixed over time, but IP addresses are often "leased" from a DHCP server and can change over time. One could temporarily connect a monitor and keyboard to the Raspberry Pi to log in and determine the IP address, but that is cumbersome and impractical.

Raspberry Pi Connect is a service that provides a way to connect to your Raspberry Pi from anywhere in the world with no knowledge of its IP address. There is a "Lite" version that only supports remote shell access, and to install the Raspberry Pi Connect Lite software on the Raspberry Pi, type:

```
sudo apt install rpi-connect-lite
```

Next, use the `rpi-connect` command to start Connect for your current user as follows:

```
rpi-connect on
```

Similarly, when you want to stop Connect, run:

```
rpi-connect off
```

It is also recommended to enable "user-lingering," which allows you to log in remotely even when you are not logged in locally. Type:

```
loginctl enable-linger
```

Once Connect is running, use the following command to generate a link that will allow you to use the device:

```
rpi-connect signin
```

Point a browser to the link provided. If you don't already have a Raspberry Pi ID account, you will need to create one. Raspberry Pi ID also provides options for two-factor authentication (2FA) which you may enable for enhanced security.

Once you have a Raspberry Pi ID, you can follow the link and log in to your Raspberry Pi ID account. You should see a link to your Raspberry Pi device. Choose a unique name to identify your Raspberry Pi and click "Create device." You should receive an email notification that a new device is available. Once the Raspberry Py is setup, you will be able connect to a shell remotely from anywhere by pointing a browser to connect.raspberrypi.com and clicking on the `Connect` button next to the device name.

> If you are having problems connecting, try using a different browser, such as Google Chrome.

To learn more, visit Raspberry Pi Connect.

> ***Warning*** If you receive an email reporting a strange sign-in on Connect, immediately change your Raspberry Pi ID password and remove the device from your account. Consider enabling two-factor authentication for greater security.

Editing Python Programs over SSH

Next, practice editing and running a Python program over `ssh`. Use `ssh` to connect to the Raspberry Pi and then enter the following command to create a new folder:

```
mkdir lab1
```

Next, change directories into the folder you created and edit a new Python program called `hello.py` by typing:

```
cd lab1
nano hello.py
```

Enter the following code into the source file

```
# Lab 1
name = input('What is your name? ')
print(f'Hi {name}, welcome to Lab 1!')
print('Good Bye')
```

Next, run this program by typing:

```
python3 hello.py
```

The program should run as expected. Note that nano is a simple, minimalist (and somewhat cumbersome) editor. Other nifty editors are available for the command line environment, like emacs and vim, which are simple but powerful alternatives to nano. If you are so inclined, you may want to learn one of these editors. In a future lab we will learn how to remotely edit programs on the Raspberry Pi over SSH using vscode.

Next, shutdown the Raspberry Pi by typing the following command:

```
sudo halt
```

Remove the power cable only when shutdown completes.

> **Note:** Note that a green LED on the Raspberry Pi indicates when the SD card is being accessed, so *do not remove power until the green LED stops flashing*. Once Linux has completely shut down and the green LED has stopped flashing, you may unplug the power supply. Always perform a `sudo halt` before removing power from your Raspberry Pi.

Since we normally plan to run your Raspberry Pi in a *headless* configuration (without a monitor or keyboard), you will need to connect to your Raspberry Pi using either `ssh` or the Raspberry Pi Connect service.

Questions

Total 10 marks (1 mark):

1. What do the red and green LEDs on the Raspberry Pi board indicate? (1 point)
2. For your Raspberry Pi: (1 point)
    - Record your WiFi (`wlan0`) IP address
    - Record your WiFi **MAC** address
3. Briefly explain the difference between an IP address and a MAC address. (1 point)
4. Why might your Wi-Fi IP address change when you reboot your Raspberry Pi? (1 point
5. Record and explain the following measurements from your system: (1 point)
    - Total memory (using `free -h` command)
    - Initial swap memory size
    - Final swap memory size after disabling swap
6. To display a list of mounted disk partitions and their sizes, type the following command:

    ```
    df -h
    ```

    How much storage space is free on the root partition (the `/` partition) on your SD card? (1 point)

7. Take a screenshot of the Raspberry Pi Connect webpage showing the overview of your configured Raspberry Pi with the **UUID** and **serial number** showing.
8. What is the kernel version running on your Raspberry Pi? (1 point)

> Hint: use the command to display operating system information included the Shell commands section in chapter 1.

9. What is the proper procedure for shutting down the Raspberry Pi? Why is it important to wait for the green LED to stop flashing before removing power? (1 point)

Lab #2: Editing and Running Programs on the Raspberry Pi

The goals of this lab are to learn:

- Editing and running programs *remotely* in a terminal window
- Editing and running programs remotely with vscode
- Comparing C and Python

Connecting over Wi-Fi

Your Wi-Fi adapter was setup in Lab 1 so we can now run the Raspberry Pi in a *headless* configuration (with no screen or keyboard) and connect over the network. Power up your Pi and determine your IP address using *one* of the following two methods:

1. Connect a monitor and keyboard and power up the Pi, login and type `ifconfig` to list your IP address.
2. Connect to a shell using Raspberry Pi Connect and type `ifconfig` to determine your Wi-Fi IP address

> Recall that each time you connect to Wi-Fi (or Ethernet) your Raspberry Pi may get assigned a different IP address!

Once you have determined the IP address of your Raspberry Pi, you can connect using `ssh`. Note that `ssh` is typically included with Linux and OSX. For Windows, you could use Putty (a common open source SSH client) or the Windows subsystem for Linux (WSL). WSL includes support for a basic shell (including `ssh` and many other commands) and can be launched by typing `WSL` or `bash` in the start menu. Once your desktop or laptop has access to `ssh`, open a terminal window and type:

```
ssh user@1.2.3.4
```

where `user` is the username you selected in the first lab and `1.2.3.4` is the Wi-Fi IP address you determined for your Raspberry Pi. If everything is working, you should be able to log in successfully after you are prompted for a password.

You can also use a tool related to `ssh` called **secure copy** (`scp`) to transfer files between your computer and the Raspberry Pi. To use `scp`, first create a test file named `testfile.txt` on your laptop or desktop and then use the following command to to transfer it from your desktop or laptop to your Raspberry Pi:

```
scp testfile.txt user@10.1.2.3:
```

This will transfer a local file named `testfile.txt` *from* your desktop or laptop *to* the home folder of `user` on the Raspberry Pi at IP address `1.2.3.4`. Note that you will be prompted for a password as well. Use the `ls` command on the Raspberry Pi to confirm that the file was successfully transferred. To learn more about `scp`, consult the `man` pages as follows:

```
man scp
```

Next, create a new folder for this lab as follows:

```
mkdir lab2
```

This creates a new folder (directory) named `lab2`. Note that Linux is case sensitive! Type `ls` to list all the files and folders in your home directory and you should see the new folder there. Change directories to the new folder by typing:

```
cd lab2
```

Running an Assembly Language Program on the Raspberry Pi

The Raspberry Pi uses an ARM processor, one that has its own Instruction Set Architecture (ISA). The ISA is a model that defines the set of instructions, registers, and operations that a processor can perform. The ARM processor has two different Instruction Set Architectures: a 32-bit architecture referred to as ARM32, and a 64-bit architecture referred to as ARM64. The 64-bit ISA is sometimes referred to as **AArch64**, and the 32-bit ISA is sometimes referred to as **AArch32**. Hence the Raspberry Pi OS has two different versions, a 32-bit version and a 64-bit version. All recent versions of the Raspberry Pi (since 2016) support the ARM64 architecture, but are backwards compatible with ARM32, allowing you to to execute 32-bit applications on ARM64 processors. To determine which version of the Raspberry Pi OS you are running, simply type:

```
uname -m
```

`aarch32` indicates a 32-bit OS and `aarch64` indicates a 64-bit OS.

In this section you will enter an ARM assembly program and run it. Assembly language is a *machine-*

specific programming language that depends on the Instruction Set Architecture and has a one-to-one correspondence between its statements and the computer's native machine language. Programs written in Assembly Language are translated into the native *Machine Language* using an *Assembler*.

To begin editing an assembly language program, type the following:

```
nano program1.s
```

As introduced in the last lab, nano is a general-purpose text editor that you can use to edit the source code of a program or other text files in a shell.

Of course, the assembly code you enter will vary depending on whether you are running a 64-bit or 32-bit OS. Assuming a 64-bit OS, enter the assembly language program as follows:

```
// ARM64 Assembler program to print a greeting
// CS326 Embedded System and IoT

.data
message: .ascii  "Hello World!\nAssembled for a 64-bit Raspberry Pi.\n"

.text
.global _start
_start:
// Linux system call to print message
    mov     w0, #1          // File descriptor 1 (stdout)
    ldr     x1, =message    // Load address of message
    mov     w2, #57         // Message length
    mov     w8, #64         // system call for write operation
    svc     #0              // perform system call

// Linux system call to terminate program
    mov     w0, #0          // Return code 0 (success)
    mov     w8, #93         // exit system call
    svc     #0              // terminate program
```

Next, assemble, link, and run the program by typing the following sequence of commands:

```
as -o program1.o program1.s
ld -o program1 program1.o
./program1
```

The dot-slash in front of the program name tells the shell to execute the program in the current folder. If everything was entered correctly, your program should assemble and run and a message should be displayed.

Question 1: What does the first system call do in the assembly program? What does the file descriptor represent?

Next, type the following command to view your resulting object file:

```
objdump -S program1.o
```

The second part of the output will display a disassembly of the program, showing the machine code (in hexadecimal) alongside the corresponding assembly instructions.

Question 2: What is the machine code corresponding to the ARM instruction: `mov w0, #0x1`

To learn more about ARM assembly, see: https://diveintosystems.org/book/C9-ARM64/index.html

Editing and compiling a C program locally on the Raspberry Pi

Next, we will create a new program using the C programming language. Connect to your Raspberry Pi using `ssh` and launch an editor program by typing:

```
nano program1.c
```

Good coding style requires that C programs start with some header comments that provide information about the program. Type the source code below:

```
/* My first Raspberry Pi C program
Name: your name
*/
#include <stdio.h>

int main(void)
{
   printf("Hello world.\n");
   printf("Compiled and run on a Raspberry Pi.\n");
   return 0;
}
```

When you have finished editing, save your file and exit by hitting `ctrl-x`. To compile your program in the terminal window, type the following compile command:

```
gcc -Wall program1.c -o program1
```

Note that the `-Wall` command line option tells the compiler to show all warnings, and the `-o` option specifies the filename for the compiled output.

Once the compile command has been entered, press return. If the compiler detects any errors or warnings, they will be reported. If there are any errors or warnings, you will need to return to the editor to make any corrections as necessary.

Once your editing and compiling are complete and no warnings or errors are generated, you can run your program in a terminal window. Type the command ls to show a listing of the files, which should now include a compiled program named `program1`. Run the program by typing:

```
./program1
```

Recompile the program *without* the `-o` program option. Use the `ls` command to view the files in the directory now.

