Projects

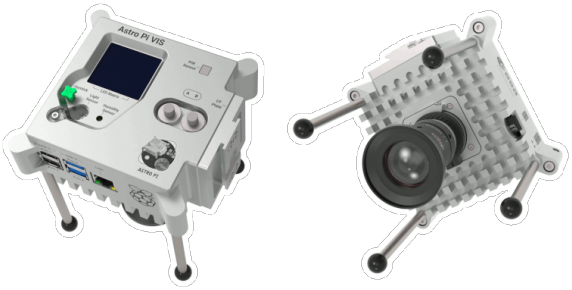
Astro Pi Mission Space Lab Phase 2 guide

Guide for participants of the Mission Space Lab Astro Pi competition 2022-23

Step 1 Introduction

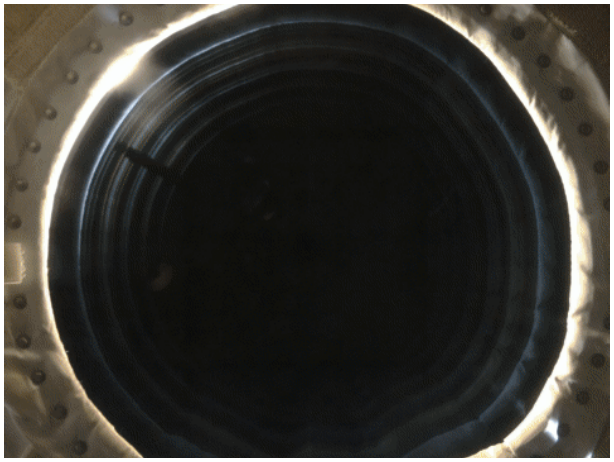
In this project, you will learn how to turn your Phase 1 experiment idea into a real experiment that can run on your own Astro Pi hardware and on the International Space Station (ISS).

> Don't know about Mission Space Lab? Don't panic! Head over to the Astro Pi website (https://astro-pi.org/mission-space-lab/) for more information and to sign up for next year.



What you will make

You will build your very own Astro Pi and write a Python program to gather data to support or reject your experiment hypothesis. You will learn how to test your program to give it the best chance of running successfully aboard the International Space Station (ISS).



What you will learn

By reading this guide you will learn how to:

1. Build an Astro Pi and how to use it
2. Plan your experiment
3. Write a Python program using the libraries available aboard the ISS, including how to:
    - Save data captured from the sensors and the camera
    - Run your program for 3 hours
    - Find the location of the ISS
    - Improve your program with error handling and logging
4. Test and review your work to give your team the best chance of running your experiment on the ISS

Guidelines and pre-submission checklist

In order to have your program run aboard the ISS, it's really important that you read the guidelines on the Astro Pi website (https://astro-pi.org/mission-space-lab/guidelines) and that you look over the Phase 2 program checklist (https://astro-pi.org/mission-space-lab/guidelines/program-checklist) before submitting your work. Bear in mind that there are some differences in what you are allowed to do depending on if your experiment theme is 'Life On Earth' or 'Life In Space'.

Step 2 Kit assembly

In this step we are going to build an Astro Pi using the official kit you received. Before starting the assembly, let's familiarise ourselves with what's in the box.

What's in the box?

It may be small but your kit box really packs a punch! Your official Astro Pi kit includes:

1x Raspberry Pi 4 (https://www.raspberrypi.com/products/raspberry-pi-4-model-b/) 4GB

1x Power supply unit

1x 16GB microSD card

1x HDMI cable

1x Sense HAT (V2)

Bag of spacers, screws, and a GPIO header

If your experiment involves using the camera, you will also have:

1x HQ Camera

1x 6mm Camera lens

and if you will be using infrared photography, your kit will also include:

1x Red optical filter

1x Allen key (1.5mm)

If your experiment involves detecting movement, your kit will include:

A passive infrared (PIR) sensor

3x F-F jumper wires

1x Tall header pins

If your experiment involves machine-learning at runtime, you will also have the following in your box:

A Coral ML accelerator (https://coral.ai/products/accelerator)

1x USB-C to USB-A cable

If you want to, you can make a 3D-printed flight case (https://projects.raspberrypi.org/en/projects/astro-pi-flight-case-mk2). However, you don't need one to take part in Mission Space Lab.

Method

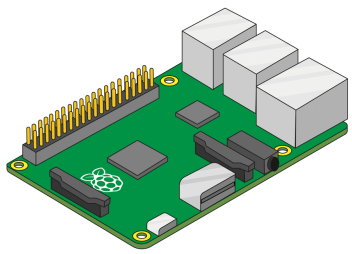
Before getting started, familiarise yourself with what's in the box and unpack everything. Make a note of whether your box includes the camera module, a Coral TPU machine learning accelerator (Coral ML accelerator), or a PIR sensor using the checkboxes below. This will show or hide the relevant assembly instructions accordingly.

- My box contains a camera ☑
- My box contains an infrared filter ☑
- My box contains a PIR sensor ☑
- My box contains a Coral ML accelerator ☑

Place the Raspberry Pi 4 on a flat surface. Make sure it is turned off with nothing plugged in. ☑

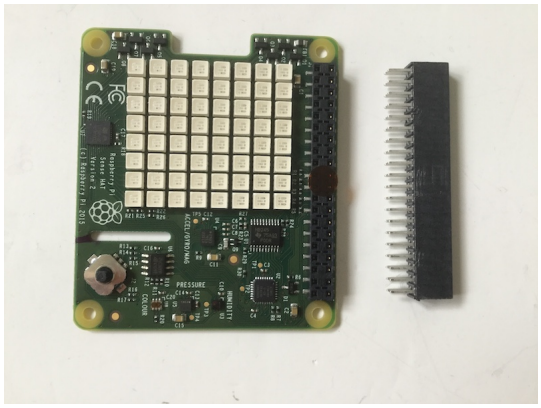
Attaching the Sense Hat

Let's start by assembling the Sense HAT.

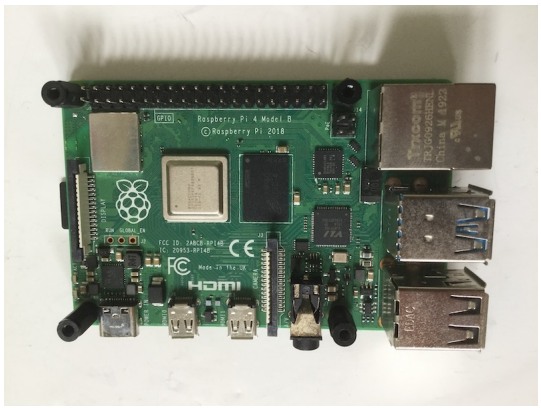


Find the Sense HAT and the small bag that comes with it that includes some screws and spacers, as well as a regular GPIO header.

Remove any stickers on the top of the Sense HAT.

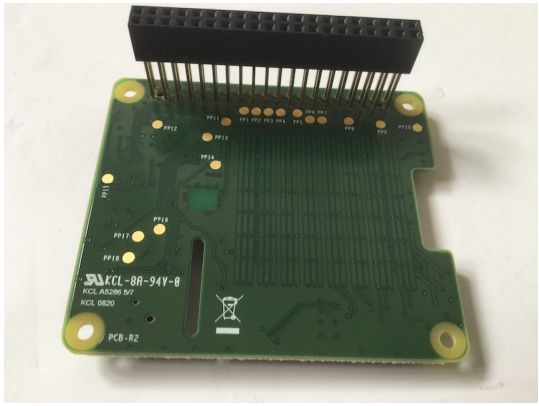


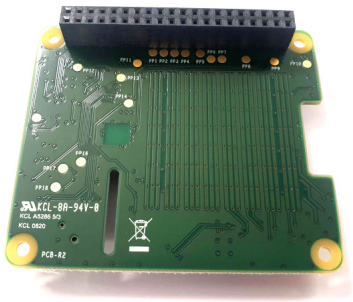
Take the black hexagonal spacer columns from the small bag that comes with the Sense HAT. Use the accompanying screws to connect them to the bottom of the Raspberry Pi 4.



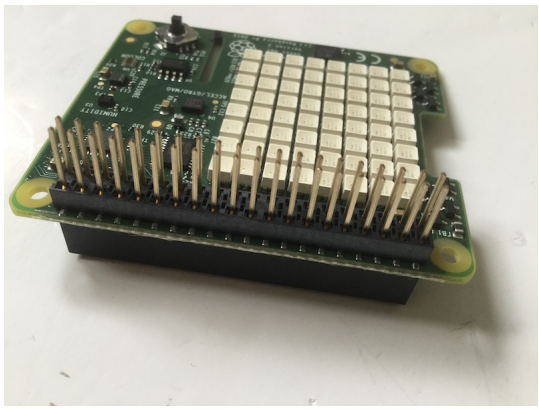
In a separate bag, locate the tall GPIO header. We will use this header instead of the regular header to allow enough space for the PIR sensor.

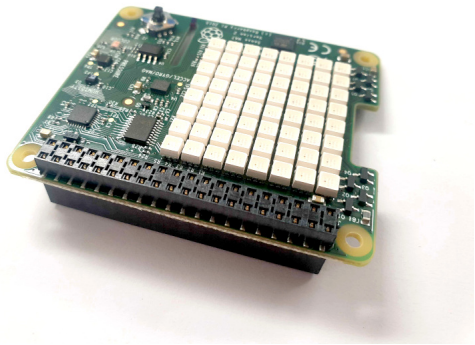
Line up the header with the corresponding holes on the bottom of the Sense HAT.

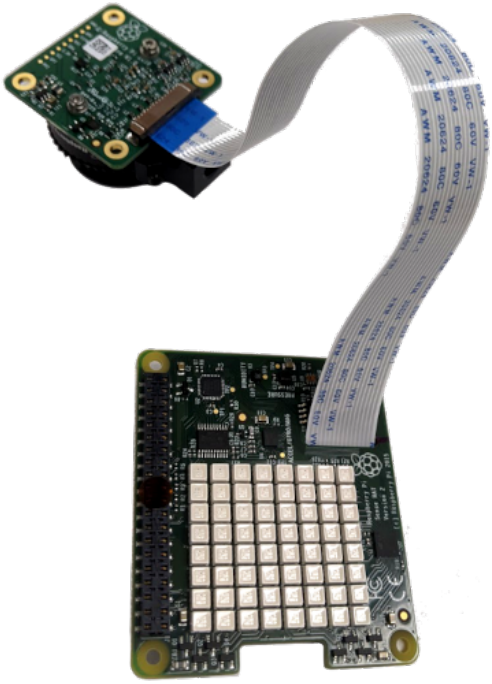




Push the header all the way through, making sure none of the pins are obstructed and that they are lined up correctly so that they do not become bent.



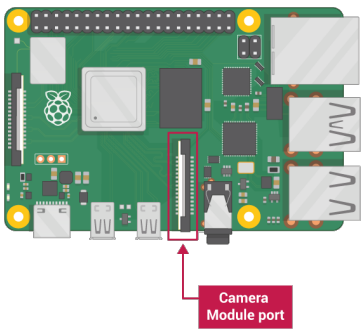


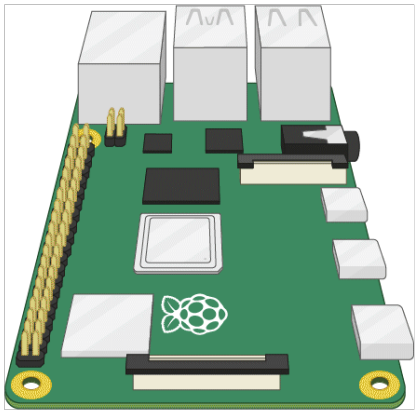


With the Raspberry Pi High Quality Camera and connector cable unboxed, take the connector cable and feed it through the gap in the Sense HAT. The silver side of the connector cable should face the LED matrix and not the blue side.

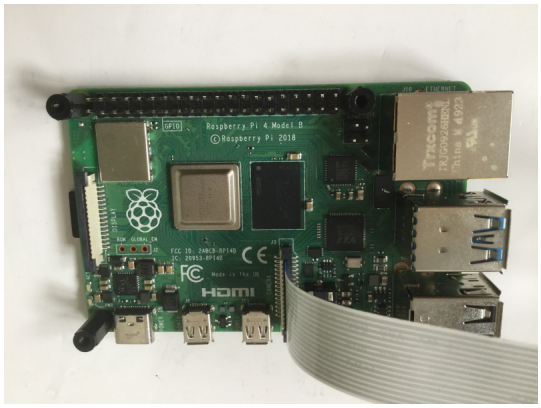
Here is a video of the process:

Find the CSI (Camera Serial Interface) port on the Raspberry Pi and gently pull the edges of the port's plastic cap.



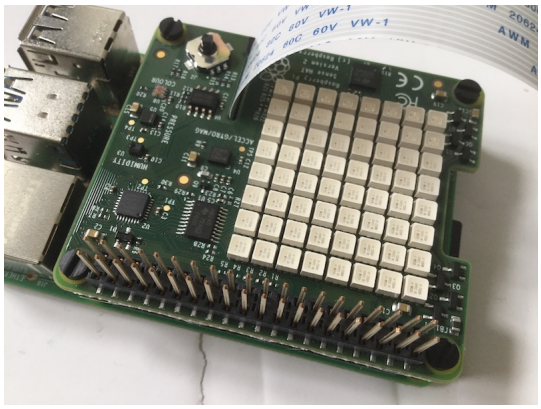


Insert the camera ribbon cable into the Raspberry Pi CSI socket, making sure that the silver side (and not the blue side) is facing the LED matrix. There should be 1 or 2mm of silver still remaining when the cable has been put in correctly.
Then, push the plastic clip back into place.



Place the Sense HAT onto the Raspberry Pi and ensure the 40 GPIO pins line up with the corresponding holes in the header.

