Belgium

esero

Fiche pour l'enseignant

Mission Space Lab

Apprendre à programmer en Python en mesurant la vitesse de l'ISS

Grâce à cette ressource, les élèves acquières des compétences de base en programmation tout en s'essayant au défi de la Mission Space Lab : estimer la vitesse de la Station Spatiale Internationale à partir de photos prises à son bord en programmant le mini-ordinateur Astro Pi.

esa

belspo

La Scientothèque
L'ÉGALITÉ DES CHANCES PAR LES SCIENCES

ULB

Avec le soutien financier de

Mission Space Lab

Apprendre à programmer en Python en mesurant la vitesse de l'ISS

Guide de l'enseignant

Grâce à cette ressource, les élèves acquières des compétences de base en programmation tout en s'essayant au défi de la Mission Space Lab : estimer la vitesse de la Station Spatiale Internationale à partir de photos prises à son bord en programmant le mini-ordinateur Astro Pi.

La Scientothèque
L'ÉGALITÉ DES CHANCES PAR LES SCIENCES

ULB

Table des matières

Vue d'ensemble de l'activité

Public	Matières	Durée
S3 à S6	Informatique, Physique	1 à 2 périodes

Résumé

Dans cette activité, une introduction aux éléments les plus important du langage Python est présentée au travers du défi Mission Space Lab : estimer la vitesse de l'ISS. Cela permet d'acquérir les notions minimales pour affronter les activités suivantes de Mission Space Lab tout en permettant déjà une expérience complète.

Objectifs d'apprentissage

- Maitrisier les bases de la programmation en language Python
- Révision en cinématique
- Révision en gravitation Newtonienne
- Révision sur le mouvement circulaire uniforme

Matériel

- Ordinateur
- Accès à Internet
- Compte Google (Gmail ou autre) *(facultatif)*

Métiers STEM en lien

- Informaticien
- Physicien
- Chercheur
- Ingénieur

Auteurs : La Scientothèque (ESERO Belgium)

Date de publication : Septembre 2025

INTRODUCTION

Mission Space Lab

Le projet Mission Space Lab, proposé par ESERO aux professeurs du secondaire, vise à éveiller l'intérêt des élèves pour les sciences spatiales et l'exploration de l'espace. Il s'agit d'une initiative concrète qui permet aux jeunes de se confronter à la recherche scientifique en conditions réelles, à bord de la Station spatiale internationale (ISS).

Ce projet poursuit plusieurs objectifs pédagogiques ambitieux :

- **Sensibiliser** les élèves aux merveilles de l'univers et aux enjeux de l'exploration spatiale.
- **Permettre** aux jeunes de réaliser des expériences scientifiques concrètes.
- **Développer** leurs compétences en programmation, analyse de données et résolution de problèmes.
- **Favoriser** la collaboration et le travail en équipe pour mener à bien un projet scientifique complexe.

Lien avec les autres activités Mission Space Lab

Le challenge **Mission Space Lab**, proposé par l'**Agence Spatiale Européenne (ESA)** en collaboration avec **ESERO**, invite les élèves à estimer la vitesse de la Station Spatiale Internationale (ISS) à partir de photographies prises depuis un hublot à l'aide d'un **AstroPi** (un micro-ordinateur fonctionnant grâce à un Raspberry Pi). Il s'agit d'un projet éducatif ambitieux et particulièrement stimulant, qui permet d'explorer des compétences variées en programmation tout en mobilisant des notions fondamentales du cours de physique.

Toutefois, la participation au challenge dans sa version originale requiert des connaissances préalables en programmation, ce qui peut constituer un obstacle pour certains élèves.

Afin de rendre cette expérience accessible à un plus grand nombre, cette ressource propose un **scénario interactif** permettant une première initiation à la programmation tout en découvrant les principes essentiels du challenge Mission Space Lab. Cette approche "deux-en-un" vise à offrir une expérience éducative riche, même aux élèves débutants en programmation.

En quoi cette version diffère-t-elle du challenge officiel ?

Cette version simplifiée se concentre principalement sur la découverte des **structures de données couramment utilisées en Python**. Les aspects les plus complexes du traitement des données, notamment le calcul précis des variations de position de l'ISS, ont été allégés. L'objectif est de permettre une compréhension progressive des concepts tout en conservant l'essence scientifique du projet original.

Les contenus liés à la physique, quant à eux, sont intégralement conservés afin de maintenir l'intérêt pédagogique du projet dans le cadre des cours scientifiques.