Question 3: What is the default name given to the output program if the `-o` option is *not* specified? i.e. if the program is compiled as follows:

```
gcc -Wall program1.c
```

Remote editing using VScode over SSH

Using text editors like `nano` on the Raspberry Pi can be cumbersome and cross-compiling can be complicated. Another option is to use a remote graphical editor to edit source files on the Raspberry Pi from a desktop or laptop computer. One program for doing this is **Visual Studio Code**, often referred to simply as **vscode**.

Using a Remote SSH connection with VScode

If `vscode` is not already installed, visit code.visualstudio.com/download and download the appropriate installer for Windows, OSX, or Linux. Once `vscode` is installed, perform the following steps:

- launch `vscode` on your desktop or laptop
- type `ctrl-shift-x` (or `cmd-shift-X` on OSX) to bring up the `vscode` extensions on the left
- type **Remote SSH** in the search box
- select and click "install" for the **Remote SSH** extension (from Microsoft)

After the installation is complete, you should see a new blue "connect" icon appear in the bottom left corner of `vscode`. Click on the connect icon and select "Connect current window to host" (there's also a "Connect to host" option that will create a new window).

Enter the `ssh` connection details using the format `user@1.2.3.4` where `1.2.3.4` is the WiFi (or Ethernet) IP address of your Raspberry Pi and `user` is your Raspberry Pi username. Next, you will be prompted to enter your Raspberry Pi password and then you will need to wait briefly as `vscode` sets up and initializes.

Once the setup and initialization are complete, click on the left link to "open folder" (or select File→Open Folder) and your home folder on the remote Raspberry Pi should appear! Select the `lab2` folder you created earlier in this lab and it will become your *working folder*. A list of files should appear on the left where you can select and open the `program1.c` file. Edit the file and make some changes to add another print statement, and then save.

The `vscode` editor provides a quick and convenient way to open a shell on the Raspberry Pi by typing `crtl-shift-` ` which opens a new remote terminal window at the bottom of the screen (connected via `ssh` to the Raspberry Pi). In the terminal window type the following to compile and run the program you just edited:

```
gcc -Wall program1.c -o program1
./program1
```

Note how much more delightful it is to use a graphical editor over SSH rather than editing code locally with the `nano` editor! VScode is the editor I would recommend using throughout this course.

Using SSHFS extension with VScode

Another `vscode` extension, SSHFS, allows mounting remote folders using SSH as if they were local workspace folders. Try installing the SSHFS extension and editing files your Raspberry Pi.

> Remote editing on the Raspberry Pi using `vscode` can also be accomplished using a *code tunnel*. A *code tunnel* is a secure connection that allows developers to work with code on remote machines. To learn more, see developing with remote tunnels. Yet another tool for remote development of embedded systems is PlatformIO, a platform for remote development in embedded systems.

Comparing C and Python

Which programming language is more efficient: C or Python? In this next section we will run a program to solve the same computational problem using C and Python and compare the run times. We will write programs in both languages to numerically solve the integral $\int_1^4 \sqrt{x}\,dx$ and then record the execution time required.

First, use `vscode` to remotely edit the following Python program as `compute.py` on the Raspberry Pi to perform the numerical integration:

```
import math

NUM_STEPS = 1000000 # number of integration steps
a = 1  # lower limit of integration
b = 4  # upper limit of integration
DELTA_X = (b-a)/NUM_STEPS
sum = 0.0
x = a
for step in range(NUM_STEPS):
    y = math.sqrt(x)
    sum += y
    x += DELTA_X
integral = sum * DELTA_X
print(f'restult = {integral}')
```

Next, execute the program as follows:

```
time python3 compute.py
```

Record the execution time reported. Repeat this ten times and determine the average execution time.

Next, use `vscode` to remotely edit the following C program as `compute.c` on the Raspberry Pi to compute the same integral.

```
#include <math.h>
#include <stdio.h>
#define NUM_STEPS 1000000  // number of integration steps
#define a 1  // lower limit of integration
#define b 4  // upper limit of integration
#define DELTA_X  (double)(b-a)/NUM_STEPS

int main()
{
    double sum = 0.0;
    double y, integral;
    double x = a;

    for (int step = 0; step<NUM_STEPS; step++)  {
        y = sqrt(x);
        sum += y;
        x += DELTA_X;
    }
    integral = sum * DELTA_X;
    printf("%f\n", integral);
}
```

Compile the code as follows:

```
gcc compute.c -lm -o compute
```

Run the compiled program and record the execution time as follows:

```
time ./compute
```

Record the execution time. Repeat this ten times and determine the average execution time.

Question 4: What are the average execution times using Python and C? Which language is faster and why?

As a last step, recompile the C program and add the `-Ofast` flag as follows:

```
gcc -Ofast compute.c -lm -o compute
```

Run the compiled program ten times and determine the average execution time.

Question 5: What does the `-Ofast` flag do and what was the result? To learn more, visit the gcc online documentation.

Lab #3: Using the GPIO Port

The goals of this lab are to learn about GPIO interfacing and programming. This lab requires:

- a breadboard
- a red LED (HLMP-4700 or similar)
- a 120 ohm resistor
- a tactile push-button switch
- an assortment of male-to-female jumper wires

The breadboard should look something like the image below. The breadboard allows you to conveniently connect components and wires by plugging them into the holes of the breadboard. The internal wiring of the breadboard connects lines of holes together such that that the inner holes are connected in vertical lines on either side of the "gutter" and the outside holes are connecting in lines that run horizontally as a "bus." The outside horizontal lines are typically used to carry power and ground while the inner holes are used to connect integrated circuits, components, and wires. The holes are electrically connected as illustrated with the blue lines below.

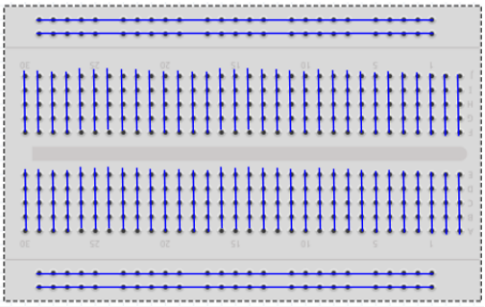


Figure 37: Breadboard with blues lines showing internal wiring.

Wiring an LED

The next part this lab requires is an LED. For our first circuit, we will wire an LED to the GPIO port on the breadboard. LEDs are "current-driven" devices where the current required to achieve a reasonable brightness is specified in the datasheet. Exceeding the maximum LED current can result in damage or thermal failure of the LED. The output of the GPIO port is a set voltage, so a "current-limiting" resistor is normally put in series with the LED to set the current appropriately. Note that the LED is a *polarized* device; the longer lead is the positive lead (anode), and the shorter lead is the negative lead (cathode).

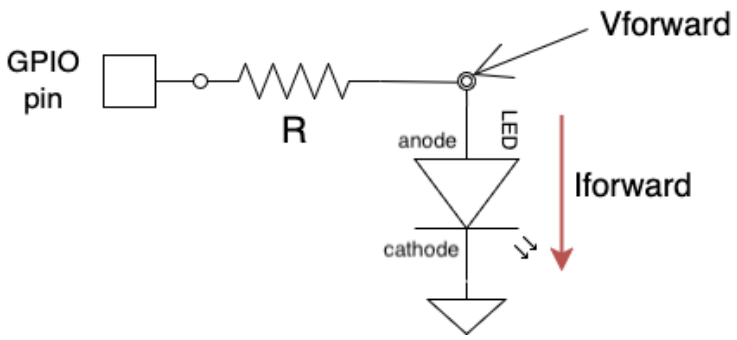


Figure 38: LED with current-limiting resistor connected to Raspberry Pi GPIO pin.

The diagram above indicates the schematic symbols for an LED and a series resistor. Selecting a suitable resistance, R, will depend on the forward current required to light up the LED, as well as the the forward voltage of the LED. The required resistance, R, to achieve the appropriate LED current is determined by solving the following equation (based on Ohm's Law):

$$R = \frac{V_{GPIO} - V_{forward}}{I_{forward}}$$

where:
R = the value of series resistor required
V_{GPIO} = the voltage of the GPIO port when turned on (approximately 3.3V)
$V_{forward}$ = the "forward voltage" of the LED when it is on (from the datasheet)
$I_{forward}$ = the forward current required to illuminate the LED (from the datasheet)

By consulting the datasheet for the LED, we can determine the LED forward voltage, $V_{forward}$, and the illumination current, $I_{forward}$. Assuming a $V_{forward} = 2.1V$ and $I_{forward} = 10mA$, we get a resistance of roughly $120ohms$. A suitable wiring diagram connecting the LED to the Raspberry Pi GPIO port pin is shown below (note the resistor color codes displayed in the diagram are different than the value you will use). Use a resistor Color Code Calculator to find out how to identify a $120ohm$ resistor.

Warning: Remember to perform all your wiring with the power OFF! Have a lab instructor check your wiring before applying power. Also, when checking your wiring, make sure that the exposed leads on the resistors and the LED do not touch each other as this could cause short circuits.

Putting the circuit together we get the circuit illustrated above. Use male-female jumper wires to

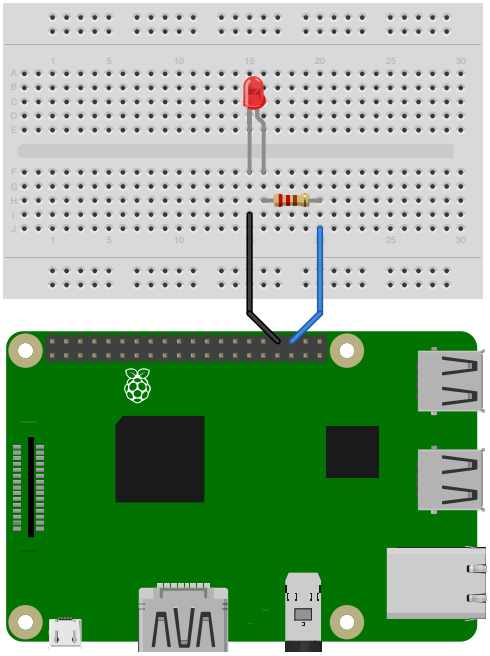


Figure 39: BCM 16 connected via resistor to an LED

connect the GPIO header pins on the Raspberry Pi to your breadboard. Pick the red LED from your kit and wire the **long** lead of the LED to a $120ohm$ resistor which connects to BCM 16 on the Pi. (Note this is **not pin number 16 of the connector**, but rather **BCM 16**). Use a wire to connect the ground (physical pin #34) to the *shorter* lead of the LED.

> **Note:** the holes on a new breadboard may initially be quite tight making it difficult to insert components with flimsy leads, like LEDs and resistors. Pushing components into tight breadboard holes may result in bending the leads. One thing that may help is to first insert the header pin from one end of a jumper wire into the tight holes to "loosen" them up before attempting to insert other components.

> **Note**: Depending on the context, there are different pin numbering conventions used with the Raspberry Pi. We will be using the BCM numbering convention for these labs. Note that BCM numbers do *not* correspond the pin numbers on the physical connector. The BCM pin numbers can be viewed by typing `pinout` in a shell or by visiting https://pinout.xyz/.

