Projets

Guide de la Phase 2 d'Astro Pi Mission Space Lab

Guide à l'attention des participants du concours Astro Pi Mission Space Lab 2021-2022

Étape 1 Introduction

Le défi européen Astro Pi est un projet éducatif de l'ESA mené en collaboration avec la Fondation Raspberry Pi. Le défi Mission Space Lab d'Astro Pi offre aux jeunes la formidable opportunité de mener des recherches scientifiques dans l'espace en écrivant des programmes informatiques exécutés sur des ordinateurs Raspberry Pi, les Astro Pi, installés à bord de la Station spatiale internationale (ISS).

Ce guide couvre la phase 2 pour les deux thèmes de Mission Space Lab : La vie sur Terre et La vie dans l'espace. Nous souhaitons que ton expérience se déroule correctement à bord de l'ISS, et ce guide t'aidera à démarrer rapidement et à mettre toutes les chances de ton côté pour exécuter ton programme sans problème.

Même si tu as déjà participé au concours Astro Pi, nous te recommandons de lire et de suivre ce guide, car beaucoup de choses ont changé par rapport aux années précédentes.

Ce que tu vas réaliser :

Ce guide contient des informations pour assembler ton kit, écrire le code de ton expérience et tester ton programme. Il comprend également des informations importantes sur les possibilités offertes ou non par le matériel et le logiciel Astro Pi.

https://projects.raspberrypi.org/en/projects/code-for-your-astro-pi-mission-space-lab-experiment/print
https://esero.fr/projets/astro-pi/

Traduction et adaptation par ESERO France

Ton projet doit répondre à certaines exigences raisonnables pour être admis à la phase suivante. Elles sont expliquées clairement tout au long de ce guide, ainsi que dans la dernière étape, sous forme de liste de contrôle. Tu dois également suivre certaines suggestions ou bonnes pratiques, afin que ton programme obtienne une bonne note lors du processus d'évaluation et qu'il puisse fonctionner facilement sur l'ISS sans nécessiter de modifications. Si tu estimes que tu dois faire les choses différemment, contacte-nous avant de soumettre ton projet.

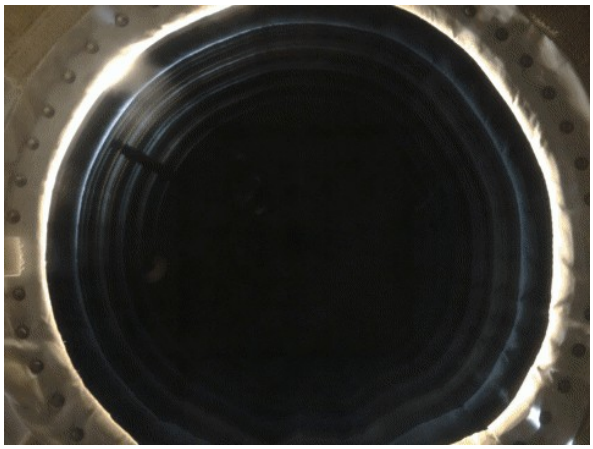
La plupart des informations contenues dans ce guide s'appliquent à la fois aux expériences de La vie dans l'espace et de La vie sur Terre. Cependant, selon le thème que tu as choisi, il existe quelques différences que tu dois prendre en compte.

Remarques pour les expériences de La vie dans l'espace

- Bien que tu puisses utiliser la caméra dans le cadre de ton expérience (par exemple, pour déterminer la luminosité à bord de l'ISS), il est interdit de l'utiliser pour prendre des photos des astronautes. De plus, aucune image ni vidéo prise ne peut rester stockée dans le dossier de l'expérience après la fin de celle-ci.
- Ton programme doit afficher un message ou une image utile sur la matrice LED du Sense HAT afin que les astronautes qui se trouvent à proximité sachent qu'une expérience est en cours. Le contenu affiché doit évoluer régulièrement pour indiquer que tout fonctionne correctement.

Remarques pour les expériences de La vie sur Terre

- Tu ne dois pas utiliser la matrice LED pendant l'exécution de ton expérience. La matrice LED sera désactivée et l'Astro Pi sera couvert pour empêcher la lumière parasite de gâcher les images prises depuis le hublot.



Ce que tu vas apprendre

Tu vas apprendre comment connecter au Raspberry Pi le matériel dont tu auras besoin (par exemple, le Sense HAT, le module caméra, un capteur de mouvement PIR ou le Coral TPU), et comment transformer ton idée de la Phase 1 de Mission Space Lab en une expérience qui fonctionne, en écrivant un programme Python qui peut être exécuté sur les Astro Pi de l'ISS.

Ce qu'il te faut

Matériel

- Un Raspberry Pi 4
- Un Sense HAT

Et si nécessaire pour ton expérience :

- Un module caméra
- Un capteur de mouvement infrarouge passif (PIR)
- Un TPU Coral

Logiciels

Tu auras besoin de la version de bureau du système d'exploitation de Vol pour l'Astro Pi. Il s'agit d'une conception personnalisée de la version de bureau du système d'exploitation du Raspberry Pi, qui comprend toutes les bibliothèques logicielles présentes sur les unités Astro Pi de l'ISS.

Ressources supplémentaires

Si tu le souhaites, tu peux imprimer en 3D une malle renforcée (https:/projects.raspberrypi.org/en/projects/astro-pi-flight-case-mk2), et l'utiliser pour simuler encore plus fidèlement l'environnement de l'ISS, afin d'effectuer des tests plus réalistes. Cependant, ce n'est pas obligatoire, et tu peux participer à Mission Space Lab sans construire une réplique de malle renforcée.

Remarque pour les mentors : collaboration pendant la pandémie de coronavirus

Nous comprenons que les restrictions dues à la pandémie de coronavirus peuvent empêcher les équipes de se réunir pour travailler avec leur kit Astro Pi. Afin d'aider les équipes à tenir ces sessions en ligne et à collaborer à distance, nous avons créé des modèles de plans de cours à utiliser par les mentors pendant leurs sessions :

Première session (https:/rpf.io/first-session-spacelab)
Sessions en cours (https:/rpf.io/ongoing-sessions-spacelab)
Vous les trouverez aussi en français ici : https://esero.fr/ressources/?projet[]=76

Lorsque vous collaborez à distance avec votre équipe, il est important d'appliquer des politiques et procédures fiables de protection des enfants, ainsi que de respecter toutes les lois locales.

Nous vous recommandons de suivre les directives et bonnes pratiques sur la protection mises en place par la Fondation Raspberry Pi (https:/rpf.io/safeguarding).

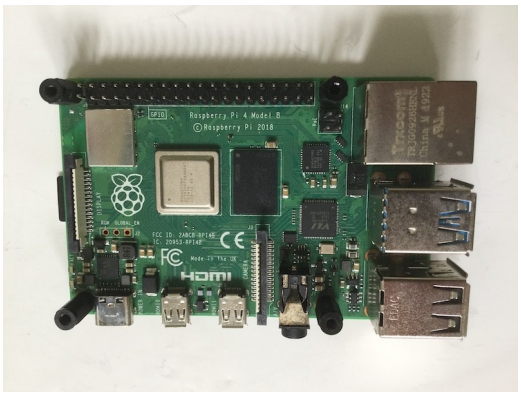
Étape 2 Démarrer

Si tu as reçu un kit Astro Pi officiel de l'ESA, tu as tout le nécessaire pour développer et tester ton programme de Phase 2 pour Mission Space Lab (MSL).Si tu le souhaites, tu peux même créer ta propre malle renforcée Astro Pi (https: /projects.raspberrypi.org/en/projects/astro-pi-flight-case-mk2), mais ne t'inquiète pas, ce n'est pas obligatoire, et la participation à Mission Space Lab ne dépend pas de la fabrication de cette malle.

Assembler le matériel

Déballe tout ce qui se trouve dans ton kit.

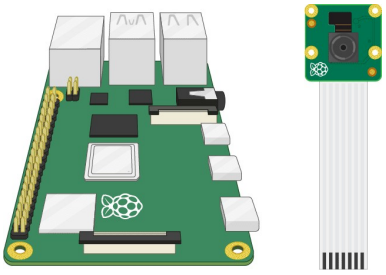
Prends les entretoises d'écartement hexagonales noires dans le petit sac qui accompagne le Sense HAT. À l'aide des vis fournies, fixe-les sur la partie inférieure du Raspberry Pi 4.



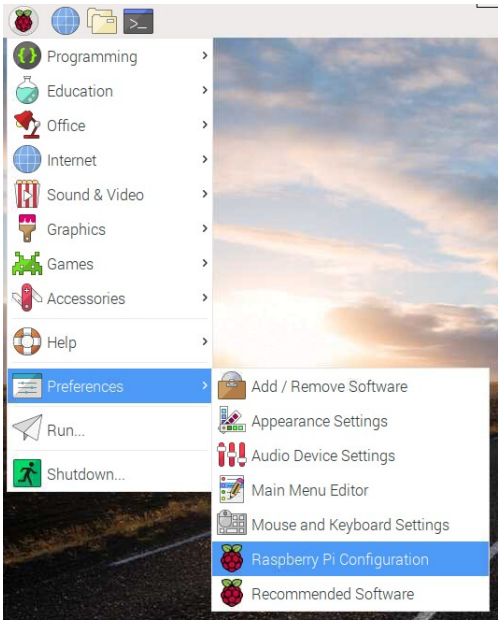
Insère le câble de la caméra dans la prise CSI (Camera Serial Interface) du Raspberry Pi.

Connecter un module caméra du Raspberry Pi

- Vérifie que le Raspberry Pi est éteint.
- Localise le port du module caméra.
- Tire doucement sur les bords du clip en plastique du port.
- Insère le câble ruban du module caméra ; vérifie que le câble est dans le bon sens.
- Remets en place le clip en plastique.

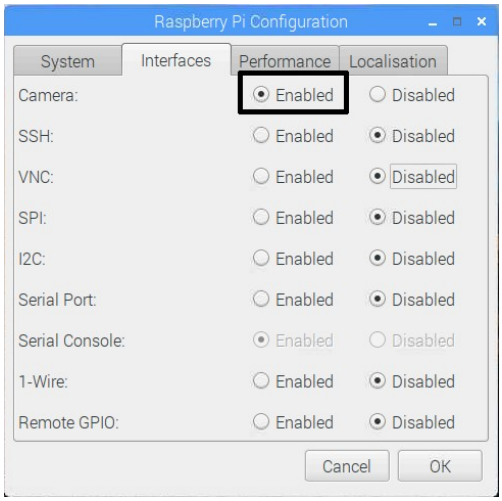


- Démarre le Raspberry Pi.
- Va dans le menu principal et ouvre l'outil de configuration du Raspberry Pi.



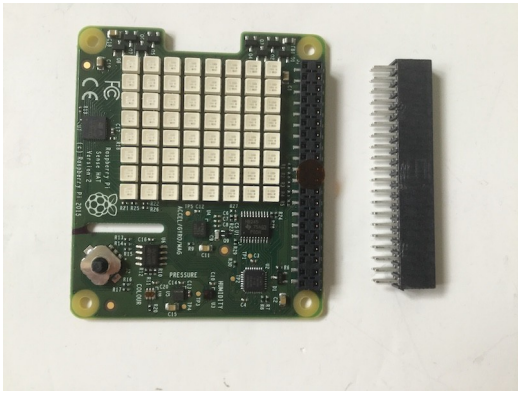
- Sélectionne l'onglet Interfaces et vérifie que la caméra est activée :

https://projects.raspberrypi.org/en/projects/code-for-your-astro-pi-mission-space-lab-experiment/print
https://esero.fr/projets/astro-pi/

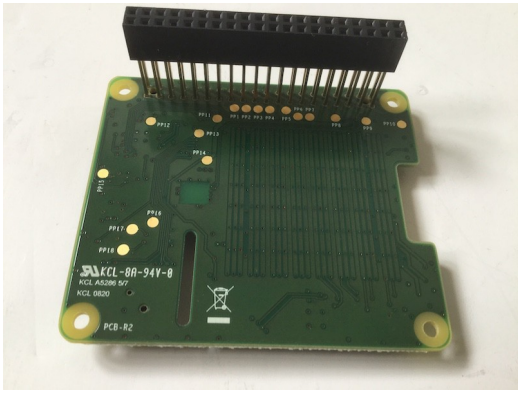


- Redémarre le Raspberry

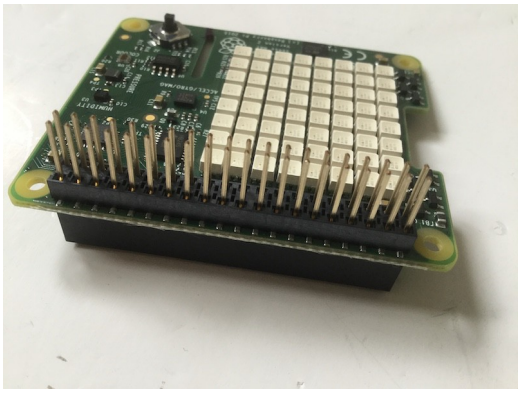
Prends le Sense HAT et retire l'embase courte si elle est attachée.