Objectifs pédagogiques

À l'issue de cette activité, les élèves auront acquis les bases de la programmation en **Python** et pourront, s'ils le souhaitent, poursuivre avec le challenge **Mission Space Lab** dans sa version complète. Cette dernière nécessite une maîtrise plus avancée, notamment en traitement d'image et en interaction directe avec les ordinateurs AstroPi, aspects volontairement contournés dans cette ressource introductive.

Ressources complémentaires

Pour les enseignants qui le souhaitent, une **introduction à Python** indépendante du contexte spatial est également proposée. Elle permet aux élèves de se familiariser d'abord avec les bases du langage avant d'aborder le challenge en une seule séquence. Les deux approches – intégrée ou séquentielle – sont possibles et peuvent être choisies en fonction du niveau et des besoins spécifiques de votre classe.

Outils mis à disposition

Il est possible de réaliser cette activité de deux manières différentes :

- **Avec Google Colab :** c'est un environnement de programmation en ligne accessible depuis un navigateur web et sans aucune installation. Son avantage principal est de permettre de mélanger texte et code. Le code de l'activité peut ainsi être progressivement écrit et testé dès qu'il est expliqué, évitant de d'éparpiller son attention sur plusieurs supports/environnements.
- **Avec Thonny :** c'est l'environnement de développement intégré (IDE) préconisé par l'ESA pour la réalisation de l'ensemble du projet. Il permet l'exécution de code localement sans connexion internet et propose un simulateur Astro Pi pour tester le code dans des conditions réalistes avant l'envoi à l'ESA.

Caractéristique	Google Collab	IDE Thonny
Pas d'installation requise	✔	✘
Simulateur Astro Pi	✘	✔
Configuration non requise	✔	✘
Internet non requis	✘	✔
Fiche élève non requis	✔	✘
Compte Google non requis	✘	✔

Voici le lien vers le notebook Google Collab associé à cette ressource :

https://colab.research.google.com/drive/1PbifrKEHZOWEG6kR1_2nek_iOCoJLFwv?usp=sharing

ACTIVITE AVEC GOOGLE COLAB

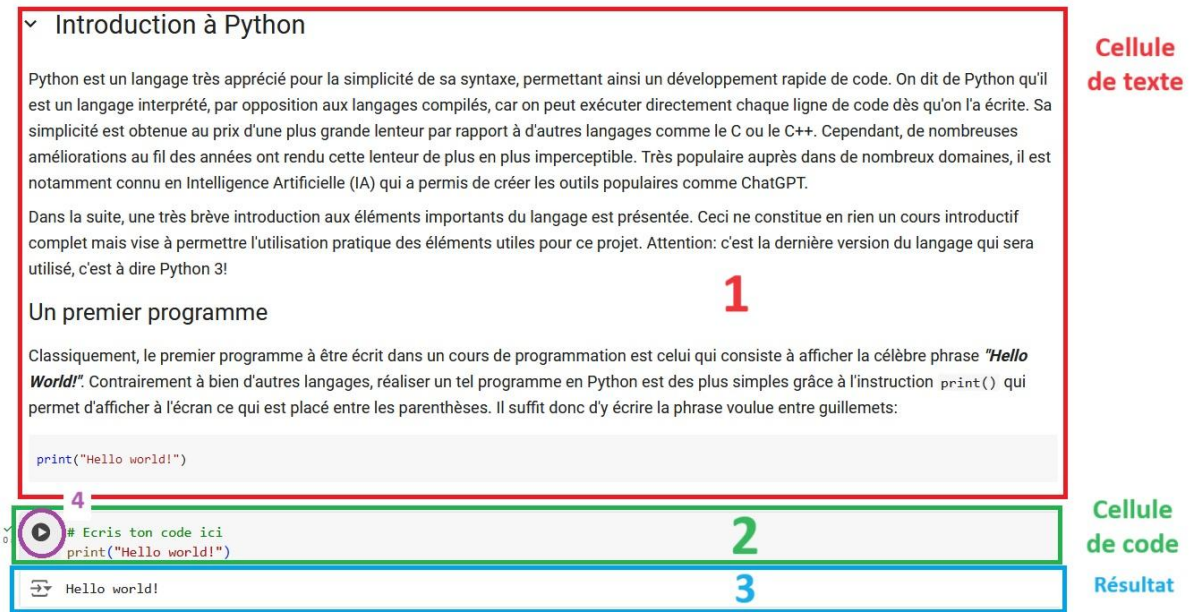
Présentation

Google Colab, ou Colaboratory, est une plateforme en ligne gratuite qui permet d'exécuter du code Python dans un environnement cloud. C'est comme un ordinateur puissant dans votre navigateur, accessible à tous et sans installation !