Power up and turn on the LED

Now power up the Raspberry Pi. Open a terminal window on your desktop PC and connect to the pi using `ssh` or Raspberry Pi Connect. (Refer to lab 1 if you need recall how to connect to the Raspberry Pi).

Begin testing your GPIO connection using the `raspi-gpio` command line utility (which should be installed by default). To show the state of all the GPIO pins, type:

```
raspi-gpio get
```

This should display the state of all the GPIO pins. Note that there are several different pin numbering schemes that can be used with the Raspberry PI, which can lead to some confusion. Note that we will be using the **BCM** numbering scheme. BCM represents the *Broadcom SOC channel* and reflects the numbering scheme used by the Broadcom ARM processor and the `raspi-gpio` utility.

We now need to set BCM 16 as an output as follows:

```
raspi-gpio set 16 op
```

Now we can turn on the LED by issuing the command to set BCM 16 high:

```
raspi-gpio set 16 dh
```

To turn off the LED, issue the command to set BCM 16 low:

```
raspi-gpio set 16 dl
```

To learn more about the raspi-gpio utility, type:

```
raspi-gpio help
```

and the various tool options will be displayed.

Write a C program to flash an LED

Create a new lab folder named **lab3** on your Raspberry Pi. Next, use `vscode` to connect to your pi and enter the following C program and save it to a file named `blink.c` (refer back to the previous lab if you need a refresher on how to edit using `vscode`). This code makes use of the pigpio library:

```
#include <stdio.h>
#include <pigpio.h>
#include <unistd.h>

#define LED 16
#define DELAY 1

int main (int argc, char *argv[])
{
    if (gpioInitialise() < 0)
        return 1;
    gpioSetMode(LED, PI_OUTPUT);

    while (1)
    {
```

```
        gpioWrite(LED, PI_ON);
        printf("LED ON\n");
        sleep(DELAY);
        gpioWrite(LED, PI_OFF);
        printf("LED OFF\n");
        sleep(DELAY);
    }
    gpioTerminate();
    return 0;
}
```

Next, open a terminal window in `vscode` and compile the program from the command line as follows:

```
gcc -Wall blink.c -o blink -lpigpio
```

Finally, run the program in the terminal by typing:

```
sudo ./blink
```

Note that this code requires `sudo` privileges. Once started, the LED should begin blinking. Type `ctrl-C` to exit.

Write a Python program to flash an LED

Make a new Python source file called `blink.py` and enter the following Python code using vscode:

```
# CS326 Lab 3
# Blinking LED
from gpiozero import LED
import time

# Create LED object using GPIO pin 16
led = LED(16)
DELAY = 0.5

# Blink the LED 20 times
for count in range(20):
    led.on()
    print('LED: on')
    time.sleep(DELAY)
    led.off()
    print('LED: off')
    time.sleep(DELAY)

led.close()
print("Done!")
```

Save the file as `blink.py` and then run it as follows:

```
python3 blink1.py
```

Next, modify the above code to implement the same blinking functionality, but use the `toggle()` method in the `gpiozero` library instead of explicitly turning the LED on or off. Consult the gpiozero LED class documentation for details on the `toggle()` method.

Finally, modify the above code to implement the same blinking functionality, but use the `blink()` method in the gpiozero LED class.

> Note that in embedded systems there is often "more than one way to crack an egg," but some methods are better than others.

To learn more about the `gpiozero` library used in this code, visit the gpiozero doumentation.

Read an input switch

Wire up an input switch as shown below, connecting to BCM 12 on the Raspberry Pi (leave the LED wired as it was from the previous step). Once the switch is properly wired, power up your pi once again and connect over `ssh`.

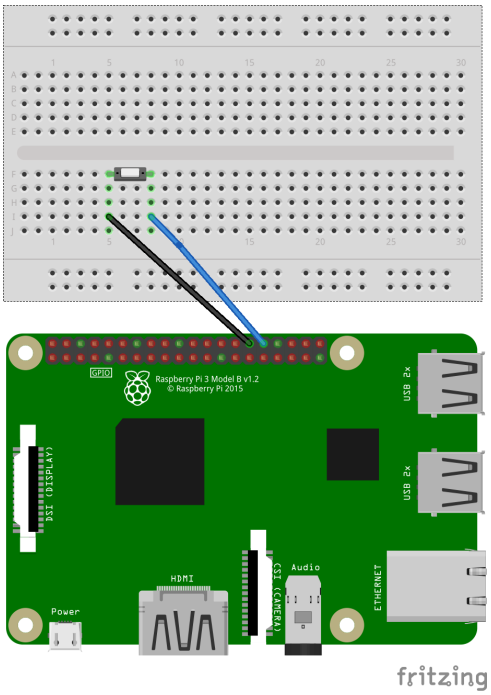


Figure 40: Input switch connected to Raspberry Pi GPIO input.

> Be sure to remove power whenever you are performing any wiring.

We can now quickly test our circuit using the `raspi-gpio` utility. First, set BCM 12 as an *input* as follows:

```
raspi-gpio set 12 ip
```

We can also turn on a weak internal "pull up" resistor to ensure the input "floats" high when the switch is open:

```
raspi-gpio set 12 pu
```

When the switch closes, the input will be forced low. Now we can read the switch input on BCM 12 as follows:

```
raspi-gpio get 12
```

We can repeatedly call this command to monitor the input (use the up arrow key so you do not have to re-type the command over and over). Push the switch in and out and observe the changing state of the input pin.

Python program to read the switch

Next, we will create a Python program to count the number of input switch transitions by *polling* the input. Using `vscode`, enter the program **`switch.py`** as shown below:

```
# CS326 Lab 3
# Count number of input switch transitions

from gpiozero import Button

# Create a Button object with pull_up=True
button = Button(12, pull_up=True)

count = 0
previous_state = False  # Keeps track of the last state of the button
    input

try:
    while True:
        # is_active will be True when button is pressed
        if button.is_active and previous_state == False:
            count += 1
            print(count)
            previous_state = True
        # Check if button is released
        if not button.is_active and previous_state == True:
            previous_state = False
```

```
except KeyboardInterrupt:
    pass
```

Run the code by typing:

```
python3 switch.py
```

This program runs a loop that continuously *polls* the button input. Observe how the output changes in response to switch transitions. Note whether the count always increments by one when the button is pressed. To exit the program, type `ctrl-c`.

Python event-based program to read the switch

The following Python program, `switch_event.py`, counts the number of input switch transitions using input *events* rather than *polling*:

```
# CS326 Lab 3
# Count input switch events
from gpiozero import Button
from signal import pause

button = Button(12, pull_up=True)
count = 0

def count_press():
    global count
    count += 1
    print(count)

# Trigger a callback whenever the button is pressed
button.when_pressed = count_press

try:
    pause() # Run script and listen for button presses
except KeyboardInterrupt:
    pass
```

Note that this code uses *events* instead of *polling* an input pin. Does the count now increment by one when the button is pressed?

Python events with switch de-bouncing

Rewrite the button code to include switch *debouncing*, which is available in the `gpiozero` library, as follows:

```
from gpiozero import Button
from signal import pause

# Set bounce_time to debounce the switch
```

```
button = Button(12, pull_up=True, bounce_time=0.1)
count = 0

def count_press():
    global count
    count += 1
    print(count)

# Trigger a callback whenever the button is pressed
button.when_pressed = count_press

try:
    pause()  # Run script and listen for button presses
except KeyboardInterrupt:
    print("Exiting...")
```

Note the addition of a `bounce_time` parameter. Now confirm that the count always increments by one when the button is pressed. To learn more about using the GPIO pins with Python, see the documentation for the gpiozero library.

Lab Questions

1. Why is a current-limiting resistor necessary for an LED?
2. If you have a 3.3V GPIO output and an LED with Vf = 1.8V that requires 5mA, what resistance value should you use? Show your calculations. Note that 5mA = 0.005Amps.
3. Power down your circuit and reverse the wiring to the LED. Power up your circuit and run the `blink1.py` program again. What happens and why?
4. Why are we using a pull-up resistor with the button input?
5. When running the **`switch.py1`** program, why does the count sometimes increase by *more* than 1 count when the switch is pressed?
6. Contrast *polling* versus *event-based* approaches for reading button inputs. What is the advantage of using events?
7. Wire the LED as shown in Figure 38 and the switch as shown Figure 40. Make a small modification to `switch_event.py` so that when the switch is momentarily pressed it *latches* on the LED: i.e. when the switch is pressed the LED should come on and remain on. When the switch is pressed again, the LED should turn off and remain off. Each time the switch is pressed the code should alternate between turning the LED on and off. Write a program that uses a button callback function with debouncing and which uses the `toggle()` method that is part of the LED class. Be sure to include header comments and other comments, appropriate constant definitions, and meaningful variable names.

Lab #4: Using the PWM output

Goals: working with PWM (Pulse Width Modulation) signals

The parts required for this lab incliude:

- a red LED (HLMP-4700 or similar)
- a 220ohm resistor
- a microservo motor (TowerPro SG92R or equivalent)
- 4 tactile button switches

Using a Software PWM to vary the brightness of an LED

We will use a PWM (Pulse Width Modulation) signal to vary the brightness of an LED. To begin, wire up an LED as shown below. Connect the LED *anode* to the resistor and the *cathode* to ground. Recall that the *anode* of the LED has the longer lead. Use a 220ohm current limiting resistor.

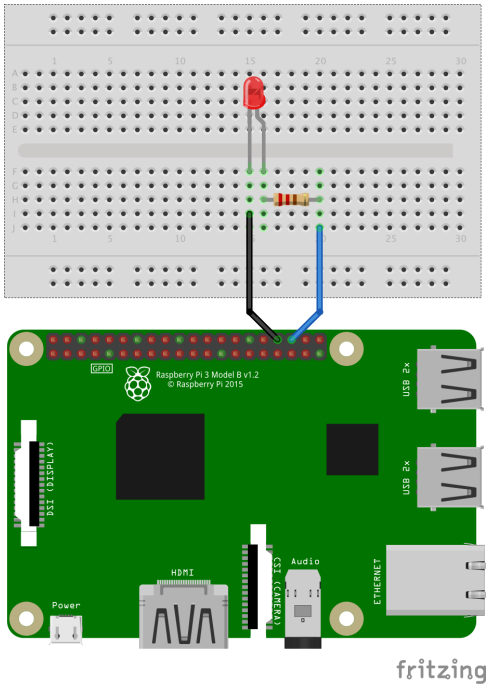


Figure 41: Raspberry Pi with LED output.

Enter the following Python program to control a software PWM signal to the LED:

```
from gpiozero import PWMLED

# Set LED to BCM 16 with 50Hz frequency
LED = PWMLED(16, frequency=50)
```

```
while True:
   try:
      duty_cycle = int(input('Enter a PWM duty cycle from 0-100 (enter -1
         to end): '))
      if duty_cycle == -1:
         break
      if duty_cycle < 0 or duty_cycle > 100:
         print('Error: Duty cycle must be between 0 and 100\n')
         continue

      # Convert percentage to 0-1 range for gpiozero
      LED.value = duty_cycle / 100
      print(f'Duty cycle = {duty_cycle}%')
   except ValueError:
      print('Error: enter a number from 1 to 100.\n')

print('Done.')
```

Question 1: How does changing the duty cycle of a PWM signal affect the brightness of an LED? Why are you unable to see the 50Hz ON-OFF pulsing of the PWM? (1 point)

> Note: If you have a phone with a camera, try recording the LED to see how it appears with a PWM less than 100%. If it is flickering, why might this be so?