Aligne la grande embase avec les trous correspondants du Sense HAT.



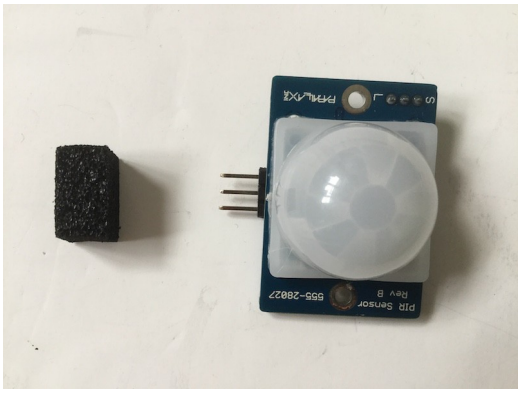
Enfonce l'embase jusqu'au bout. Vérifie qu'aucune des broches n'est obstruée et qu'elles sont alignées correctement afin de ne pas se plier.



Fais passer le câble de la caméra à travers la fente du Sense HAT, puis place le Sense HAT sur le dispositif Raspberry Pi. Vérifie que les 40 broches GPIO sont alignées avec les trous correspondants de la grande embase.

À l'aide des quatre vis noires restantes, fixe le bloc Sense HAT aux entretoises.

Prends maintenant le PIR et retire le bloc de protection en mousse des broches.



Connecte trois câbles aux broches du PIR. Regarde les étiquettes à l'arrière du circuit imprimé du PIR qui indiquent l'utilisation de chaque broche :

- Le GND doit être connecté à la broche de terre correspondante du Raspberry Pi
- Le VCC doit être connecté à une broche 3V3 du Raspberry Pi
- Le OUT doit être connecté à la broche GPIO 12 du Raspberry Pi

Connecte les câbles du PIR aux broches GPIO appropriées du Raspberry Pi. Tu peux utiliser les diagrammes disponibles ici (https://www.raspberrypi.org/documentation/usage/gpio/) pour vérifier que tu connectes les câbles aux broches appropriées.

Convertir une caméra pour des expériences sensibles aux IR du thème La vie sur Terre

Le capteur de la caméra haute qualité peut détecter la lumière infrarouge (IR). Toutefois, le boîtier du capteur contient un filtre IR, qui sert à réduire considérablement la sensibilité de la caméra à la lumière IR. Ainsi, les images prises par le capteur de la caméra haute qualité ressemblent à ce que nous voyons avec nos propres yeux (qui ne sont pas sensibles à la lumière IR).

Si ton expérience de La vie sur Terre requiert une caméra sensible aux IR (comme celle de l'Astro Pi IR), tu devras convertir le capteur de la caméra haute qualité que tu as reçu dans le kit en enlevant le filtre IR. Si tu prévois de mener une expérience de La vie dans l'espace ou si ton expérience de La vie sur Terre requiert de prendre des photos uniquement dans le spectre de la lumière visible, alors tu ne dois pas convertir le capteur de ta caméra haute qualité. Passe directement aux étapes d'assemblage finales indiquées ci-dessous.

Convertir une caméra pour des expériences sensibles aux IR

Remarque : une fois ce processus terminé, il est impossible de l'inverser ou de l'annuler.

- [ ] Assure-toi d'avoir vraiment besoin de la caméra sensible aux IR avant de suivre ces instructions (https: /www.raspberrypi.org/documentation/accessories/camera.html#raspberry-pi-hq-camera-f ilter-removal).

Tu peux maintenant ajouter le filtre passe-bande double rouge/NIR DB660/850-25.4. Il a été conçu principalement pour les applications d'imagerie NDVI (indice de végétation par différence normalisé). En l'ajoutant au capteur de la caméra haute qualité, seules la lumière rouge réfléchie (660 nm) et la lumière proche infrarouge réfléchie (850 nm) seront captées par le capteur. Voir notre projet NDVI (https:/projects.raspberrypi.org/en/projects/astropi-ndvi) pour plus d'information.

- [ ] Vérifie que la bague de mise au point arrière est vissée à fond.

- [ ] Dévisse le capuchon du capteur de la caméra haute qualité et l'adaptateur C/CS.

- [ ] Place le filtre sur le trou au centre du capteur de la caméra.

- [ ] En utilisant uniquement tes doigts, commence à tourner doucement le filtre dans le sens des aiguilles d'une montre, afin de le visser sur le capteur de la caméra. Fais attention à ne pas toucher la partie en verre de l'objectif et à ne pas laisser de traces de doigts !

- [ ] À l'aide de l'outil fourni avec le filtre, aligne les deux bouts de chaque extrémité avec les rainures correspondantes sur le filtre. Si tu as une imprimante 3D, tu peux imprimer une poignée pour l'outil afin qu'il soit plus facile à saisir. Une poignée semblable a été imprimée et envoyée à l'ISS afin que les astronautes puissent l'utiliser lors de l'exécution de cette tâche, mais elle n'est pas obligatoire.

- [ ] Continue de faire doucement tourner le filtre à l'aide de l'outil. Fais attention à ne pas toucher la partie en verre de l'objectif avec l'outil, il pourrait la rayer !

- [ ] Tu dois commencer à sentir une résistance croissante à mesure que tu visses le filtre. Après environ neuf tours complets, le filtre sera aussi bas que possible et tu ne pourras pas le visser davantage. Veille à ne pas trop le serrer.

Étapes d'assemblage finales

- [ ] Insère l'autre extrémité du câble de la caméra dans la prise CSI du capteur de la caméra haute qualité.

- [ ] Retire le capuchon du capteur de la caméra.

- [ ] Retire la bague de l'adaptateur C/CS du capteur.

- [ ] Retire les capuchons de l'extrémité de l'objectif de 6 mm et visse l'objectif sur le capteur.

Remarque : le capteur de la caméra haute qualité fournie dans le kit de l'ESA est le même que celui des nouveaux Astro Pi installés à bord de l'ISS. Tu peux lire la documentation sur la caméra HQ ici (https://www.raspberrypi.org/documentation/hardware/camera/), et de nombreuses informations techniques détaillées se trouvent également dans la section correspondante de la documentation sur la bibliothèque de la PiCamera (https://picamera.readthedocs.io/en/latest/fov.html#camera-hardware).

Ton kit Astro Pi doit maintenant être complet. Insère ta carte SD, connecte-toi à un moniteur, un clavier et une souris et enfin, branche le câble d'alimentation USB-C. Pour plus d'information sur l'utilisation du Raspberry Pi, tu peux consulter le guide suivant : https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started (version anglaise) ou https://esero.fr/wp-content/uploads/2020/10/T05.1b_Getting-started-with-astro-pi_teacher-guide_FR.pdf (version française).

Performances

Les kits de l'ESA pour l'Astro Pi 2020-21 contiennent des dispositifs Raspberry Pi 4 dotés de 4 Go de mémoire (RAM). Ils sont presque identiques aux nouveaux Astro Pi de l'ISS, à ceci près que ces derniers disposent de 8 Go de mémoire. Cela signifie qu'il devrait y avoir très peu de différence de performances entre l'ordinateur sur lequel tu développes et testes ton code et l'ordinateur sur lequel il est exécuté, à moins que ton expérience ne consomme beaucoup de mémoire.

Logiciels et environnement de développement

Après l'assemblage du matériel, tu dois configurer l'environnement de développement. Si tu as reçu un kit de l'ESA, il contient une carte SD avec la version de bureau du système d'exploitation de Vol : une version personnalisée du système d'exploitation Raspberry Pi dont tu auras besoin.

Les packages disponibles dans la version de bureau du système d'exploitation de Vol correspondent étroitement à ceux du système d'exploitation de Vol installé sur les Astro Pi de l'ISS. Cependant, le système d'exploitation de Vol est un environnement « ligne de commande uniquement » qui n'est pas vraiment pratique pour développer des programmes, c'est pourquoi les kits de l'ESA fournissent une version de bureau qui inclut également un ensemble d'outils de programmation. Voici une présentation vidéo (https://youtu.be/i57kwOiR7UM) de la version 2020 !

Tu utiliseras cet environnement pour développer et tester le code de ton expérience. Vérifier que ton programme fonctionne correctement dans cet environnement est la meilleure façon de garantir que ton expérience respecte notre procédure de test et qu'elle peut fonctionner sur les Astro Pi de l'ISS sans aucune modification.

N'effectue aucune mise à jour et n'installe pas de packages ni de bibliothèques Python supplémentaires dans cet environnement, car ils ne seront pas disponibles lorsque ton expérience s'exécutera.

Accéder à distance à la version de bureau du système d'exploitation de Vol

Le bureau du système d'exploitation de vol exécute le VNC Server de RealVNC, vous pouvez donc utiliser le client VNC compatible (https :www.realvnc.com/en/connect/download/viewer/) pour vous connecter à votre bureau depuis un autre ordinateur.

Tu peux également te connecter au système d'exploitation de Vol de bureau depuis un navigateur. Sur un ordinateur connecté au même réseau que ton kit Astro Pi, ouvre un navigateur et entre `http://astro-pi-kit.local/vnc.html` dans la barre d'adresses. Cela doit t'amener sur la page de connexion noVNC. Clique sur le bouton `Connect` puis saisi `raspberry` comme mot de passe : la version de bureau de système d'exploitation de Vol s'affiche dans ton navigateur !

Téléchargement d'images du système d'exploitation (facultatif)

Si tu souhaites, crées des cartes SD supplémentaires à utiliser pour l'Astro Pi, tu peux télécharger

le fichier image du bureau du système d'exploitation de Vol (http:/downloads.raspberrypi.org/AstroPi_latest) utilisé dans les Kits de l'ESA. Après le téléchargement, tu peux utiliser n'importe quel logiciel pour enregistrer le fichier image sur ta propre carte SD. Consulte ce guide (https: /www.raspberrypi.org/documentation/installation/installing-images/) pour savoir comment procéder.

Données d'échantillon

La version de bureau du système d'exploitation de dossier `Données` contenant des exemples de données d'une mission précédente vol inclut également un que tu peux utiliser pour tester et affiner ton code.

Relevés des capteurs

Il existe un fichier de valeurs séparées par des virgules (CSV) contenant l'équivalent de 3 heures de données issues de tous les capteurs du Sense HAT. Les colonnes de ce fichier suivent l'ordre suivant :

Date
Heure
Humidité
Température
Pression
Tangage (mesuré par le gyroscope)
Roulis (mesuré par le gyroscope)
Lacet (mesuré par le gyroscope)
Tangage (mesuré par l'accéléromètre)
Roulis (mesuré par l'accéléromètre)
Lacet (mesuré par l'accéléromètre)
Valeur X brute de l'accéléromètre
Valeur Y brute de l'accéléromètre
Valeur Z brute de l'accéléromètre
Valeur X brute du magnétomètre
Valeur Y brute du magnétomètre
Valeur Z brute du magnétomètre
Degrés de latitude
Minutes de latitude
Secondes de latitude
Degrés de longitude
Minutes de longitude
Secondes de longitude

LibreOffice Calc est un programme de tableur similaire à Microsoft Excel, installé sur le système d'exploitation de Vol de bureau. Tu peux l'utiliser pour étudier les données et tracer des graphiques.