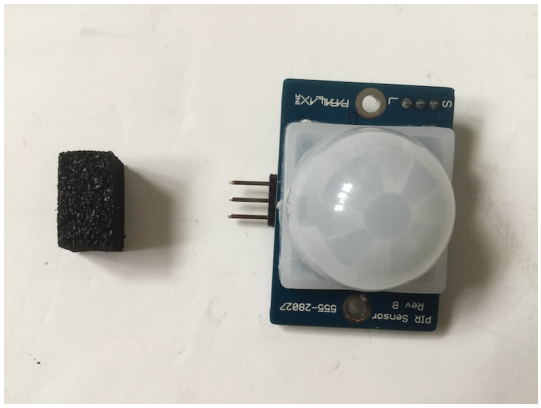
Use the four remaining black screws to secure the Sense HAT stack to the spacers.



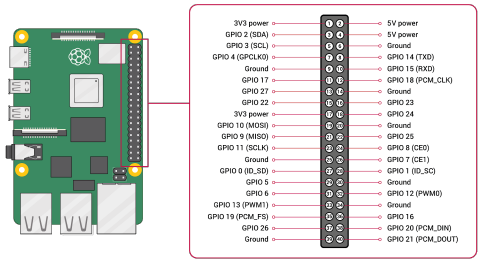
You've finished putting the Sense HAT on. On to the next step.

Passive infrared (PIR) sensor

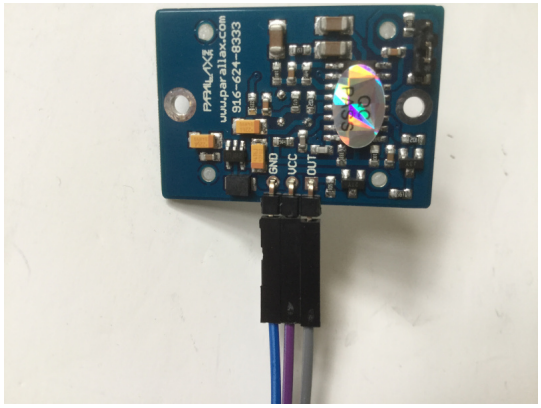
Take the PIR and remove the foam pin protector block.



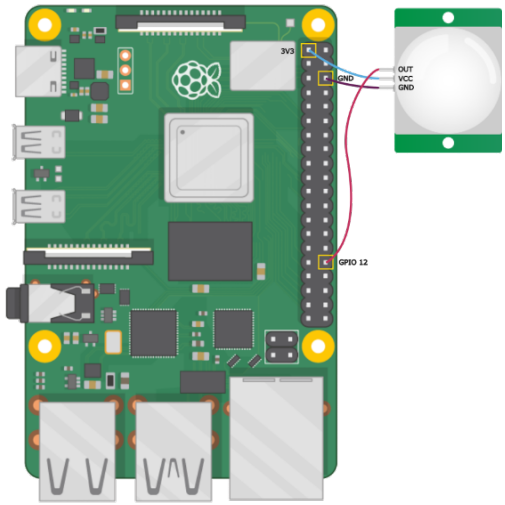
Take a moment to familiarise yourself with the layout of the Raspberry Pi pins (https://pinout.xyz). Notice that the odd-numbered pins are on the left-hand side, the even-numbered pins are on the right-hand side, and that the pin number increases by 2 each time we go down a row.



Take a moment to look at the PIR sensor. Do you see the labels GND, VCC, and OUT?

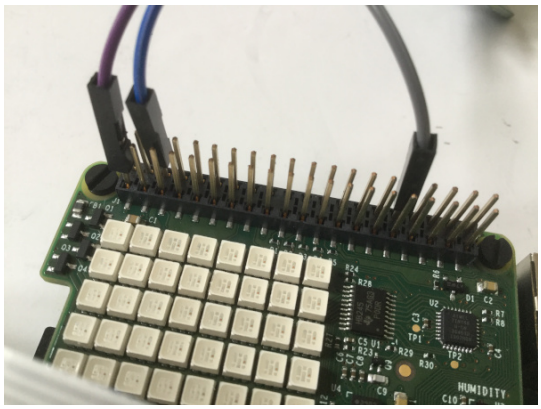


Now we are going to connect each pin on the PIR sensor to an appropriate pin on the Raspberry Pi using the three F-F jumper wires provided in the kit.



- Connect the VCC pin on the PIR sensor to pin 1 (3V3) on the Raspberry Pi
- Connect the GND pin on the PIR sensor to pin 6 (GND) on the Raspberry Pi
- Connect the OUT pin on the PIR sensor to pin 32 (GPIO 12)

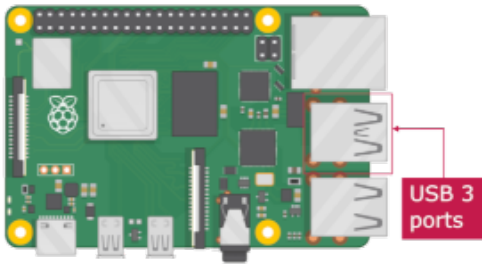
Note: Your jumper wires may be a different colour to the ones in the photos.



Here is a video of the PIR setup:

You've just finished assembling the PIR sensor. Great work!

Setting up the Coral ML accelerator



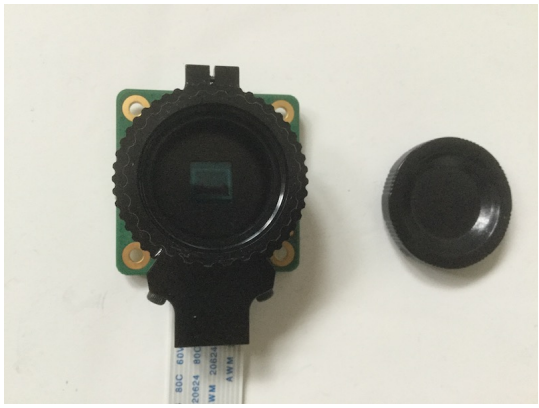
The Coral ML accelerator requires no assembly — simply locate the USB-C cable and plug it into the accelerator,

and then plug the other end into any of the blue USB (USB 3) ports.

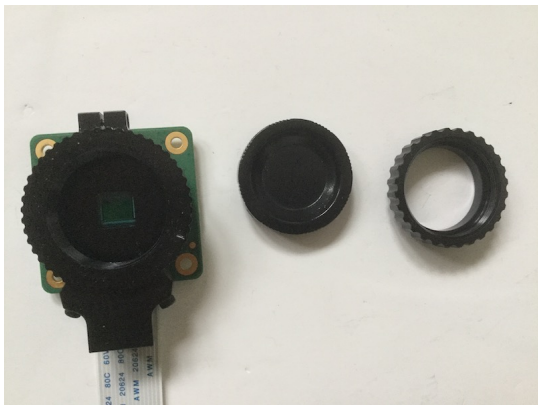
Camera assembly

Find the high-quality camera board and check that the back focus ring is screwed all the way in.

Remove the protective cap from the high-quality camera.



Remove the C/CS adapter ring from the high-quality camera.



Converting the camera to use infrared

If your Life on Earth experiment requires an infrared-sensitive (IR-sensitive) camera — for example, you are doing a NDVI (Normalised Difference Vegetation Index) experiment — then you will need to convert your camera using the steps below.

Note: If you are programming a Life in Space experiment, or your Life on Earth experiment requires photos to be taken in the visible light spectrum only, then please don't convert your high-quality camera sensor as you can't reverse/undo it later.

How does the infrared camera work?

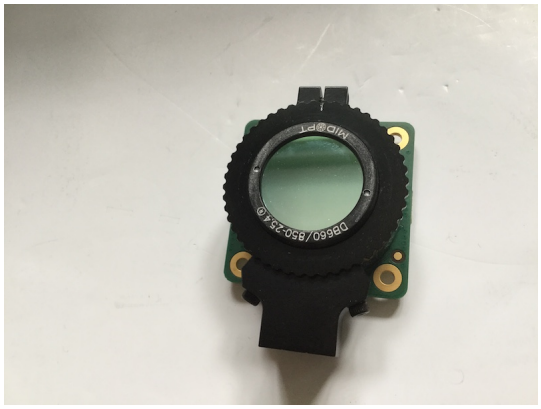
The high-quality camera sensor can detect infrared (IR) light. However, the sensor housing contains an IR filter, which is used to greatly reduce the camera's sensitivity to IR light. This is so that the images captured by the high-quality camera sensor look the same as what we see with our eyes (which are not sensitive to IR light). By removing this filter, we allow the IR light to pass through along with visible light.

In the next steps we will replace the built-in filter with a separate red filter which allows only reflected red light (660nm) and reflected near-infrared light (850nm) through to the sensor. See our NDVI project (https://projects.raspberrypi.org/en/projects/astropi-ndvi) for more information.

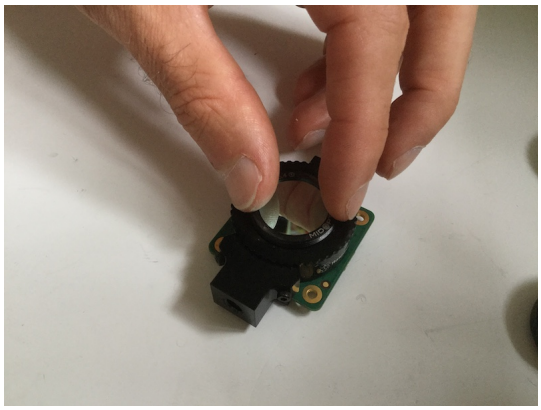
Remove the built-in infrared filter from the high-quality camera by following the instructions here (https://www.raspberrypi.org/documentation/accessories/camera.html#raspberry-pi-hq-camera-filter-removal). You will need to use the 1.5mm Allen key.

To help you even more, here is a video of this process:

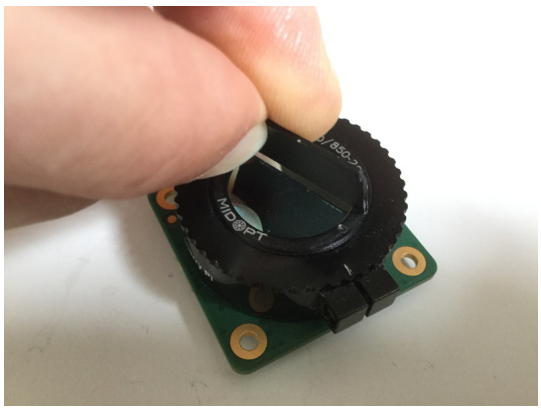
Take the MIDOPT filter and sit it onto the hole in the centre of the high-quality camera sensor.



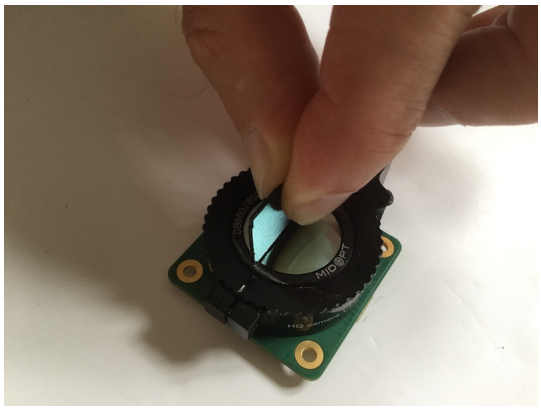
Gently start turning the filter clockwise using just your fingers, so that the filter screws down into the high-quality camera sensor. Take care not to touch the glass part of the lens and leave greasy fingerprints.



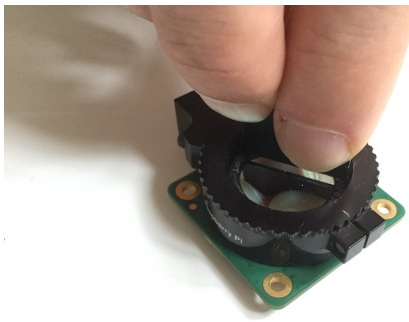
Take the tool provided with the filter and line up the two knobbly bits at each end with the corresponding dimples in the filter. If you have a 3D printer, you might like to print a handle (https://www.google.com/url?q=https://www.tinkercad.com/things/31aXmXN5y1a?sharecode%3DzKyKeuZ0fRedhVhzYGubLbc-m-ffZI-uq_EGv07lS9M&sa=D&source=docs&ust=1668524338706246&usg=AOvVaw2rUbx_xg6m_2a4R2SkQ4_K) for the tool to make it easier to grip. One of these handles has been printed and sent to the ISS for the astronauts to use when completing the task — but it isn't required.



Continue gently turning the filter using the tool. Take care not to touch the glass part of the lens with the tool — it will scratch it.

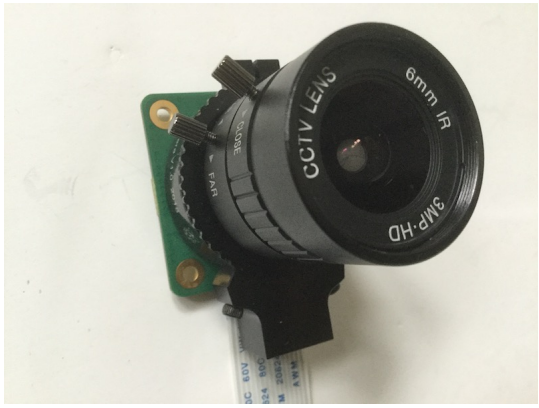


You should start to feel increasing resistance as the filter gets lower. After about nine full turns, the filter should be as low as it can go and you won't be able to turn it any further. Be careful not to over-tighten.



Remove the cap from the narrower end of the 6mm lens.

Screw the 6mm lens onto the high-quality camera sensor.

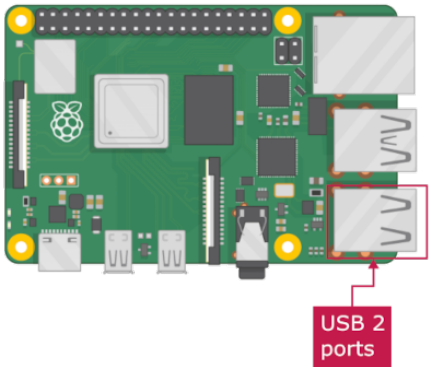


Congratulations! Now you have a complete infrared-sensitive camera.