Parfait pour l'apprentissage automatique, la science des données et l'enseignement, Colab offre des notebooks Jupyter qui vous permettent d'écrire et d'exécuter du code, d'analyser des données, de créer des visualisations et bien plus encore. Le tout, avec l'accès à des GPU et TPU puissants pour le développement en Intelligence Artificielle.

Un notebook Jupyter est constitué de *cellules* de deux types différents:

- **Cellule de texte (voir 1)** : Elles contiennent du texte formaté, des images, des liens et des formules mathématiques écrite en Markdown. Vous pouvez les utiliser pour rédiger des explications, des titres ou des commentaires.
- **Cellule de code (voir 2) :** Elles contiennent du code exécutable dans un langage de programmation spécifique (Python, R, etc.). Vous pouvez les utiliser pour analyser des données, créer des visualisations, ou implémenter des algorithmes.



L'exécution des cellules de code se fait séquentiellement, ce qui permet de suivre le déroulement d'un programme ou d'une analyse. Pour exécuter le code d'une cellule de code, il suffit d'appuyer sur le bouton *Play* à gauche (entouré en 4). Les résultats de l'exécution du code s'affichent directement sous la cellule correspondante (voir 3).

Pour pouvoir travailler avec Google Collab, **il faut cependant disposer d'un compte Google**. Toute adresse email permettant de se conencter à Youtube, Gmail ou un autre service Google comme les adresses @gmail.com fonctionnent.

Sauvegarder son travail

La sauvegarde de toute modification est régulièrement effectuée automatiquement dans Google Colab. Il n'est pas donc pas nécessaire d'effectuer une action. Il est néanmoins possible de manuellement enregistrer le travail en allant dans **Fichier > Enregistrer** ou en utilisant les touches **CTRL+S.**

Exporter le code et sauvegarde du notebook

Il pourrait être utile d'exporter le code pour l'utiliser dans Thonny par exemple. Il y a deux formats d'exportation disponibles :

- *Notebook Jupyter :* c'est le format notebook identique à Google Collab qui peut être utilisé dans tout client Jupyter Notebook sur votre ordinateur sans internet. L'extension du fichier est *.ipynb et peut être téléchargé sous **Fichier > Télécharger > Télécharger le fichier .ipynb**
- *Code Python :* c'est un simple document texte contenant uniquement le contenu des cellules de code tandis que les cellules de texte sont mis en commentaires. L'extension du fichier est *.py et peut être téléchargé sous **Fichier > Télécharger > Télécharger le code .py**

Pour garder votre propre copie du notebook Google Collab dans votre propre espace Google Drive, il convient d'aller dans **Fichier > Enregistrer une copie dans le Drive.** Vous pourrez alors ensuite le modifier et le partager à vos élèves à votre guise comme vous le feriez pour un document Google Docs ou Sheet.

Mise en place en classe

Il convient de prévoir un ordinateur par étudiant et disposant d'une connection internet fiable. Ensuite, les élèves sont invités à ouvrir le navigateur web pour accéder au notebook à partir du lien suivant :

https://colab.research.google.com/drive/1PbifrKEHZOWEG6kR1_2nek_iOCoJLFwv?usp=sharing