Images

Il y a aussi une séquence de photos prises par la caméra IR de l'Astro Pi Mark I. La séquence commence la « nuit », c'est pourquoi les premières photos sont noires.

Ensuite, le hublot apparaît progressivement lorsque la lumière commence à jaillir.

Dès l'image 150, nous pouvons voir la Terre située en dessous.

Et finalement, la zone qui entoure le hublot n'est plus du tout visible.

Tu peux utiliser ces images pour entraîner un algorithme d'apprentissage automatique à reconnaître différents types de vues. Toutefois, souviens-toi qu'il n'y a aucune garantie que l'emplacement, la vue et l'orientation de l'Astro Pi soient exactement les mêmes lorsque ton programme sera exécuté à bord de l'ISS. Par conséquent, ton programme doit être assez souple pour s'adapter à tout changement.

Étape 3 Écrire ton programme

Maintenant, tu peux commencer à écrire le programme de ton expérience. Pour ce faire, tu devras planifier tes sessions de codage, comprendre quelle est la meilleure façon d'écrire le programme de ton expérience et t'assurer qu'il fonctionnera sur les Astro Pi de l'ISS.

Version Python

Tous les programmes de Mission Space Lab doivent être écrits en Python 3. La version de l'interpréteur Python actuellement disponible sur le système d'exploitation de Vol est la version 3.7.3.

Ton fichier principal de programme Python

Lorsque tu soumets ton expérience MSL, ton fichier principal de programme Python doit s'appeler `main.py`.

Idéalement, l'ensemble de ton code doit être contenu dans ce fichier. Toutefois, si ton expérience est complexe et que tu dois décomposer ton code dans des modules individuels, des fichiers supplémentaires sont autorisés.

Documenter ton code

Lorsque tu crées un logiciel utile et que tu souhaites le partager avec d'autres personnes, créer de la documentation est crucial pour aider les gens à comprendre ce que fait le programme, comment il fonctionne et comment il peut être utilisé. Cela est particulièrement important pour ton expérience MSL, car ton programme doit montrer clairement comment tu atteindras les objectifs de ton expérience.

Ce projet (https://projects.raspberrypi.org/en/projects/documenting-your-code) t'explique comment ajouter des commentaires utiles dans ton programme.

Remarque : toute tentative de masquer ou de rendre difficile à comprendre ce que fait une partie du code entraînera la disqualification du projet. Et bien sûr, ton code ne doit pas contenir de langage grossier ou injurieux.

Bibliothèques Python

Le système d'exploitation de Vol de l'Astro Pi offre une sélection de bibliothèques Python que tu peux utiliser pour ton expérience, en plus de celles qui sont disponibles par défaut. Les mêmes bibliothèques (y compris les mêmes versions de packages) ont également été installées sur la version de bureau du système d'exploitation de Vol, afin que l'environnement où tu développes et testes ton code corresponde le plus possible à celui de l'ISS.

Tu trouveras ci-dessous des informations pour utiliser ces bibliothèques et trouver la documentation pertinente.

Skyfield

Utilisation

Skyfield, le successeur de PyEphem, est un programme d'astronomie qui calcule la position des étoiles, des planètes et des satellites qui sont en orbite autour de la Terre.

La rubrique *Localiser l'ISS* (4) explique comment utiliser Skyfield pour obtenir la position de la Station spatiale internationale au-dessus de la Terre.

Documentation

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

picamera

La bibliothèque Python qui sert à contrôler le module caméra de Raspberry Pi s'appelle `picamera`.

Utilisation

```
from picamera import PiCamera from time import sleep

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

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

Documentation

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

colorzero

`colorzero` est une bibliothèque de manipulation de couleurs, ayant pour objectif d'être simple d'utilisation et dont la nature est proche de celle de Python.

Utilisation

`colorzero` facilite la transition entre deux couleurs :

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

sense = SenseHat()

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

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

Documentation

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

gpiozero

GPIO Zero est une bibliothèque GPIO simple mais puissante. Dans le cadre de Mission Space Lab, la plupart de ses fonctionnalités sont limitées. Par exemple, tu n'es pas autorisé à accéder à des broches GPIO autres que la broche GPIO 12, à laquelle le capteur de mouvement est connecté. Cependant, certaines de ses fonctionnalités peuvent être utiles pour ton expérience, tel que le dispositif interne `CPUTemperature`.

Utilisation

Compare les relevés de température du processeur du Raspberry Pi aux relevés de température du Sense HAT :

```
from sense_hat import SenseHat from gpiozero import CPUTemperature

sense = SenseHat() cpu = CPUTemperature()

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

Documentation

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

GDAL

La Geospatial Data Abstraction Library (GDAL) est un ensemble de bibliothèques multiplateformes et open source, ainsi que d'outils de bas niveau permettant de travailler avec des données géospatiales sous de nombreux formats.

Documentation

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

numpy

`numpy` est un package de traitement de tableau à usage général conçu pour manipuler efficacement de grands tableaux multidimensionnels d'enregistrements arbitraires sans sacrifier trop de vitesse pour de petits tableaux multidimensionnels.

Utilisation

`numpy` est particulièrement utile pour capturer des données de caméra que tu souhaites manipuler :

```
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 est une librairie Python libre et open source utilisée pour l'informatique scientifique et technique. SciPy contient des modules pour l'optimisation, l'algèbre linéaire, l'intégration, l'interpolation, les fonctions spéciales, la FFT (transformation de Fourier rapide), le traitement de signaux et d'images, les solveurs d'EDO (équations différentielles ordinaires) et d'autres tâches fréquentes en science et en ingénierie.

Documentation

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

TensorFlow, TensorFlow Lite et PyCoral

TensorFlow est le cadre d'apprentissage automatique de Google. Tu en auras besoin si tu veux construire et entraîner tes propres modèles. Si tu as seulement besoin d'utiliser ou de ré-entraîner des modèles existants pour inférence, tu peux utiliser TensorFlow Lite ou, mieux encore, la bibliothèque PyCoral. Cette bibliothèque est construite sur TensorFlow Lite mais dispose d'une interface plus simple et de plus haut niveau, et te permet d'utiliser facilement l'accélérateur USB Coral (Edge TPU).

Documentation

- Tensorflow 2.4 (https:/www.tensorflow.org/versions/r2.4/api_docs/python/tf)
- Tensorflow Lite (https:/www.tensorflow.org/lite/api_docs/python/tf/lite)
- PyCoral (https:/coral.ai/docs/edgetpu/tflite-python/)

pandas

`pandas` est une bibliothèque open source offrant des structures de données et des outils d'analyse de données très performants et simples à utiliser

Documentation

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

logzero

`logzero` facilite la journalisation avec Python.

Utilisation

```
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/)

Keras

Keras est une API de réseaux neuronaux de haut niveau qui peut fonctionner sur TensorFlow.

Documentation

- keras.io (https://keras.io/)

matplotlib

`matplotlib` est une bibliothèque de traçage 2D qui produit des tracés de qualité publication dans une variété de formats papier et d'environnements interactifs.

Utilisation

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

sense = SenseHat() cpu = CPUTemperature()

st, ct = [], []
 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/)

pisense

`pisense` est une interface alternative au Sense HAT de Raspberry Pi. La principale différence avec `sense_hat` est que dans `pisense` les différents composants du Sense HAT (écran, joystick, capteurs environnementaux, etc.) sont représentés par des classes distinctes qui peuvent être utilisées individuellement ou par la classe principale qui les comprend toutes.

L'écran comporte quelques atouts supplémentaires, dont la prise en charge de toutes les polices de PIL, la représentation sous forme de tableau numpy (ce qui rend très simple le défilement en assignant des sections d'une image plus grande), et plusieurs fonctions d'animation rudimentaires. Le joystick et l'ensemble des capteurs disposent également d'une interface itérable.

Utilisation

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

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

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

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

hat.screen.fade_to(flag)
```

Documentation

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

Pillow

Pillow est une bibliothèque de traitement d'images. Elle offre une prise en charge étendue des formats de fichier, une représentation interne efficace et des capacités de traitement d'image assez puissantes.

La bibliothèque d'images de base est conçue pour offrir un accès rapide aux données stockées dans quelques formats de pixel de base. Elle peut fournir une base solide pour un outil de traitement d'image général.

Documentation

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

opencv

`opencv` est une bibliothèque de vision numérique open source.
Plus particulièrement, c'est le package `opencv-contrib-python-headless` qui est installé sur les Astro Pi, et qui inclut l'intégralité de `opencv`, ainsi que des modules supplémentaires (listés dans les documents opencv (https://docs.opencv.org/master/)) et qui exclut toutes les fonctionnalités de l'interface utilisateur graphique.

Documentation

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

exif

`exif` permet de lire et de modifier les métadonnées d'image EXIF à l'aide de Python.

Documentation

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

scikit-learn

`scikit-learn` est un ensemble d'outils simples et efficaces pour l'exploration et l'analyse de données accessibles à tous, et réutilisables dans différents contextes. Il a été conçu pour interagir avec `numpy`, `scipy` et `matplotlib`

Documentation

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

scikit-image

`scikit-image` est une bibliothèque de traitement d'images open source. Elle comprend des algorithmes pour la segmentation, les transformations géométriques, la manipulation de l'espace colorimétrique, l'analyse, le filtrage, la morphologie, la détection de fonctions, et bien plus encore.

Documentation

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

reverse-geocoder

`reverse-geocoder` prend une coordonnée latitude/longitude et donne le village ou la ville la plus proche.

Utilisation

Lorsqu'il est utilisé avec `skyfield` et `reverse-geocoder`, il peut déterminer la position actuelle de l'ISS :

```
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)
```

Selon ce résultat, l'ISS se trouverait au-dessus de Hamilton, dans l'état de 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)

Tu ne dois pas installer d'autres packages Python ni modifier les versions de ceux déjà disponibles dans le système d'exploitation de Vol de bureau. Si tu le fais, ton programme pourra fonctionner correctement lors du test dans ton environnement modifié, mais il échouera lors du test sur le système d'exploitation de Vol réel.

Pour t'assurer que tu n'utilises pas un package Python que tu n'es pas censé utiliser, suis les instructions de ce guide et vérifie que ton code se termine correctement, sans générer d'erreurs, lorsqu'il sera exécuté dans un terminal à l'aide de `python3 main.py`, dans une version non modifiée du système d'exploitation de Vol de bureau.

Si ton expérience requiert des bibliothèques Python qui ne sont pas disponibles dans le système d'exploitation de Vol, contacte-nous et nous essaierons de t'aider à trouver une solution.

Note que certaines bibliothèques Python peuvent inclure des fonctions qui exécutent une requête Web pour rechercher des informations ou retourner une valeur qui dépend de l'heure ou de l'emplacement. Même si elles peuvent s'avérer très utiles, elles ne sont pas autorisées (voir la section « Mise en réseau » de ce guide).

Étape 4 Capteurs de l'Astro Pi

Le Sense HAT de l'Astro Pi héberge une gamme de capteurs dont tu peux extraire des données d'entrée utiles pour tes expériences :

- Accéléromètre
- Gyroscope
- Magnétomètre
- Capteur de température
- Capteur d'humidité
- Capteur de pression barométrique
- Capteur de lumière et de couleurs

Si tu n'as jamais utilisé le Sense HAT auparavant, commence par ce court projet (https:/projects.raspberrypi.org/en/projects/getting-started-with-the-sense-hat/). Tu trouveras la version française ici : https://esero.fr/wp-content/uploads/2020/10/T05.1b_Getting-started-with-astro-pi_teacher-guide_FR.pdf.
Reviens dans cette section une fois que tu auras pris connaissance des utilisations de base du Sense HAT.

L'Astro Pi comprend également un capteur de mouvement infrarouge passif (PIR), utilisé dans les alarmes anti-intrusion. Il est capable de détecter si un objet se déplace dans la portée de son champ de vision (par exemple, un astronaute) et de fournir ces données d'entrée via l'une des broches GPIO du Raspberry Pi.

Remarque : pour les expériences de La vie dans l'espace, tu peux utiliser uniquement les données du capteur de lumière ou du capteur de mouvement PIR. Pour La vie sur Terre, l'Astro Pi est positionné avec la caméra face au hublot et placé sous un « capot » noir, pour éviter les reflets. Le capteur de lumière et le capteur de mouvement PIR sont dirigés dans la direction opposée de celle de la caméra, loin du hublot, afin d'être dans l'obscurité et couverts.

Récupérer les données du capteur du Sense HAT

La documentation du Sense HAT (https:/pythonhosted.org/sense-hat/) contient des informations sur la récupération des données des capteurs environnementaux (https:/pythonhosted.org/sense-hat/api/#environmental-sensors) (température, humidité, pression) et de l'unité de mesure inertielle (IMU) (https:/pythonhosted.org/sense-hat/api/#imu-sensor) (accélération, orientation). Une documentation supplémentaire est disponible pour interagir avec les capteurs de lumière et de couleurs (https:/gist.github.com/boukeas/e46ab3558b33d2f554192a9b4265b85f). Tu peux également explorer le large éventail de projets Sense HAT (https:/projects.raspberrypi.org/en/projects?hardware%5B%5D=sense-hat) disponible auprès de la Fondation Raspberry Pi.

Voici un petit exemple montrant comment obtenir des mesures à partir du capteur de couleurs :