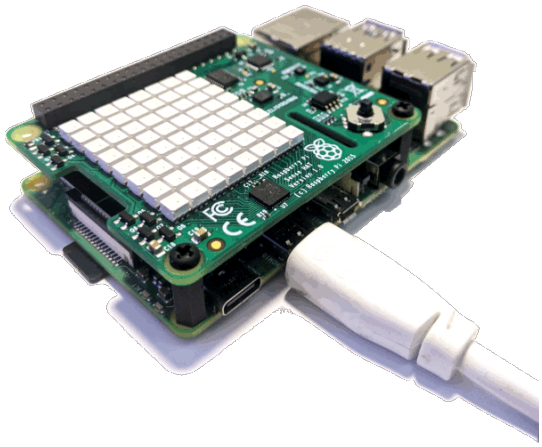
Final assembly steps

You're almost done. Finish your Astro Pi with these steps:

Connect your keyboard and mouse into the two (black) USB 2 ports.

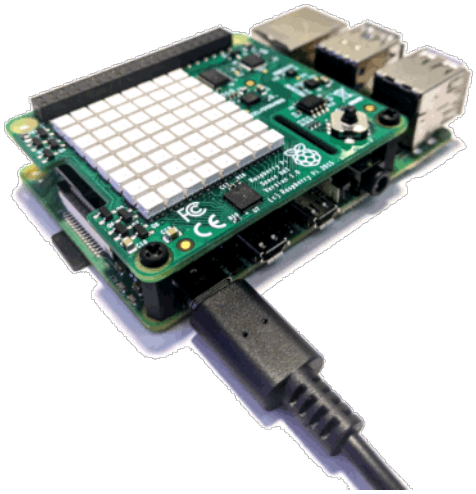


Connect the smaller micro-HDMI cable head into the HDMI 1 port of the Raspberry Pi and connect the other end into your screen.



The SD card comes in a large adapter, so remove it from the adapter and insert it into the SD port on the bottom of the Raspberry Pi — make sure you put it in the right way.

Finally, connect the USB-C power cable into the Raspberry Pi and watch it boot up.

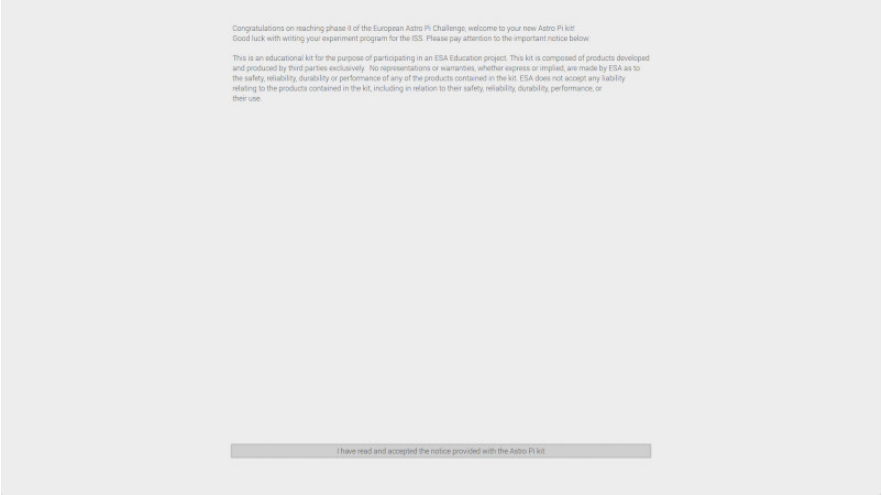


Congratulations, you have made an Astro Pi! 🚀
Continue to the next stage to learn about the Operating System, the Kit OS.

Step 3 The Kit OS

Setting up your OS

When you power on your Astro Pi for the first time you will be invited to accept the ESA Licence Agreement, as shown below:



Troubleshooting - ESA screen does not appear

The initial agreement page will only display on the HDMI1 output of the Raspberry Pi. If you only see the background image make sure you have connected your screen to HDMI1.

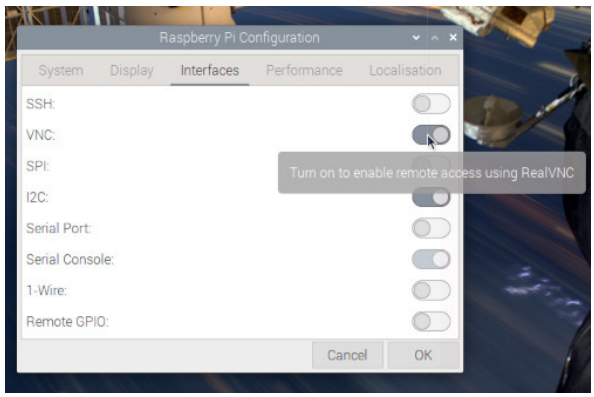
Once you have accepted the licence agreement, you will be asked to create a new username and password, to set the system time and language settings, and to connect to a WiFi network. For more help on setting up your Raspberry Pi, take a look at this guide (https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started/4).

After you have completed these steps, the Astro Pi will reboot itself and you will be ready to start using the Kit OS. 🚀

Optional setup - Accessing the Kit OS remotely

> Note: Enabling VNC may expose your Raspberry Pi to security threats and could allow a hacker to access your machine. Make sure you understand the risks before enabling it.

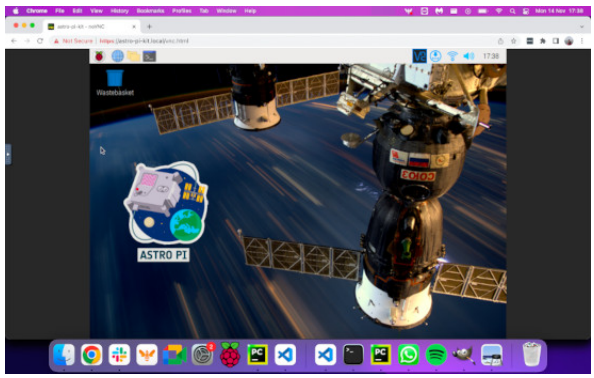
The Kit OS can be configured so that you are able to connect to it from another desktop. To do this, you will need to install a compatible VNC client (https://www.realvnc.com/en/connect/download/viewer/) on the other desktop, and then enable VNC on the Astro Pi itself by entering the commands below or via the graphical Raspberry Pi Configuration tool (Menu > Preferences > Raspberry Pi Configuration > Interfaces).

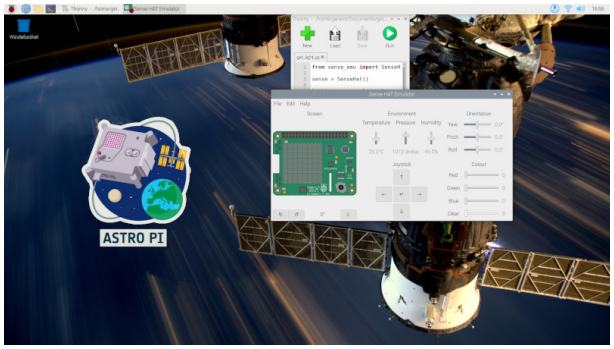


The default password to connect via VNC is `raspberry` but you are highly encouraged to change this using `vncpasswd -service` before enabling VNC.

```
sudo raspi-config nonint do_vnc 0
sudo systemctl enable novnc
sudo systemctl start novnc
sudo systemctl unmask avahi-daemon
sudo systemctl enable avahi-daemon
sudo systemctl start avahi-daemon
```

You can also connect to the Kit OS using just a browser, albeit less securely. On a machine that is connected to the same network as your Astro Pi kit, open up a browser and type `https://astro-pi-kit.local/vnc.html` in the address bar.
You will have to tell your browser to trust your unique Astro Pi SSL certificate to continue (e.g. on Chrome, type `thisisunsafe` while the browser tab has focus), but once you have done so you will be lead to the noVNC connection page. Click on the `Connect` button, enter the password you set in `vncpasswd`, and you should see the Flight OS desktop in your browser.





Note: Be careful about installing new software on your Kit OS, as it will increase the chances of your experiment not running successfully on the Astro Pis aboard the ISS. But, if you have accidentally installed anything, you can always redownload and reinstall the Kit OS and start again.

Optional - Downloading a fresh version of the Kit OS

If you want to create additional SD cards to use for Astro Pi, or if you accidentally installed something, you can download the Kit OS image file (https://downloads.raspberrypi.org/AstroPi_latest) used in the official kits. After downloading, you can use any software tool to write the image file to your own SD card. See this guide (https://www.raspberrypi.org/documentation/installation/installing-images/) for instructions on how to do this.

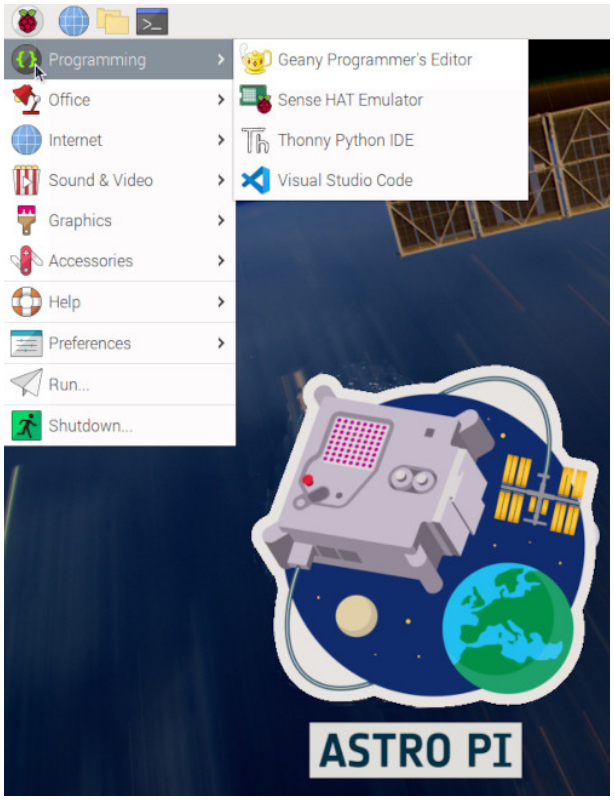
Now it's time to take a tour of the OS.

A tour of the OS

The Kit OS is a special version of the Raspberry Pi Desktop OS (Bullseye 32 bit) that contains the same programs and libraries as the Astro Pis aboard the International Space Station. It includes everything you need to develop and test your experiment, and more!
Making sure that your program runs successfully in this environment is the best way to ensure that your experiment can run on the Astro Pis on the ISS.

Applications

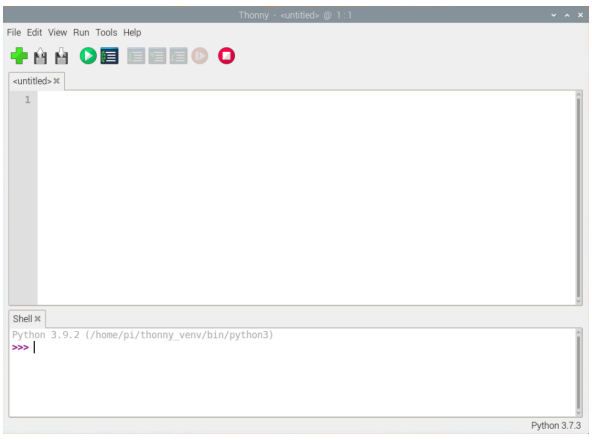
There are a lot of applications that are installed in the Kit OS — take a moment to explore the start menu in the top left-hand corner. How many apps can you see?



Familiarise yourself with the Kit OS by finding the apps below.

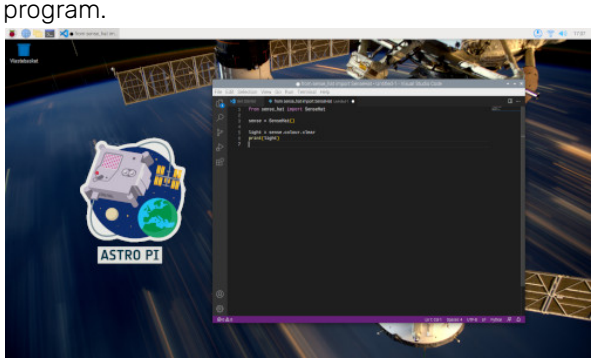
Thonny

Thonny is a Python IDE (Integrated Development Environment (https://simple.wikipedia.org/wiki/Integrated_development_environment)), and is easy to use to get started writing your experiment program.



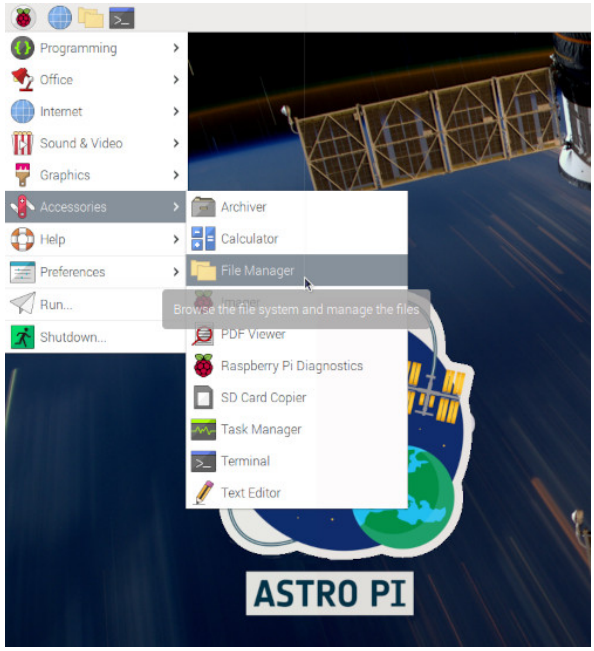
Visual Studio Code

Visual Studio Code is a text editor (https://simple.wikipedia.org/wiki/Text_editor) and can be used to write programs in many programming languages, including Python. It isn't quite as beginner-friendly as Thonny, but you could also use this to write your program.



