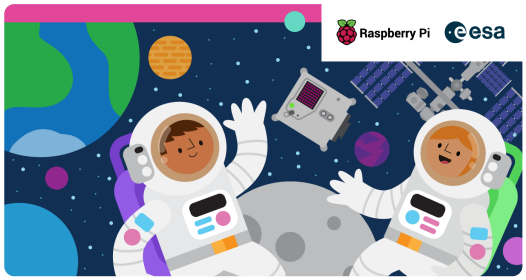
Projects

Astro Pi Mission Space Lab creator guide



The Mission Space Lab creator guide is designed to help teams to write programs that calculate the speed of the ISS. It introduces the task and program requirements, points towards other project guides, and provides guidance on how and what to submit.

Step 1 Introduction

This guide is designed to help you and your team create your program for Mission Space Lab 2023/24. In this Mission, your task is to create a program that will gather data using an Astro Pi computer's sensors and visual light camera and use this data to calculate the speed at which the International Space Station (ISS) is travelling. We will provide you with lots of supporting materials to help you write and develop your program, including an example project using historical photos. We will also help you to adapt and test your program so that it can run for 10 minutes on board the ISS and produce a value for the speed of the ISS, in real time.

This is not a complete step-by-step guide on how to create a program that will solve the problem posed in this Mission. You and your team will need to come up with ideas and solutions and work out how to implement them.

Don't know about Mission Space Lab? Don't worry! Head over to the **Astro Pi website** (https://astro-pi.org/mission-space-lab/) for more information.

Answers to many of the questions that arise can probably be found by searching online, and we encourage you to do some research and try out different solutions if you get stuck. We will also be running a couple of scheduled online webinars where you can ask questions of the team at Astro Pi Mission Control, or you can email us at **enquiries@astro-pi.org** (mailto:enquiries@astro-pi.org). Don't hesitate to contact us — we would love to hear from you.

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

> If you get stuck, please **contact us** (mailto:enquiries@astro-pi.org) and we will do our best to help you!

What you will need to make

Your task is to design a program that will run for 10 minutes aboard the ISS, and in that time, gather data and use it to estimate the speed of the ISS. At the end of the 10 minutes, your program must have written a file containing your estimate for the speed of the ISS in kilometres per second.

You can use our Astro Pi Replay plug-in to simulate your code running live on the Astro Pis on the ISS, to test that your program will work in real time.

We will also give you information on how to improve your program to make sure it runs smoothly on the ISS while also following the security rules.

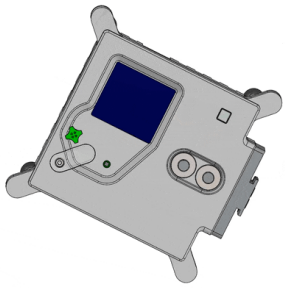
What you will need

To complete this project, you will need:

- **A computer running Python 3.9.2 or above.** You can use any Microsoft Windows, macOS, or Linux computer. You can find **instructions for installing Python here** (https://projects.raspberrypi.org/en/projects/generic-python-install-python3). A full description of the Python requirements for Mission Space Lab appears later in this guide.
- **Thonny integrated development environment (IDE).** We recommend you use Thonny, as it is easy to use, and it is available for Microsoft Windows, macOS, Linux, and Raspberry Pi OS operating systems. You can find **instructions for installing Thonny here** (https://thonny.org/).
- **An internet connection.** You will need to access the internet to install the Python packages and to use the Astro Pi Replay plug-in, and to submit your program.

Step 2 The Astro Pi computers

The Astro Pis aboard the ISS are two modified Raspberry Pi 4 8GB computers, kitted out with a Sense HAT add-on board and camera, and packed into a custom aluminium flight case. The Sense HAT (V2) includes sensors such as temperature, humidity, gyroscope, magnetometer, accelerometer, and light/colour sensors, allowing you to measure things like the local magnetic field and acceleration. The computers are equipped with a powerful Raspberry Pi High Quality Camera with a 5mm lens that can take amazing pictures of the Earth. Plus, these computers can do real-time machine learning thanks to the attached Coral machine learning accelerators. You can **find out more about the computers and sensors here** (https://astro-pi.org/about/the-computers).



With an understanding of what the available sensors on the Astro Pis can do, think creatively about how to use them to find the speed of the ISS. Don't worry about getting everything perfect at first. Try to think of different ways, even if they seem unusual. Either by yourself or as a team, how many ways can you think of to calculate the speed using these tools?

Come up with several different ways of calculating the speed of the ISS using the Astro Pi hardware. Be creative, and try to think out of the box. Once you have a few options, discuss them as a team and choose the one you think will give the most accurate result.

In the next section, you will learn about the different Python libraries available that can help you with your project, and also about some that you cannot use for security reasons.