```
from sense_hat import SenseHat

sense = SenseHat() sense.color.gain = 16 light = sense.color.clear if light < 64:
   print('Dark') else:
print('Light')
```

Récupérer des données du capteur de mouvement

Tu peux récupérer les données du capteur de mouvement sur l'Astro Pi en utilisant la bibliothèque `gpiozero` pour créer un objet `MotionSensor` attaché spécifiquement à la broche GPIO 12 :

Consulte la documentation (https://gpiozero.readthedocs.io/en/stable/api_input.html#motionsensor-d-sun-pir) to find out about the different ways in which you can interact with the motion sensor.

```
from gpiozero import MotionSensor

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

Remarque : tu dois t'assurer que ton programme utilise uniquement la broche GPIO 12. Attacher un objet `MotionSensor` à une autre broche ne fonctionnera pas du tout. Si tu tentes de manipuler d'autres broches GPIO, tu risques d'entraîner un dysfonctionnement du matériel ou de l'endommager.

Enregistrer les données des capteurs dans des fichiers

Les données d'expérience que ton programme recueille à partir des capteurs doivent être stockées dans des fichiers. Les fichiers CSV sont fréquemment utilisés pour cela. Il s'agit de fichiers texte ordinaires contenant des données sous forme de valeurs séparées par des virgules : dans les lignes de données, chaque valeur individuelle est séparée de ses voisines par une virgule.

Par exemple, voici un extrait d'un fichier CSV où la date, l'heure, l'humidité et la température ont été enregistrées à des intervalles d'environ une minute. Note que les fichiers CSV incluent généralement un en-tête avec les noms des colonnes.

```
Date, Time, Humidity, Temperature 05/05/2018, 10:23:56, 45.60, 21.05
05/05/2018, 10:24:58, 45.62, 21.10
05/05/2018, 10:25:57, 45.68, 21.10
05/05/2018, 10:26:58, 45.72, 21.13
```

Ce type de fichier serait nommé par exemple `data.csv`, avec l'extension `.csv` indiquant le type du fichier.

Remarque : normalement, les expériences génèrent un ou deux fichiers `.csv` . Si ton programme génère un nombre considérable de fichiers de données (par exemple, plus de cinq) au cours de l'expérience, cela indique une erreur logique ou simplement une mauvaise approche, et il est fort probable que ton expérience ne sera pas admise à la phase suivante.

Structure du répertoire et noms des fichiers

Tu ne dois pas faire d'hypothèses quant à l'emplacement de stockage de ton programme lorsqu'il sera déployé sur l'ISS, d'autant plus que la structure du répertoire dans le système d'exploitation de Vol réel est différente de la version de bureau. Ton programme ne doit jamais utiliser de chemins de dossiers absolus, c'est-à-dire qu'il ne doit pas faire référence à des dossiers spécifiques tels que `/home/pi` ou `/home/pi/Desktop`. À la place, ton programme Python principal doit utiliser le code ci-dessous pour déterminer le dossier dans lequel il est actuellement stocké, c'est-à-dire le `base_folder` :

```
from pathlib import Path

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

Tous les fichiers créés par ton programme doivent être enregistrés dans ce `base_folder`, c'est-à-dire dans le même dossier où le fichier Python principal sera stocké lors de son exécution sur les Astro Pi de l'ISS.

De plus, tous les fichiers générés par ton programme doivent avoir des noms pratiques et informatifs. Utilise uniquement des lettres, des chiffres, des points (.), des tirets (-) ou des tirets bas (_) dans les noms de tes fichiers. Aucun autre caractère n'est autorisé. N'utilise pas d'espaces dans les noms de fichiers, car ils peuvent causer des problèmes lorsque les fichiers sont transférés d'un ordinateur à un autre.

Enregistrer des données dans un fichier CSV

Pour créer et écrire facilement dans un fichier CSV, nous te recommandons d'utiliser `csv`, qui est inclus dans la bibliothèque Python standard. Voici un exemple simple qui implique de spécifier le nom du fichier de données, de l'ouvrir avec des autorisations d'écriture et d'ajouter une ligne d'en-tête avant d'écrire selon un mode itératif une ligne supplémentaire de données du capteur toutes les 60 secondes.

```
import csv
from sense_hat import SenseHat from datetime import datetime from pathlib import Path
from time import sleep sense = SenseHat()

base_folder = Path( file ).parent.resolve() data_file = base_folder/'data.csv'

h open(data_file, 'w', buffering=1) as f: writer = csv.writer(f)
header = ("Date/time", "Temperature", "Humidity") writer.writerow(header)
for i in range(10):
row = (datetime.now(), sense.temperature, sense.humidity) writer.writerow(row)
sleep(60)
```

Il est important d'enregistrer l'horodatage avec tes points de données, afin que tu saches quand la mesure a été prise, combien de temps s'écoule entre chaque mesure et à quel moment les actions se sont produites. Tu peux également calculer rétrospectivement la position de l'ISS à l'aide d'un horodatage avec `skyfield`

Remarque : l'argument `buffering=1` utilisé dans la fonction `open` est indispensable. Il permet d'assurer que la sortie générée par le programme est écrite dans le fichier de données ligne par ligne. Sinon, par défaut, la sortie générée serait accumulée dans une mémoire tampon et serait écrite dans le fichier de données uniquement en gros blocs (pour plus d'efficacité).
Chaque année, certaines équipes reçoivent des fichiers de données vides, car leur programme est interrompu lorsqu'il atteint la limite de 3 heures et les données de la mémoire tampon sont perdues avant d'avoir été purgées vers un fichier. Sinon, il est possible d'utiliser `f.flush()` après chaque ligne de code.

Comment peux-tu modifier le code ci-dessus pour enregistrer également les relevés de pression barométrique du Sense HAT ?

J'ai besoin d'un indice

Ton programme doit ressembler à cela :

```
import csv
from sense_hat import SenseHat from datetime import datetime from pathlib import Path
from time import sleep sense = SenseHat()

base_folder = Path( file ).parent.resolve() data_file = base_folder/'data.csv'

h open(data_file, 'w', buffering=1) as f: writer = csv.writer(f)
header = ("Date/time", "Temperature", "Humidity", "Pressure") writer.writerow(header)
for i in range(10):
row = (datetime.now(), sense.temperature, sense.humidity, sense.pressure)
writer.writerow(row)
sleep(60)
```

Tu peux préférer fermer ton fichier chaque fois que tu ajoutes une ligne de données du capteur, puis l'ouvrir de nouveau pour la prochaine ligne. Cela est moins efficace, mais fermer le fichier est une autre façon de s'assurer que les données générées ont été enregistrées, et donc qu'aucune de ces données ne sera perdue si ton programme se termine prématurément.

Tu peux utiliser des fonctions distinctes pour créer le fichier de données (avec sa ligne d'en-tête) et écrire des lignes individuelles :

```
import csv
from sense_hat import SenseHat from datetime import datetime from pathlib import Path
from time import sleep

def create_csv(data_file):
:h open(data_file, 'w') as f: writer = csv.writer(f)
header = ("Date/time", "Temperature", "Humidity") writer.writerow(header)

f add_csv_data(data_file, data): with open(data_file, 'a') as f:
writer = csv.writer(f) writer.writerow(data)

sense = SenseHat()

base_folder = Path( file ).parent.resolve() data_file = base_folder/'data.csv'

create_csv(data_file) for i in range(10):
row = (datetime.now(), sense.temperature, sense.humidity) add_csv_data(data_file, row)
sleep(60)
```

Remarque : la première fois que tu écris dans un fichier, tu dois l'ouvrir avec `w` (mode écriture). Si tu souhaites y ajouter des données ultérieurement, tu dois utiliser `a` (mode ajout).

Quota de stockage des données

Ton expérience est autorisée à produire un maximum de 3 Go de données. Veille ainsi à calculer la quantité maximale d'espace de stockage nécessaire pour les données enregistrées de ton expérience, y compris les photos, et fais attention à ne pas dépasser 3 Go.

Journalisation avec logzero

La bibliothèque`logzero` Python permet de consigner facilement des notes sur ce qui se passe dans ton programme. Si tu récupères les données de ton expérience et qu'il manque de nombreuses données sans aucune explication, tu ne pourras pas savoir ce qui s'est passé. C'est pourquoi tu dois consigner autant d'informations que possible sur ce qui se passe dans ton programme. Consigne chaque itération de boucle et chaque fois qu'une fonction importante est appelée, et si tu as des conditions dans ton programme, consigne le chemin parcouru par le programme (`if` or `else`).

Voici un exemple basique sur la façon dont logzero peut être utilisé pour suivre les itérations de boucle :

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

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

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

Les deux principaux types d'entrée de journal que tu peux utiliser sont : `logger.info ()` pour consigner les informations et `logger.error ()` quand tu rencontres un erreur inattendue ou que tu gère une exception. Il existe aussi : `logger.warning()` et `logger.debug()`

Par exemple, si tu as une fonction pour détecter la nuit ou l'obscurité sur des photos, tu pourrais également enregistrer ces informations :

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

Si tu souhaites gérer une exception, et enregistrer que tu l'as gérée, tu peux utiliser `logger.error` :

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

Par exemple, si tu divises par zéro dans `do_something` cela crée l'entrée de journal suivante :

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

Ton programme continuerait de s'exécuter sans échec, mais plutôt que de ne voir aucune entrée de journal, tu verrais qu'une erreur s'est produite à ce moment.

Remarque : tu peux (et dois) utiliser à la fois la bibliothèque `csv` (pour l'enregistrement des données expérimentales) et la bibliothèque `logzero` (pour consigner les événements importants qui ont lieu pendant ton expérience).

Étape 5 Localiser l'ISS

En utilisant la bibliothèque Python `skyfield` , tu peux calculer la position des objets spatiaux dans notre système solaire. Cela comprend le Soleil, la Lune, les planètes et de nombreux satellites terrestres comme l'ISS. Tu peux utiliser l'emplacement actuel de l'ISS au-dessus de la Terre pour déterminer si elle survole la terre ou la mer, ou savoir au-dessus de quel pays elle passe.

Qu'est-il arrivé à la bibliothèque Ephem ?

Si ton équipe a participé aux défis précédents, tu te souviens peut-être que la bibliothèque `ephem` était utilisée pour calculer la position de l'ISS. Cette bibliothèque est devenue obsolète et a été remplacée par son successeur, `skyfield`.

Des données de télémétrie à jour sont nécessaires pour calculer avec précision la position de l'ISS (ou de tout autre satellite en orbite autour de la Terre). Pour t'éviter de devoir obtenir et manipuler ces données, le système d'exploitation de Vol offre le package Python, qui utilise `skyfield` pour créer un objet `ISS` que tu peux importer dans ton programme :

Données de télémétrie

Pour des calculs précis, `skyfield` requiert l'ensemble d'éléments à deux lignes (TLE) le plus récent pour l'ISS. TLE est un format de données utilisé pour transmettre des ensembles de paramètres orbitaux qui décrivent les orbites des satellites terrestres.

Lorsque tu importes l'objet `ISS` à partir de la bibliothèque `orbit`, une tentative de récupération des données TLE à partir d'un fichier `iss.tle` dans le dossier `/home/pi` est effectué. Si le fichier n'est pas présent mais qu'un connexion Internet est disponible les dernières données seront téléchargées automatiquement dans le fichier `iss.tle`, tu n'as donc pas besoin de t'en soucier

Toutefois, si ton kit Astro Pi n'a pas accès à Internet, tu dois télécharger manuellement les dernières données TLE de l'ISS (http:/www.celestrak.com/NORAD/elements/stations.txt), copier les trois lignes ISS dans un fichier appelé `iss.tle`, puis placer ce fichier dans le dossier `/home/pi`. Les données TLE ressembleront à ceci :

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

Lorsque ton code sera exécuté sur la Station spatiale, nous nous assurerons d'utiliser les données de télémétrie les plus précises et les plus à jour.

Tu peux utiliser `ISS` comme toute autre objet `earthsatellite` dans `skyfield` (voir la référence (https:/rhodesmill.org/skyfield/api-satellites.html#skyfield.sgp4lib.EarthSatellite) et exemples (https: /rhodesmill.org/sky field/earth-satellites.html)). Par exemple, voici comment calculer les coordonnées de l'emplacement de la Terre qui se trouve actuellement directement sous l'ISS :

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

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

Si tu n'es pas intéressé par la définition ou l'enregistrement de l'heure `t`, alors l'objet `ISS` offre également une méthode `coordinates` pratique que tu peux utiliser comme alternative pour récupérer les coordonnées de l'emplacement de la Terre qui se trouve actuellement directement sous l'ISS :

