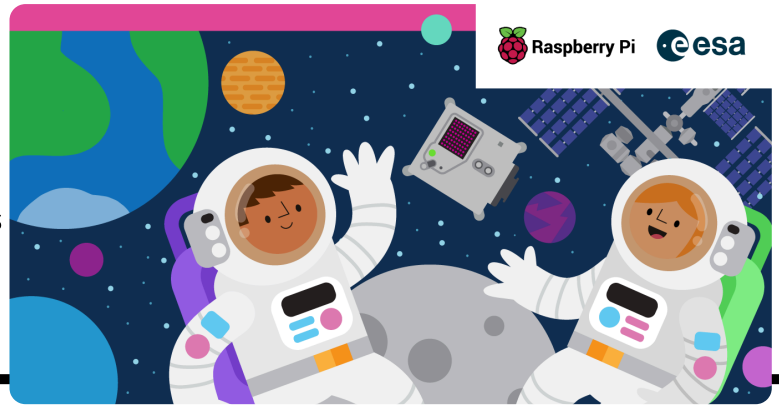
Projects

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 2025/26. 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 **contact@astro-pi.org** (mailto:contact@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 Tool (https://rpf.io/replay) 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.

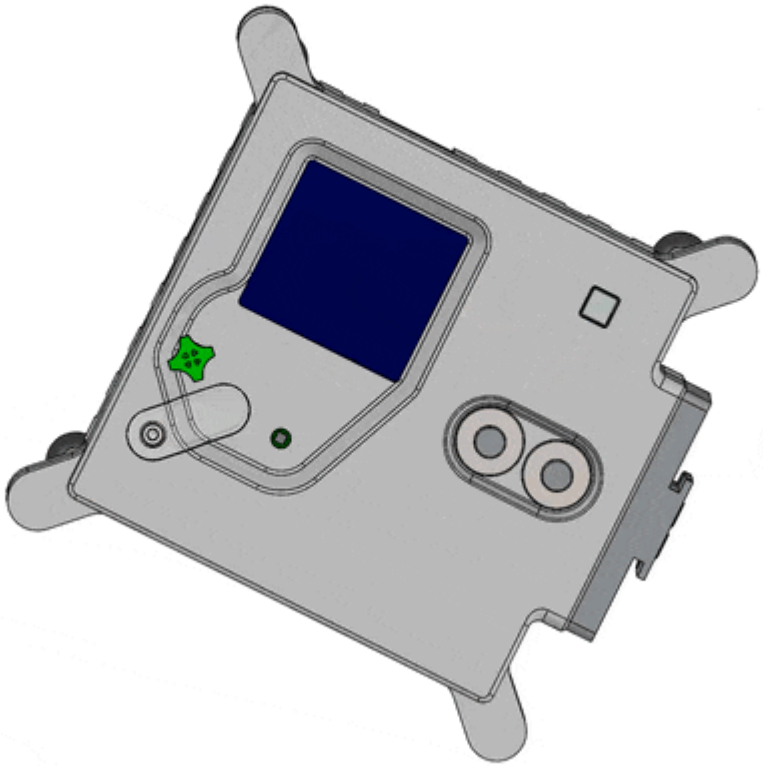
What you will need

To complete this project, you will need:

- **A computer running Python 3.11 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.
- **An internet connection.** You will need to access the internet to test and submit your program.
- **A code editor.** This is where you will write and edit your Python code. Any text editor will work, but we recommend using a dedicated code editor like Thonny. In this guide, we will be using instructions for Thonny.

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. 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. You do not have to use all of the Python libraries in this section, only the ones that you decide to use in your program to help it perform the way you want it to.

The Astro Pi Python environment

The Astro Pi computers on the ISS have Python version 3.11 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 +