The Astro Pi Python environment

The Astro Pi computers on the ISS have Python version 3.9.2 installed, so you will need to be using this version, or higher. If you are using a higher version, be aware that there may be some new functions that work on your computer but not on the Astro Pis.

There are some restrictions on the modules (parts) of the standard library that you can use. The following modules are not allowed, and if you do use them, your program will not be accepted:

Disallowed libraries (https://docs.google.com/spreadsheets/u/0/d/1EoVzgA8gOiDXsJ1k9dQBdPyFC8U3bXFca2dRmdKNbcI/edit)

Alongside the Python standard environment, the Astro Pis have extra libraries installed to help you complete the Mission. Each one is explained briefly below with examples. There are also links for more details if you need them. 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 **Finding the location of the ISS (2)** section you can find out how to use Skyfield to obtain the position of the International Space Station above the Earth.

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 guide (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)**

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)

NumPy

`numpy` is a general-purpose array processing library designed to efficiently manipulate large multidimensional arrays (e.g. matrices) 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 (ML) 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 Flight 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 with the Coral accelerator** (2) 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.

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.

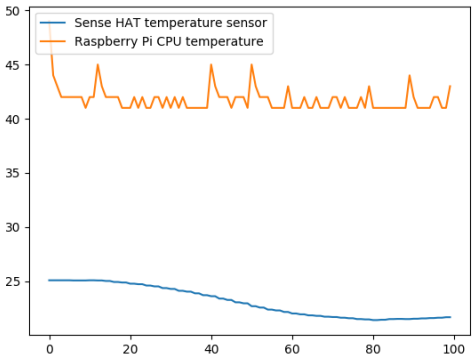
Usage

```
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 documentation** (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/master/)

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 is 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 that 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 guide (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 light and colour sensor (https://gist.github.com/boukeas/e46ab3558b33d2f554192a9b4265b85f)**

> Because there are lots of security restrictions when running a program on board the ISS, these are the only third-party libraries that you will be allowed to use if your program runs on the Astro Pis. Please **contact us (enquiries@astro-pi.org)** if you think anything is missing or have any suggestions.

Setting up your programming environment

We recommend using Thonny to create your program.

Install Thonny

Install Thonny on a Raspberry Pi

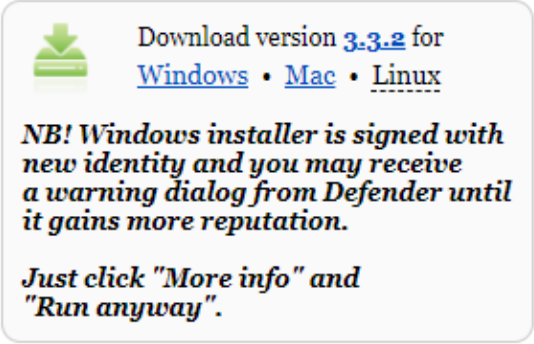
- Thonny is already installed on Raspberry Pi OS, but may need to be updated to the latest version
- Open a terminal window, either by clicking the icon in the top left-hand corner of the screen or by pressing the Ctrl+Alt+T keys at the same time
- In the window, type the following to update your OS and Thonny