```
from orbit import ISS
location = ISS.coordinates() # Equivalent to ISS.at(timescale.now()).subpoint()
print(location)
```

Remarque : la position actuelle de l'ISS est une estimation basée sur les données de télémétrie et l'heure actuelle. Par conséquent, lorsque tu testes ton programme sur la version de bureau du système d'exploitation de Vol, tu dois vérifier que l'heure du système a été correctement réglée.

En outre, `location` est une `GeographicPosition`, tu peux donc consulter la documentation et voir comment accéder à ses éléments individuels :
(https:/rhodesmill.org/skyfield/api-topos.html#skyfield.toposlib.GeographicPositio n) :

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

Note que la latitude et la longitude sont des `Angles` et que l'altitude est une `Distance`. La documentation explique comment passer d'une représentation d'`Angle` à une autre (https:/rhodesmill.org/skyfield/api-units.html#skyfield.units.Angle) ou comment exprimer la html#skyfield.units.Distance) :dans différentes unités (https:/rhodesmill.org/skyfield/api-units.

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

Il existe plusieurs façons de représenter la latitude et la longitude, et il est important de choisir la méthode appropriée, surtout lorsque l'on travaille avec des logiciels et des bibliothèques qui attendent un format de données spécifique.

Le code ci-dessus donne la latitude et la longitude en utilisant le format des degrés décimaux (DD), où l'unité de mesure des coordonnées est le degré (°). Il y a 180° de latitude : 90° au nord et 90° au sud de l'équateur. Il y a 360° de longitude : 180° à l'est et 180° à l'ouest du premier méridien (avec une latitude égale à zéro et défini comme un point situé à Greenwich, en Angleterre). Pour spécifier précisément un emplacement, chaque degré peut être exprimé par un nombre décimal, par exemple (-28.277777, 71.5841666).

Une autre approche est le format degrés:minutes:secondes (DMS), où chaque degré est divisé en 60 minutes (') et chaque minute est divisée en 60 secondes ("). Pour une précision encore plus fine, on utilise des fractions de seconde indiquées par un point décimal. Le signe de l'angle indique si le point auquel se réfère la

coordonnée est au nord ou au sud de l'équateur (pour la latitude) et à l'est ou à l'ouest du méridien (pour la longitude).

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

Exemple : quel hémisphère ?

Si tu souhaites que ton expérience se déroule lorsque l'ISS se trouve au-dessus d'un emplacement terrestre particulier, tu peux utiliser les valeurs de latitude et de longitude pour déclencher une autre action. N'oublie pas que l'orbite de l'ISS ne survole pas toute la Terre, et que la plus grande partie de la surface de notre planète est constituée d'eau et non de terres. Ainsi, dans ta fenêtre expérimentale de 3 heures, les chances de passer au-dessus d'une ville ou d'un emplacement très précis seront faibles.

Pour savoir comment cela pourrait être utile dans ton programme, modifie le code ci-dessus afin qu'il affiche un message lorsque l'ISS se trouve au-dessus de l'hémisphère sud.

J'ai besoin d'un indice

Ton code doit ressembler à cela :

```
from orbit import ISS

location = ISS.coordinates() latitude = location.latitude.degrees if latitude < 0:
   print("In Southern hemisphere") else:
print("In Northern hemisphere")
```

Exemple : ISS éclairée par la lumière du Soleil

Le comportement de ton code peut différer selon que l'ISS est éclairée ou non par la lumière du Soleil. La bibliothèque `skyfield` permet d'obtenir facilement ces informations pour n'importe quel objet `EarthSatellite`. Peux-tu consulter la documentation et écrire un programme qui affiche toutes les 30 secondes si l'ISS est éclairée ou non par la lumière du Soleil ?

J'ai besoin d'un indice

Ton code doit ressembler à cela :

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

ephemeris = load('de421.bsp') timescale = load.timescale()

while True:
t = timescale.now()
ISS.at(t).is_sunlit(ephemeris): print("In sunlight")
else:
   print("In darkness") sleep(30)
```

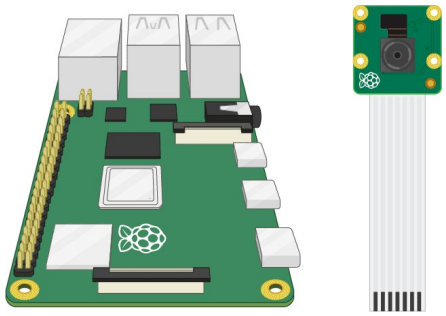
Remarque : en raison de l'altitude de l'ISS, le soleil se lève un peu plus tôt sur cette dernière que sur la surface terrestre, qui se trouve en dessous de la Station spatiale. De même, le soleil se couche sur l'ISS un peu plus tard que sur la surface terrestre, qui se trouve directement en dessous de l'ISS.

Étape 6 Enregistrer des images à l'aide de la caméra

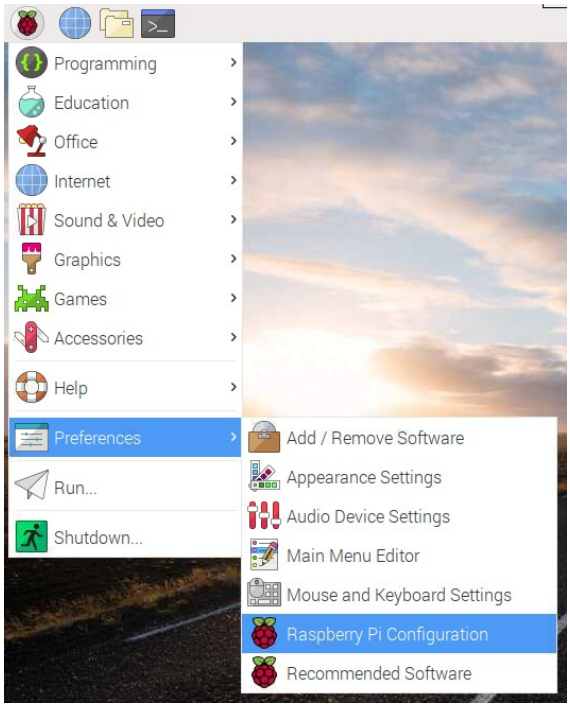
Tout d'abord, si tu ne l'as pas déjà fait, connecte ton module caméra au Raspberry Pi.

Connecter un module caméra du Raspberry Pi

- Vérifie que le Raspberry Pi est éteint.
- Localise le port du module caméra.
- Tire doucement sur les bords du clip en plastique du port.
- Insère le câble ruban du module caméra ; vérifie que le câble est dans le bon sens.
- Remets en place le clip en plastique.



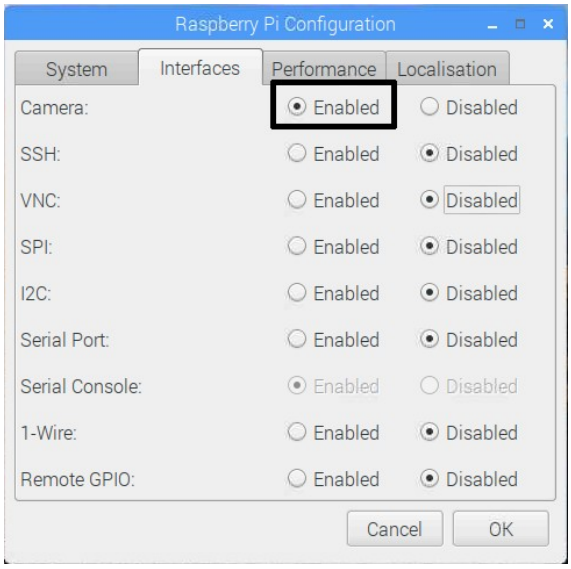
- Démarre le Raspberry Pi.
- Va dans le menu principal et ouvre l'outil de configuration du Raspberry Pi.



https://projects.raspberrypi.org/en/projects/code-for-your-astro-pi-mission-space-lab-experiment/print
https://esero.fr/projets/astro-pi/

Traduction et adaptation par ESERO France

- Sélectionne l'onglet Interfaces et vérifie que la caméra est activée :



- Redémarre le Raspberry Pi.

Une fois que tu as suivi ces étapes, rallume le Raspberry Pi et prends quelques photos de test

Prendre une photo avec PiCamera

Tu peux utiliser Python et le module `picamera` pour prendre des photos avec le Raspberry Pi et son module caméra.

- Importe la `PiCamera` et crée un objet `camera`

```
from picamera import PiCamera

camera = PiCamera()
```

- `capture()` `selfie.png` `/home/pi/`

```
from picamera import PiCamera

camera = PiCamera()
camera.capture('home/pi/selfie.png')
camera.close()
```

- Exécute ton code, puis utilise le Gestionnaire de fichiers ou le Terminal pour rechercher le fichier `selfie.png`.

Si tu n'as encore jamais utilisé le module caméra, commence par réaliser ce projet destiné aux débutants (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/). Tu trouveras la version française ici : https://esero.fr/wp-content/uploads/2020/10/T05.1b_Getting-started-with-astro-pi_teacher-guide_FR.pdf.
Reviens dans cette section une fois que tu auras utilisé les fonctions `picamera` de base de la bibliothèque Python.