astro_pi_orbit +

picamzero +

GPIO Zero +

NumPy +

SciPy +

pandas +

logzero +

Matplotlib +

Pillow +

OpenCV +

exif +

scikit-learn +

scikit-image +

reverse-geocoder +

sense_hat +

ai_edge_rt

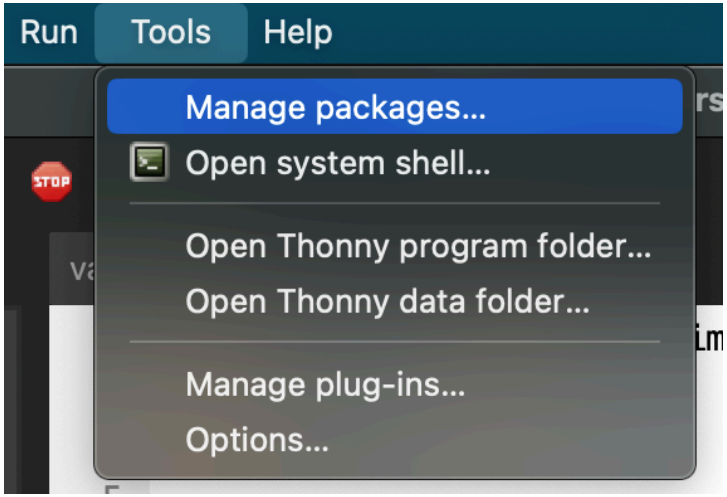
Because there are lots of security restrictions when running a program on board the ISS, these are the only third-party libraries that you will be allowed to use if your program runs on the Astro Pis. Please contact us (enquiries@astro-pi.org) if you think anything is missing or have any suggestions.

Setting up your programming environment

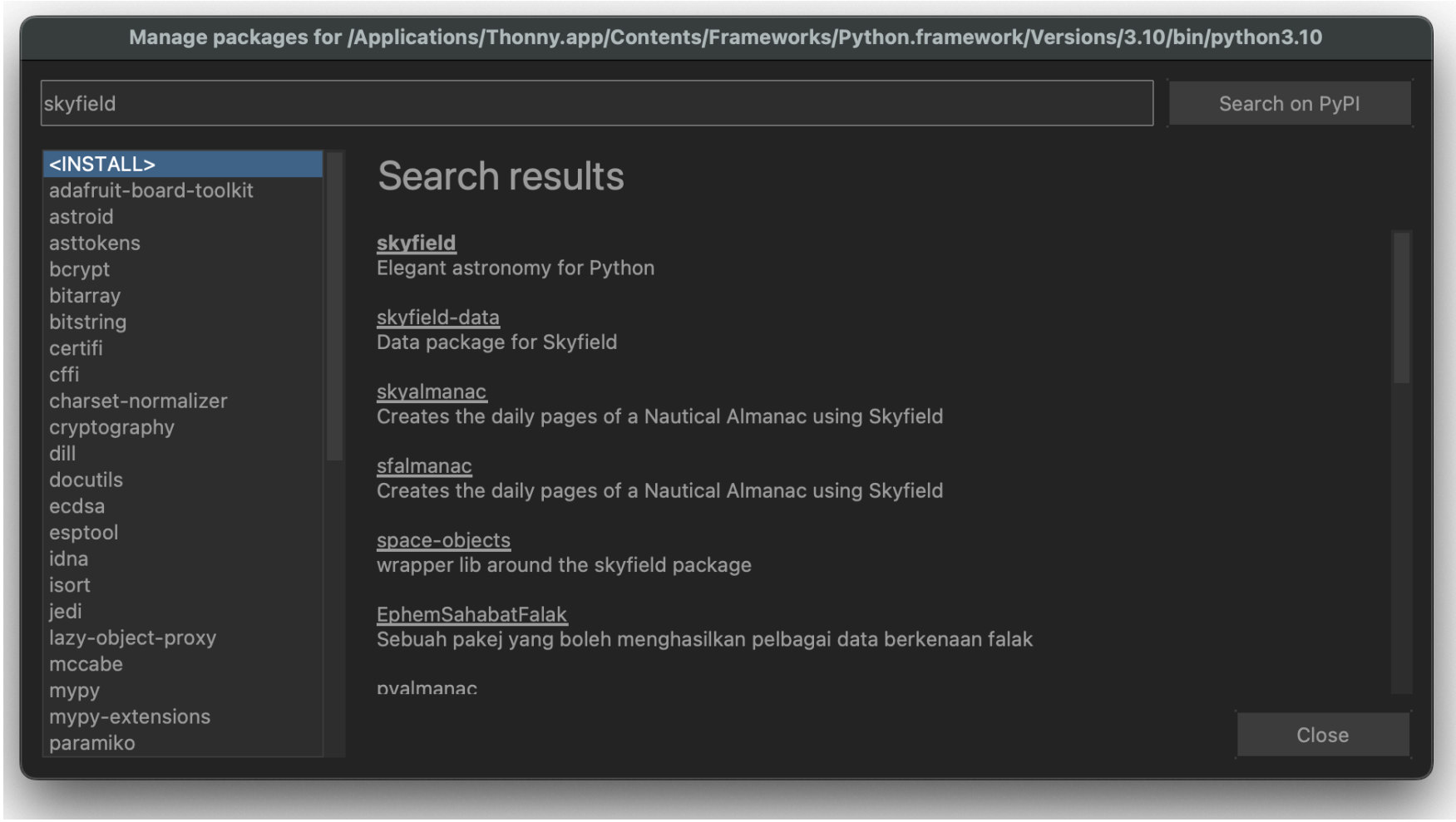
We recommend using Thonny to create your program.