```
sudo apt update && sudo apt upgrade -y
```

Install Thonny on other operating systems

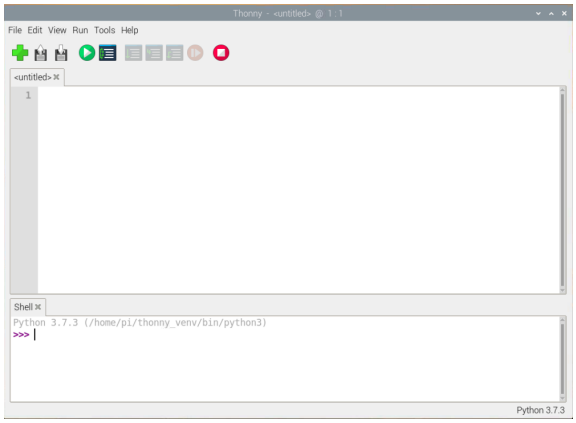
- On Windows, macOS, and Linux, you can install the latest Thonny IDE or update an existing version
- In a web browser, navigate to **thonny.org (https://thonny.org/)**

- In the top right-hand corner of the browser window, you will see download links for Windows and macOS, and instructions for Linux
- Download the relevant files and run them to install Thonny



Opening Thonny

Open Thonny from your application launcher. It should look something like this:



You can use Thonny to write standard Python code. Type the following in the main window, and then click the **Run** button (you will be asked to save the file).

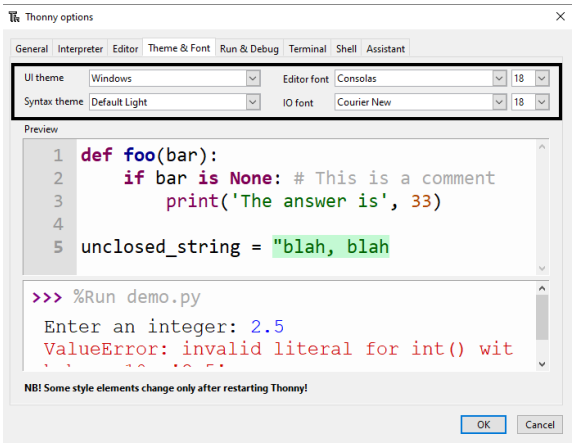
```
print('Hello World!')
```

Change the theme and font in Thonny

Thonny allows you to change the theme and font of the software. This feature means that you can increase the font size and change the background and text colours to suit your needs.

To change the theme and font:

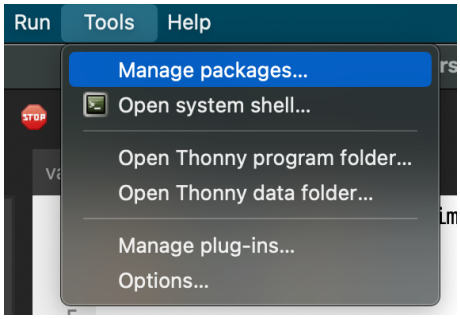
- Click on Tools -> Options.
- Click on the 'Theme & Font' tab.
- Click on the drop down boxes for each option until you find the settings that best suit your needs.



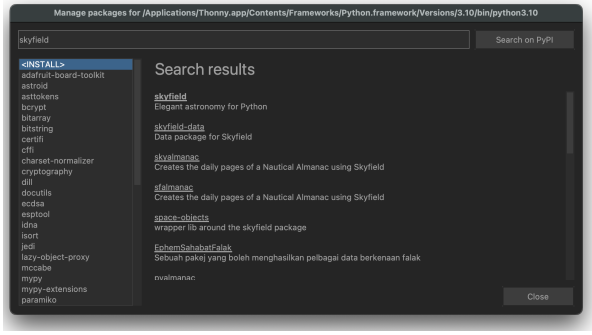
- Press OK when you are finished.

Warning: Stick to simple, clean fonts. If you use a handwriting style font then it can make it difficult to read and debug.

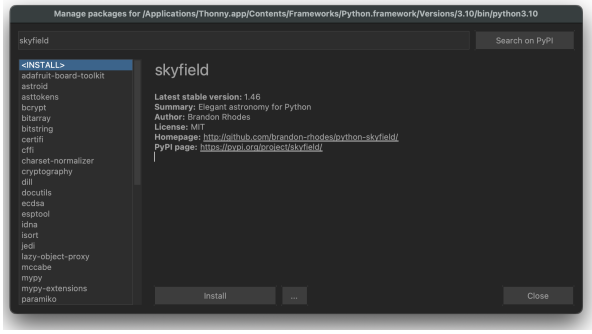
To install any of the Python libraries, open Thonny and click on **Tools > Manage packages…**.



Search for the library you want by typing its name into the search bar.



Select the correct file from the search results, then press **Install**.



The Astro Pi Replay plug-in

The Astro Pi Replay plug-in acts as a kind of simulator you can use on Earth that will make your program act as if it is running on an Astro Pi on board the ISS. It allows you to test your code before it goes to space without needing to have a Raspberry Pi, camera, or Sense HAT. The simulation is not perfect, however, and will only produce photos and sensor data from within its own data set, but it should still allow you to test that your program would work when running on board the ISS.

You can find instructions for downloading and using the Astro Pi Replay tool later in this creator guide.

Looking ahead

Now it's time to think about how your team is going to approach this Mission. Discuss how you will choose your method, divide up the tasks, and plan your program. Speak to your team mentor about your ideas, your progress, and any obstacles along the way. They will have lots of ideas to help you plan.

Step 3 Writing your program and resources to help