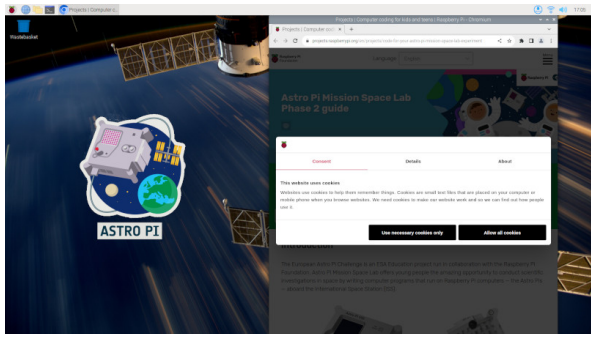
File Manager

The File Manager is used to … manage files. Every file on your computer can be accessed using this application. For more information, check out this page (https://projects.raspberrypi.org/en/projects/raspberry-pi-using/7).



Chromium

Chromium is the default web browser on Raspberry Pi OS — use it to get help preparing your experiment.



Terminal

Terminal is an application that you can use to run other programs and automate almost any task on your computer. As an example to get started, type `python3` to open up a python interpreter. Check out this page (https://projects.raspberrypi.org/en/projects/raspberry-pi-using/8) for more information about Terminal.

Python and Python libraries

What is Python?

Python is the general purpose programming language that you will use to run your experiment. Python programs are instructions formed of special words telling the computer what to do. Using Python, you can do almost anything you can imagine. The application that runs a Python program is called the 'interpreter'. If this is your first time with Python, check out this project (https://projects.raspberrypi.org/en/codeclub/python-module-1).

The Kit OS comes with Python version 3.9.2 and many additional Python libraries pre-installed to help you write the program for your experiment. The same Python and library versions have been installed on the Astro Pis aboard the ISS, so running your program on the Kit OS should behave similar as it would in space.

> Because the ISS is a secured environment, these are the only libraries that you will be allowed to use in your experiment if it runs on the Astro Pis on board. Please contact us (mailto:enquiries@astro-pi.org) if you think anything is missing or have any suggestions.

Python libraries

What is a Python library?

A 'library' or 'package' is a collection of program 'blocks' that solve a specific problem or achieve a specific goal. You use them by 'importing' them into your Python program: `import numpy as np`, for example. Using libraries avoids having to solve the same problem twice, and makes the process of programming easier.

Take some time now to familiarise yourself with the libraries available to use on the Astro Pis, which are listed below. Each library is briefly described and is presented with an example of how to use it, together with links to external documentation where you can learn more. Remember to bookmark this page for later.

Skyfield

Usage

Skyfield is an astronomy package that computes the positions of stars, planets, and satellites in orbit around the Earth.

In the How to find the location of the ISS (6) section you can find out how to use Skyfield to obtain the position of the International Space Station above the Earth and how to determine whether the ISS is sunlit.

Documentation

- rhodesmill.org/skyfield (https://rhodesmill.org/skyfield/)

picamera

The Python library for controlling the Raspberry Pi Camera Module is `picamera`. To get started, check out this project (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/4) for a handy walkthrough of how to use it.

Usage

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()
camera.resolution = (2592, 1944)

for i in range(3*60):
    camera.capture(f'image_{i:03d}.jpg')  # Take a picture every minute for 3 hours
    sleep(60)
```

Documentation

- picamera.readthedocs.io (https://picamera.readthedocs.io)

colorzero

`colorzero` is a colour manipulation library that aims to be simple to use.

Usage

`colorzero` makes it easy to transition between two colours:

```
from colorzero import Color
from sense_hat import SenseHat
from time import sleep

sense = SenseHat()

start = Color('yellow')
end = Color('cyan')

# Slowly transition the Sense HAT from the `start` to the `end` color
for color in start.gradient(end, steps=100):
    sense.clear(color.rgb_bytes)
    sleep(0.1)
```

Documentation

- colorzero.readthedocs.io (https://colorzero.readthedocs.io)

GPIO Zero

GPIO Zero is a simple but powerful GPIO (General Purpose Input Output) library. Most of its functionality is restricted aboard the ISS — for example, the only pin you are allowed to access is GPIO pin 12, where the motion sensor is connected. However, some of its other features can be handy in your experiment, such as the internal device `CPUTemperature`.

Usage

Compare the Raspberry Pi's CPU temperature to the Sense HAT's temperature reading:

```
from sense_hat import SenseHat
from gpiozero import CPUTemperature

sense = SenseHat()
cpu = CPUTemperature()

while True:
    print(f'CPU: {cpu.temperature}')
    print(f'Sense HAT: {sense.temperature}')
```

Documentation

- gpiozero.readthedocs.io (https://gpiozero.readthedocs.io)

GDAL

The Geospatial Data Abstraction Library (GDAL) is an open-source, cross-platform set of libraries and low-level tools for working with geospatial data in many formats. For most purposes on Astro Pi, you may want to look at using `earthpy` or `geopandas` instead which are simpler to use.

Documentation

- GDAL Cookbook (http://pcjericks.github.io/py-gdalogr-cookbook/index.html)
- pypi.org/project/GDAL (https://pypi.org/project/GDAL/)

GeoPandas

GeoPandas is a package to make working with geospatial data in Python easier. GeoPandas extends the datatypes used by the `pandas` package to allow spatial operations on geometric types. The examples gallery (https://geopandas.org/en/stable/gallery/index.html#) shows how to create a variety of graphs using geospatial data.

Usage

Start exploring a dataset interactively:

```
import geopandas

path_to_data = geopandas.datasets.get_path("nybb")
gdf = geopandas.read_file(path_to_data)
gdf = gdf.set_index("BoroName")
gdf["area"] = gdf.area
gdf.explore("area", legend=False)
```

Documentation

- https://geopandas.org/en/stable/docs.html (https://geopandas.org/en/stable/docs.html)

EarthPy

EarthPy is a package that makes it easier to plot and work with spatial raster and vector data.
The examples gallery (https://earthpy.readthedocs.io/en/latest/gallery_vignettes/index.html) includes examples on how to plot bands of satellite imagery and how to calculate NDVI.

Usage

```
import earthpy.plot as ep
import numpy as np

arr = np.random.randint(4, size=(3, 5, 5))
ep.plot_bands(arr)
```

Documentation

- https://earthpy.readthedocs.io/en/latest/index.html (https://earthpy.readthedocs.io/en/latest/index.html)

NumPy

`numpy` is a general purpose array processing library designed to efficiently manipulate large multidimensional arrays (e.g. matrixes) of arbitrary records without sacrificing too much speed for small multidimensional arrays.

Usage

`numpy` is particularly handy for capturing camera data for manipulation:

```
from picamera import PiCamera
from time import sleep
import numpy as np

camera = PiCamera()

camera.resolution = (320, 240)
camera.framerate = 24
output = np.empty((240, 320, 3), dtype=np.uint8)
sleep(2)
camera.capture(output, 'rgb')
```

Documentation

- docs.scipy.org/doc (https://docs.scipy.org/doc/)

SciPy

SciPy is a free open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimisation, linear algebra, integration, interpolation, special functions, FFT (Fast Fourrier Transform), signal and image processing, ODE (Ordinary Differential Equations) solvers, and other tasks common in science and engineering. You may need to use this library to solve a particular equation.

Documentation

- docs.scipy.org/doc (https://docs.scipy.org/doc/)

TensorFlow Lite and PyCoral

TensorFlow Lite and the PyCoral library can be used to use or re-train existing machine learning models for inference. The latter is built on top of TensorFlow Lite but has a simpler, higher-level interface and allows you to easily use the Coral ML Accelerator (Edge TPU). Note that Tensorflow (as opposed to TensorFlow Lite) is not supported by the Kit OS because Tensorflow requires a 64-bit operating system. You may want to use these libraries to create object classifiers, for example. For more information, see the Machine Learning and computer vision (7) section.

Documentation

- Tensorflow Lite (https://www.tensorflow.org/lite/api_docs/python/tf/lite)
- PyCoral (https://coral.ai/docs/edgetpu/tflite-python/)

pandas

`pandas` is an open-source library providing high-performance, easy-to-use data structures and data analysis tools. You may want to use it when you are analysing the results of your program test runs.

Usage

```
import pandas as pd

df = pd.read_csv("my_test_data.csv")
df.describe()
```

Documentation

- pandas.pydata.org (https://pandas.pydata.org/)

logzero

`logzero` is a library used to make logging easier. Logs are records of what happened while a program was running, and can be really useful for debugging.

Usage

Logs are categorised into different levels according to severity. By using the various levels appropriately, you will be able to tune the amount of information you get about your program according to your debugging needs.

```
from logzero import logger

logger.debug("hello")
logger.info("info")
logger.warning("warning")
logger.error("error")
```

Documentation

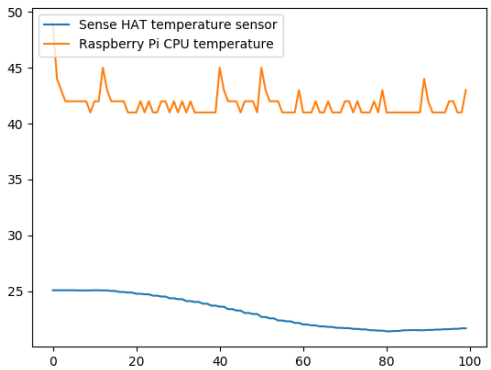
- logzero.readthedocs.io (https://logzero.readthedocs.io/en/latest/)

Matplotlib

`matplotlib` is a 2D plotting library that produces publication-quality figures in a variety of hard copy formats and interactive environments. You may want to use it to analyse the results of your test runs.

Usage

```
from sense_hat import SenseHat
from gpiozero import CPUTemperature
import matplotlib.pyplot as plt
from time import sleep

sense = SenseHat()
cpu = CPUTemperature()

st, ct = [], []
for i in range(100):
    st.append(sense.temperature)
    ct.append(cpu.temperature)
    sleep(1)

plt.plot(st)
plt.plot(ct)
plt.legend(['Sense HAT temperature sensor', 'Raspberry Pi CPU temperature'], loc='upper left')
plt.show()
```



Documentation

- matplotlib.org (https://matplotlib.org/)

Pillow

Pillow is an image processing library. It provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Documentation

- pillow.readthedocs.io (https://pillow.readthedocs.io/)

OpenCV

`opencv` is an open-source computer vision library. The Astro Pi units specifically have the `opencv-contrib-python-headless` package installed, which includes all of `opencv` plus additional modules (listed in the opencv docs (https://docs.opencv.org/master/)), and excludes all GUI functionality. You may want to use OpenCV for edge detection (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/3), for example.

Documentation

- docs.opencv.org (https://docs.opencv.org/4.4.0/)

exif

`exif` allows you to read and modify image EXIF metadata using Python. You may want to use it to embed GPS data into any images you take, or to analyse photos taken aboard the ISS (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/1).

Documentation

- pypi.org/project/exif (https://pypi.org/project/exif/)

scikit-learn

`scikit-learn` is a set of simple and efficient tools for data mining and data analysis that are accessible to everybody, and reusable in various contexts. It's designed to interoperate with `numpy`, `scipy`, and `matplotlib`.

Documentation

- scikit-learn.org (https://scikit-learn.org)

scikit-image

`scikit-image` is an open-source image processing library. It includes algorithms for segmentation, geometric transformations, colour space manipulation, analysis, filtering, morphology, feature detection, and more.

Documentation

- scikit-image.org (https://scikit-image.org/)

reverse-geocoder

`reverse-geocoder` takes a latitude/longitude coordinate and returns the nearest town/city.

Usage

When used with `skyfield`, `reverse-geocoder` can determine where the ISS currently is:

```
import reverse_geocoder
from orbit import ISS

coordinates = ISS.coordinates()
coordinate_pair = (
    coordinates.latitude.degrees,
    coordinates.longitude.degrees)
location = reverse_geocoder.search(coordinate_pair)
print(location)
```

This output shows the ISS is currently over Hamilton, New York:

```
[OrderedDict([
    ('lat', '42.82701'),
    ('lon', '-75.54462'),
    ('name', 'Hamilton'),
    ('admin1', 'New York'),
    ('admin2', 'Madison County'),
    ('cc', 'US')
])]
```

Documentation

- github.com/thampiman/reverse-geocoder (https://github.com/thampiman/reverse-geocoder)

sense_hat

The `sense_hat` library is the main library used to collect data using the Astro Pi Sense HAT. Look at this project (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat) to get started.

Usage

You can log the humidity to the display using the code below:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message(str(sense.get_humidity()))
```

Documentation

- https://sense-hat.readthedocs.io/en/latest/ (https://sense-hat.readthedocs.io/en/latest/)
- Additional documentation for the colour sensor (https://gist.github.com/boukeas/e46ab3558b33d2f554192a9b4265b85f)

pisense

`pisense` is an alternative interface to the Raspberry Pi Sense HAT. The major difference to `sense_hat` is that in `pisense` the various components of the Sense HAT (the screen, the joystick, the environment sensors, etc.) are each represented by separate classes that can be used individually or by the main class that comprises them all.

The screen has a few more tricks including support for any fonts that PIL supports, representation as a numpy array (which makes scrolling by assigning slices of a larger image very simple), and several rudimentary animation functions. The joystick, and all sensors, have an iterable interface too.

Usage

```
from pisense import SenseHAT, array
from colorzero import Color

hat = SenseHAT(emulate=True)
hat.screen.clear()

B = Color('black')
r = Color('red')
w = Color('white')
b = Color('blue')

black_line = [B, B, B, B, B, B, B, B]
flag_line = [B, b, b, w, w, r, r, B]
flag = array(black_line * 2 + flag_line * 4 + black_line * 2)

hat.screen.fade_to(flag)
```

Documentation

- pisense.readthedocs.io (https://pisense.readthedocs.io/en/latest/)

Next steps

This concludes our brief tour of the Kit OS. As you've seen, there are a lot of applications and libraries to become familiar with. With a good plan of action you will be soon using them in your own experiment. In the next step, you will learn about how to plan Phase 2 of your experiment.

Step 4 Planning your work

Teams have often reflected in previous years that they wished they had spent more time in the planning and design stages of their experiment. Being organised should make the most of the opportunity to run your code aboard the ISS, and in this section we've put together some team activities for you to try. Before you start, though, check out this video from last year:

Five questions to get started

To get started planning your Phase 2, you could meet together as a team and try to answer the questions below. This should help you decide how you are going to work together and set some expectations. Make sure everyone gets a turn to speak.