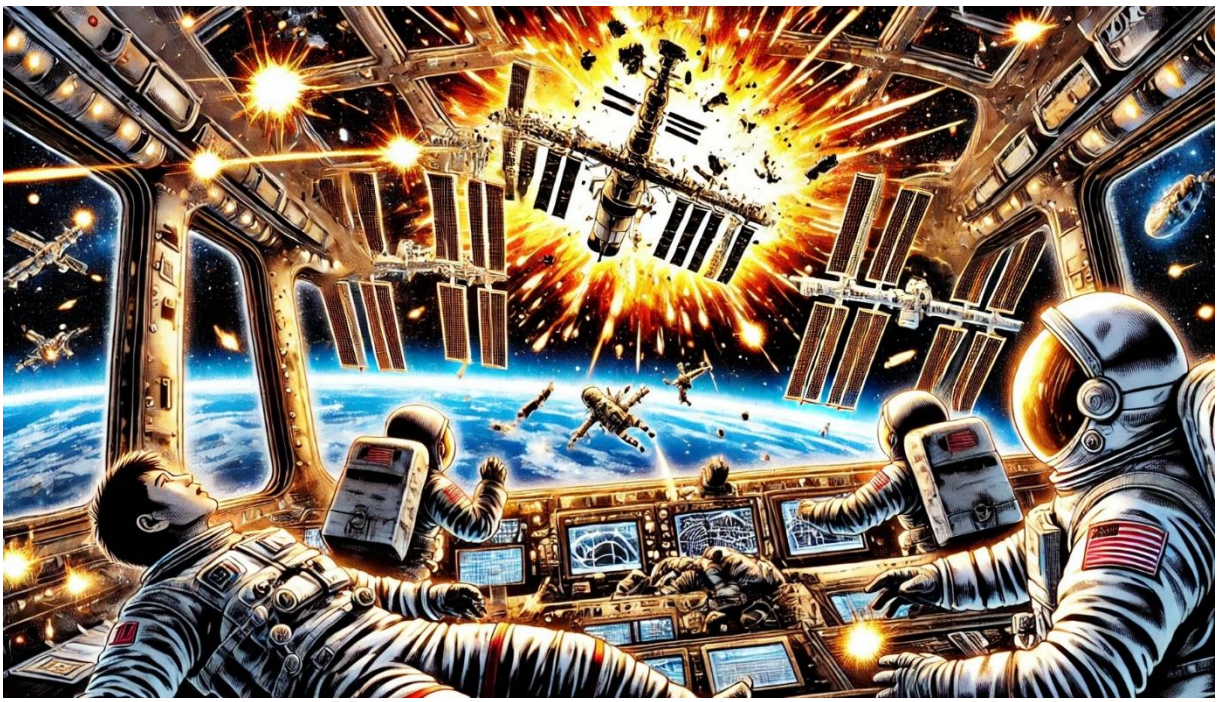
En haut à droite, ils doivent se connecter sur un compte Googe en appuyant sur *Connexion* :

Si vous ne voulez pas perdre de temps sur la connection sur le compte Google, il est préférable de créer des comptes Google à l'avance et de s'y connecter sur les ordinateurs avant l'activité.

Tout est prêt ! Vous pouvez commencer l'activité !

Déroulé de l'activité & Réponses aux exercices

Les alarmes résonnent dans la station spatiale internationale (ISS) depuis déjà presque 10 minutes! Elle se sont déclenchées au moment de l'explosion qui a ravagé le module Unity et une partie importante de la station. Depuis, plus de communication radio avec la Terre…



Vous êtes seuls, peut-être en danger de mort. Tu repenses aux dernières années écoulées : tes cours de physique de secondaire et ton génial professeur, Monsieur Ducobu, qui t'inçita à poursuivre des études de biophysicien.ne à l'Université; ton inscription au programme de sélection des futurs astronautes de l'agence spatiale européenne – au départ un défi entre copains qui est devenu de plus en plus sérieux au fur et à mesure des étapes; ensuite l'entrainement au centre de Cologne; puis ta sélection dans la nouvelle promotion, jusqu'à ton décollage et ton arrivée dans la station spatiale internationale...

Mais y aura-t-il une suite à cette folle aventure?

La prochaine heure sera déterminante.

L'explosion et la dépressurisation du module ont peut-être induit un changement dans la vitesse de la station qui mènera inexorablement à sa rentrée dans l'atmosphère et à sa désintégration. Heureusement, les moteurs du module Zvesda sont en état de fonctionnement. Mais comment savoir si vous devez ou non les allumer? Tout dépend de la vitesse de la station. Comment la mesurer? Tu te souveniens que le micro-ordinateur AstroPi, situé dans le module européen Columbus, est connecté à une caméra qui prend des photos régulières de la surface de la Terre. Tu pourrais écrire un petit programme en Python pour calculer la vitesse de la station… C'est parti!

Avant tout, il faut récupérer les images prises par l'AstroPi. Pour ce faire, tu disposes du code ci-dessous que tu peux éxécuter pour qu'il télécharge les images. Il n'y a qu'à lancer la cellule de code.