This section will help you get started with writing your program, and provide links to other project guides that will help you develop some of the coding skills you may need. You can choose which project guides you want to look at depending on which of the sensors and/or camera you are going to use in your program. At this point, you should have already spent some time with your team and your team mentor to plan your program, and have decided what data you are going to collect to make your calculations.

Getting started

We recommend that you start writing your program in small steps, and that you do not try to do everything at once.

To keep everything organised, create a folder to store all your project files. For the name of the folder, you may wish to use your team name.

The main.py file

Every submission must include a file named `main.py`. This is the file from which your program will run, and which will be tested by Astro Pi Mission Control. When you run the finished program, it should do everything you need to estimate the speed of the ISS. Start by making a file for your main program, and add in the code that you get working as you go along.

Create a new file in Thonny and **Save as** `main.py` in your project folder.

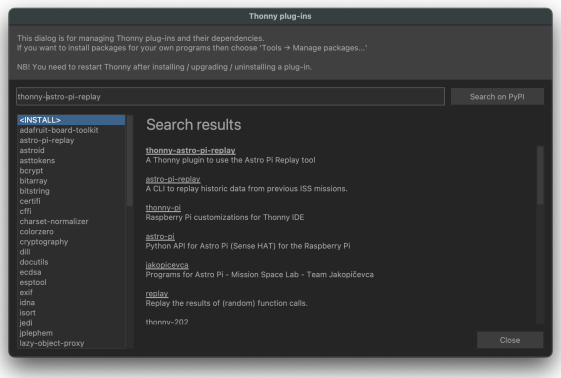
Installing Astro Pi Replay

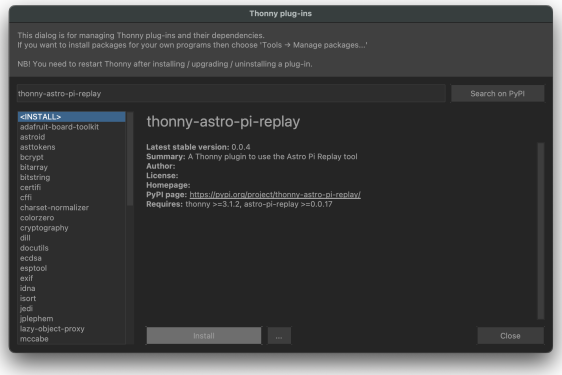
Next, you will install the Astro Pi Replay tool, which allows you to simulate using an Astro Pi Sense HAT or camera to capture data from space.

Installing on Raspberry Pi Bookworm

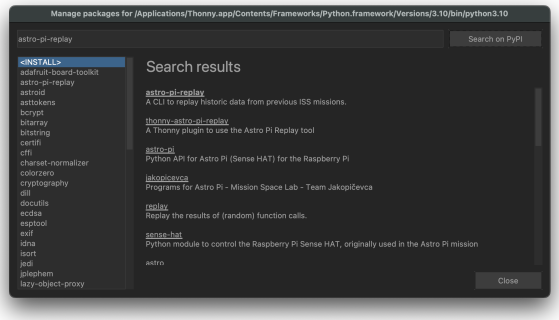
If you are on Raspberry Pi OS Bookworm, please follow the instructions on how to configure Thonny to use a virtual environment on the **raspberrypi website** (https://www.raspberrypi.com/documentation/computers/os.html#using-the-thonny-editor) before proceeding with the instructions below.

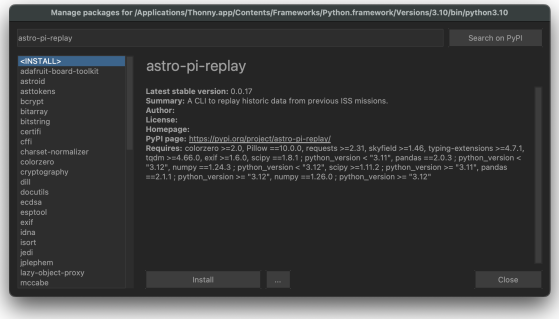
To install the Astro Pi Replay tool, open Thonny, then click on **Tools > Manage plug-ins...**, and search for `thonny-astro-pi-replay`. Select the correct plug-in, then press **Install**.





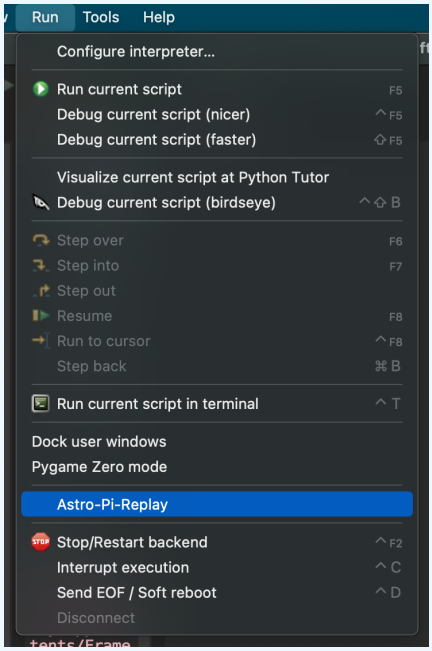
Then, click on **Tools > Manage packages…**, and search for `astro-pi-replay`. Select the correct package, then press **Install**.





You will need to close and restart Thonny for the installation to complete.

The Astro Pi Replay tool works by replaying a set of old pictures taken on the ISS. When your code goes to take a picture, instead of accessing some camera hardware, the library selects a picture to replay and acts as if it has just been captured 'live'.



How to use the Astro Pi Replay plug-in