Meet together and answer the five questions below.

- How are you going to make decisions as a team? Will you all have a say, or will you elect someone to take charge?
- How can you utilise everyone's strengths? What is everyone good at, and how can you help each other?
- What does everyone want to learn? Don't just do what you're good at — try something new!
- How much time you do you have? Decide when and where you will meet, and how often.
- How will you work together? Will you work online or mostly in person?

Timeline

As a team, decide on the ideal date you would like to finish writing your program. Make sure you have enough time to test it fully before submitting it, and to check it against the requirements checklist (https://astro-pi.org/mission-space-lab/guidelines/program-checklist).

Create a timeline for your project that includes your ideal finish date and the date at which you will start checking and testing your program.

Identify your measurements

Once you have answered the five questions above and created a rough timeline for your work, you may want to note down the measurements and data you will need to test your hypothesis. If you need help with this, take a look at this project (https://projects.raspberrypi.org/en/projects/experiment-design) and in particular the define your measurements (https://projects.raspberrypi.org/en/projects/experiment-design/2) page.

Write down the measurements that you will need to take in order to test your hypothesis.

> This step will really help you to make sure the outputs of your program are useful for later stages of Mission Space Lab, and that you remain focused throughout Phase 2!

Limitations of the sensors

Be aware of the limitations of the sensors and their constraints. In particular, be mindful that the temperature and humidity sensors are affected more by the temperature of the CPU than anything else. If you would like to take readings of the ISS environment, it's advisable that you test the temperature and humidity readings in a controlled (known) environment and come up with a strategy to compensate for this limitation.

Do some research

There are lots of resources on how to use the Astro Pi hardware on the projects.raspberrypi.org (https://projects.raspberrypi.org) website. To make the most of the Mission Space Lab opportunity, we recommend you complete either of the pathways below, depending on the nature of your experiment:

- Life in Space pathway (https://projects.raspberrypi.org/en/pathways/life-in-space)
- Life on Earth pathway (https://projects.raspberrypi.org/en/pathways/life-on-earth)

Pick a project pathway to look at and create a plan to study it as a team.

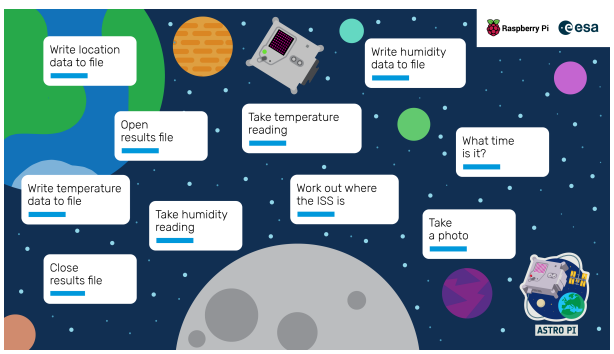
You don't need to restrict yourself to the projects site though! You could use a search engine to try and find examples of other teams working on a similar idea, to find more data for a machine learning experiment, or to engage with real scientific literature.

Check out the Resources section (11) for more inspiration.

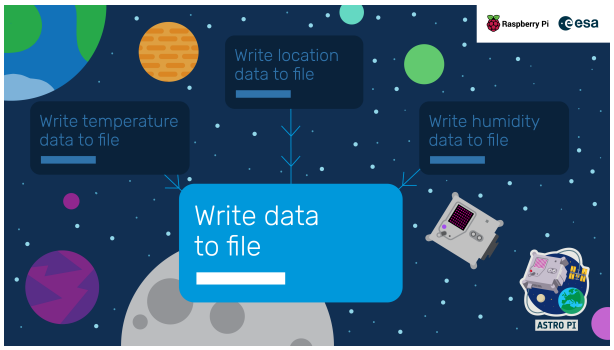
Work out the key program tasks visually

Once you have a good idea of what you are trying to achieve with your program, the next step is to work out the tasks that the program will need to do. We recommend doing this visually using pen and paper, or a whiteboard, or online using a tool like Miro (https://miro.com).

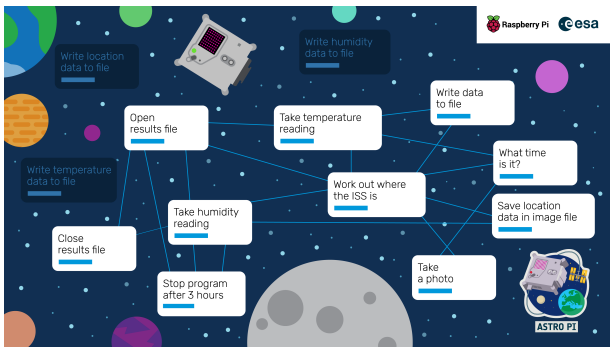
List all of the key tasks that your program will need to perform. You don't need to worry about the order or the actual functions and commands at this stage — just note down the specific things that need to be achieved like in the image below.



Have a closer look at each task and think about whether it can be split into smaller subtasks. Check to see if there are any actions that can be combined with one another, or if there are any tasks that need to be repeated.



Try to put everything in a logical order, using lines to connect the various tasks. It will start to get messy, but you will probably discover that there are some obvious repeated tasks — these tasks are probably going to be written as functions that you will reuse.



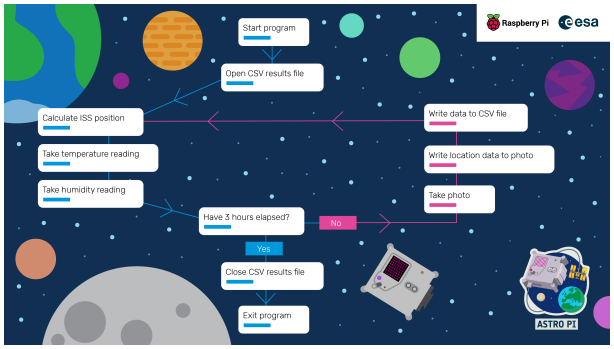
If you have been using a whiteboard or pen and paper, don't forget to take a photo of your work to save it for later!

Draw a flow chart

Using your notes from the previous step, try and refine the tasks into a flow chart (https://simple.wikipedia.org/wiki/Flow_chart), which is a diagram of all of a program's tasks, in the right order, but doesn't contain any actual programming language commands. To do this you can follow these steps:

- Identify the natural order of the tasks and try and use arrows to connect them in a sequence to create a 'flow' along the page, zig-zagging a bit like a snakes and ladders board (https://upload.wikimedia.org/wikipedia/en/b/ba/Cnl03.jpg)
- Identify any decisions that the program needs to make and check that all outcomes are catered for
- Include a 'start' and 'end' block to make it very clear where the program begins and finishes

A flow chart for a typical experiment might look something like this:



Create a flow chart of your program using the help above.

Consider 'What if' scenarios

An important aspect of programming and design is making sure you are ready for when things go wrong. Most experiments will have a main loop that runs repeatedly over the 3-hour period. An unexpected error encountered in this loop could be disastrous if it causes the program to stop or stall and prevent further data collection. So, think of some 'what if' scenarios. For example, if you're reading data from a sensor, what will happen if it gives you an unexpected result? Will your program cope with this? How are you dealing with hardware errors?

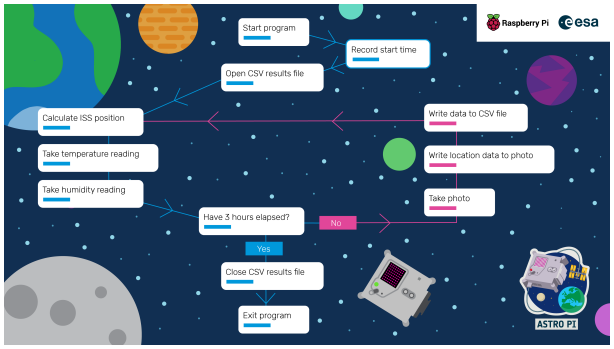
Identify points in your flow chart where errors might occur and add new blocks to cater for them.

Assign tasks to members of the team

After finishing your flow chart, it is a good idea to review it all together. Double-check that there is no unnecessary work and that you're satisfied that the final result will help you test your hypothesis. Once you're satisfied, assign tasks by following the steps below.

- Give descriptive names to each task block
- Assign responsibility for each block to different members of the team, keeping in mind their experience and desires
- Remember that someone needs to be responsible for the scaffold of the final program that will contain the various function calls in the right order

You may want to use a project management tool to keep track of your tasks. Something as simple as Google Keep (https://www.google.com/keep/) could work, or you could use a Trello (https://www.trello.com) or Monday.com (https://www.monday.com) board.



Assign tasks to each member of your team.

Whatever method you choose to track your work, make sure to schedule a time to meet regularly to discuss progress and work through any major challenges as a group. It can be useful to update your pseudocode flow diagram to reflect any changes that your team realises are necessary as they write the actual program.

Get coding!

Now that you have a much better idea of what your experiment program is going to do, it's time to get coding. Continue reading for specific guidance on how to write your program.

Step 5 Writing your program - recording data and images

In this section, we are going to start writing your experiment program and learn how to record data using the sensors and camera. By the end of this page, you will be able to collect measurements and images to support your hypothesis — neat!

To get started, create a file called `main.py`. In this file we will write all our functions to take measurements and capture images.

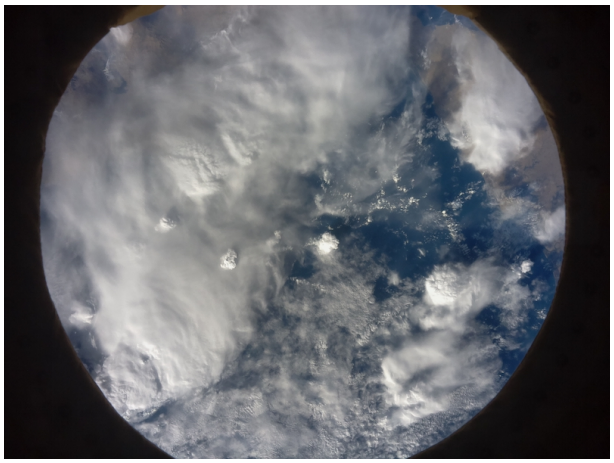
Create a file called `main.py`

Using multiple Python source files

Ideally, all of your code should be contained within `main.py`. Don't worry if your experiment is complex and you need to break down your code into individual modules though. Additional files are allowed.

Recording images using the camera

> Check the Mission Space Lab (MSL) Guidelines (https://astro-pi.org/mission-space-lab/guidelines/program-checklist) to make sure you are allowed to use the camera before reading this section.



The Astro Pis on the ISS are equipped with a high-quality camera each so that you can take pictures of Earth — something normally only astronauts can do. Take some time now to read over the Getting started with the Camera Module (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/) project to learn how to use the camera.

Read through the Getting started with the Camera Module (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/) project.

Choosing camera settings

> If your team has completed Phase 2 of Mission Space Lab before, please note that the AstroPis may not have exactly the same view as last year as they may be deployed in a different window on the ISS.

As you will have noticed by reading the Getting started with the Camera Module (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/) project, the picamera library (https://picamera.readthedocs.io/en/release-1.13/) allows you to set a huge selection of camera settings. When deciding which settings to use, or whether to leave the settings on automatic, be mindful that you will be taking pictures in a variety of conditions with a range of weather, landscapes, and lighting. Nighttime photography using the Astro Pi's Camera Module is particularly difficult: the ISS is travelling so fast that a long exposure time is needed, and this makes the photos come out very blurry in low light conditions.

What camera and lenses are used on the real Astro Pis?

The camera sensor in the ESA kit is the same high-quality camera as the one found in the new Astro Pis on the ISS. You can read the documentation about the HQ camera here (https://www.raspberrypi.org/documentation/hardware/camera/), and find a lot of detailed technical information in the relevant section of the picamera library documentation (https://picamera.readthedocs.io/en/latest/fov.html#camera-hardware).

All Life On Earth experiments will use a 5mm Kowa Lens (https://lenses.kowa-usa.com/10mp-jc10m-series/397-lm5jc10m.html), which has a large aperture range. All Life in Space experiments will use the 6mm Raspberry Pi lens (https://uk.farnell.com/raspberry-pi/rpi-6mm-lens/rpi-6mm-wide-angle-lens/dp/3381607). In all experiments the focal length will be set to infinity due to the altitude.

The Astro Pi sensors

The Astro Pi includes a range of easy to use sensors that are ready to use for your experiments:

- Accelerometer
- Gyroscope
- Magnetometer
- Temperature sensor
- Humidity sensor
- Barometric pressure sensor
- Light and colour sensor

All of these sensors are accessed using the Sense HAT, which provides a simple way to take measurements from the environment. Take some time to look at the Getting started with the Sense HAT (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/7) and Sense HAT data logger (https://projects.raspberrypi.org/en/projects/sense-hat-data-logger/1) projects to learn how to log measurements from these sensors to a CSV file.

There is also a PIR (passive infrared) motion sensor on the Astro Pis on the ISS, which can be accessed using the `gpiozero` library to create a `MotionSensor` object attached specifically to GPIO pin 12:

```
from gpiozero import MotionSensor

print("Inititating motion detection")
pir = MotionSensor(pin=12)
pir.wait_for_motion()
print("Motion detected")
pir.wait_for_no_motion()
```

You can also use the `gpiozero` to read the CPU temperature:

```
from gpiozero import CPUTemperature

cpu = CPUTemperature()
print(cpu.temperature)
```

For more details about the interface for the Sense HAT and gpiozero libraries, make sure to look at the documentation — this is a really useful resource.

Sense HAT and gpiozero documentation

The Sense HAT documentation (https://pythonhosted.org/sense-hat/) contains sections on how to retrieve data from the environmental sensors (https://pythonhosted.org/sense-hat/api/#environmental-sensors) (temperature, humidity, pressure) and the Inertial Measurement Unit (IMU) (https://pythonhosted.org/sense-hat/api/#imu-sensor) (acceleration, orientiation). Additional documentation is available for interacting with the light and colour sensor (https://gist.github.com/boukeas/e46ab3558b33d2f554192a9b4265b85f). You can also explore the wide range of Sense HAT projects (https://projects.raspberrypi.org/en/projects?hardware%5B%5D=sense-hat) available from the Raspberry Pi Foundation.