L'extrait de code ci-dessous montre comment prendre une photo avec les modules caméra des Astro Pi, à l'aide de la bibliothèque `picamera` et comment l'enregistrer dans le répertoire adéquat. La bibliothèque `picamera` est très puissante et dispose d'une documentation excellente. (https: /picamera.readthedocs.io/en/latest/)

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

base_folder = Path( file ).parent.resolve() camera = PiCamera()
camera.resolution = (1296,972) camera.start_preview()
# Camera warm-up time sleep(2)
camera.capture(f"{base_folder}/image.jpg")
```

Si ton expérience porte sur le thème La vie dans l'espace, ton programme doit faire en sorte que toutes les images prises soient supprimées avant la fin de l'expérience :

```
(base_folder/"image.jpg").unlink()
```

Si ton expérience porte sur le thème La vie sur Terre, alors tu obtiendras des images magnifiques de la Terre vue de l'ISS. Même si ton programme traitera ces images et n'utilisera que les données extraites, nous te recommandons de ne pas supprimer toutes les images (à moins que ton programme ne génère tellement d'images que tu risques de manquer d'espace disque sur l'Astro Pi). En plus d'être un souvenir unique de ta mission, les images peuvent également t'aider à résoudre tout problème inattendu avec tes résultats expérimentaux. Voici quelques exemples d'images prises par la caméra infrarouge de l'Astro PI IR (https: /www.flickr.com/photos/raspberrypi). Si tu traites des images (par exemple avec la bibliothèque Python OpenCV), tu dois tester ton code sur certaines de ces images.

Le reste de cette étape s'applique principalement aux expériences de La vie sur Terre. Aucune image des expériences de La vie dans l'espace ne peut être enregistrée.

Données de localisation (La vie sur Terre)

Normalement, prendre des photos de la Terre depuis un hublot de l'ISS est réservé aux astronautes. Nous te recommandons d'enregistrer la position de la Station spatiale pour toutes les images que tu captes. Tu peux le faire en enregistrant la latitude et la longitude dans un fichier CSV, avec le nom de fichier associé à l'image.

Une meilleure méthode consiste à ajouter les informations d'emplacement dans les champs EXIF de chaque fichier image lui-même. Ces métadonnées sont « attachées » au fichier image et n'ont pas besoin du fichier de données CSV qui l'accompagne.

Dans l'extrait ci-dessous, une fonction`capture` est appelée pour prendre une image, après avoir défini les données EXIF sur la latitude et la longitude actuelles. Les coordonnées dans les données EXIF des images sont enregistrées en utilisant une variante du format degrés:minutes:secondes (DMS), et tu peux voir comment la fonction prend les données reçues `iss.coordinates ()` et les convertir dans un format adapté au stockage sous forme de données EXIF. L'utilisation de fonctions pour effectuer ces tâches permet d'avoir un programme organisé.

La complication supplémentaire ici est que la valeur des degrés ne peut pas être négative. Une information supplémentaire doit être incluse pour chaque valeur : la référence de la latitude et celle de la longitude. Cela indique seulement si le point auquel se réfère la coordonnée est au nord ou au sud de l'équateur (pour la latitude) et à l'est ou à l'ouest du méridien (pour la longitude). L'exemple ci-dessus s'afficherait ainsi : (28:16:40 S, 71:35:3 E).

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

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

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

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

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

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

# Capture the image camera.capture(image)

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

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

Localiser des images sur une carte

Lorsque les coordonnées sont incluses dans les métadonnées EXIF des images que tu as prises, tu peux utiliser des logiciels tels que DigiKam (inclus dans le système d'exploitation de Vol de bureau) ou un service en ligne pour localiser automatiquement sur une carte la position où l'image a été prise.

Tu peux également extraire les coordonnées des métadonnées EXIF d'une image grâce à un programme. Par exemple, l'image ci-dessous représente une partie des données du système d'exploitation de vol de bureau. En utilisant la librairie Python exif, tu peux en déduire que l'image a été prise aux coordonnées 35°24'20.0"N 112°10'46.2"O

Il s'agit du Grand Canyon, avec le lac Mead en haut à gauche !

Au lieu d'utiliser des données EXIF, il est possible de superposer du texte sur l'image visible elle-même, comme un filigrane. Cependant, cela risque toujours de masquer une partie utile de l'image, et peut entraîner une confusion pour le code qui examine la luminosité des pixels dans l'image. En outre, il est difficile de supprimer ces superpositions. Contrairement à la méthode EXIF, elles ne facilitent pas non plus le traitement automatique des images à partir des métadonnées, ni la recherche d'images à partir de l'emplacement où elles ont été prises. Par conséquent, nous te recommandons de ne pas utiliser la méthode du filigrane pour enregistrer la latitude et la longitude, et d'utiliser plutôt les données EXIF.

https://projects.raspberrypi.org/en/projects/code-for-your-astro-pi-mission-space-lab-experiment/print
https://esero.fr/projets/astro-pi/

Numéroter les fichiers

Avec une séquence d'images de l'ISS, tu peux également créer quelque chose de très intéressant : un timelapse, comme celui de la première section de ce projet. Tu peux le faire sur un Raspberry Pi avec une seule commande, si les images sont enregistrées avec des noms de fichiers pratiques qui incluent un numéro de séquence évident. Ainsi, la convention de nomenclature de tes fichiers image doit être `image_001.jpg`, `image_002.jpg`, etc. N'inclus pas d'espaces ni de symboles de ponctuation (à l'exception des tirets bas (_) et des tirets (-)) dans les noms de tes fichiers !

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

 base_folder = Path( file

 ).parent.resolve() camera =

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

Ensuite, une fois que tu as récupéré tes images de l'ISS, tu peux utiliser la commande suivante pour créer un timelapse. Selon la fréquence à laquelle tu as pris des images, tu peux modifier la valeur `–framerate` pour créer un mouvement plus fluide.

```
ffmpeg -framerate 10 -i %*.jpg -c:v libx264 -crf 17 -pix_fmt yuv420p timelapse.mp4
```

.

Cette étape doit être réalisée uniquement après la fin de ton expérience. Tu ne dois pas utiliser tes 3 heures d'expérience à bord de l'ISS pour essayer de créer un timelapse !

Photographie en basse lumière et photographie nocturne

Il est difficile de prendre des photos de nuit avec le module caméra de l'Astro Pi, car il est très peu probable que ton programme soit exécuté pendant que l'ISS survole une ville lumineuse sans couverture nuageuse. La caméra dispose d'une assez bonne sensibilité à la lumière, mais elle doit être utilisée avec les meilleurs réglages logiciels pour cette situation particulière, or, il est difficile d'anticiper ces réglages et de les inclure dans ton programme. Faire en sorte que la caméra s'adapte aux variations de luminosité en temps réel est également délicat, surtout lorsque la caméra se déplace par rapport à la source lumineuse, comme c'est le cas pour les Astro Pi de l'ISS.

Taille et nombre d'images

N'oublie pas que ton expérience peut produire une quantité de données limitée à 3 Go. Veille à calculer la quantité maximale d'espace nécessaire pour tes mesures, y compris les fichiers images enregistrés, et fais attention à ne pas dépasser 3 Go. Souviens-toi que la taille d'un fichier image dépend non seulement de la résolution, mais également de la quantité de détails présents dans l'image : une photo d'un mur blanc et nu sera plus petite qu'une photo de paysage.

Étape 7 Faire plusieurs choses à la fois

Dans un programme simple, chaque ligne est exécutée dans l'ordre, et les actions se produisent les unes après les autres. Le programme ne peut pas faire deux choses à la fois : il doit attendre qu'une tâche en cours d'exécution soit terminée avant de commencer la prochaine. Il s'agit d'un processus à thread unique.

Multithreading

Si tu as besoin de faire plus d'une chose à la fois, tu peux utiliser un processus multithread. Plusieurs bibliothèques Python permettent d'inclure ce type de mode multitâche dans ton code. Cependant, pour le faire sur les Astro Pi, tu as le droit d'utiliser uniquement la bibliothèque `threading`

Utilise la bibliothèque `threading` seulement si elle est absolument essentielle pour ton expérience. Gérer des threads peut être complexe, et comme ton expérience sera exécutée dans le cadre d'une séquence de programmes, nous devons nous assurer que le programme précédent s'est terminé sans problème avant de commencer le suivant. Les threads non fiables peuvent devenir incontrôlables et accaparer des ressources système, ils doivent donc être évités. Si tu décides d'utiliser des threads dans ton code, tu dois vérifier qu'ils sont tous gérés avec soin et fermés proprement à la fin de l'expérience. En outre, tu dois veiller à ce que les commentaires inclus dans ton code expliquent clairement comment cela est réalisé.

Étape 8 Exécuter ton expérience pendant 3 heures

Ton expérience disposera de 180 minutes pour être exécutée à bord de l'ISS. Par conséquent, ton code ne doit pas s'exécuter pendant plus de 3 heures et il doit mettre fin proprement à toutes les activités (par exemple, fermer la caméra, fermer les fichiers ouverts, effacer les contenus affichés sur la matrice LED). Au bout de 3 heures, ton code sera automatiquement interrompu par l'Astro Pi, mais cela peut entraîner la perte ou l'enregistrement incorrect de données, tu ne dois donc pas compter sur cela pour arrêter ton programme.

Afin d'arrêter ton programme après une durée spécifique, tu peux utiliser la bibliothèque Python `datetime` Cette bibliothèque permet de travailler facilement avec les heures et de les comparer. Effectuer cette action sans recourir à cette bibliothèque n'est pas toujours simple : il est facile de se tromper en utilisant les mathématiques normales. Par exemple, c'est facile de déterminer la durée écoulée entre 10 h 30 et 10 h 50 (soustraire 30 de 50), mais c'est un peu plus compliqué avec 10 h 44 et 11 h 17 (ajouter (60 - 44) à 17). Les choses deviennent encore plus compliquées si les deux heures sont réparties sur deux jours (par exemple, calculer la durée écoulée en minutes entre 23 h 07 le lundi 31 mai et 11 h 43 le mardi 1er juin). Grâce à la bibliothèque `datetime`, tu peux effectuer
beaucoup plus facilement ce type d'opération en créant des objets `datetime` que tu peux ajouter ou soustraire les uns par rapport aux autres.

En enregistrant et en stockant l'heure au début de ton expérience, tu peux vérifier à plusieurs reprises si l'heure actuelle est supérieure à l'heure de début, plus un certain nombre de minutes, de secondes ou d'heures. Cette différence est appelée `timedelta`.

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

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

Remarque : lorsque tu définis la durée d'exécution de ton code, tiens compte de la durée de réalisation d'un cycle par ta boucle. Ainsi, si tu veux exploiter l'intégralité du créneau de 3 heures (180 minutes) disponible pour ton expérience, mais que terminer la boucle de ton code prend 6 minutes, alors ton `timedelta` doit être de être `180-6 = 174` minutes, pour certain que ton code se termine avant la fin des 3 heures.

Étape 9 Uniquement pour les expériences de La vie dans l'espace : utiliser l'affichage LED

La matrice LED est le seul affichage disponible pour l'ordinateur Astro Pi, qui n'est jamais connecté à un moniteur ni à un écran de télé normal à bord de l'ISS. Si rien ne s'affiche sur la matrice pendant un certain temps, l'équipage peut se demander si l'Astro Pi est en panne. L'équipage devra alors perdre du temps à vérifier l'ordinateur et/ou à appeler le contrôle sol pour signaler un problème. Pour éviter cela, ton code doit régulièrement mettre à jour la matrice LED d'une manière ou d'une autre, pour indiquer que l'expérience est en cours.

Remarque : si ton expérience porte sur La vie sur Terre, tu ne dois pas utiliser la matrice LED. Lorsque l'Astro Pi exécute des expériences de La vie sur Terre, la matrice LED est désactivée et l'unité est placée sous un « capot » noir, afin d'empêcher les reflets et la lumière parasite de gâcher les images prises depuis le hublot de l'ISS.

L'argument `sense_hat` possède des fonctions pour écrire des messages sur la matrice LED ou éclairer des pixels individuels.

Paramétrer un seul pixel sur le Sense HAT

Tu peux utiliser la commande set-pixel pour contrôler les LED individuelles sur le Sense HAT. Pour ce faire, tu définis les variables x et y que la commande set-pixel prend.

x indique l'axe horizontal du HAT et peut avoir une valeur comprise entre 0 (à gauche) et 7 (à droite). y indique l'axe vertical du HAT et peut avoir une valeur comprise entre 0 (en haut) et 7 (en bas). Par conséquent, les coordonnées x, y et 0,0 correspondent à la LED en haut à gauche, et les coordonnées x, y et 7,7 correspondent à la LED en bas à droite.

La grille ci-dessus correspond au Raspberry Pi lorsqu'il est orienté de cette façon :

Fais un test avec cet exemple pour définir une couleur différente dans chaque coin de la matrice LED du Sense HAT. Tu devras utiliser la commande `set_pixel` à plusieurs reprises dans ton code, comme ceci :