To run your code using the Astro Pi Replay plug-in, do **not** press the green **Run** button. Instead, open the **Run** menu, then click on **Astro-Pi-Replay**. This will run your code as if it was running on Astro Pi hardware. **Note:** Although all of the functions of the `picamera` library are available, many of the `picamera` settings and parameters that would normally result in a different picture being captured are silently ignored when the code is executed using Astro Pi Replay. Additionally, most attributes on the `PiCamera` object are ignored. For example, setting the resolution attribute to anything other than `(4056,3040)` has no effect when simulated on Astro Pi Replay, but would change the resolution when run on an Astro Pi in space.

Calculating with historical data

You may wish to start by learning how to write a program that estimates the speed of the ISS using photos with our **Calculate the speed of the ISS using photos** (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/0) project guide. Once you have written a program, you can try it out using different images or data sets to improve the accuracy of your estimate. Here are some examples of images and data you can use:

- **Astro Pi Mission Space Lab 2022/23 photos** (https://www.flickr.com/photos/raspberrypi/collections/72157722152451877/)
- **Astro Pi Mission Space Lab 2022/23 data** (https://docs.google.com/spreadsheets/d/1RjPEp2IHVB6For65wuUQdWntsg1H5sHWpYUtLzK9LCM/edit?usp=sharing)

Don't forget that you will only be able to use the visual light camera on the ISS this year.

Simulate running your program in real time

You may prefer to get started by using the `sense_hat` and `picamera` libraries and simulating running your program in real time. To simulate reading data from the Sense HAT and capturing photos from the camera, you will use the Astro Pi Replay tool. Using the tool is simple — instead of running your program by pressing the green **Run** button, open the **Run** menu, then click on **Astro-Pi-Replay**.

Taking measurements with the Sense HAT

In order to calculate the speed of the ISS, you may wish to gather data from the sensors on the Sense HAT. Check out our **Getting started with the Sense HAT** (https://projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat) project guide to learn how to do this.

Remember that you will need to run your code using the **Astro Pi Replay** plug-in on Thonny.

Taking photos with the camera

You may also wish to use the camera to take photos of the Earth to use in your program. You can use our **Getting started with the Camera Module** (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera) project guide to learn how to do this. However, if you do not have a Raspberry Pi and High Quality Camera to test your code on, you can still run the same code using the Astro Pi Replay plug-in.

Here is an example of a simple program to test the Astro Pi Replay plug-in:

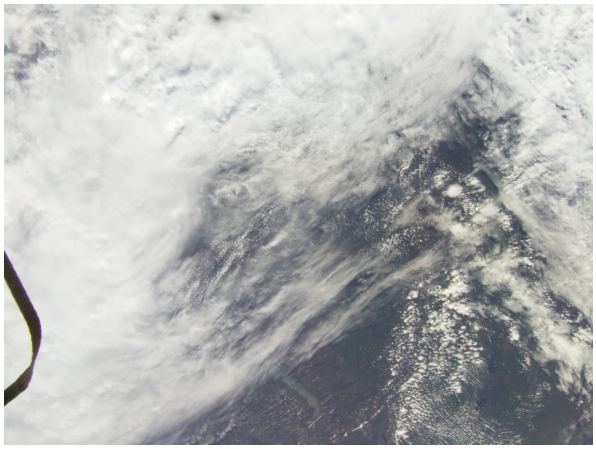
```
# Import the PiCamera class from the picamera module
from picamera import PiCamera