For the PIR sensor, check out the gpiozero documentation (https://gpiozero.readthedocs.io/en/stable/api_input.html#motionsensor-d-sun-pir), which shows the different ways in which you can interact with the sensor.

Where to save your data

Your program is going to be stored in a different location when it is deployed on the ISS, so it's really important to avoid using absolute file paths. Use the code below to work out which folder the `main.py` file is currently stored in, which is called the `base_folder`:

```
from pathlib import Path

base_folder = Path(__file__).parent.resolve()
```

Then you can save your data into a file underneath this `base_folder`:

```
data_file = base_folder / "data.csv"

for i in range(10):
    with open(data_file, "w", buffering=1) as f:
        f.write(f"Some data: {i}")
```

Make sure to check the MSL Guidelines Phase 2 checklist (https://astro-pi.org/mission-space-lab/guidelines/program-checklist) for rules on files and filenames.

Numbering plans for images and files

Normally, experiments generate one or two `.csv` files, but it is very common to take lots of pictures. When dealing with lots of files of the same type, it's a good idea to follow a naming convention. In the example below, we use an obvious sequence number: `image_001.jpg`, `image_002.jpg`, etc., to keep our files organised:

```
from time import sleep
from picamera import PiCamera
from pathlib import Path

base_folder = Path(__file__).parent.resolve()

camera = PiCamera()
camera.start_preview()
sleep(2)
for filename in camera.capture_continuous(f"{base_folder}/image_{counter:03d}.jpg"):
    print(f'Captured {filename}')
    sleep(300) # wait 5 minutes
```

Your experiment

Having read the Getting started with the Sense HAT (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/7), Sense HAT data logger (https://projects.raspberrypi.org/en/projects/sense-hat-data-logger/1), and Getting started with the Camera Module (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/) projects you are in a good position to start taking your own pictures and measurements.

Update your `main.py` file to include a function to collect data from the sensors or to capture an image.

Check out the example below if you need more help.

Measuring temperature and humidity

A team wants to investigate whether the environment on the ISS is affected by the day and night cycle: Does the ISS get colder at night, or drier in the day? To do this, they will first have to collect temperature and humidity data. The function to do this might look like this:

```
from sense_hat import SenseHat

sense = SenseHat()

def collect_data(sense):
    return sense.get_temperature(), sense.get_humidity()
```

Step 6 Running an experiment for 3 hours

In this section we are going to modify our `main.py` so that it will run and stop itself after 3 hours. Each experiment on the ISS has a 3 hour slot to use, after which it will be abruptly stopped. We need to shut down our program gracefully to ensure we don't lose any data.

The datetime library

One way to stop a Python program after a specific length of time is using the `datetime` Python library. This library makes it easy to work with times and compare them. Doing so without the library is not always straightforward: it's easy to get it wrong using normal mathematics. For example, it's simple to work out the difference in time between 10:30 and 10:50 (subtract 30 from 50), but slightly more complicated when you have 10:44 and 11:17 (add (60 - 44) to 17). Things become even trickier if the two times are split across two days (for example, the difference in minutes between 23:07 on Monday 31 May and 11:43 on Tuesday 1 June). The `datetime` library makes this type of operation much simpler by allowing you to create `datetime` objects that you can simply add to or subtract from each other.

By recording and storing the time at the start of the experiment, we can then check repeatedly to see if the current time is greater than that start time plus a certain number of minutes, seconds, or hours. In the program below, this is used to print "Hello from the ISS" every second for 2 minutes:

```
from datetime import datetime, timedelta
from time import sleep

# Create a `datetime` variable to store the start time
start_time = datetime.now()
# Create a `datetime` variable to store the current time
# (these will be almost the same at the start)
now_time = datetime.now()
# Run a loop for 2 minutes
while (now_time < start_time + timedelta(minutes=2)):
    print("Hello from the ISS")
    sleep(1)
    # Update the current time
    now_time = datetime.now()
```

Instead of calling the `print` function, the code would be more useful if it could call a function to collect data or capture an image. Can you think of how to do this?

Alter the code above to make it call an arbitrary function instead of the `print` function.

Solution

Remember that in Python you can use a function just like any other value. For example, you could assign it to a variable:

```
def say_hello():
    return "Hello world"
action = say_hello
```

and then call it later using the variable:

```
action()
```

Or you could pass the function as an input to another function:

```
def one():
    return 1

def two(one):
    return one() + one()
```

We can use this idea of assinging a function to a variable and passing it as an input to another function to modify the while loop so that it accepts any function:

```
from datetime import datetime, timedelta
from time import sleep

def run_for(time_delta, action):
    """
    Repeats an action/function until the timedelta
    has elapsed

    :param time_delta: The timedelta to wait for
    :type timedelta
    :param: The action/function to repeat
    :type function
    """
    # Create a `datetime` variable to store the start time
    start_time = datetime.now()
    # Create a `datetime` variable to store the current time
    # (these will be almost the same at the start)
    now_time = datetime.now()
    # Run a loop for time minutes
    while (now_time < start_time + time_delta):
        # Do the action
        action()
        # Update the current time
        now_time = datetime.now()
```

Now we can call our new function with any `timedelta` and any function:

```
from datetime import datetime

def write_time_to_file():
    with open("data.txt", "a") as f:
        f.write(str(datetime.now()))

run_for(timedelta(minutes=100), write_time_to_file)
```

Stopping gracefully

At the end of the experiment it's a good idea to close all resources you have open. This might mean closing any files you have open:

```
file = open(file)
file.close()
```

or clearing the `sense_hat` LED matrix:

```
from sense_hat import SenseHat

sense = SenseHat()
sense.clear()
```

or closing the camera:

```
from picamera import PiCamera

cam = PiCamera()
cam.close()
```

> Don't forget to call the `flush` and `os.fsync` methods regularly on any files you modify in your experiment to ensure they are saved to disk.

Your experiment

Modify your `main.py` program so that it will finish gracefully and close all its resources before the 3 hours have elapsed.

> Note: When deciding on the runtime for your code, make sure you take into account how long it takes for your loop to complete a cycle. So if you want to make use of the full 3-hour (180-minute) experiment slot available, but each loop through your code takes 6 minutes to complete, then your `timedelta` should be `180-6 = 174` minutes, to ensure that your code finishes before the 3 hours have elapsed.

Step 7 Finding the location of the ISS

For many experiments, it's really useful to be able to calculate the location of the ISS. Knowing how to work out where in the world the ISS is flying over can you help you do all sorts of handy things, like calculating if the Earth and ISS are sunlit — you could even change your program logic to respond to the location of the ISS!

The orbit library

The easiest way to find the location of the ISS is to use the `orbit` Python library on your Kit OS. You can import this library and use the `coordinates` function to work out where on Earth the ISS is flying over:

```
from orbit import ISS
location = ISS.coordinates()
print(location)
```

How does it work?

The `orbit` library provides a simpler application programming interface (API) for the skyfield (https://rhodesmill.org/skyfield/) library. Skyfield is a Python library that allows you to calculate the positions of space objects within our solar system. For accurate calculations, `skyfield` requires the most recent two-line element (TLE) data for the ISS. TLE (https://en.wikipedia.org/wiki/Two-line_element_set) is a data format used to describe the orbits of Earth's satellites. A typical file looks something look something like this:

```
ISS (ZARYA)
1 25544U 98067A   21162.24455464  .00001369  00000-0  33046-4 0  9995
2 25544  51.6454  12.1174 0003601  83.6963  83.5732 15.48975526287678
```

When you import the `ISS` object from the `orbit` library, an attempt is made to retrieve the TLE data from a file called `iss.tle` in the home folder (https://simple.wikipedia.org/wiki/Home_directory). If the file is not present but an internet connection is available, the latest data will be downloaded automatically into the `iss.tle` file, so you don't need to worry about it. However, if your Astro Pi kit has no internet access, then you need to manually download the latest ISS TLE data (http://www.celestrak.com/NORAD/elements/stations.txt), copy the three ISS-related lines into a file called `iss.tle`, and then place this file into your home folder.

The `ISS` object is implemented as a skyfield `EarthSatellite` object (see the skyfield reference (https://rhodesmill.org/skyfield/api-satellites.html#skyfield.sgp4lib.EarthSatellite) and examples (https://rhodesmill.org/skyfield/earth-satellites.html)), which you can use, for instance, to compute the coordinates of the Earth location that is currently directly beneath the ISS:

```
from orbit import ISS
from skyfield.api import load

# Obtain the current time `t`
t = load.timescale().now()
# Compute where the ISS is at time `t`
position = ISS.at(t)
# Compute the coordinates of the Earth location directly beneath the ISS
location = position.subpoint()
print(location)
```

> Note: The current position of the ISS is an estimate based on the telemetry data and the current time. Make sure that the system time has been set correctly.

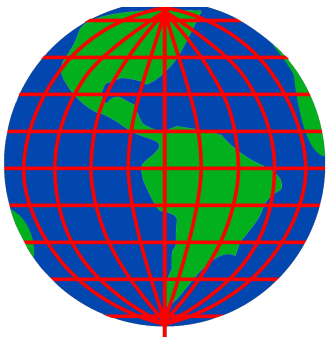
If you need to, you can access the individual elements of the location using the `GeographicPosition` API (https://rhodesmill.org/skyfield/api-topos.html#skyfield.toposlib.GeographicPosition).

```
print(f'Latitude: {location.latitude}')
print(f'Longitude: {location.longitude}')
print(f'Elevation: {location.elevation.km}')
```

> Note that the latitude and longitude are `Angle` objects while the elevation is a `Distance`. The skyfield documentation describes how to switch between different angle representations (https://rhodesmill.org/skyfield/api-units.html#skyfield.units.Angle) and how to express distance in different units (https://rhodesmill.org/skyfield/api-units.html#skyfield.units.Distance).

What are latitude and longitude?

Latitude and longitude are a system of lines used to describe the location of anywhere on Earth. Latitude measures the distance north or south of the equator, while longitude measures the distance west or east of the equator. In the diagram below, latitude lines are horizontal and longitude lines are vertical.



There are a couple of different ways of expressing latitude and longitude which are useful to be aware of:

Decimal Degrees format

```
print(f'Lat: {location.latitude.degrees:.1f}')
print(f'Long: {location.longitude.degrees:.1f}')
```

The code above outputs latitude and longitude values in the Decimal Degrees (DD) format (https://simple.wikipedia.org/wiki/Decimal_degrees), where coordinates are written using degrees (°) as the unit of measurement. There are 180° of latitude: 90° north and 90° south of the equator. There are 360° of longitude: 180° east and 180° west of the prime meridian (the point of zero longitude, defined as a point in Greenwich, England). To precisely specify a location, each degree can be reported as a decimal number, starting with the latitude then the longitude. For example, the point (-28.277777, 71.5841666) describes a precise location in the Indian ocean (https://goo.gl/maps/VRRhtg4seGzHNmn18).

Degrees Minutes Seconds

```
print(f'Lat: {location.latitude.signed_dms()}')
print(f'Long: {location.longitude.signed_dms()}')
```

Another approach is the Degrees Minutes Seconds (DMS) format (https://www.latlong.net/lat-long-dms.html), where each degree is split into 60 minutes (') and each minute is divided into 60 seconds ("). For even finer accuracy, fractions of seconds given by a decimal point are used. The sign of the angle can indicate whether the point that the coordinate refers to is north or south of the equator (for latitude) and east or west of the meridian (for longitude), but to reduce confusion often the abbreviations N W S E are used. The same point in the Indian Ocean (https://goo.gl/maps/VRRhtg4seGzHNmn18) has a DMS value of 28°16'40.0"S 71°35'03.0"E.

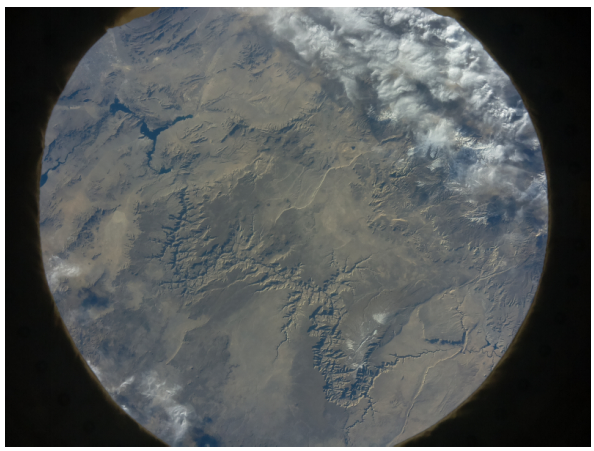
Capturing location data

It is very useful to record the position of the Space Station for any images that you capture. You can do this by attaching metadata to the image file itself using the `exif` (https://exif.readthedocs.io/en/latest/usage.html) library.

In the snippet below, a function called `capture` is called to capture an image, after setting the EXIF data to the current latitude and longitude. The coordinates in the EXIF data of images are stored using a variant of the DMS format, and you can see how the `convert` function takes the data from `ISS.coordinates()` and converts it into a format suitable for storing as EXIF data. Using functions to perform these tasks keeps the program tidy.

```
from orbit import ISS
from picamera import PiCamera
from pathlib import Path

def convert(angle):
    """
    Convert a `skyfield` Angle to an EXIF-appropriate
    representation (positive rationals)
    e.g. 98° 34' 58.7 to "98/1,34/1,587/10"

    Return a tuple containing a boolean and the converted angle,
    with the boolean indicating if the angle is negative.
    """
    sign, degrees, minutes, seconds = angle.signed_dms()
    exif_angle = f'{degrees:.0f}/1,{minutes:.0f}/1,{seconds*10:.0f}/10'
    return sign < 0, exif_angle

def capture(camera, image):
    """Use `camera` to capture an `image` file with lat/long EXIF data."""
    point = ISS.coordinates()

    # Convert the latitude and longitude to EXIF-appropriate representations
    south, exif_latitude = convert(point.latitude)
    west, exif_longitude = convert(point.longitude)

    # Set the EXIF tags specifying the current location
    camera.exif_tags['GPS.GPSLatitude'] = exif_latitude
    camera.exif_tags['GPS.GPSLatitudeRef'] = "S" if south else "N"
    camera.exif_tags['GPS.GPSLongitude'] = exif_longitude
    camera.exif_tags['GPS.GPSLongitudeRef'] = "W" if west else "E"

    # Capture the image
    camera.capture(image)

cam = PiCamera()
cam.resolution = (1296,972)

base_folder = Path(__file__).parent.resolve()
capture(cam, f"{base_folder}/gps1.jpg")
```

Finding the location of an image



When coordinate information is included in the EXIF metadata of your captured images, you can use software such as DigiKam (https://www.digikam.org/about/) (included in the Kit OS) or an online service to automatically locate the position where the image was taken on a map. Alternatively, you can extract the coordinates from the image using the `exif` library.