```
from sense_hat import SenseHat

sense = SenseHat() # This clears any pixels left on the Sense HAT. You may not need this step and may want to choose when to add it in.

sense.clear() sense.set_pixel(0, 0, 255, 0, 0)
sense.set_pixel(0, 7, 0, 255, 0)
sense.set_pixel(7, 0, 0, 0, 255)
sense.set_pixel(7, 7, 255, 0, 255)
```

Test pour définir la couleur de différents pixels à l'aide de l'émulateur du Sense HAT :

Il s'agit de fonctions bloquantes. En d'autres termes, rien d'autre ne peut se produire pendant l'exécution de ces tâches. Donc, si tu utilisais `show_message` pour afficher une très longue chaîne de texte, cela occuperait un temps de mission précieux.
Par conséquent, les messages d'introduction au début de ton programme doivent durer moins de 15 secondes à partir du début du programme.

Évidemment, tu peux utiliser un thread de programme pour effectuer une autre tâche en arrière-plan pendant que des contenus sont affichés sur la matrice LED. Toutefois, comme indiqué ci-dessus dans la section « Faire plusieurs choses à la fois », tu dois éviter d'utiliser des threads, à moins qu'ils ne soient absolument essentiels pour ton expérience.

La matrice LED peut produire des images très colorées. Cependant, souviens-toi que l'ISS est un environnement de travail : tu dois éviter d’afficher des contenus qui clignotent trop, car ils pourraient déranger les astronautes.

```
from sense_hat import SenseHat from time import sleep
import random sense = SenseHat()

# Define some colours – keep brightness low g = [0,128,0]
o = [0,0,0]

# Define a simple image image = [
g,g,g,g,g,g,g,g,
o,g,o,o,o,o,g,o,
o,o,g,o,o,g,o,o,
o,o,o,g,g,o,o,o,
o,o,o,g,g,o,o,o,
o,o,g,g,g,g,o,o,
o,g,g,g,g,g,g,o,
g,g,g,g,g,g,g,g,
]

# Define a function to update the LED matrix def active_status():
# A list with all possible rotation values rotation_values = [0,90,180,270]
# Pick one at random
rotation = random.choice(rotation_values) # Set the rotation sense.set_rotation(rotation)

# Display the image sense.set_pixels(image) while True:
# Do stuff (in this case, nothing) sleep(2)
# Update the LED matrix active_status()
```

Tu dois mettre à jour l'écran toutes les 15 secondes au minimum. Si ton expérience a une période de « repos » plus longue, tu peux diviser la période d'attente :

```
sh.set_pixels(image) while True:
# Do stuff (in this case, nothing) sleep(15)
# Update the LED matrix active_status()
# More doing nothing sleep(15)
# Update LEDs again active_status()
```

Remarque : tu n'es pas autorisé à modifier le niveau de lumière des LED. N'utilise pas `sense.low_light`, `sense.gamma`, `sense.reset_gamma` ni `pisense.hat.screen.gamma` dans le programme que tu soumets.

Étape 10 Mise en réseau et processus du système

Pour des raisons de sécurité, ton programme n'est pas autorisé à accéder au réseau de l'ISS. Il ne doit pas essayer d'ouvrir une prise, d'accéder à Internet, ou d'établir une quelconque connexion réseau. Cela inclut les connexions réseau locales vers l'Astro Pi lui-même. Afin de tester ton programme, tu dois désactiver la connectivité sans fil et débrancher le câble Ethernet de ton Raspberry Pi pour vérifier que ton expérience s'exécute correctement sans connexion Internet.

De plus, ton programme n'est pas autorisé à exécuter un autre programme ou une commande que tu saisirais normalement dans la fenêtre du terminal du Raspberry Pi, comme `vcgencmd`.

Température du processeur

Il est fréquent de recourir à un sous-processus pour mesurer la température du processeur. Toutefois, nous te recommandons d'utiliser l'interface Température du processeur (https://gpiozero.readthedocs.io/en/stable/api_internal.html#cputemperature) fournie par GPIO Zero :

```
from gpiozero import CPUTemperature

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

Étape 11 Erreurs fréquentes

Mission Space Lab existe depuis quelques années maintenant et nous avons reçu des expériences étonnantes. Cependant, chaque année, certains projets fantastiques ne peuvent pas fonctionner à bord de l'ISS en raison de problèmes avec leur code final.

Il y a également des expériences qui fonctionnent mais ne génèrent pas de données pour leurs équipes en raison d'erreurs évitables. Voici quelques problèmes fréquents que tu dois éviter.

Ne compte pas sur les entrées effectuées par les utilisateurs

Ton programme ne doit pas s'appuyer sur l'entrée utilisateur faite via le joystick ou les boutons. L'équipage n'aura pas le temps de faire fonctionner manuellement les Astro Pi, donc ton expérience ne peut pas dépendre de l'entrée utilisateur. Par exemple, si un astronaute doit appuyer sur un bouton pour lancer une expérience, cette action n'aura jamais lieu et l'expérience ne s'exécutera pas. C'est aussi pourquoi les expériences menées sur l'équipage, comme les tests de vitesse de réaction humaine ou de mémoire, ne sont pas adaptées comme projets de Mission Space Lab.

Enregistre tes données

Assure-toi que toutes les données expérimentales sont écrites dans un fichier dès qu'elles sont enregistrées. Évite d'enregistrer des données dans une liste interne ou un dictionnaire au fur et à mesure, pour ensuite les écrire en intégralité dans un fichier à la fin de l'expérience, car si ton expérience se termine brusquement à cause d'une erreur ou parce qu'elle dépasse la limite de 3 heures, tu ne récupéreras aucune donnée.

N'utilise pas de chemins de fichier absolus

Assure-toi de ne pas utiliser de chemins spécifiques pour tes fichiers de données. Utilise la variable `_file_` comme indiqué dans « Enregistrer les données de ton expérience ».

Recherche les erreurs liées à la « division par zéro »

L'une des raisons fréquentes de l'échec des programmes est le fait qu'une fonction mathématique tente de diviser une valeur par zéro. Cela peut se produire si tu lis une valeur émanant d'un capteur et que tu l'utilises ensuite dans un calcul. Vérifie toujours que ton programme peut fonctionner si l'une des valeurs retournées par un capteur (en particulier l'accéléromètre) est égale à zéro.

Vérifie que ton code peut gérer les erreurs (exceptions)

Une `exception` est un événement qui survient pendant l'exécution d'un programme et qui perturbe l'exécution normale de ses instructions. Par exemple, si ton programme prend deux nombres et les divise, ceci fonctionnera dans de nombreux cas :

```
>>> a = 1
>>> b = 2
>>> c = a / b
>>> print(c) 0.5
```

Mais si le deuxième nombre est zéro, l'opération de division échouera :

```
>>> a = 1
>>> b = 0
>>> c = a / b
aceback (most recent call last): File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Afin de gérer cette situation potentielle, tu peux gérer le cas du zéro dès le départ :

```
if b != 0:
   c = a / b else:
print("b cannot be zero")
```

Il est également possible d'essayer de terminer l'opération, mais de gérer l'exception si elle se produit :

```
try:
c = a / b
ept ZeroDivisionError: print("b cannot be zero")
```

Voici un bon exemple d'exception qui peut survenir lorsque tu utilises le Sense HAT : ton programme utilise une variable comme valeur de couleur d'un pixel, mais la valeur attribuée à la variable dépasse la plage autorisée (entre 0 et 255).

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

Il est important d'anticiper tous les endroits de ton programme où une variable pourrait atteindre une valeur qui entraînerait des problèmes. Par exemple, si tu utilises la mesure de l'humidité pour déterminer le niveau de rouge des pixels, assure-toi que cette valeur ne peut pas dépasser la plage de 0 à 255, pas seulement pendant les tests, mais dans toutes les situations possibles :

```
red = int(max(0, min(sh.humidity / 100 * 255, 255)))
```

Cette ligne de code signifie que si la mesure de l'humidité est égale ou inférieure à `0`, la valeur de `red` sera 0, et si la mesure est de 100 ou plus, la valeur `red` sera de 255. Pour les mesures comprises entre 0 et 100, la valeur de `red` sera proportionnelle. En outre, la valeur `red` sera toujours un entier.

Plusieurs exceptions

La manière la plus simple de traiter les exceptions est de repérer toutes les exceptions et de les gérer de la même manière :