# Create an instance of the PiCamera class
cam = PiCamera()

# Set the resolution of the camera to 4056×3040 pixels
cam.resolution = (4056, 3040)

# Capture an image
cam.capture("image1.jpg")
```

This will simulate taking a picture on the ISS and save it in a file called `image1.jpg`. If you open this file, you should see the exact photo below.



The `picamera` library offers a huge number of features and camera settings. You can see some more advanced examples by going to the **'Basic Recipes' page** (https://picamera.readthedocs.io/en/release-1.13/recipes1.html) on the picamera website, but be mindful that if your code is run on the ISS, it will be taking pictures of a variety of weather conditions with a range of clouds, landscapes, and lighting. However, your program is always guaranteed to be run in daylight.

While all features of the `picamera` library will be available on the Astro Pi in space, not all can be simulated by the Astro Pi Replay plug-in.

Capturing sequences

Using a `for` loop, it is very simple to take a sequence of pictures by repeatedly calling the `capture` function. The example below takes three pictures in succession. It also saves the images as PNG files instead of JPEG files.

Create a new file called `camera-sequence.py`, and in it, type the following lines:

```
# Import the PiCamera class from the picamera module
from picamera import PiCamera

# Create an instance of the PiCamera class
cam = PiCamera()

# Set the resolution of the camera to 4056x3040 pixels
cam.resolution = (4056, 3040)

# Capture three images using a loop
for i in range(3):
    # Capture an image and save it
    # with a file name like
    # "image0.png", "image1.png", etc.
    cam.capture(f"image{i}.png")
```

Run this code using the Astro Pi Replay plug-in by clicking on **Run > Astro-Pi-Replay**.

Numbering plans for images and files

When dealing with lots of files of the same type, it is a good idea to follow a naming convention. In the example above, we use an obvious sequence number — `image1.png`, `image2.png`, etc. — to keep our files organised.

If you need more help with using the camera, check out the **'Take still pictures with Python code' step** [(https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/5)](https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/5) in our 'Getting started with the Camera Module' project guide.

Update your `main.py` file to capture images or Sense HAT data in real time.

Finding the location of the ISS

You will be able to download up to 42 pictures that you take on the ISS. It can be nice to know where exactly an image was taken, and this is something you can do easily with the `orbit` and `exif` libraries available on the Astro Pis.

The following is an example of a program that will, when run using the Astro Pi Replay plug-in, create a new image called `gps_image1.jpg`. The `custom_capture` function will have set the Exif metadata for the image to include the current latitude and longitude of the ISS. There are several ways of formatting **latitude and longitude** [(https://www.britannica.com/science/latitude)](https://www.britannica.com/science/latitude) angles, and using the `custom_capture` function. You will have to adapt this code to suit your particular program.

```
from orbit import ISS
from picamera import PiCamera

cam = PiCamera()
cam.resolution = (4056,3040)

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 custom_capture(iss, 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)


capture(ISS(), cam, "gps_image1.jpg")
```

> 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).

Machine learning with the Coral accelerator

If you have access to a Coral machine learning accelerator, check out our **Image classification with Google Coral** (https://projects.raspberrypi.org/en/projects/image-id-coral/2) project guide. You will walk through the process of training a machine learning model to classify images, and experience using the TensorFlow Lite library. You can then use a similar approach to classify images played back when you run your program using Astro Pi Replay, or on the ISS.

Once you have 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.

Writing your result file

For your submission to pass testing by Astro Pi Mission Control, your program needs to write a file called `result.txt` that contains your estimate for the speed of the ISS. This file must be in text file format (.txt), and will contain your estimate to up to five significant figures. Please do not include any other data in this file.

```
7.1234
```

Example result.txt for an average speed estimate.

The following is an example of a program that will write a .txt file called `result.txt` with an estimated speed value in kilometres per second (km/s) to 5 significant figures. You will have to adapt this code to suit your particular program.

```
estimate_kmps = 7.1234567890  # Replace with your estimate

# Format the estimate_kmps to have a precision
# of 5 significant figures
estimate_kmps_formatted = "{:.5f}".format(estimate_kmps)

# Create a string to write to the file
output_string = estimate_kmps_formatted

# Write to the file
file_path = "result.txt"  # Replace with your desired file path
with open(file_path, 'w') as file:
    file.write(output_string)

print("Data written to", file_path)
```

Update your `main.py` file so that it writes a file called `result.txt` when it is executed.

Make sure to check the **Mission Space Lab rulebook** (https://astro-pi.org/mission-space-lab/rulebook) for rules on files and file names.

Step 4 Optimising your program for the ISS

For astronauts, working in space means working under some very strict constraints, and the same applies to you! This section sets out how to ensure your code behaves as expected while running on the ISS, and how to manage things like resources and errors.

Running an experiment for 10 minutes

Every program run on the Astro Pis has a 10-minute time slot in daylight to estimate the speed of the ISS. Your program will need to keep track of the time and shut down gracefully before the 10 minutes are over to make sure no data is lost.

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 is easy to get it wrong using normal mathematics.

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 1 minute:

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

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

Update your `main.py` file to make use of the `datetime` library to stop your program before the 10-minute time slot has finished.

Note: When deciding on the runtime for your program, make sure you take into account how long it takes for your loop to complete a cycle. For example, if you want to make use of the full 10-minute slot available, but each loop through your code takes 2 minutes to complete, then your `timedelta` should be **10-2 =** `8` minutes, to ensure that your program finishes before 10 minutes have elapsed.

Using relative paths

Your program is going to be stored in a different location when it is deployed on the ISS, so it is really important to avoid using absolute file paths when writing your `result.txt` file (or any other file you might want to write). 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 **Mission Space Lab rulebook** (https://astro-pi.org/mission-space-lab/rulebook) for the rules on files and file names.

Closing resources

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

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

or closing the camera:

```
from picamera import PiCamera

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

> Avoid closing and reopening the camera in a loop — this may cause the Raspberry Pi to run out of memory and prevent your program from being allowed to run on the ISS. Only close the camera after you have finished taking photos.

