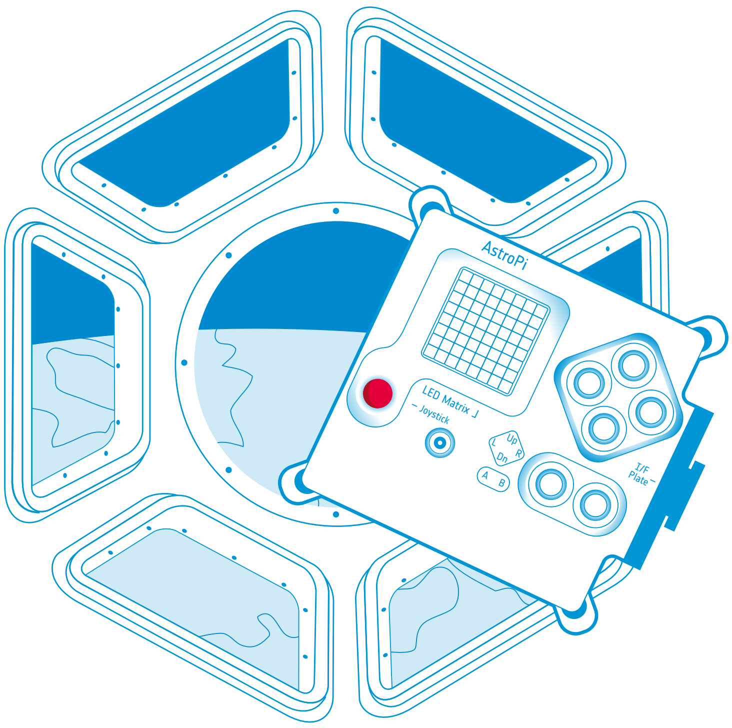
teach with space

→ EUROPEAN ASTRO PI CHALLENGE 2017-2018

Mission Space Lab Coding Rules



European Space Agency

→ MISSION SPACE LAB CODING RULES

In this document the student teams participating in Phase 2 of the European Astro Pi Challenge - Mission Space Lab will find the rules their code must obey in order to guarantee the feasibility of their mission on the ISS. Read them carefully to make sure your entry will be accepted.
Enjoy and good luck!

Coding rules - Codes MUST		
Rules	**Explanation**	**Example code (for illustration only)**
Codes must: Run the experiment proposed in Phase 1 of the Challenge	Astro Pi kits are given to teams on the merit of the experiment idea they proposed in Phase 1 of the Challenge. Because of this, ESA requires teams to stick to their ideas in Phase 2. If teams are not carrying out the original experiment they proposed (for instance, an experiment easier than the one initially proposed), they will be disqualified. If, however, teams need to modify their experiment for a legitimate reason, they will first have to contact ESA, who will review the situation on a case by case basis, and assess if such an entry can be accepted.	`n/a`

Codes must: Be written in Python version 2.7.x or 3.4.x	The Astro Pi computers onboard the ISS only have Python versions 2.7 and 3.4 installed, as do the Astro Pi kits supplied by ESA. Entries that require a different version of Python will be disqualified.	`import sys`
		`print(sys.version)`

Codes must: Use the LED matrix	The LED matrix is the only display available to the Astro Pi computer, and it is never connected to a normal monitor or TV screen on the ISS. The crew may begin to wonder if the Astro Pi computer has crashed if nothing is shown on its display for some time. It will then cost crew time if they need to check it and/or call ground control to report a problem. To avoid this, any entries that do not use the LED matrix at all will be disqualified.	`from time import sleep`
		`from sense_hat import SenseHat`
		`sense = SenseHat()`
		`sense.clear(0, 255, 0)`
		`sleep(1)`
		`sense.clear()`
		`sleep(1)`
		`sense.clear(255, 0, 0)`
		`sleep(1)`
		`sense.clear()`