```
try:
   do_something() except:
print("An error occurred")
```

Cependant, cela ne t'indique pas ce qui s'est mal passé dans ton programme. Tu dois plutôt envisager quels types d'exceptions peuvent se produire. Il est possible de gérer différentes exceptions de différentes manières :

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

En combinant évitement et bonne gestion des exceptions, tu peux éviter les erreurs qui empêcheraient ton programme de terminer son exécution et ainsi, tu ne seras pas déçu. Il serait dommage de récupérer les journaux d'une expérience ratée pour se rendre compte qu'il y avait une exception que tu aurais pu gérer, ou un message d'erreur n'ayant rien révélé de ce qui s'est mal passé.

Étape 12 Exemple pratique

Voici un exemple d'idée d'expérience Mission Space Lab sur l'Astro Pi : l'équipe de CoderDojo Tatooine veut déterminer si l'environnement de l'ISS est affecté par la surface terrestre qu'elle survole. L'ISS se réchauffe-t-elle quand elle survole un désert, ou devient-elle plus humide quand elle se trouve au-dessus de la mer ?

Cet exemple servira de modèle pour illustrer comment tu peux combiner tous les éléments décrits jusqu'à présent dans ce guide pour planifier et écrire ton programme informatique.

Pour cet exemple spécifique, le programme de l'expérience doit :

- Prendre régulièrement des mesures de la température et de l'humidité toutes les 30 secondes, et
- enregistrer les valeurs dans un fichier CSV Calculer la latitude et la longitude de l'ISS et enregistrer ces informations dans le fichier CSV
- Prendre une photo à l'aide de la caméra IR d'Astro Pi, qui est orientée vers la Terre depuis un hublot, afin de recueillir des données pour savoir si la couverture nuageuse peut également jouer un rôle.
- Écrire les données de latitude et de longitude dans le fichier CSV et également dans les balises EXIF des images, dont les noms de fichier sont numérotés de façon séquentielle.
- Gérer les erreurs inattendues et enregistrer les détails.

Planifier tes sessions de codage

Pour t'aider avec la planification, nous avons mis au point des conseils utiles pour la phase 2 qui faciliteront le codage pour ton équipe.

Conseils pour la planification et le déroulement des sessions de codage

Comment aborder l'écriture du programme pour la phase 2 de Mission Space Lab

- Lis l'intégralité de ce guide. Familiarise-toi avec les exigences que ton programme doit respecter afin d'être admis à la phase suivante et de fonctionner sans problème sur les Astro Pi de l'ISS. Lis les conseils utiles fournis dans ce guide pour connaître la meilleure façon de développer ton programme et tirer le meilleur parti de tes résultats expérimentaux.

Liste les tâches principales

- Réunis ton équipe pour commencer à établir les grandes lignes du fonctionnement de ton programme. Pour cela, tu peux rassembler le groupe et demander à chacun d'exprimer ses idées, ou tous les membres de l'équipe peuvent travailler individuellement et se réunir pour comparer leurs résultats.

- Utilise un tableau blanc ou une grande feuille pour lister toutes les tâches clés que ton programme devra effectuer. À ce stade, tu n'as pas besoin de te soucier de l'ordre ou des fonctions et commandes réelles, il suffit de noter les actions spécifiques qui doivent être réalisées. Voici à quoi cela ressemblerait pour l'exemple de scénario ci-dessus :

- Examine chaque tâche de plus près et réfléchis à la possibilité de la diviser en sous-tâches plus petites. En outre, certaines actions peuvent-elles être raisonnablement combinées avec d'autres actions ? Vérifie également s'il y a des tâches qui doivent être répétées.

- Essaie de mettre l'ensemble dans un ordre logique, en traçant des lignes pour relier les différentes tâches. Ça sera un peu la pagaille, mais tant mieux ! Tu vas probablement découvrir que certaines tâches se répètent. C'est le bon moment pour présenter ou revoir les concepts de programmation de la répétition et des boucles.

- Si certaines tâches se répètent, apparaissent-elles une seule fois dans ton diagramme, avec des lignes qui les traversent plusieurs fois, ou les retrouves-tu à plusieurs endroits ? Discute avec ton équipe du fait que les tâches répétées ne doivent être codées qu'une seule fois, afin que des parties du programme puissent être réutilisées.

Crée un organigramme

- Prends une feuille vierge ou trouve une zone libre sur ton tableau blanc (n'oublie pas de copier ou de prendre en photo les contenus avant d'effacer quoi que ce soit). Réorganise les étapes et leur déroulement en un diagramme plus ordonné, par exemple en tournant dans le sens des aiguilles d'une montre sur la feuille ou en commençant en haut et en descendant. Essaie quelques versions différentes pour savoir laquelle est la plus facile à suivre. Inclus un bloc « début » et « fin » pour indiquer clairement où le programme commence et se termine. Lors de ces étapes, y a-t-il des actions que tu dois effectuer ?
  Le résultat final est ce que l'on appelle un organigramme : un diagramme qui décrit toutes les tâches d'un programme, dans le bon ordre, mais qui ne contient aucune commande de langage de programmation.

- Passe en revue ta liste de tâches et identifie les actions manquantes. S'il manque des fonctionnalités, ajoute-les dans ton organigramme.

- Tu dois également réfléchir pour déterminer où traiter les exceptions et les erreurs dans ton programme. La plupart des expériences présentent une boucle principale qui s'exécute à plusieurs reprises pendant la période de 3 heures. Une erreur inattendue rencontrée dans cette boucle pourrait être désastreuse si elle provoque l'arrêt du programme et empêche la collecte de données supplémentaires. Tu dois donc réfléchir aux scénarios possibles. Par exemple, si tu lis les données d'un capteur, que se passera-t-il si le relevé te donne un résultat inattendu ? Ton programme saura-t-il gérer cette situation ? Comment gères-tu les erreurs liées au matériel ?

Attribue des tâches aux membres de l'équipe

- Donne des noms descriptifs à chaque bloc de tâches.

- Attribue la responsabilité de chaque bloc à différents membres de l'équipe. Pense aux niveaux d'expérience et aux capacités de programmation des membres de l'équipe et attribue les tâches en conséquence. En fonction du nombre de personnes présentes dans ton équipe et de la complexité de ton programme, il peut être judicieux d'affecter plus d'une personne à un bloc ou à une fonction spécifique.

- N'oublie pas que quelqu'un doit être responsable de l'assemblage du programme final qui contiendra les divers appels de fonction dans le bon ordre.

Commence à coder !

- Tu vas probablement découvrir que certaines fonctions sont vraiment faciles à créer en utilisant les bibliothèques Python recommandées, peut-être même avec une seule ligne. D'autres seront plus complexes, et nous avons inclus des extraits de code utiles dans le document des exigences de codage (par exemple, pour ajouter des informations de latitude et de longitude aux données EXIF d'une photo) que tu peux copier dans ton projet. Afin de concevoir des fonctions plus complexes, si nécessaire, tu peux utiliser la même approche pseudo-code que celle utilisée dans l'ensemble du programme.
- Rappelle à ton équipe que vous ne pouvez pas installer de bibliothèques Python supplémentaires ni accéder à Internet sur les ordinateurs Astro Pi de l'ISS, vous ne devez donc pas utiliser de commandes faisant une requête Web ou recherchant une information à partir d'une source en ligne.
- Incite chaque membre ou sous-équipe responsable d'une partie spécifique du programme à coder sa section de façon à ce qu'elle fonctionne indépendamment des autres, tout comme les exemples donnés dans le document sur les exigences de codage. Suggère-leur d'ajouter des commentaires et des docstrings à mesure de leur progression.
- Réunissez-vous régulièrement pour discuter des progrès accomplis et travailler en groupe pour résoudre les problèmes majeurs. Il est utile de mettre à jour l'organigramme de ton pseudo-code pour refléter les changements que ton équipe estime nécessaires au fur et à mesure de l'écriture du programme.
- Garde à l'esprit la date butoir de soumission de ton programme. Si tu manques de temps, certaines parties du programme peuvent-elles être laissées de côté ? Si tu prévois d'analyser les résultats en temps réel, cela peut-il être fait après l'exécution du programme, lorsque tu auras reçu les résultats ?

Teste ton programme

- N'oublie pas de tester ton programme dans la version de bureau du système d'exploitation de Vol : ouvre un terminal, tape `python3 main.py` et vérifie que ton programme se termine au bout de 3 heures sans générer d'erreurs (y compris les exceptions qui ont pu être consignées et enregistrées dans le journal).
- Vérifie que les données et les images collectées par ton programme correspondent à ce que tu attendais.
- Passe en revue la liste de contrôle des exigences indiquée dans la dernière étape de ce guide.

Le code de l'expérience

Voici à quoi peut ressembler le code final qui concrétise ton idée d'expérience :

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

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

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

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

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

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

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

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

    # Capture the image
    camera.capture(image)


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

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

# Set up Sense Hat
sense = SenseHat()

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

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

# Initialise the photo counter
counter = 1
# Record the start and current time
start_time = datetime.now() now_time
= datetime.now()
# Run a loop for (almost) three hours
while (now_time < start_time + timedelta(minutes=178)):
    try:
        humidity = round(sense.humidity, 4)
```

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

Voici un extrait du fichier `data.csv` qui est généré :

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

Note que le traitement des exceptions dans ce programme est plutôt rudimentaire : toutes les exceptions soulevées seront repérées et consignées. Par conséquent, il est très peu probable qu'un tel programme se termine brusquement et affiche une erreur. Même si des erreurs sont générées et que le programme ne parvient pas à atteindre son objectif, cela ne deviendra évident qu'en vérifiant les journaux à la recherche d'erreurs. Lorsque tu testes ton programme, vérifie également les journaux qu'il génère.

Étape 13 Teste ton code sur le système d'exploitation de Vol

Nous arrivons à la dernière partie de la phase 2, qui est aussi la plus importante.

Avant de soumettre ton programme, il est essentiel de le tester sur un Astro Pi qui fonctionne avec la version de bureau du système d'exploitation de Vol inclus dans les kits de l'ESA.

Les packages installés dans cette version spéciale du système d'exploitation du Raspberry Pi correspondent aux packages du système d'exploitation de Vol installé sur les Astro Pi de l'ISS. Il simule tout simplement la même configuration qu'à bord de l'ISS, y compris les versions exactes des packages. Vérifier que ton programme fonctionne sans erreur dans cet environnement est la meilleure façon de garantir que ton expérience respecte notre procédure de test et qu'elle peut fonctionner sur les Astro Pi de l'ISS sans aucune modification.

Lorsque nous recevrons ton code, nous l'exécuterons sur le système d'exploitation de Vol réel. Des centaines d'équipes soumettent chaque année des programmes au défi et, malheureusement, nous ne sommes pas en mesure de rechercher des erreurs ou de déboguer des erreurs de code complexes : si ton programme ne fonctionne pas correctement lorsque nous le testons sur le système d'exploitation de Vol, ton équipe ne sera pas admise à la phase 3 et ton code ne sera pas exécuté sur l'ISS.

Ainsi, pour donner à ton programme les meilleures chances de réussite, il est important que tu le testes autant de fois que possible. Teste minutieusement ton programme, débogue les erreurs et vérifie qu'il respecte les exigences de codage. Il est particulièrement important d'envisager les erreurs qui pourraient survenir lors de l'exécution de ton programme sur le système d'exploitation de Vol des Astro Pi, telles que les erreurs de chemin d'accès de fichier ou l'écrasement de fichiers.

Contrôle final

Afin que ton programme puisse fonctionner en toute sécurité et correctement à bord de l'ISS, il doit respecter certaines exigences. Si tu as travaillé en te fiant à ce guide, ton programme doit déjà être parfait. Toutefois, avant de soumettre ton programme, effectue cette dernière étape et vérifie à nouveau que :

- [ ] Ton expérience ne repose pas sur l'interaction avec un astronaute.
- [ ] Ton programme est écrit en Python 3 et il est nommé `main.py`. Il doit fonctionner sans erreurs lorsqu'il est exécuté sur la ligne de commande du système d'exploitation de Vol en utilisant `python3 main.py`.
- [ ] Ton programme ne s'appuie sur aucune autre bibliothèque que celles énumérées dans ce guide.
- [ ] Ton programme surveille sa durée d'exécution et s'arrête après 3 heures.
- [ ] Ton programme ne contient pas de langage grossier ou injurieux.

- [ ] Ton programme est documenté et facile à comprendre. Il n'y a aucune tentative de cacher ou d'obscurcir ce que fait le code.
- [ ] Ton programme ne contient pas de code malveillant, c'est-à-dire du code qui tente délibérément de perturber le fonctionnement du système.
- [ ] Ton programme ne démarre pas de processus système et n'exécute pas un autre programme ou une commande généralement saisie sur le terminal, par exemple `vcgencmd`.
- [ ] Ton programme n'a pas recours à la mise en réseau.
- [ ] Si ton programme utilise des threads, il le fait uniquement en ayant recours à la bibliothèque de `threading`. Les threads sont gérés avec soin, fermés proprement, et leur utilisation est clairement expliquée par des commentaires ajoutés dans le code.
- [ ] Ton programme enregistre les données uniquement dans le dossier où se trouve le fichier Python principal, comme décrit dans ce guide (c'est-à-dire en utilisant la variable `_file_`). Tu n'utilises aucun nom de chemin absolu.
- [ ] Tous les fichiers générés par ton programme portent des noms qui incluent uniquement des lettres, des chiffres, des points (.), des tirets (-) ou des tirets bas (_).
- [ ] Ton programme ne consomme pas plus de 3 Go d'espace pour stocker des données.
- [ ] Si tu as choisi le thème La vie dans l'espace, ton programme doit faire en sorte qu'aucune image ni vidéo prise ne reste stockée dans le dossier de l'expérience après la fin de l'expérience.
- [ ] Si tu as choisi le thème La vie dans l'espace, ton programme doit afficher régulièrement des messages ou des images sur la matrice LED pour indiquer qu'une expérience est en cours.
- [ ] Si tu as choisi le thème La vie sur Terre, ton programme ne doit pas utiliser la matrice LED.

Exécute ton code

Connecte le Sense HAT et le module caméra (si nécessaire).

Démarre le Raspberry Pi en exécutant la version de bureau du système d'exploitation de Vol.

Le point d'entrée principal de ton expérience doit être nommé `main.py`. Exécute-le directement avec la commande suivante uniquement :

```
python3 main.py
```

Ton code doit fonctionner pendant 3 heures, puis s'arrêter.

Une fois que le projet est terminé, examine les fichiers de sortie qu'il a générés. As-tu prévu de recevoir des fichiers image de la caméra ? Des fichiers de données ? Ou quelque chose d'autre ? Trouves-tu des rapports d'erreurs dans tes journaux ?

Si tu constates des erreurs, ou si l'expérience ne fait pas ce que tu avais prévu, tu dois résoudre ces problèmes avant de soumettre ton code, afin de te donner toutes les chances d'atteindre la phase de jugement final.

Remarque : lors des tests, il est conseillé de désactiver la connexion Internet du Raspberry Pi pour s'assurer que ton expérience n'a pas accès à Internet.