Example: ISS in the sunlight

The behaviour of your code might differ depending on whether or not the ISS is in sunlight.

The `skyfield` library includes an `is_sunlit` method (documentation here (https://rhodesmill.org/skyfield/earth-satellites.html#find-when-a-satellite-is-in-sunlight)) which you can use with an 'ephemeris' table to calculate whether the ISS is sunlit at a specific time:

```
from orbit import ISS, ephemeris
from skyfield.api import load

timescale = load.timescale()
t = timescale.now()
if ISS.at(t).is_sunlit(ephemeris):
    print("In sunlight")
else:
    print("In darkness")
```

What is an ephemeris?

An ephemeris is a high accuracy table of the position of celestial objects. In this case, it is needed to compute the positions of the Earth and the Sun at a specific point in time.

The `orbit` library makes it easy to import an ephemeris - all you have to do is put `from orbit import ephemeris` at the top of your file and then pass the `ephemeris` object to skyfield's `is_sunlit` function. In the background, `orbit` will download an ephemeris file named `de421.bsp` to your home folder. If you don't have internet access on your Raspberry Pi you will need to do this yourself by saving this file (https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/a_old_versions/de421.bsp) on another computer and put it in your home folder manually. Fortunately these files never change so you only need to do this once.

> The `is_sunlit` method is a good approximation but it is not 100% reliable. Remember that because of the altitude of the ISS, the sun rises on the ISS slightly earlier than it does on the surface of the Earth below the ISS. Similarly, the sun sets on the ISS slightly later than it does on the surface of the Earth directly below it.

Step 8 Computer vision and machine learning

Getting started with computer vision

If you are interested in computer vision, have a look at the Calculate the speed of the ISS (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/4) project. You will learn how to start using the `opencv` library to track key points in a sequence of photos of the Earth, and use them to calculate how fast the ISS is flying.

Complete the Calculate the speed of the ISS (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/4) project.

After completing this project, you may want to look at the OpenCV Python tutorials (https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html) for information on how to do things like edge detection and object tracking — techniques you may decide to use in your own experiment.

Getting started with machine learning

If you have a Coral Machine Learning accelerator, check out the Image classification (https://projects.raspberrypi.org/en/projects/image-id-coral/2) project. You will walk through the process of training a machine learning model to classify images and experience using the `tensorflow_lite` library.

Complete the Image classification (https://projects.raspberrypi.org/en/projects/image-id-coral/2) project.

Once you've completed this project you may want to look at the Coral examples page (https://coral.ai/examples/) and this GitHub page (https://github.com/robmarkcole/satellite-image-deep-learning#datasets) for some inspiration on how to apply machine learning techniques to your own experiment. Perhaps you'd like to classify clouds or 'segment landscapes', or discover some underlying structure in your data?

Data

The success of both machine learning and computer vision relies on having a representative dataset, and to this end you will find a folder called `Data` in the Kit OS. In this folder you will find a CSV file and a selection of images taken from a previous mission which can be used to help test and refine your model.

> Can't find the `Data` folder? Don't worry! You can redownload all of the data (images + csv file) here (http://rpf.io/ap-data-all).

data.csv

The `data.csv` (http://rpf.io/ap-sample-data) file is a comma-separated values file (https://simple.wikipedia.org/wiki/Comma-separated_values) containing 24 hours worth of data from all of the Sense HAT sensors. Click below to find out more.

What is in data.csv?

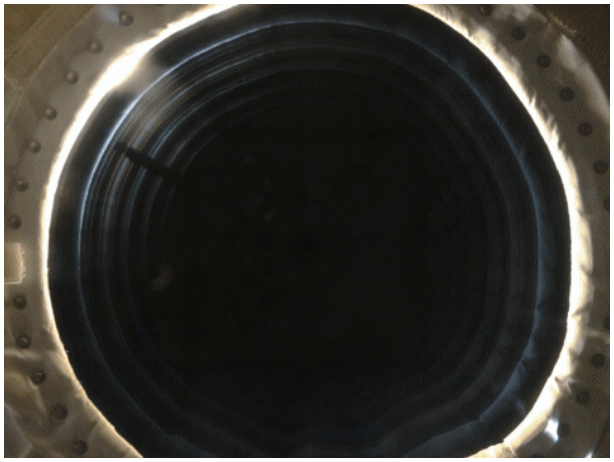
The columns in this file are in the order of this table:

Header	Description
Date/Time	The date and time the measurements were taken.
Latitude	An angle partly describing where over the Earth the measurements were taken.
Longitude	An angle partly describing where over the Earth the measurements were taken.
Temperature	The current temperature in degrees Celsius from the humidity sensor.
Humidity	The percentage of relative humidity from the humidity sensor.
Pressure	The pressure in millibars from the pressure sensor.
Compass	The angle north from the magnetometer in degrees.
MagX	The raw x axis magnetometer data in microteslas (μT).
MagY	The raw y axis magnetometer data in microteslas (μT).
MagZ	The raw z axis magnetometer data in microteslas (μT).
Pitch	A component of the current orientation from the gyroscope in radians.
Roll	A component of the current orientation from the gyroscope in radians.
Yaw	A component of the current orientation from the gyroscope in radians.
AccelX	The raw x axis accelerometer data in Gs.
AccelY	The raw y axis accelerometer data in Gs.
AccelZ	The raw z axis accelerometer data in Gs.
R	The amount of incident red light, scaled to 0-256.
G	The amount of incident green light, scaled to 0-256.
B	The amount of incident blue light, scaled to 0-256.
C	The amount of incident clear light, scaled to 0-256.
Motion	A Boolean signalling whether motion was detected by the PIR sensor.

You can use LibreOffice Calc (https://www.libreoffice.org/discover/calc/) to open this file on the Kit OS.

Images

There is also a sample of photos taken by the Mark II Astro Pi IR and Vis cameras in the `Data` folder. You could use these images to train a machine learning algorithm to recognise different types of views, for example. However, please note that there is no guarantee that the location, view, and orientation of the Astro Pi will be exactly the same when your program runs on the ISS. Therefore, your program should be flexible enough to adapt to any changes.



> It's possible that the Astro Pis may have a different window view to last year.

Overfitting

When creating a machine learning model or writing a computer vision algorithm, ideally the code will be generic enough to cope with slight variations in images. That is, you should be careful not to "overfit" to your data (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit). For example, an ideal land detector would work in a variety of lighting conditions and a variety of camera angles. In general, this is difficult to achieve but there are a few strategies that you can use to generalise your model:

- Play with the scale, crop, colour, and perspective of your training data. For example, you could use a program like GIMP (https://www.gimp.org/) to subtly change the camera angle or the time of day.
- Use more data. You are not limited to only using the Astro Pi images and could consider training your algorithm on other datasets (https://github.com/Seyed-Ali-Ahmadi/Awesome_Satellite_Benchmark_Datasets).
- If you are not using Teachable Machine, keep a portion of your data for testing only so you can avoid getting inaccurate results.

Citizen science

The number of public datasets are vast, but more data means better models! If you complete your experiment to Phase 4 and annotate your images (https://github.com/robmarkcole/satellite-image-deep-learning#annotation-tools-with-geo-features) from the Astro Pi you could contribute too.

Step 9 Preparing for the unexpected

A program can fail for many reasons, but with some foresight and planning it is possible to deal with these failures instead of crashing and losing the chance to capture data and images aboard the ISS. In this section we are going to try and find ways to improve your program so that it stands the best chance of working as intended if something unexpected happens.

Error handling

The first tool in your toolbox is Python's `try-except` statement, which is used to handle exceptions thrown at runtime. Normally, when exceptions are thrown and not handled, the program will crash immediately and data will potentially be lost. However, by using the 'try-except' statement to catch the exception, we can let the program carry on.

A common cause of failed programs is when a mathematical function tries to divide a value by zero. This can happen if you're reading a value from a sensor and then using that as part of a calculation. What will happen if you run the program below at freezing point? Will it print "Hello"?

```
from sense_hat import SenseHat

sense = SenseHat()
b = 5

a = b / sense.get_temperature()
print("Hello")
```

Answer

At freezing point, this program will not print "Hello" and instead will crash with a `ZeroDivisionError`.

One way to handle this situation is to catch the zero case early:

```
if sense.get_temperature() != 0:
    a = b / sense.get_temperature()
    print("Hello")
else:
    print("Temperature is zero")
```

The preferred way of resolving this situation is to try to complete the operation, but handle the exception using a 'try-except' statement:

```
try:
    a = b / sense.get_temperature()
    print("Hello")
except ZeroDivisionError:
    print("Temperature is zero")
```

In this way, we can carry on with our program even in the event of a divide by zero error, and we can deal with other errors by catching them in their own `except` clause:

```
try:
    a = b / sense.get_temperature()
    print("Hello")
except ZeroDivisionError:
    print("Temperature is zero")
except TypeError:
    print("Woops, that's not a number!")
```

Handling multiple exceptions

The easiest way to handle exceptions is to catch all exceptions and deal with them in the same way:

```
try:
    do_something()
except Exception as e:
    print(f"An error {e} occurred")
```

This can be useful to keep the program running in the event of an error, and to provide some information of what went wrong, but it's usually always better to limit the scope of the except statement to specific exceptions that are treated uniquely:

```
try:
    divide(a, b)
except ZeroDivisionError:
    print("b cannot be zero")
except TypeError:
    print("a and b must be numbers")
```

Sadly this doesn't guarantee that the program will be useful if it carries on. In the program above, we will never be able to use the value of `a` in the event that the temperature is zero. Yet, on balance, it's probably a good addition to your experiment program to make the most of the time available on the ISS.

An example - ValueError from the Sense HAT

Another common exception in experiments using the Sense HAT is caused by trying to set a pixel value to a value outside of the range allowed.

```
>>> r = a + b
>>> sense.set_pixel(x, y, r, g, b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.5/dist-packages/sense_hat/sense_hat.py", line 399, in set_pixel
    raise ValueError('Pixel elements must be between 0 and 255')
ValueError: Pixel elements must be between 0 and 255
```

It's important to anticipate all the places in your program where a variable may reach a value that would cause problems, and use the `try-except` statement to your advantage. You can do this using a combination of reading the documentation and testing your code in a variety of environmental conditions.

Using a combination of avoidance and good exception handling, you can avoid errors that would prevent your program from completing its run and causing you disappointment. Imagine receiving the logs from a failed experiment only to see that there was an exception that could have been handled, or an error message that didn't reveal anything about what went wrong.

Review your program and consider if you want to catch any of these errors:

- `DivideByZeroError`
- `ValueError`
- `TypeError`
- `SenseHatError`

Logging

The second tool we have in our toolbox is the `logzero` library. If your experiment data came back with lots of missing data and no explanation, you wouldn't be able to find out what happened. The `logzero` Python library makes it easy to make notes about what's going on in your program. You can log as much information about what happens in your program — every loop iteration, every time an important function is called — and if you have conditionals in your program, `logzero` will log which route the program went (`if` or `else`).

Here's a basic example of how logzero can be used to keep track of loop iterations:

```
from logzero import logger, logfile
from pathlib import Path
from time import sleep

base_folder = Path(__file__).parent.resolve()
logfile(base_folder/"events.log")

for i in range(10):
    logger.info(f"Loop number {i+1} started")
    ...
    sleep(60)
```

The two main types of log entry you can use are `logger.info()` to log information, and `logger.error()` when you experience an unexpected error or handle an exception. There's also `logger.warning()` and `logger.debug()`.

For example, if you had a function to detect night or dark from photos, you could log this information too:

```
for i in range(10):
    if night_or_dark() == 'night':
        logger.info('night - wait 60 seconds')
        sleep(60)
    else:
        ...
```

If you want to handle an exception, but log that you did so, you can use `logger.error`:

```
try:
    do_something()
 except Exception as e:
    logger.error(f'{e.__class__.__name__}: {e})')
```

For example, dividing by zero in `do_something` would create the following log entry:

```
[E 190423 00:04:16 test:9] ZeroDivisionError: division by zero
```

Your program would continue without crashing, but rather than seeing no log entry, you see that an error occurred at this time.

> It's a good idea to use both the `csv` library (for recording experiment data) and the `logzero` library (for logging important events that take place during your experiment).

Your experiment

Using these strategies, you can quickly improve the reliability of your experiment and you can observe your program while its running. In the next section, we will look further into how to increase your confidence in your Python program, to improve its chances of success, and to get ready for submission.

Step 10 Test and checking your program

Once you've finished writing your program, it's vital that you check that it meets the guidelines and test it using the Kit OS. Taking time to check your program against the guidelines and making sure that it runs without errors is the best way to ensure that your experiment passes the Astro Pi Mission Control testing procedure and can run on the ISS.

Check your program against the Mission Space Lab Guidelines Phase 2 checklist (https://astro-pi.org/mission-space-lab/guidelines/program-checklist).

In addition to this checklist, we have prepared a list of common mistakes for you to look out for while reviewing your work:

User input

Your program should not rely on human input via the joystick or buttons. The crew will not have time to manually operate the Astro Pis, so your experiment cannot depend on human input. For example, if an experiment needs a button to be pressed by an astronaut to begin, that button press will never happen, and the experiment will not run. This is also why experiments on the crew, like human reaction speed or memory tests, are not suitable as Mission Space Lab entries.