Install Thonny

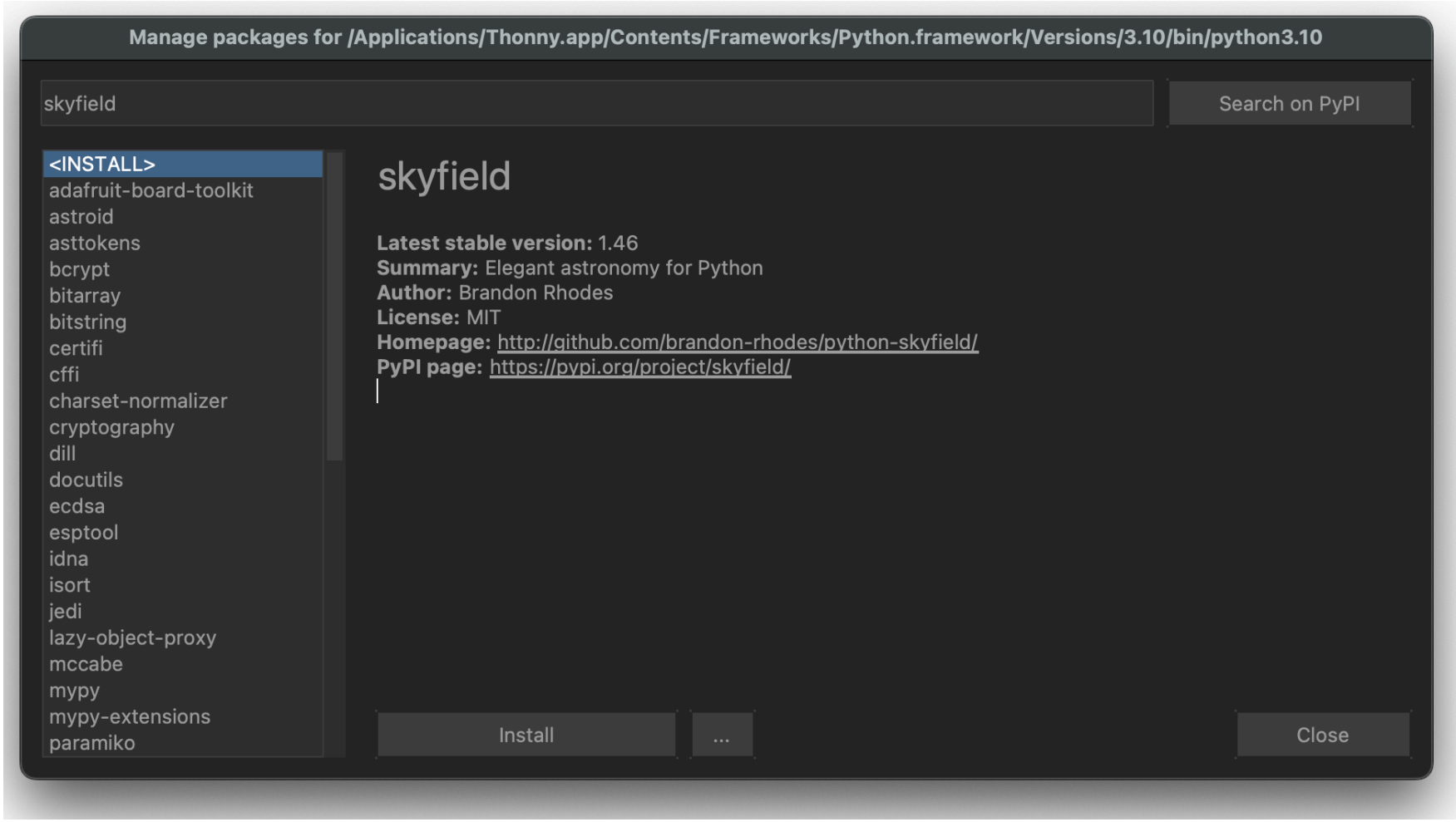
To install any of the Python libraries, open Thonny and click on **Tools > Manage packages...**