Using a Software PWM to Control a Microservo

Servo motors are actuators that allow you to add motion to a system. They're useful because you can specify an angle to turn and the micro servo will automatically adjust the position for you. An ordinary motor will simply turn when power is applied, but the micro servo includes electronics, gears, and a feedback sensor to control the position of the output. Servos typically come with multiple attachments, such as wheels or levers (known as "horns") that attach to the shaft and can be coupled to whatever mechanical device they are operating.

The position of the servo motor is set by the length of a control pulse which the servo typically expects to receive roughly every 20 milliseconds. By precisely setting the width of the pulse one can adjust the position of the servo.

Pick one of the "horns" in your kit and press it onto the servo shaft (don't worry about tightening it with a screw for now). The servo motor also comes with three wires: power (red), ground (brown), and a wire to send commands (yellow). In general, a servo can draw substantial power, especially when the motor is accelerating or requires a holding torque, which may cause its power supply to fluctuate. The motor in your kit requires only modest power, so we will run it using the 5VDC power directly from the Raspberry Pi. However, depending on the power draw, you may require a separate power supply for your servo motors.

Your microservo motor should have the wires as indicated in the table below:

Color	Description
yellow	Control input
red	Power (+5VDC)
brown	GND

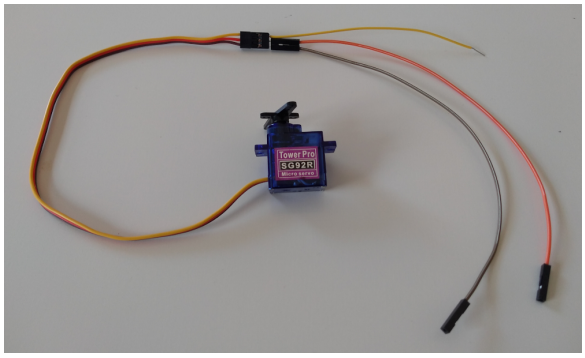


Figure 42: Microservo motor with wire extensions.

First, note that the servo motor has a female connector which, because of the pinout, cannot be directly plugged into the GPIO header. This can be addressed by inserting jumper wires into the servo cable header as illustrated with the yellow wire in the photo above. Insert two regular jumper cables for power and ground and connect them directly to +5V and GND on the GPIO port. Insert a regular hookup wire into the microservo cable socket to run a connection to the breadboard (see the wiring diagram below). It is strongly recommended that you use wires of the same colors to ensure you don't get confused!

Next, without applying power, connect the servo motor in a circuit as shown to the right. Note that 220ohm resistor is not strictly required, but it is used to help protect the GPIO pin in the event a fault develops in the servo.

Double check your circuit before you apply power since an improperly wired 5VDC signal may destroy your GPIO inputs!

We will use BCM 18 for a *software PWM* output and program it for the correct frequency and duty cycle (pulse width). Servo motors typically expect a pulse frequency of around 50Hz and pulse widths that vary from roughly 1-2ms. The pulse width is set to a value that corresponds to the angle we want to command.

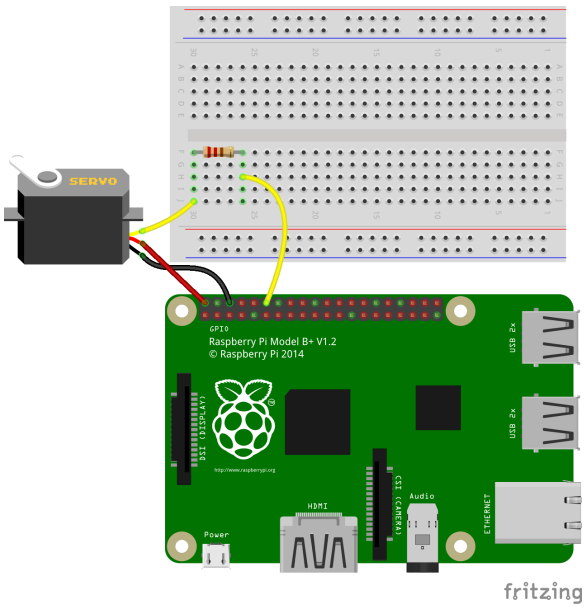


Figure 43: Wiring the microservo motor to the Raspberry Pi.

For our microservo, a pulse width of 1.5ms corresponds to the midpoint of travel for the servo motor (0 degrees). A pulse width of 2ms will turn the servo all the way to the right (90 degrees), and a pulse width of 1ms will turn the servo all the way to the left (-90 degrees). For this step, we will move the servo just shy of the full sweep to +/-72 degrees.

For a 50Hz frequency, the period $T = \frac{1}{50}$ evaluates to 20 milliseconds. The relationship between pulse width (PW) and duty cycle(D) can be calculated using the following equation:

$$D = \frac{PW}{T} = \frac{PW}{20ms}$$

Solving this equation for 3 different angles is summarized in the table below:

Servo Angle (degrees)	Pulse width (milliseconds)	Duty Cycle (%)
0	1.5 ms	7.5%
72	1.9 ms	9.5%
-72	1.1 ms	5.5%

Attempting to drive micro servos beyond their mechanical limits can cause stripped gears! To keep a safe margin, ensure that you limit the servo angle to an absolute maximum of +90 and -90 degrees!

Enter the following code to step the servo motor through 0, 72, and -72 degrees in a loop:

```
from gpiozero import Servo
import time

DELAY = 2

# Use the gpiozero Servo class
SERVO = Servo(18)  # Connect servomotor to BCM 18

try:
    while True:
        print('setting minimum angle...')
        SERVO.value = -1  # Equivalent to 5.5% duty cycle
        time.sleep(DELAY)

        print('centering...')
        SERVO.value = 0  # Equivalent to 7.5% duty cycle
        time.sleep(DELAY)

        print('setting maximum angle...')
        SERVO.value = 1  # Equivalent to 9.5% duty cycle
        time.sleep(DELAY)

except KeyboardInterrupt:
    # return to 0 degrees position before exiting
    print('Reset angle...')
    SERVO.value = 0
```

Run the servo program and ensure that the servo motor is moving back and forth. The issues with a software PWM can be exacerbated under an increased CPU load.

Run the servo program again, but this time induce a "fake load" on the CPU. A fake load can be initiated using the `stress-ng` utility, which can be installed as follows:

```
sudo apt install stress-ng
```

Run the serove program and then make a second `ssh` connection to the Raspberry Pi to run the `stress-ng` program to load the CPUs for 60 seconds by typing:

```
sudo stress-ng --hdd 4 --timeout 60s
```

Observe the behavior of the servo motor. Hit `ctrl-c` to exit the program.

Using a hardware PWM to control a Microservo

In order to use a *hardware* PWM signal, we need to turn to a different library: the pigpio library. The `pigpio` library should already be installed on your Raspberry Pi.

Power up your Pi and start the `pipgio` service by typing the following from the command line:

```
sudo service pigpiod start
```

To verify that the pipgio service is running, type:

```
sudo service pigpiod status
```

Now, enter the following code which will continuously cycle through the positions -90, 0 , and 90 degrees. The `set_servo_pulsewidth` method sets a pulse width in microseconds and automatically sets the PWM frequency to 50Hz. Note that the `pigpio` library uses standard BCM pin numbers.

```
import time
import pigpio

# Constants
PWM = 18 # Use hardware PWM on BCM 18
DELAY = 2

# connect to the pigpio service (which must be running)
pi = pigpio.pi()
if not pi.connected:
    exit(0)

pi.set_PWM_frequency(PWM,50); # Set PWM frequency to 50Hz

try:
    while True:
        print('setting angle = -72 degrees')
        pi.set_servo_pulsewidth(PWM, 1100)
        time.sleep(DELAY)
        print('setting angle = 0 degrees')
        pi.set_servo_pulsewidth(PWM, 1500)
        time.sleep(DELAY)
        print('setting angle = 72 degrees')
        pi.set_servo_pulsewidth(PWM, 1900)
        time.sleep(DELAY)
except KeyboardInterrupt:
    pi.set_servo_pulsewidth(PWM, 0) # turn pulses off
pi.stop()
```

As before, run the servo program while inducing a “fake load” on the CPU. Initiate a second SSH connection to the Raspberry Pi in another terminal window and use the `stress-ng` program to load the

CPUs for 60 seconds by typing:

```
sudo stress-ng --hdd 4 --timeout 60s
```

Observe the behavior of the servo motor. Note that you can stop the program by typing `ctrl-c`. If you wish to stop the `pigpiod` service, type:

```
sudo service pigpiod stop
```

Question 2: How does a hardware PWM compare to the software PWM? Why? (1 point)

Question 3: Wire **four** pushbuttons on your breadboard (if need be, refer to the last lab) connected to four suitable GPIO pins. One button will move the servo left (-72 degrees), another right (+72 degrees), another to the center (0 degrees), and the last button will stop the servo. Using a hardware PWM and the `pigpio` library, create a program that uses a state machine that shifts states depending on the button presses. When the stop button is pressed, the program should exit the state machine, stop the servo and exit the program. Your program should be written using a state machine class that must include the following:

- use the pigpio library for all GPIO functions and use a *hardware* PWM
- define a set of state variables: `'SERVO_LEFT,'SERVO_CENTER','SERVO_RIGHT'`
- Create a `State_Machine` class
  - Include all the following methods in this same class:
    * `__init__(self,pi)`
      · copy `pigpio` object passed as a constructor variable to `self.pi`
      · `self.states= {'SERVO_LEFT,'SERVO_CENTER','SERVO_RIGHT'}`
      · initialize all input buttons with a pullup and a "glitch filter" of 1000us (see pullup docs and glitch filter docs)
      · define a callback method for all four buttons to call the appropriate method on a `FALLING_EDGE` event within this same class (see the callback docs)
      · initialize a variable to store the current state called `self.state` and initialize to `SERVO_LEFT`
      · move servo to -72 degrees
    * `left_button_callback(self,...)`
      · update the state variable to `SERVO_LEFT`
      · print the new state
      · move the servo to -72 degrees and return
    * `center_button_callback(self,...)`
      · update the state variable to `SERVO_CENTER`
      · print the new state

        - center the servo at 0 degrees and return
    * `right_button_callback(self,...)`
        - the state variable to `SERVO_RIGHT`
        - print the new state
        - move the servo to +72 degrees and return
    * `stop_button_callback(self,...)`
        - Turn PWM off
        - print message and exit program
    * `get_state(self)`
        - ensure current state is in the set of `self.states`
        - return current state

Your man program should define the state machine class, instantiate a state machine object as indicated in the following code skeleton:

```
from signal import pause
import pigpio

# Define State_Machine class here

# connect to the pigpio service (which must be running)
pi = pigpio.pi()
if not pi.connected:
    exit(0)
pi.set_PWM_frequency(PWM,50); # Set PWM frequency to 50Hz

sm = State_Machine(pi)  # Instantiate state machine
try:
    pause()
except KeyboardInterrupt:
    pi.stop()
print("exiting")
```

Question 4: Draw an official UML state diagram for your code in question 5.
For UML state diagrams, consult this UML documentation and be sure to include the symbol to indicate the *initial state* and the *final state*. Consider using draw.io.