Inappropriate use of the Sense HAT LED matrix and camera

It is a requirement for all Life in Space submissions to regularly use the LED matrix. If you need help with this, check out this project (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/3). By constrast, Life on Earth experiments are not allowed to use the LED matrix at all — make sure you know what type of experiment you are running and check the mission-specific rules.

Poor documentation

When you've created a useful piece of software and you want to share it with other people, a crucial step is creating documentation that helps people understand what the program does, how it works, and how they can use it. This is especially important for your MSL experiment, because it should be obvious from your program how you will achieve your experiment's aims and objectives.

This project (https://projects.raspberrypi.org/en/projects/documenting-your-code) shows you the recommended way to add useful comments to your program.

Note: Any attempt to hide, or make it difficult to understand, what a piece of code is doing may result in disqualification. And of course, there should be no bad language or rudeness in your code.

Overfitting to one dataset

Your code must be able to deal with variations in conditions aboard the ISS. For example, captured images might have small differences in the field of view.

Use of absolute file paths

Make sure that you don't use any specific paths for your data files. Use the `__file__` variable as described in the How to record data and images (4) section.

Not saving data immediately

Make sure that any experimental data is written to a file as soon as it is recorded. Avoid saving data to an internal list or dictionary as you go along and then writing it all to a file at the end of the experiment, because if your experiment ends abruptly due to an error or exceeding the 3-hour time limit, you won't get any data. To save data immediately, call the `flush` method on a file object and then call `os.fsync`.

Running out of space

Your program is limited to producing 3GB of data. Make sure that you calculate the maximum amount of space that your measurements, including any saved image files, will take up, and that this does not exceed 3GB. Remember that the size of an image file will depend not only on the resolution, but also on how much detail is in the picture: a photo of a blank white wall will be smaller than a photo of a landscape. Test your program in a variety of lighting and atmostpheric conditions if possible to get a good idea of how much space it will use.

Forgetting to call your function

We've seen cases where teams have written a function only to forget to call it in their `main.py` — how frustrating!

Saving into directories that don't exist

A number of teams want to organise their data into directories such as `data`, `images`, etc. This in and of itself is a really good thing, but it's easy to forget to make these directories before writing to them.

Networking

For security reasons, your program is not allowed to access the network on the ISS. It should not attempt to open a socket, access the internet, or make a network connection of any kind. This includes local network connections back to the Astro Pi itself. As part of testing your program, you should disable wireless connectivity and unplug the Ethernet cable from your Raspberry Pi to make sure that your experiment runs successfully without an internet connection.

Trying to run another program

In addition to not being able to use any networking, your program is not allowed to run another program or any command that you would normally type into the terminal window of the Raspberry Pi, such as `vcgencmd`.

Multiple threads

If you need to do more than one thing at a time, you can use a multithreaded process. There are a number of Python libraries that allow this type of multitasking to be included in your code. However, to do this on the Astro Pis, you're only permitted to use the `threading` library.

Only use the `threading` library if absolutely necessary for your experiment. Managing threads can be tricky, and as your experiment will be run as part of a sequence of programs, we need to make sure that the previous one has ended smoothly before starting the next. Rogue threads can behave in an unexpected manner and take up too much of the system resources. If you do use threads in your code, you should make sure that they are all managed carefully and closed cleanly at the end of your experiment. You should additionally make sure that comments in your code clearly explain how this is achieved.

 Forgetting to delete photos from a Life in Space experiment

Life in Space experiments are not allowed to save photos. You must delete them before your experiment finishes.

 Setting the program execution time too short

Some teams set their program execution time to a small value (e.g. 5 minutes) for testing and then forget to switch it back to an appropriate value. Make sure to use as much of your allocated 3-hour time slot as possible.

Review your program again. Can you spot any of the common mistakes in your program?

Testing your program

Having checked your program against the guidelines and reviewed it for common mistakes, you are ready to test it using the Kit OS. Doing this gives your entry the best chance of success and confidence that it will work aboard the ISS. When Astro Pi Mission Control receive your program, it will be tested on an actual Flight OS. Hundreds of teams submit programs to the challenge each year and, unfortunately, there is not enough time to check for mistakes or debug complex code errors. If your program has errors when we test it on the Flight OS, your team will not progress to Phase 3 and your code will not run on the ISS.

To ensure that your entry has the best chance of success, it's important that you test your program thoroughly, debug any errors, and check it against the coding requirements. It's especially important for you to consider any errors that could occur during your program's run on the on-board Astro Pis' Flight OS, such as file path errors or overwriting of files.

> If you have installed any additional software on your Kit OS we recommend that you reflash your SD card with the Kit OS again. Instructions for doing this are in the Kit OS section (2).

To test your program, disconnect your Raspberry Pi from the internet, navigate to your project folder, and run it directly with the following command:

```
python3 main.py
```

Your code should run for 3 hours and then stop.

Test your experiment by running `python3 main.py` from the base directory with the internet disconnected.

When it's finished, observe any output files created by your project. Are you expecting image files from the camera? Data files? Anything else? Are there reports of errors in your logs?

If you see any errors, or the experiment doesn't do what you expected it to, you'll need to address this before you submit your code to ensure that you have a chance of reaching the final judging round.

Review the output of your test run for any problems or unexpected behaviour.

Submitting your experiment

If you tested your experiment program and everything went well, congratulations! You are ready to submit your experiment. All that remains to be done is to zip up your work and upload it to your (mentor's) Raspberry Pi account.

It's easy to `zip` your work using a Terminal:

```
zip -r your_team_name.zip /path/to/your/experiment/base/folder
```

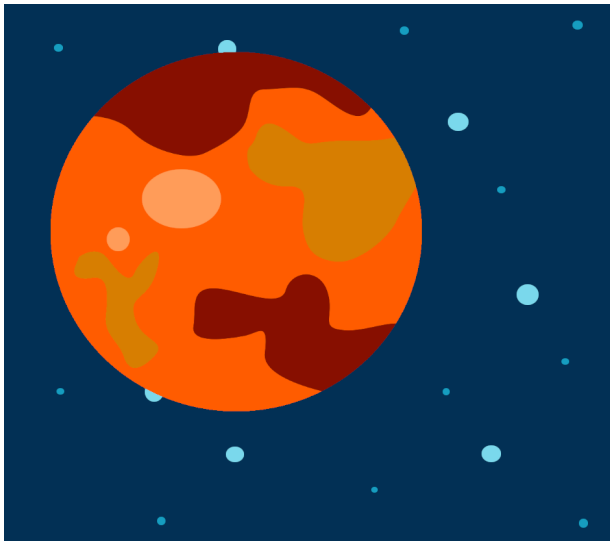
You should see output like below:

```
adding: example_experiment/ (stored 0%)
adding: example_experiment/images/ (stored 0%)
adding: example_experiment/main.py (stored 0%)
adding: example_experiment/data/ (stored 0%)
adding: example_experiment/model.tflite (stored 0%)
```

Don't forget to double-check you have included every file in the zip!

Step 11 A complete worked example

The team from CoderDojo Tatooine wants to investigate whether the environment on the ISS is affected by the surface of the Earth it is passing over. Does the ISS get hotter when it passes over a desert, or wetter when it is above the sea?



This example will serve as a template, to illustrate how you can combine all the elements described so far in this guide to plan and write your computer program.

For this particular example, the program for the experiment should:

- Take regular measurements of temperature and humidity every 30 seconds, and log the values in a CSV file
- Calculate the ISS's latitude and longitude and log this information in the CSV file
- Take a photo using the camera on Astro Pi IR, which is pointing out of a window towards Earth, to gather data on whether cloud cover might also be a factor
- Write the latitude and longitude data in the CSV file and also into the EXIF tags of the images, which have sequentially numbered file names
- Handle any unexpected errors and log the details

The experiment code

Here is what the final code that implements the experiment idea might look like:

```
from pathlib import Path
from logzero import logger, logfile
from sense_hat import SenseHat
from picamera import PiCamera
from orbit import ISS
from time import sleep
from datetime import datetime, timedelta
import csv

def create_csv_file(data_file):
    """Create a new CSV file and add the header row"""
    with open(data_file, 'w') as f:
        writer = csv.writer(f)
        header = ("Counter", "Date/time", "Latitude", "Longitude", "Temperature", "Humidity")
        writer.writerow(header)

def add_csv_data(data_file, data):
    """Add a row of data to the data_file CSV"""
    with open(data_file, 'a') as f:
        writer = csv.writer(f)
        writer.writerow(data)

def convert(angle):
    """
    Convert a `skyfield` Angle to an EXIF-appropriate
    representation (rationals)
    e.g. 98° 34' 58.7 to "98/1,34/1,587/10"

    Return a tuple containing a boolean and the converted angle,
    with the boolean indicating if the angle is negative.
    """
    sign, degrees, minutes, seconds = angle.signed_dms()
    exif_angle = f'{degrees:.0f}/1,{minutes:.0f}/1,{seconds*10:.0f}/10'
    return sign < 0, exif_angle

def capture(camera, image):
    """Use `camera` to capture an `image` file with lat/long EXIF data."""
    location = ISS.coordinates()

    # Convert the latitude and longitude to EXIF-appropriate representations
    south, exif_latitude = convert(location.latitude)
    west, exif_longitude = convert(location.longitude)

    # Set the EXIF tags specifying the current location
    camera.exif_tags['GPS.GPSLatitude'] = exif_latitude
    camera.exif_tags['GPS.GPSLatitudeRef'] = "S" if south else "N"
    camera.exif_tags['GPS.GPSLongitude'] = exif_longitude
    camera.exif_tags['GPS.GPSLongitudeRef'] = "W" if west else "E"

    # Capture the image
    camera.capture(image)


base_folder = Path(__file__).parent.resolve()

# Set a logfile name
logfile(base_folder/"events.log")

# Set up Sense Hat
sense = SenseHat()

# Set up camera
cam = PiCamera()
cam.resolution = (1296, 972)

# Initialise the CSV file
data_file = base_folder/"data.csv"
create_csv_file(data_file)

# Initialise the photo counter
```

```
counter = 1
# Record the start and current time
start_time = datetime.now()
now_time = datetime.now()
# Run a loop for (almost) three hours
while (now_time < start_time + timedelta(minutes=178)):
    try:
        humidity = round(sense.humidity, 4)
        temperature = round(sense.temperature, 4)
        # Get coordinates of location on Earth below the ISS
        location = ISS.coordinates()
        # Save the data to the file
        data = (
            counter,
            datetime.now(),
            location.latitude.degrees,
            location.longitude.degrees,
            temperature,
            humidity,
        )
        add_csv_data(data_file, data)
        # Capture image
        image_file = f"{base_folder}/photo_{counter:03d}.jpg"
        capture(cam, image_file)
        # Log event
        logger.info(f"iteration {counter}")
        counter += 1
        sleep(30)
        # Update the current time
        now_time = datetime.now()
    except Exception as e:
        logger.error(f'{e.__class__.__name__}: {e}')
```

Here's a snippet from the `data.csv` file that is produced:

```
Counter,Date/time,Latitude,Longitude,Temperature,Humidity
1,2021-02-24 10:46:39.399823,39.740617143761526,3.3473845489216094,27.4958,42.934
2,2021-02-24 10:47:10.221346,38.53934241569049,5.26367913685018,27.6456,42.7503
3,2021-02-24 10:47:40.890616,37.309551077336856,7.1032053271899365,27.7018,42.5886
4,2021-02-24 10:48:11.571371,36.047429941325575,8.879601929060437,27.5894,42.6544
```

> Exception handling in this program is rather crude: all raised exceptions will be caught and logged. This means that such a program is very unlikely to terminate abruptly and display an error. Even if errors are generated and the program fails to achieve its goal, this will only become apparent by checking the log files for errors. When testing your program, make sure you also check any log files it generates.

Step 12 Resources

There are a wealth of resources available to help you succeed at every stage of your Astro Pi journey.

> Stuck? Please contact us (mailto:enquiries@astro-pi.org) and we will do our best to help you!

Here are a few Raspberry Pi projects to whet your appetite:

- For a Life In Space experiment, a great resource is the Life In Space pathway (https://projects.raspberrypi.org/en/pathways/life-in-space). This includes projects to help you get started with the Sense HAT (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/3), collect data (https://projects.raspberrypi.org/en/projects/sense-hat-data-logger/4), and design your experiment (https://projects.raspberrypi.org/en/projects/experiment-design/3).
- For a Life On Earth experiment, there is a separate pathway (https://projects.raspberrypi.org/en/pathways/life-on-earth) that includes projects to help you use the camera (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/6) and even calculate the speed of the ISS using computer vision (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/5).
- NDVI experiments will find the NDVI project (https://projects.raspberrypi.org/en/projects/astropi-ndvi) very helpful.
- Machine learning experiments will find the image classification project (https://projects.raspberrypi.org/en/projects/image-id-coral) an excellent guide to get rolling.

In addition to these projects, there are many other useful and interesting tutorials and guides available on the internet:

- The Coral example projects (https://coral.ai/examples/) and the Coral camera examples (https://github.com/google-coral/examples-camera) give a good overview of what's possible with the Coral ML accelerator.
- The OpenCV Python tutorials (https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html) explain how to do all sorts of cool things like machine learning, edge detection, and object tracking.
- The EarthPy examples gallery (https://earthpy.readthedocs.io/en/latest/gallery_vignettes/index.html) walks you through how to process satellite imagery, including for calculating NDVI.
- This GitHub page (https://github.com/orbitalindex/awesome-space) contains a curated list of space-related resources.
- This list of annotated satellite datasets (https://github.com/Seyed-Ali-Ahmadi/Awesome_Satellite_Benchmark_Datasets) can be used to train your machine learning model.
- Teams wanting a greater challenge will find a comprehensive resource on aerial and satellite machine learning here (https://github.com/robmarkcole/satellite-image-deep-learning#datasets) — be warned that it is quite advanced.

Finally, don't forget that links to the documentation for each library are available in the Kit OS section (2) if you need specific information on how to use a particular library.