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**.



If you are using a different IDE to write your code, you will need to follow local instructions for downloading the libraries you want from PyPi (https://pypi.org/).

Looking ahead

Now that you have set up your coding environment 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.

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.

Test your program with the Astro Pi Replay Tool

The Astro Pi Replay Tool 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.

There is an online version and an offline version, available as a Thonny plug-in, for you to test your program. We recommend you use the online version of the tool.

Accessing the Astro Pi Replay Tool online

Installing Astro Pi Replay Tool on Raspberry Pi Bookworm

Note: Although all of the functions of the `picamera-zero` library are available, many of the `picamera-zero` 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 `Camera` 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.
</p>

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-zero` 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 online or with Thonny.

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.

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://rpf.io/gswpicamera) 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 Tool.

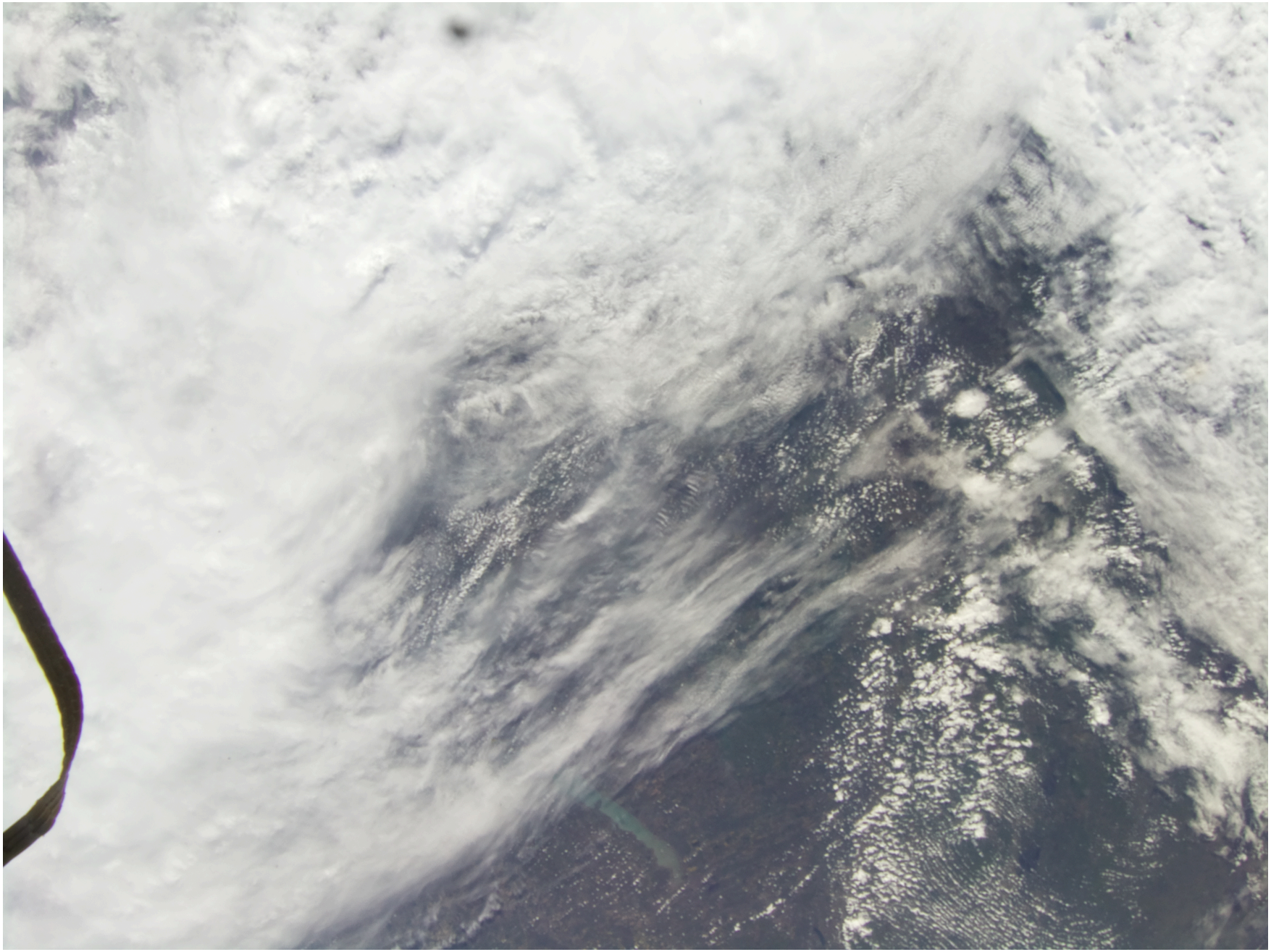
Here is an example of a simple program to test the Astro Pi Replay plug-in, if you are using the offline version in Thonny:

```
# Import the Camera class from the picamera-zero module
from picamzero import Camera

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

# Capture an image
cam.take_photo("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-zero` library supports a variety of features and camera settings. You can see some examples by going to the 'Recipes' page (https://raspberrypifoundation.github.io/picamera-zero/recipes/) on the picamera-zero 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-zero` library will be available on the Astro Pi in space, not all can be simulated by the Astro Pi Replay Tool.

Capturing sequences

Using `picamzero` it is very simple to take a sequence of pictures by calling the `capture_sequence` function. The example below takes three pictures in succession, with a 3 second gap between each one.

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