Review your `main.py` file and update it so that it closes all resources appropriately.

Preparing for the unexpected

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

Exception handling

An exception is when something happens while a program is running that it does not know how to handle. This can cause the program to crash, unless it has a procedure to follow in the event of something going wrong.

Visit Ada Computer Science to learn more about **exception handling** (https://adacomputerscience.org/concepts/design_exception?examBoard=all&stage=gcse).

File buffering

When you write to a file using the `open` function, Python normally does not save the file to disk immediately. Instead, it keeps the file contents to save in a temporary storage area in the computer's memory called a buffer. Python does this so that it can choose the best time to write to the disk — something that normally does not matter us. But while the data is in the buffer and not yet saved to the disk, there is a chance that it could be lost if an error occurs. To prevent this from happening, we can tell Python to save the buffer to disk at the end of every line of text by setting the `buffering` argument to `1`:

```
with open("some_file.txt", "w", buffering=1) as f:
    f.write("example data")
```

Note: If you are writing bytes to a file (with argument `"wb"`), then you should tell Python to not use a buffer at all and to write the data to disk immediately. You can do this by setting the `buffering` argument to `0`.

Review your program and consider if you need to set the buffering mode when writing to a file.

Logging

If your program fails, then it is always helpful to have a record of what happened, so that you can fix it for next time. The `logzero` Python library (**documentation here** (https://logzero.readthedocs.io/en/latest/)) makes it easy to make notes about what's going on in your program. You can log lots of 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`). But remember that you cannot download more than 250MB of data from the ISS.

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

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

logfile("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 your program experiences an unexpected error or handles an exception. There is also `logger.warning()` and `logger.debug()`.

We recommend that you always use the `logzero` library (for logging important events that take place during your experiment), even if you also write sensor data to a file.

Once you have finished writing your program and you believe it provides the ISS speed estimate in the correct format and follows best practices like logging and handling errors, it is crucial to thoroughly test your program using Astro Pi Replay.

Step 5 Improving the accuracy of your program

The average speed of the ISS is not a secret, but the ISS's speed is not really constant and there are several factors that can affect it, for example the altitude. If the altitude has changed, then the ISS must have fired its boosters and therefore the speed must have changed. To increase your chances of getting your program run on the ISS, ask yourself if your program is sensitive enough to the subtle changes that will affect the speed that the ISS is travelling.

On the **N2YO.com website** (https://www.n2yo.com/?s=25544), you can see live data of the ISS which shows how its altitude changes during orbit. You can also see when the ISS will next be passing over your location.

Averages

If your program calculates multiple estimates for the speed of the ISS (for example, by calculating the speed from sequences of two photos), then you will need to decide how to reduce these estimates into a single number when writing your `result.txt` file. If you used a simple average (**mean** (https://en.wikipedia.org/wiki/Mean)), could you explore the accuracy of other statistical measures, such as the median and other percentiles?

There is a lot of scope for being creative when improving the accuracy of your estimate. One method is to be selective about which photos or data you use to calculate your estimate. If you can determine that a specific sequence of data is the most reliable, then you could weight this data more highly in your final estimate.

> Be cautious about training your program to be oversensitive to the exact sequence shown when using Astro Pi Replay — the sequence on the ISS will be different, and you want your program to be accurate on the ISS most of all!

If your method of calculating the speed is based on the **Calculate the speed of the ISS using photos** (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/0) project, then perhaps you could use techniques from computer vision or machine learning to classify photos that are easier to estimate from. For example, you could perform machine learning inference in real time to evaluate the accuracy of your estimate, or the conditions below the ISS.

- [ ] Evaluate the accuracy of the estimate of the ISS speed that your program outputs using Astro Pi Replay.

Step 6 Testing your program

You should now test your program using **Astro Pi Replay**. Doing this gives your entry the best chance of success and of ensuring that it will work aboard the ISS. When Astro Pi Mission Control receives your program, it will be tested and evaluated using Astro Pi Replay, and if it succeeds, on an Astro Pi on Earth. Hundreds of teams submit programs to Mission Space Lab each year and, unfortunately, there is not enough time to check for mistakes or debug code errors. If your program has errors when we test it, your program will not be eligible to run on the ISS.

If you have been following this creator guide from the start, you should have Astro Pi Replay already installed. The installation instructions can be found earlier in this guide and in the Astro Pi Replay documentation.

To test your program and simulate it running aboard the ISS, open Thonny and run your `main.py` code through the Astro Pi Replay plug-in by opening the **Run** menu and clicking on **Astro-Pi-Replay**.

Your code should run for less than 10 minutes and then stop.

When it has finished, double-check that it created a `result.txt` file in your project folder with a valid structure. Additionally, observe any other output files created by your project. Did your saved files exceed the 250MB limit, or include file types that are not allowed in the rules? Finally, check your logs for any errors.