Coding rules - Codes MUST

Rules	Explanation	Example code (for illustration only)
Codes must: Use at least one sensor, or the infrared (NoIR) camera	Teams will have to collect data and input from the Astro Pi sensors or the camera. Without this the program is display-only, and arguably it is not a science experiment. Therefore, any entries that do not use the sensors or the camera will be disqualified.	`from sense_hat import SenseHat` `sense = SenseHat()` `temp = round(sense.temp, 1)` `print (temp)`
Codes must: Save data into a file	In Phase 4 of the Challenge teams will be asked to write a short report on the results of their experiment. To get results back from space, the collected data will need to be stored in a file for download to ground; this save- to-file instruction has to be included in the student code (see example). The collected data can be images, log files, or csv files containing a table of your sensor readings. ESA strongly recommends storing a timestamp along with the data collected to help with the analysis process. Timestamps can be inside a data file, or even in the file name. Entries that do not produce any results will be disqualified.	`import time` `from sense_hat import SenseHat` `sense = SenseHat()` `with open( "team_data.csv" , "a" ) as f:` `    f.write(str(sense.temp))` `    f.write( "," )` `    f.write(str(sense.humidity))` `    f.write( "," )` `    f.write(str(time.time()))` `    f.write( "\n" )`
Codes should: Take no longer than 3 hours to accomplish their mission	Teams' experiments will run for three hours on the International Space Station (approx. two orbits). Teams do not need to add a special time code to terminate their experiment on time. ESA will take care of that. ESA recommends that the teams write their code as if they had to run indefinitely. They must, however, make sure that they gather all their relevant experiment data within three hours.	n/a

Coding rules - Codes must NOT		
Rules	**Explanation**	**Example code (for illustration only)**
Codes must not: Use the visible camera to save pictures or videos	Experiments related to theme A (Life in Space) may not use the visible camera to save pictures or record videos. This is an operational constraint of the mission due to crew privacy policies. Teams can, however, use the visible camera purely as a sensor where no imagery is stored. Teams can, for example, convert the image data into RGB format, process it in flight and then delete the image file. Therefore, in such a case, the teams' code has to contains instructions to delete the original image file.	`n/a`
Codes must not: Use networking	For security reasons, student codes are not allowed to access the network on the ISS. Any attempt to open a socket, call out to the Internet, or make a network connection of any kind will result in disqualification. This includes local network connections back to the Astro Pi itself. ESA recommends that teams test their code with WiFi off and Ethernet unplugged.	`n/a`

Codes must not:
Contain syntax errors
	A syntax error is a mistake in the structure or sequence of the Python code that prevents the computer from understanding an instruction. These errors will actually stop the code from running so should be easy to spot by the teams. Entries containing syntax errors show lack of testing, and will therefore be disqualified.	`from sense_hat import SenseHat`
		`sense = SenseHat()`
		
		`# Correct syntax:`
		`temp = round(sense.temp, 1)`
		
		`# Incorrect syntax, note the missing bracket:`
		`temp = round sense.temp, 1)`

Coding rules - Codes must NOT		
Rules	**Explanation**	**Example code (for illustration only)**
Codes must not: Contain obfuscated code	Any attempt to deliberately hide or make unintelligible what a piece of code is doing will result in disqualification. This prevents the judges from doing their job, and the judges will automatically assume the code has malicious intent. This includes textually-encoded binary payloads.	`n/a`
Codes must not: Rely on the joystick or manual push of buttons	Since the crew will not have time to manually operate the Astro Pi, the experiments cannot depend on human input. For example, if an experiment needs a button pressed by an astronaut to begin, that button press will never happen and the experiment will not run for three hours. This is also why experiments on the crew, like human reaction speed or memory tests, are not admissible. Any entries which rely on human input via the joystick or buttons will be disqualified. Note: if access to buttons and/or the joystick is performed through the code (without the need for human input), then the code will not be disqualified.	`n/a`
Codes must not: Contain bad language or unpleasant themes	Entries will be manually screened by a human. Any entries that contain swear words, in any language, or any unpleasant meaning will be disqualified. This includes blanking things out with other characters or asterisks.	`n/a`

Codes must not:
Start another process or run a system command
	For security reasons, student codes are not allowed to run another program or any command that you would normally type into the terminal window of the Astro Pi. Any attempt of the teams to do this in their code will result in a disqualification. However there is one exception: the terminal command to obtain the central processing unit (CPU) temperature will be allowed (see example).	`In a terminal window:`
		`vcgencmd measure_temp`

Codes must not:
Raise unhandled exceptions
	An exception is a signal used by the Python programming language when the syntax of the code is correct but an error (different from a syntax error) or unusual situation stops it from running. Python has a number of built-in exceptions that include running out of memory, dividing by zero, or opening a file that doesn't exist. Thorough testing will allow the teams to identify and find these exceptions, but teams can also compensate for them by using some extra code statements such as the try-except block (see example). Any entries presenting unhandled exceptions will be disqualified.	`# Unhandled divide by zero error:`
		`a = 1 / 0`
		
		`# Handled divide by zero error:`
		`try :`
		`    a = 1 / 0`
		`except Exception as ex:`
		`    print ( "oops" )`
		`    print (ex)`