Question 5: What kind of state machine is this (Mealy or Moore)?

Question 6: (a) Is this state machine *deterministic*? (b) Is this state machine *receptive*?

Lab #5: Scheduling and Kernel latency

Purpose: experimentally determine latency in a regular Linux kernel and compare it with a modified *preemptible* kernel.

Introduction

The term *latency*, when used in the context of the OS Kernel, is the time interval between the occurrence of an event and the time when that event is handled. This can be critical in certain embedded applications where a computer must respond to an external event within a guaranteed time.

How can we measure latency on the Raspberry Pi? One technique is to connect two GPIO pins with a wire and set one as an output and the other as an input. The output can be used to trigger an edge on the input allowing us to measure the elapsed time between the trigger and the event handler. This process can be repeated many times to compute an average response time. While this will not provide an exact measurement, it will give a good approximation of the type of latencies we might expect.

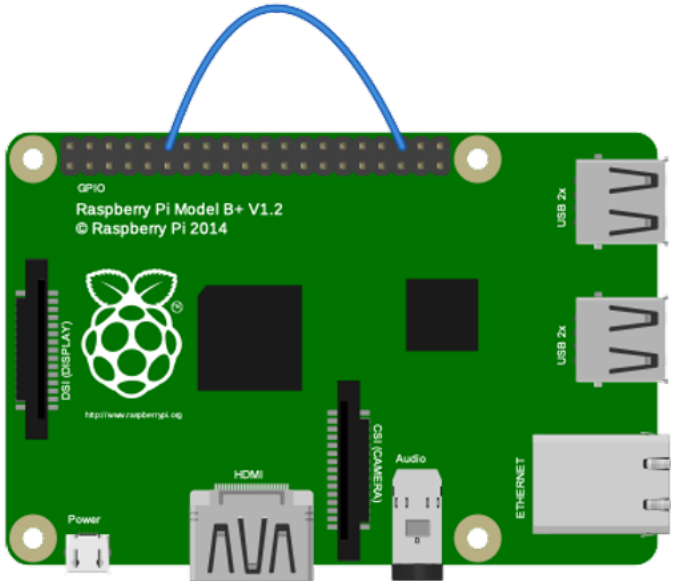


Figure 44: Wire jumper to test latency.

To begin, connect GPIO pin BCM 16 to pin BCM 18 as shown in the figure above. If you do not have a female-to-female jumper wire, the two pins can be connected with two jumpers to the breadboard where they can be connected.

Next, power up your Raspberry Pi and connect using `ssh`. In order to make accurate timing measurements, we will make use of the `time_ns()` function, which provides nanosecond resolution of time. This function is only available in Python version 3.7 and later versions, so confirm you are running a supported version of Python by typing:

```
python3 --version
```

We can estimate latency in Python by measuring the delay from the time that the output pulse is asserted to the time it takes for a callback function to respond to the input event. To do our test, create a program called `latency.py` as follows:

```
# CS326 Lab 6
# Experimentally record latency from GPIO event to callback function
from gpiozero import DigitalInputDevice, DigitalOutputDevice
import time

# Constants
COUNT = 5000  # Number of samples
HISTOGRAM_SIZE = 1000
NANOSECS_PER_MICROSEC = 1000

# GPIO pin objects
pin = DigitalInputDevice(16)  # Input pin
pout = DigitalOutputDevice(18)  # Output pin

# Global variables for timing measurements
t1 = 0
sum_of_latencies = 0
max_latency = 0
histogram = [0] * HISTOGRAM_SIZE

# Callback function for input
def input_callback():
    global t1, max_latency, sum_of_latencies, histogram

    # record time elapsed and store
    latency = time.time_ns() - t1
    if latency > max_latency:
        max_latency = latency

    latency_in_microseconds = int(latency/NANOSECS_PER_MICROSEC)
    if latency_in_microseconds < HISTOGRAM_SIZE:
        histogram[latency_in_microseconds] += 1

    sum_of_latencies += latency

# Set up the input pin callback on high input
pin.when_activated = input_callback

# Loop numerous times toggling output to trigger input event
for count in range(COUNT):
    t1 = time.time_ns() # Time is measured from here, but output changes
       on next line
    pout.on()           # set output HIGH to initiate a callback
    time.sleep(0.01)    # sleep for a while before taking another
```

```
        measurement
    pout.off()          # set output LOW

# Output histogram of latencies
print('Histogram of latencies measured (in microseconds):')
for x in range(len(histogram)):
    print(f'{x+1},{histogram[x]}')

print(f'Average latency: {(sum_of_latencies/COUNT)/NANOSECS_PER_MICROSEC}
   microseconds')
print(f'Maximum latency: {max_latency/NANOSECS_PER_MICROSEC} microseconds'
    )
```

Note that the code above makes use of the `time_ns()` function but that the times are converted to microseconds. Run the code as follows:

```
python3 latency.py
```

Note that the program outputs an *average* and *maximum* value as well as a histogram of values (all in microseconds). Run the program again and redirect the output to a text file as follows:

```
python3 latency.py > histogram.txt
```

Question 1: Run the program and capture the histogram to a text file. Transfer the histogram file to your laptop or workstation and use a spreadsheet to plot a histogram of the latencies. Be sure to label the axes and use an appropriate horizontal scale and a bin size of 1 us to show the shape of the histogram.

Question 2: What do you observe about the latency? What causes variations in the latency?

Using Latency Testing Tools

The previous test code runs in a Python interpreter which adds additional overhead. The Linux kernel has a set of optimized tools that can be used for latency benchmarking. To install some of the tools on your Raspberry Pi, type the following in a terminal:

```
sudo apt install rt-tests stress-ng
```

The `rt-tests` package includes a latency measurement tool called `cyclictest`. To use this tool to test latency, type:

```
sudo cyclictest --smp -p95 -m
```

The output of `cyclictest` will show measurements for each of the CPU cores. You should see an output something like the following:

```
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.12 6.76 8.02 1/207 8001
```

```
T: 0 ( 7560) P:95 I:1000 C: 116205 Min: 5 Act: 20 Avg: 17 Max: 198
T: 1 ( 7561) P:95 I:1500 C: 77457 Min: 5 Act: 18 Avg: 18 Max: 172
T: 2 ( 7562) P:95 I:2000 C: 58086 Min: 5 Act: 17 Avg: 16 Max: 200
T: 3 ( 7563) P:95 I:2500 C: 46463 Min: 5 Act: 16 Avg: 17 Max: 139
```

In the above output, the maximum latency measured for CPU0 is 198us, for CPU2 it is 172us, for CPU3 it is 200us, and CPU4 is 139us. Type `ctrl-c` to end the test. To find out more about `cyclictest`, type:

```
man cyclictest
```

The latency measurements can be exacerbated under an increased CPU load. Repeat the `cyclictest` as above, but this time induce a "fake load" on the CPUs just before starting the test. Initiate a second `ssh` connection to the Raspberry Pi and in another terminal window use the `stress-ng` program to load the CPUs for 60 seconds by typing:

```
sudo stress-ng --hdd 4 --timeout 60s
```

Question 3: Why are there differences between average and maximum latencies and between the four processors?

Reducing Latency by using the Preemptible (low latency) Kernel Option

In this next part we are going to build the Linux kernel so that it produces more predictable timing and lower latencies. The Linux kernel can be configured with an option that minimizes the amount of kernel code that is *non-preemptible* (except places that are critical sections in the kernel). This option is not enabled by default in the standard Linux kernel, and so we will need to compile a custom kernel.

Since the Linux kernel is a large program with *millions* of lines of code, it can take considerable time to compile. To speed up the compilation time, one can *cross-compile* the modified ARM kernel on a hefty workstation and then transfer the new kernel to the Raspberry Pi. For this lab, we will compile directly on the Raspberry Pi.

First, install the build dependencies:

```
sudo apt install bc bison flex libssl-dev make libncurses-dev
```

Next, download and build a new ARM custom kernel on the Raspberry Pi by typing the following in a terminal:

```
git clone --depth=1 --branch rpi-6.12.y https://github.com/raspberrypi/
    linux
```

Note that omitting the `--depth=1` will download the *entire* repository history and will take a *very* long time!

> We have specified the `rpi-6.12.y` branch since this is the first version of the kernel that includes the patch for a fully preemptible kernel.

The next step is to prepare the default configuration of the kernel. For a 64-bit ARM kernel, this is done as follows:

```
cd linux
KERNEL=kernel8
make bcm2711_defconfig
```

Setting `KERNEL=kernel8` selects the 64-bit kernel for the ARM processor. Before proceeding further with compiling the kernel, we will need to customize the kernel settings. To make changes to kernel settings we can use the `makemenu` utility as follows:

```
make ARCH=arm64 menuconfig
```

This utility essentially provides a friendly menu interface to modify numerous settings in the kernel `make` config file. Use the arrow keys, the return key, and the tab key to navigate. These settings will later be used to guide the compilation process to produce a Linux kernel that is customized to your individual settings.

For our purposes, we want to enable the *fully preemptible* kernel model. As mentioned earlier, this feature compiles the kernel so that all kernel code is *preemptible* except for a few select critical sections. To enable the preemption model from within the `menuconfig` utility, select *General setup -> Preemption Model*. Under the *Preemption Model* setup select *Fully Preemptible Kernel (Real-Time)*.

It is also recommended to give the kernel a custom name to distinguish it from other kernels. You can change the kernel name under *General setup -> Local Version* (give it a name like `-lab5`).

Once all the customizations are set, save the configuration and exit. After making changes to the configuration, a 64-bit ARM kernel can then be compiled by typing the following:

```
make -j6 Image.gz modules dtbs
```

This command cross-compiles a new ARM kernel with related modules. At this point, find something useful to do while the kernel code is compiling.

When the compilation completes, install the kernel as follows:

```
sudo make -j6 modules_install
```

Run the following commands to create a backup image of the current kernel, install the fresh kernel image, overlays, README, and unmount the partitions:

```
sudo cp arch/arm64/boot/Image.gz /boot/firmware/$KERNEL-preempt_rt.img
```

To view all the main kernel files, type:

```
ls -al /boot/firmware
```

The file listing will display full filenames which should include new kernel files with the `lab5` label (or whatever you set the *Local Option* to in the `menuconfig` step earlier). Make note of the file name that begins with `kernel8` following by the label you selected earlier.

To enable our new custom kernel on the next boot, edit the `/boot/firmware/config.txt` file as follows:

```
sudo nano /boot/firmware/config.txt
```

Add a line pointing to the new kernel like the following to the end of `/boot/config.txt` under a section labelled with `[all]`:

```
[all]
kernel=kernel8-preempt_rt.img
```

Finally, reboot into the new kernel by typing:

```
sudo reboot
```

Once the Raspberry Pi has rebooted, connect to a shell and verify that the new kernel is running using the following command:

```
uname -a
```

The new kernel name should now be reported with the custom Linux kernel label string you set along with a string indicating it is running the `PREEMPT_RT` option.