If you see any errors, or the program does not do what you expected it to, you will need to address this before you submit your code, to make sure you have the best chance of achieving 'flight status'. You can rerun your experiment with the **Astro Pi Replay** tool as many times as needed until you are confident that your program works.

Test your program with **Astro Pi Replay** and check the output for any problems or unexpected behaviour.

Program checklist

Your program will also be analysed to make sure that it adheres to the rulebook. Take the time now to read it and re-review your program to ensure it fits the bill.

Check your program against the **Mission Space Lab rulebook** (https://astro-pi.org/mission-space-lab/rulebook).

Many Mission Space Lab teams have not had their programs run on the ISS due to some common mistakes or errors in their programs. Below you will find a list of common mistakes with descriptions of why they affect the programs running on the ISS.

Common mistakes

Opening and closing the camera repeatedly

If you close and reopen the camera multiple times, for example in a loop, you are likely to make the Raspberry Pi run out of memory and not have your program accepted by Astro Pi Mission Control.

Not using full-resolution images

Beware that the ground sampling distance, or GSD, can change with the final resolution of the image.

The value used in the **Calculate the speed of the ISS using photos** (https://projects.raspberrypi.org/en/projects/astropi-iss-speed/0) project guide is valid for images in full resolution, `(4056x3040)`, but not necessarily for smaller resolutions. For this reason, we recommend that you capture full-resolution images.

Storing more than 42 images

Your program is not allowed to retain more than 42 images at the end of the 10 minutes — though it can store more than that while it is running.

User input

Your program cannot rely on interaction with an astronaut to work.

Using the LED matrix

Your program is not allowed to use the LED matrix.

Poor documentation

When you have 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. Make sure your program contains comments and the method used to determine the speed of the ISS is well explained.

This project guide (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 the replayed data

Your code must be responsive to the images and sensor data on the ISS and not rely on specific landmarks or geographic areas shown in the sequence(s) used in Astro Pi Replay.

Use of absolute file paths

Make sure that you do not use any specific paths for your data files. Use the `__file__` variable.

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 10-minute time limit, you will not get any data.

Running out of space

You are allowed to produce up to 250MB of data. 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. Using Astro Pi Replay will give you a good idea of how many pictures you will be able to take.

Forgetting to call your function

We have seen cases where teams have written a function but forgotten to call it in their `main.py` program — watch out!

Saving into directories that do not 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 is 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 your network connection to make sure that your program 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 are only permitted to use the `threading` library.

Only use the `threading` library if absolutely necessary. Managing threads can be tricky, and as your program will be run as part of a sequence of many other 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 program. You should also make sure that comments in your code clearly explain how this is achieved.

Setting the program execution time too short

Some teams set their program execution time to a small value (e.g. 1 minute) for testing and then forget to change it back to an appropriate value. Make sure to use as much of your allocated time slot as possible.

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

Step 7 Submitting your work

If you are happy with your program and have run it using Astro Pi Replay, have read the Mission Space Lab rulebook on the Astro Pi website, and have checked and double-checked the Mission Space Lab program checklist, all that is left to do is to zip up your work and ask your team mentor to submit it to us.

Preparing a zip file

If you do not know how to compress your project folder into a zip file, speak to your mentor, who will be able to tell you the correct process for your operating system.

Removing unnecessary files

Your zip file must not be more than 3MB in size, unless it includes a .tflite model, in which case it is allowed to be up to 7MB in size to accommodate the size of the model. This means, for example, that you cannot supply your own ephemeris files (e.g. **de421.bsp** (https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/a_old_versions/de421.bsp) or **de440s.bsp** (https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/de440s.bsp)), since they are above the size limit.

Both `de421.bsp` and `de440s.bsp` files are available on the Astro Pis. If your program needs them, you can access them via the `orbit` library, as seen in the following code:

```
from orbit import de421, de440s
print(de440)
print(de440s)
```

Generate a zip file for your project, and ask your mentor to submit it to us before the deadline.

Other 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!

- 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 is possible with the Coral machine learning 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.
- This **GitHub page** (https://github.com/orbitalindex/awesome-space) contains a curated list of space-related resources.
- This **list of annotated satellite data sets** (https://github.com/Seyed-Ali-Ahmadi/Awesome_Satellite_Benchmark_Datasets) can be used to train any machine learning models.
- Teams wanting a greater challenge will find a **comprehensive resource on using machine learning with aerial and satellite imagery here** (https://github.com/robmarkcole/satellite-image-deep-learning#datasets) — be aware that it is quite advanced.

Finally, don't forget that links to the documentation for each library are available in the **Astro Pi Python environment** (1) section if you need specific information on how to use a particular library.

Good luck!