```
# Import the Camera class from the picamzero (picamera-zero) module
from picamzero import Camera

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

cam.capture_sequence("sequence", num_images=3, interval=3)
```

Run this code using Astro Pi Replay online (https://rpf.io/replay), or with the Thonny 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) 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 `astro_pi_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 Tool, create a new image called `gps_image1.jpg`. The `picamzero` library will have set the Exif metadata for the image to include the current latitude and longitude of the ISS.

```
from astro_pi_orbit import ISS
from picamzero import Camera

iss = ISS()

def get_gps_coordinates(iss):
    """
    Returns a tuple of latitude and longitude coordinates expressed
    in signed degrees minutes seconds.
    """
    point = iss.coordinates()
    return (point.latitude.signed_dms(), point.longitude.signed_dms())

cam = Camera()
cam.take_photo("gps_image1.jpg", gps_coordinates=get_gps_coordinates(iss))
```

You will need to use the Astro Pi Replay tool to run this snippet.

> 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).

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, including units e.g. `km/s`.

```
7.1235
```

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 = "{:.4f}".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.

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. For example, close all files that you have open:

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

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

File buffering

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

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 the Astro Pi Replay Tool.

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.

Step 6 Testing your program

You should now test your program using Astro Pi Replay (https://rpf.io/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 already seen the online Astro Pi Replay Tool (https://rpf.io/replay), or installed the offline version as a Thonny plug-in. 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, go to the Astro Pi Replay Tool (https://rpf.io/replay) and submit your program file. If you are using the Astro Pi Replay plug-in with Thonny, 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 complete within 10 minutes.

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 are using the online version of Astro Pi Replay, you may download a zip file of the output of your program.

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** (https://rpf.io/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

ZeroDivisionError when calculating the speed +

cv2.error when calculating image features +

Opening and closing the camera repeatedly +

Not using full-resolution images +

Storing more than 42 images +

User input +

Using the LED matrix +

Poor documentation +

Overfitting to the replayed data +

Use of absolute file paths +

Not saving data immediately +

Running out of space +

Forgetting to call your function +

Saving into directories that do not exist +

Networking +

Trying to run another program +

Multiple threads +

Setting the program execution time too short +

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 included `astro_pi_orbit` library, as seen in the following code:

```
from astro_pi_orbit import de421, de440s
print(de421)
print(de440s)
```

If you are not using a Raspberry Pi you will need to run this snippet using the Astro Pi Replay tool in order to access the `astro_pi_orbit` library.

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 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!