Congratulations! You have successfully configured, compiled, and installed a new Linux kernel on your Raspberry Pi! To learn more about building a kernel for the Raspberry Pi, see the kernel building documentation.

> Interested in being one of the cool kids who contribute to the Linux kernel? If so, visit kernelnewbies.org.

Question 4: Repeat the `cyclictest` in one terminal while running the `stress-ng` "fake load" in another terminal. How does the latency of the regular Linux kernel compared to the one with the preemptible kernel? Where does the difference appear most visible (minimum, average, or maximum latencies)?

> **Note**: When you complete this lab, reset your Raspberry Pi to boot the *regular* kernel for future labs. To return to the regular Linux kernel, comment out the lines you added in `/boot/`

> `firware/config.txt` by placing a # in front of the line. You will need to reboot each time you change your kernel configuration for the new kernel to become active.

Latency Histograms

Return your Raspberry Pi to running the *regular* Linux kernel and reboot (see the note above). Use the following argument with `cyclictest` to produce a histogram of the latencies (run `stress-ng` as well to ensure there is a load):

```
sudo cyclictest -h 100 -q -i 1000 -l 100000 -p 90 --smp
```

After about a minute or so you should see a list of columns and rows of text data. The first column is the latency bin (in microseconds) and the next four columns represent the counts for that latency for each of the four CPU cores on the ARM processor in the Raspberry Pi. To capture the output, redirect the standard output to a text file as follows:

```
sudo cyclictest -h 100 -q -i 1000 -l 100000 -p 90 --smp > histogram.txt
```

Question 5: Transfer the `histogram.txt` file to a spreadsheet to plot the histogram (the first column are the bins for latency in microseconds and the other columns are for CPUs 1-4). Plot each CPU with a different color and add a legend and a title for the plot (indicating which kernel was used) and add x and y axis labels. Include the plot in your lab report. Use an appropriate horizontal scale that shows the overall shape of the histogram.

Question 6: Repeat the histogram test, but this time use the custom preemptible Linux kernel. Just as you did in the previous question, use a spreadsheet to plot the histogram (the first column are the bins for latency in microseconds and the other columns are for CPUs 1-4). Set an appropriate horizontal scale for the data as it was printed with a separate plot using a different color for each CPU plot. Add a legend and a title for the plot (indicating the kernel used) and add appropriate x and y axis labels.

Question 7: Comment on the differences between the two histograms for the different kernels. How might the preemptible kernel make a difference for time-critical tasks?

Question 8: Read about the `PREEMPT_RT` kernel patch for Linux and describe *how* this patch improves latency and makes timing more predictable. See the article, "A realtime preemption overview".

Lab 6: M2M Communications with MQTT

Purpose: to explore Machine-to-Machine communication using MQTT

Exchanging simple messages using MQTT

In this lab we will explore machine-to-machine (M2M) communications using a protocol called MQTT. MQTT is a bandwidth-efficient, lightweight protocol allowing clients to publish and subscribe data to a special "broker" server. For this course we will use the test MQTT broker server named `test.mostquitto.org`.

We will build a system to control an LED using a button by sending messages through the network using the MQTT protocol. MQTT allows communication between machines using a simple, lightweight protocol. The protocol requires an intermediate *broker server* that a client can *publish* or *subscribe* to as illustrated in the figure below.

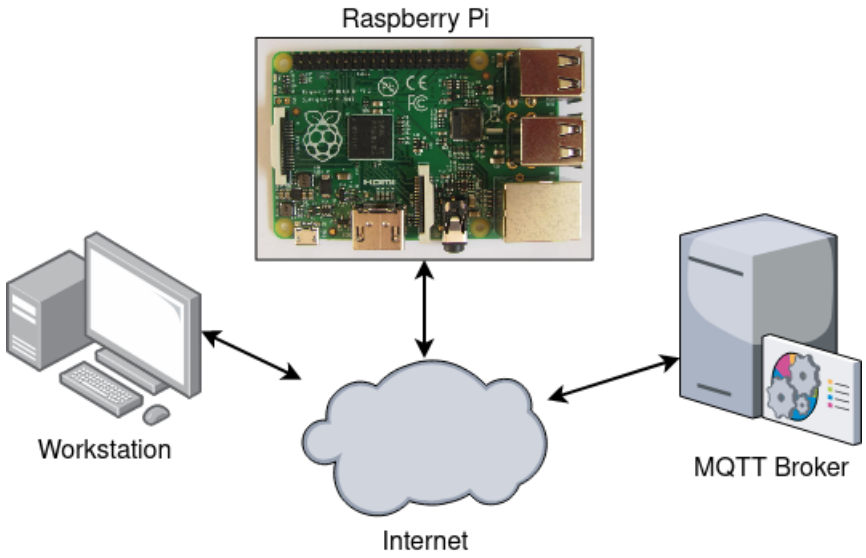


Figure 45: Raspberry Pi using in an IoT application

Before proceeding, briefly read the following background FAQ page about MQTT at mqtt.org/faq

The Eclipse Mosquitto project provides an open source MQTT message broker and tools. Open an `ssh` connection to your Raspberry Pi and install the mosquitto client tools as follows:

```
sudo apt install mosquitto-clients
```

These client tools will allow you to publish and subscribe messages to a *broker*. If you want to install them on your own computer, see https://mosquitto.org/download/ and consult the documentation.

For this lab we will make use of a public MQTT broker. Two options are:

1. mqtt.eclipse.io
2. test.mosquitto.org

> These public MQTT brokers are open and do not require usernames or passwords which comes with some security issues.

On a separate desktop or laptop workstation type the following on the command line:

```
mosquitto_sub -h test.mosquitto.org -t rpi/jcalvin
```

This command uses the MQTT protocol to subscribe with the broker server `test.mosquitto.org` to the topic `rpi/jcalvin` (you may want to replace `jcalvin` with your own username).

Next, open a shell on your local Raspberry Pi and publish a message to the same topic and broker server by entering the following:

```
mosquitto_pub -t cs326/jcalvin -m "Hello World" -h test.mosquitto.org
```

Again, replace `jcalvin` with your own username (using the same broker server you used in the last command). Note that after you hit enter you should see the `"Hello World"` message appear on the workstation. You have now successfully used MQTT to transfer a message from your Raspberry Pi to a workstation! Moreover, *you have transferred a message without needing to know the IP address of the subscriber* since messages are nicely handled by the broker server! Cool, eh?

Try sending a few more messages using different topics but add the `-d` flag (debug) to view the details of the packets that are exchanged. Note that using public MQTT brokers like `test.mosquitto.org` comes with security issues and you should not publish any sensitive information.

Publishing an event using MQTT

The following step is best done using two Raspberry Pi computers. Wire one Raspberry Pi with a push button to BCM 12 which connects the pin to ground when the switch is pressed. Power up the Raspberry Pi and connect to it using `ssh`. Install the Python MQTT library on the Raspberry Pi in a Python virtual environment as follows:

```
python3 -m venv --system-site-packages lab6
source lab6/bin/activate
pip3 install paho-mqtt
```

Next, enter the new folder created called `lab6`. Enter the following program which sends an MQTT message whenever the button is pressed.

```
# Lab 6
# This program sends an MQTT message whenever a button is pressed.

import os
from gpiozero import Button
import paho.mqtt.client as mqtt
```

```
# Constants
PORT = 1883
QOS = 0
KEEPALIVE = 60
TOPIC = 'jcalvin/button'
MESSAGE = 'Button pressed'

# Set hostname for MQTT broker
BROKER = 'test.mosquitto.org'

# Callback when a connection has been established with the MQTT broker
def on_connect(client, userdata, flags, reason_code, properties):
    if reason_code == 0:
        print(f'Connected to {BROKER} successful.')
    else:
        print(f'Connection to {BROKER} failed. Return code={rc}')

# Callback function when button is pressed
def button_callback(channel):
    global client
    (result, num) = client.publish(TOPIC, MESSAGE, qos=QOS)
    if result == 0:
        print(f'MQTT message published -> topic:{TOPIC}, message:{MESSAGE}
            ')
    else:
        print(f'PUBLISH returned error: {result}')

# define button input
button = Button(12, pull_up=True, bounce_time=0.1)

# Setup MQTT client and callbacks
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_connect=on_connect
client.connect(BROKER, PORT, KEEPALIVE)

# Detect a falling edge on input pin
button.when_pressed = button_callback

try:
    client.loop_forever()
except KeyboardInterrupt:
    client.disconnect()
print('Done')
```

Modify the code to replace `jcalvin` with your own unique username. Observe that the code uses *callback* functions to detect the button pushes and to handle MQTT events. Run the program in the Raspberry Pi terminal as follows:

```
python3 mqtt-button.py
```

Next, open a command line on anohter workstation and subscribe to the same topic on the same MQTT broker used in the program you entered on your Raspberry Pi:

```
mosquitto_sub -h tes.mosquitto -t **jcalvin**/button
```

(replace `jcalvin` as appropriate). Now, observe that when you press the button on the Pi you should see a message received on your workstation! Note once again that the program does not require the IP addresses, just the name of the broker. Test the operation of your code before proceeding to the next step.

Remote control of an LED

Next, wire up an LED circuit on a second Raspberry Pi. Refer to a previous lab to recall how to wire an LED and an appropriate value for a series current-limiting resistor.

Using `ssh`, open a terminal and install the Python MQTT library on the second Raspberry Pi in a virtual environment, just as you did earlier on the other Pi. Next, enter the following Python program named `mqtt-led.py`:

```
# Lab 6
# This program turns on an LED in response to an MQTT message.

from gpiozero import LED
import paho.mqtt.client as mqtt

# Constants
TOPIC = 'jcalvin/button'
PORT = 1883
QOS = 0
KEEPALIVE = 60

# setup LED on BCM 16
led = LED(16)

# Set hostname for MQTT broker
BROKER = 'test.mosquitto.org'

# Callback when a connection has been established with the MQTT broker
def on_connect(client, userdata, flags, reason_code, properties):
    if reason_code == 0:
        print(f'Connected to {BROKER} successful.')
    else:
        print(f'Connection to {BROKER} failed. Return code={rc}')

# Callback when client receives a message from the broker
# Use button message to turn LED on/off
def on_message(client, data, msg):
    print(f'MQTT message received -> topic:{msg.topic}, message:{msg.
        payload}')
```

```
    if msg.topic == TOPIC:
       if led.is_lit
          led.off()
       else:
          led.on()

# Setup MQTT client and callbacks
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_connect = on_connect
client.on_message = on_message

# Connect to MQTT broker and subscribe to the button topic
client.connect(BROKER, PORT, KEEPALIVE)
client.subscribe(TOPIC, qos=QOS)

try:
    client.loop_forever()
except KeyboardInterrupt:
    client.disconnect()
    print('Done')
```

Once again, modify the code to replace `jcalvin` used as the `topic` with your own unique username. Run both programs at the same time on the different Raspberry Pi's. Note that when the button is pressed on one Raspberry Pi, the LED light toggles on the other Raspberry Pi. Note that neither Raspberry Pi is aware of the IP address of the other, just the name of a common broker server. In fact, this will work even if both Raspberry Pi's are on opposite sides of the globe.

> If you only have one Raspberry Pi, you may proceed by wiring the LED on the same Raspberry Pi as the switch. In this case, the messages will be sent back and forth via the broker and the button should turn on the local LED. However, it is much less exciting to run this on one device.

Lab Questions

Question 1: What does MQTT stand for and which port does it normally use?

Question 2: Why is MQTT well-suited for the Internet of Things (IoT)?

Question 3 Using the `-d` command line switch, send a message using to the `mosquitto_sub` and `mosquitto_pub` commands. How often is a `PINGREQ` and `PINGRESP` packet sent when subscribing to a topic on a broker?

Question 4: Are your MQTT messages completely *secure and private* as they travel on the network? Consult the MQTT pages for more information.

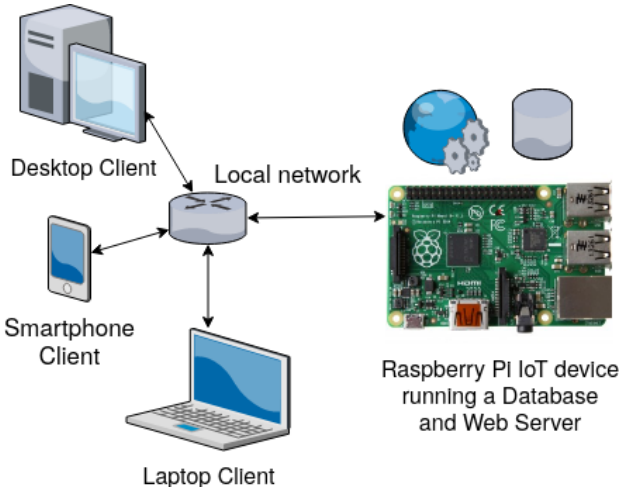
Question 5: Write a single concise program that both publishes and subscribes to the same topic and measures the roundtrip time for a published message to be received. Your program should operate with the following sequence:

Lab #7: I^{2}C with Local Web Server and Local Database

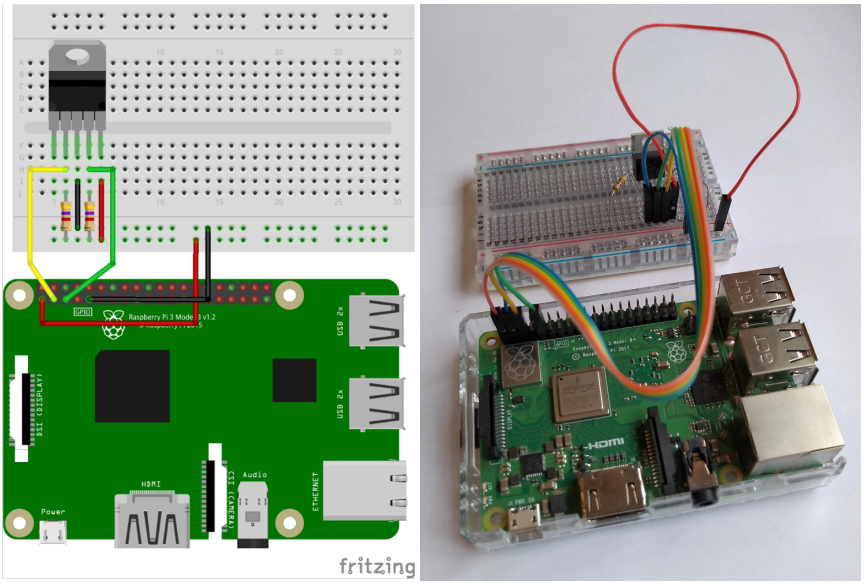
Purpose: Introduction to I^{2}C, storing sensor data in databases, and viewing with web clients.

Reading Temperature

In this lab we will use the Raspberry Pi, create a system that can measure and store temperature and display data in a helpful webpage. This lab will be restricted to running on a local network as illustrated below.



We will be reading temperature with the TC74 temperature sensor Although this sensor has only a modest precision of about ±2°C it will be adequate for the purposes of this lab. The first step will be wire up the TC74 temperature sensor. The TC74 communicates using an I^{2}C serial connection and requires two pullup resistors on the SDA a SCL lines. These pullup resistors should be placed from the SDA and SCL lines to the 3.3V supply and should have a value of 4.7kohms as shown in the photo below. Consult a GPIO pinout diagram to ensure your wiring is correct (in particular, do not confuse the 3.3V and 5V pins on the GPIO since they are next to each other). An image of the complete wiring is shown below.



Once the wiring for the Raspberry Pi GPIO pins is complete, power up your Raspberry Pi and connect using SSH. Make a new `lab8` folder as follows:

```
mkdir lab8
cd lab8
```

Next, ensure the I2C command line tools are installed to test the TC74 temperature sensor:

```
sudo apt-get install i2c-tools
```

The I2C kernel module should have been enabled in Lab 1. If it was not, enable the module as follows:

```
sudo raspi-config
```

Select Interfacing Options→I2C and then reboot by typing:

```
sudo reboot
```

Ensure your username is in the group permissions for talking to I2C devices as follows:

```
sudo usermod -a -G i2c user
```

where `user` is set to your username. Next, scan the I^2C bus for the TC74 temperature sensor to ensure it can be found:

```
/usr/sbin/i2cdetect 1
```

If the TC74 is present and wired properly, you should see a device reported at address 48hex. Next, test if it is possible to read the temperature from that address as follows:

```
/usr/sbin/i2cget -y 1 0x48 0 b
```

If everything is wired correctly and functioning, you should see a value returned from this query.

Question 1: Include a photo of your finished wiring showing your GPIO port connections and breadboard wiring.

Question 2: What value is returned from the `i2cget` command (see above)? Which base is this number displayed in and what are the units? Consult the TC74 datasheet if necessary.

Connecting to the TC74 with Python

Next, install the system management bus library for accessing the I^2C bus as follows:

```
sudo apt-get install python3-smbus
```

The SMBus enables I^2C for communications and can be used to communicate with simple devices. We can test the sensor by creating and running `temptest.py` program as follows:

```
# This program periodically reads an I2C TC74 temperature sensor and
   prints the reading.

import smbus
import time

# constants
BUS = 1             # I2C bus number
ADDRESS = 0x48      # TC74 I2C bus address
DELAY = 0.5         # delay between reads

# Connect to I2C bus
bus = smbus.SMBus(BUS)

try:
    while True:
        temp = bus.read_byte(ADDRESS)
        print(f'{temp} degrees C')
        time.sleep(DELAY)
except KeyboardInterrupt:
    bus.close()
    print('Done')
```

Once you have verified that your temperature sensor is working in Python, move on to the next part.

Storing Temperatures in an SQL database

We will store the data in a *local* database. MySQL is a common database engine, but is a little much for a simple temperature database. SQLite is a suitable, lightweight alternative which supports most SQL commands. SQLite is also convenient since it does not run as a server and stores data in a single

file which can be placed anywhere. Install the SQLite database as follows:

```
sudo apt install sqlite3
```

Setup an sqlite database in your `lab8` folder as follows:

```
sqlite3 temperature.db
```

Once SQLite launches, create a table to store temperature data by typing the following SQL command:

```
CREATE TABLE temperaturedata (datetime TEXT NOT NULL, temperature double
    NOT NULL);
```

Make sure you include a semicolon at the end of the command to indicate that your SQL command is complete. Note that SQLite does *not* have a storage class for dates and/or times so we will store the date and time as a text field. Next, check that the table was successfully created by typing:

```
sqlite> .tables
```

Question 3: Type the following `sqlite3` command and show the output from this command. What does this output represent?

```
pragma table_info('temperaturedata');
```

Next, quit SQLite3 by typing the following at the prompt:

```
sqlite> .quit
```

Your database should now be set for recording date/time and temperature data! Since we will be time-stamping our temperature data, we need to ensure the time and date settings are correct on your Raspberry Pi. Set the time zone as follows:

```
sudo dpkg-reconfigure tzdata
```

Next, enter the following Python program called `logtemp.py`:

```
# This program stores an I2C TC74 temperature sensor reading every 10
    seconds to an SQLite database.

import smbus
import sqlite3
import time
import sys
import signal

# Constants
BUS = 1             # I2C bus number
```

```
ADDRESS = 0x48      # TC74 I2C bus address
FILENAME = 'temperature.db'   # SQLite filename
TABLE = 'temperaturedata'     # SQLite table name
PERIOD = 10.0                 # Sample period (seconds)

def timer_handler(signum, frame):
    ''' Periodic timer signal handler
    '''
    global bus
    global db
    global cursor
    temp = bus.read_byte(ADDRESS)    # Read TC74 sensor
    # Insert data into database
    sqlcmd = f"INSERT INTO {TABLE} VALUES (datetime('now','localtime'),{
        temp})"
    cursor.execute(sqlcmd)
    db.commit()

# Connect to I2C bus
bus = smbus.SMBus(BUS)

# Connect to the database
db = sqlite3.connect(FILENAME)
cursor = db.cursor()

# Setup signal to call handler every PERIOD seconds
signal.signal(signal.SIGALRM, timer_handler)
signal.setitimer(signal.ITIMER_REAL, 1, PERIOD)

# Continuously loop blocking on signals
try:
    while True:
        signal.pause()      # block on signal
except KeyboardInterrupt:
    bus.close()
    db.close()
    print('Done')
```

This program runs a periodic task to read the temperature every 10 seconds and stores it in an SQLite database.

> The code also uses global variables for expediency and convenience, but global variables are normally not recommended. One could implement a single class that encapsulates the signal handler along with the variables to avoid using globals.

Question 4: What *might* happen to the SQL database and available disk space if this code were to continue running for years (or perhaps decades)?

In order to address this issue identified above, add the following line of code *after* the data is inserted

into the database and before the `db.commit()` method is called:

```
sqlcmd=f"DELETE FROM {TABLE} WHERE datetime < datetime('now','localtime
    ','-1 hour')"
cursor.execute(sqlcmd)
```

Note that this line of code will need its own separate call to `cursor.execute(sqlcmd)` in order for SQLite to execute the query.

Question 5: What does the above SQL command do? Consult the SQLite3 documentation if need be at: https://www.sqlite.org/docs.html.

Running a local webserver

Next, we want to create a webpage that displays the data from the database. For this lab, we will install a *local* webserver running on the Raspberry Pi. For this lab we will use lighttpd, a simple, lightweight web server that should be adequate for our purposes. We will also use PHP for server-side scripting. To install these packages, type the following:

```
sudo apt update
sudo apt -y install lighttpd
```

Test the web server by creating a simple web file as follows:

```
sudo nano /var/www/html/index.html
```

Type the message "hello world" into the file and then save it. Point your browser to `http://a.b.c.d` where `a.b.c.d` is the IP address of your Raspberry Pi, and a "hello world" web page should appear.

Question 6: Which network *port number* is `lighttpd` listening on for web requests? You can determine the port number by using `netstat` and typing the following at the command line:

```
sudo netstat -ltnp
```

The ports used will be listed in rows. Identify the row(s) listing `lighttpd` as the program name and note the number after the colon in the "Local address" column. (1 point)

Putting it all together

Next we will create a webpage using PHP to access the SQLite database to retrieve the temperature data and then use a Javascript plotting library to create a chart of the temperature data. Enable PHP as follows:

```
sudo apt -y install php-fpm php-cgi php-sqlite3
sudo lighty-enable-mod fastcgi
sudo lighty-enable-mod fastcgi-php
sudo service lighttpd restart
```

Next, set the permissions for the web folder so your user can edit web files as follows:

```
sudo chown -R www-data:www-data /var/www
sudo chmod -R 775 /var/www
sudo usermod -a -G www-data jcalvin
```

where `jcalvin` is the username on your Raspberry Pi. Reboot the Raspberry Pi and the `lighttpd` webserver should now be running. Next, enter the code for `chart.php`:

```
<html>
<head>
   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
<h2>Raspberry Pi Temperature Local Data</h2>
<div id='chart_div'></div>
<?PHP
# Open local SQLite database
try {
   $db = new SQLite3('/home/pi/temperature.db');
}
catch (Exception $exception) {
  echo '<p>There was an error connecting to the database!</p>';
}
?>

<script>
var data = [{
   x: [
<?PHP
     # Use PHP to query database and build JavaScript array
     $query = 'SELECT * FROM temperaturedata';
     $result = $db->query($query) or die('Query failed');
     while ($row = $result->fetchArray()) {
        echo "    '" . $row['datetime'] . "',\n";
     }
?>
   ],
   y: [
<?PHP
     while ($row = $result->fetchArray()) {
        echo "    " . $row['temperature'] . ",\n";
     }
?>
   ],
   type: 'scatter'
}];

var layout = {
    xaxis: { title: 'Add appropriate y axis label here' },
    yaxis: { title: 'Add appropriate x axis label here' }
```

```
};

Plotly.newPlot('chart_div', data, layout );
</script>
</body>
</html>
```

Observe that this file contains a combination of HTML, PHP and Javascript code (both server-side and client-side code). Place the file inside the `/var/www/html` folder.

Note that the path for the SQLite database in this program must match the path you used for the database in your python program. If vscode has trouble editing this file remotely you can also edit it directly using:

```
sudo nano /var/www/html/chart.php
```

This code uses PHP to retrieve the temperature data from the SQLite database and creates a Javascript data structure that is used by `plotly.js`, a handy Javascript library, to create a nifty chart. Note that there are a wide variety of nifty open source Javascript chart libraries that you may wish to explore. For more information about `plotly.js`, visit: https://plot.ly/javascript/

Next, you will need to set the permissions of the Sqlite3 database file so that the Lighttpd server has permissions to open and read the temperature data. You will also need to ensure your web server has execute privileges in your home folder. This can be done by navigating to the folder with your database file and typing the following commands:

```
chgrp www-data /home/jcalvin
chmod g+x /home/jcalvin
sudo chgrp www-data temperature.db
chmod 644 temperature.db
```

where `jcalvin` is your username. Next, start running your Python temperature logging program in the background as follows:

```
nohup python3 logtemp.py &
```

Finally, steer your workstation/laptop browser to the following URL and observe your webpage plotting temperature data:

```
http://a.b.c.d/chart.php
```

where `a.b.c.d` is the IP address of your Raspberry Pi.

If your webpage fails to load you will need to use some of the debugging features available in your browser. For example, with the Chrome browser, press `*crtl-shift-i` to show the debugging tools as well as the Javascript console.

Try varying the temperature by touching on your sensor and observing if it measures a rise in temperature.

Question 7: Note that your web client should be *only* accessible on your local network. What security issues might arise if the web server on your Raspberry Pi (or other IoT devices) were to be made visible on the internet?

Question 8: Include a picture of the chart in the webpage (you can download an image by clicking the download option on the menu at the top of the chart). Ensure proper axes labels are shown.

Some fine tuning

Notice that `lighttpd` runs several processes. You can see all the processes that are running by using the following command:

```
sudo service lighttpd status
```

Note the number of tasks and the list of processes in the output listed. Note that `lighttpd` runs 1 process, but several other processes are spawned relating to php-cgi. Every additional process requires memory and resources, which we like to minimize when working with embedded systems. We are not expecting high web traffic to our Raspberry pi, so we can reduce the number of PHP processes. To do this, edit the `lighttpd` configuration file as follows:

```
sudo nano /etc/lighttpd/conf-enabled/15-fastcgi-php.conf
```

Edit the file to reduce the `PHP_FCGI_CHILDREN` to `0`. Save the file and restart the web server as follows: sudo service `lighttpd` restart

Use the `service status` command from above and note how many PHP processes are running now. Test and make sure your webpage can still be viewed in your browser.

To disable the `lighttpd` web service from starting up on the next reboot boot, type:

```
sudo systemctl disable lighttpd
```

Question 9: How many processes are running that belong to `www-data` *before* and *after* the changes are made to the `lighttpd` configuration file?

Question 10: What are the disadvantages and risks of running a web server and database on a local IoT device?

Lab #8: MQTT Security

To demonstrate some security issues, we will create an application that generates MQTT traffic and then see if we can "snoop" the contents. The following Python program generates a simple periodic MQTT message containing the text 'hello world' as follows:

```
# Send periodic MQTT traffic.

import paho.mqtt.client as mqtt
import time
import os

# Constants
BROKER = ''     # broker hostname
PORT = 1883     # default MQTT port
QOS = 0
DELAY = 5.0
TOPIC = 'test/topic'

# Callback when connecting to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print(f'Connected to {BROKER}')
    else:
        print(f'Connection to {BROKER} failed. Return code={rc}')
        os._exit(1)

# Setup MQTT client and callbacks
client = mqtt.Client()
client.on_connect = on_connect

# Connect to MQTT broker
client.connect(BROKER, PORT, 60)
client.loop_start()

# Continuously publish message
try:
    while True:
        print('Sending MQTT message')
        client.publish(TOPIC, 'hello world')
        time.sleep(DELAY)

except KeyboardInterrupt:
    print('Done')
    client.disconnect()
```

Be sure to set the BROKER constant to one of the public MQTT brokers such as mqtt.eclipseprojects.io or test.mosquitto.org.

Run the program in a terminal on your Raspberry Pi as follows:

```
python3 mqtt-hello.py
```

Subscribe to the MQTT topic on another computer or laptop using the command line as follows (modify the broker hostname and topic as necessary):

```
mosquitto_sub -d -h mqtt.eclipseprojects.io -p 1883 -t 'cs326/jcalvin'
```

Using `ssh`, open a second terminal on your Raspberry Pi and install the **tcpdump** utility. The **tcpdump** tool will enable us to "snoop" the network packets. To install the tool, type:

```
sudo apt install tcpdump
```

Once the tool is installed, use it to monitor your MQTT traffic (once again, modify the hostname of your MQTT broker if you are using a different one):

```
sudo tcpdump -XA host mqtt.eclipseprojects.io
```

Note that we are using command line options to display the packet contents in ASCII and to limit our monitoring *only* to traffic exchanged with the MQTT broker. Each time an MQTT packet is sent, you should see a network packet "dumped" to the console.

> **NOTE**: Running **tcpdump** or similar tools on servers is normally not "polite" since it allows you to view network traffic, some of which is "sent in the clear." In this case, since the Raspberry Pi is your server, and the traffic is your traffic, you may give yourself permission to do this. However, in general, **tcpdump** is among a set of tools that needs to be used ethically and judiciously. Remember the famous movie line from Spiderman's uncle: *with great power comes great responsibility.*[a]
>
> [a] As cited by Peter Parker's uncle Ben in *Spiderman*.

Question 1: Answer the following questions:

a) Show the contents of one of the MQTT packets being sent between your Raspberry Pi and the MQTT broker.

b) Are the contents of the MQTT packets *visible* i.e.. can you see the topic and message contents inside the payload? What does this imply about security of the communications?

> If the topic and message contents are visible, then such a system is vulnerable to snooping, spoofing, or a "man-in-the-middle attack." As the number of connected devices grows and the amount of data being transferred increases, security continues to be a crucial topic!

Next, let's examine whether the data on the broker is private and secure. Type the following command to subscribe to *all* the topics on a public broker server. Let's see what we can see:

```
mosquitto_sub -h mqtt.eclipseprojects.io -p 1883 -t '#' -v
```

Question 2: Describe what you see when you enter the command above. What does this imply about data sent to open MQTT brokers? (1 point)

These security issues can be dealt with using *authentication* and *encryption*. Without **encryption**, your packets are like postcards sent in the mail – nothing prevents the postman or anyone along the way from reading your data!

The Transport Security Layer (TSL) provides a secure communication channel between a client and a server. TSL provides a cryptographic protocol to create a secure connection between a client and the broker. Insecure MQTT traffic is sent to port 1883 "in the clear" whereas port 8883 is used for secure MQTT connections. MQTT brokers normally provide an X509 certificate (typically issued by a trusted authority) that clients use to verify the identity of the server. Verifying the identity can help prevent so-called "man-in-the-middle" attacks.

> **Note:** using secure MQTT on embedded systems can present an issue since encryption incurs overhead and requires more CPU resources and power. This can be a problem for resource-constrained or battery-powered devices. However, the Raspberry Pi is more than capable of handling encryption.

Authentication is a security mechanism which verifies whether a person or device is who they say they are and can be used to grant access. The MQTT broker includes support for username and password uthentication. Ideally, authentication should only be used over an encrypted connection to ensure that the username and password itself are not sent in the clear.

Question 3:

a) Using the NIST National Vulnerability Database, search for recent MQTT vulnerabilities, give **one** of the vulnerability identifiers and describe it briefly in your own words.

b) How many MQTT vulnerabilities were there reported in the database for the current year (based on published date)?

Troubleshooting Tips for the Raspberry Pi

The Raspberry Pi includes a green activity LED that can aid in troubleshooting. Normally this LED flashes to indicate it is reading from the SD card. However, if an error is encountered during booting, the LED will flash an error code. The "blink codes" for the Raspberry Pi 4 comprise a series of long

and short flashes as summarized below. Note that the blink codes for the Raspberry Pi 4 and 5 are different than for prior versions of the Raspberry Pi.

Long flashes	Short Flashes	Status
0	3	Generic failure to boot
0	4	start*.elf not found
0	7	Kernel image not found
0	8	SDRAM failure
0	9	Insufficient SDRAM
0	10	In HALT state
2	1	Partition not FAT
2	2	Failed to read from partition
2	3	Extended partition not FAT
2	4	File signature/hash mismatch
4	4	Unsupported board type
4	5	Fatal firmware error
4	6	Power failure type A
4	7	Power failure type B

Closing Note

We opened by citing the remark by the respected computer scientists Edsgar Dijkstra that "computer science is no more about computers than astronomy is about telescopes." While this is true, it is certainly enhanced through widespread access to cost-effective, nifty, and powerful computing platforms designed for learning and exploring. The Raspberry Pi provides one such platform, and this guide provides ample evidence to the sweeping topics in computer science that one can explore using this modest platform. Kudos to the folks at the Raspberry Pi Foundation!