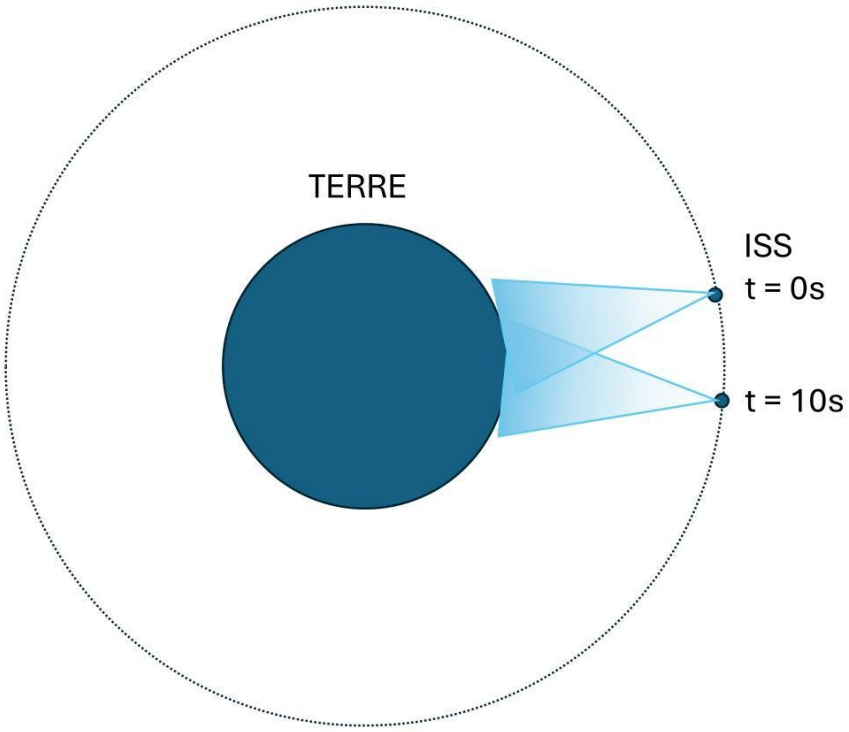
```
# Prérequis
# Pour lancer les cellules de code, presse simultanément les touches
Shift+Enter.

!pip install exif
!wget -O ./astropi-iss-speed-en-resources.zip https://rpf.io/p/en/astropi-
iss-speed-go
!unzip astropi-iss-speed-en-resources.zip
!rm astropi*
```

NOTE : Si vous réalisez l'aventure avec le simulateur Thonny, vous ne devez pas utiliser le code ci-dessus. Vous pouvez récupérer les images en ligne en suivant ce lien https://rpf.io/p/en/astropi-iss-speed-go (le téléchargement se lancera automatiquement). Il faudra ensuite mettre ces images dans un dossier accessible à Thonny pour la suite des évènements. Si vous réalisez l'aventure avec Google Colab, il suffit d'exécuter le code ci-dessus et les images seront accessibles dans Colab.

Comment déterminer la vitesse de la station spatiale internationale (ISS) ?

L'idée est la suivante: en partant de photos de la Terre prises depuis l'ISS, comment peut-on déterminer la vitesse de la station?



Pour rappel, la vitesse moyenne entre un temps t_f final et un temps initial t_i est définie par

$$v = \frac{\Delta x}{\Delta t} = \frac{x_f - x_i}{t_f - t_i}.$$

Ceci exprime que la vitesse correspond à la distance parcourue sur un certain laps de temps.

La trajectoire de l'ISS autour de la Terre est à peu près circulaire. Mais l'illustration ci-dessus est exagérée. Pendant un intervalle de temps très court, typiquement de quelques secondes, on peut considérer que l'ISS évolue presqu'en ligne droite. De manière similaire, les points de la surface de la Terre situés juste en dessous de l'ISS pendant cet intervalle de temps très court se trouvent également sur une ligne droite.

Première question: comment estimer simplement la distance parcourue à partir de deux photos de la surface de la Terre, prises par l'ISS avec cet intervalle de temps?

L'objectif est de réaliser un petit programme en langage Python qui pourra effectuer le calcul de la vitesse de l'ISS. On procèdra en trois temps:

1. Calculer de la distance parcourue Δx à partir de photos,
2. Trouver l'intervalle de temps Δt entre deux photos,
3. Calculer la vitesse v de l'ISS.

Note: Ce tutoriel ne présente que les grandes lignes et l'estimation obtenue est déjà proche de la réalité. Il y a cependant encore beaucoup d'améliorations que l'on peut apporter pour affiner l'estimation de la vitesse. C'est là que tu entres en jeu, si tu participes au concours Mission Space Lab! Mais avant celà, voici ta première leçon de Python!

Python: Mini cours 1 - Premier programme

Tes cours de Python de l'Université datent un peu, quelques rappels sont donc nécessaires.

Python est un langage de programmation parmi les plus utilisés dans le monde, en particulier pour des applications scientifiques, pour l'analyse de données ou encore pour le dévelopment de l'intelligence artificielle (par exemple, ChatGPT est programmé principalement en Python). Un programme en Python a la particularité de pouvoir s'exécuter directement ligne par ligne, on dit que c'est un langage interprété. Python est aussi un langage libre, donc tu peux l'utiliser et développer des programmes en Python gratuitement.

*Les **variables** permettent d'enregistrer des données. Cela peut être un nombre **entier** (integer), un nombre **réel à virgule** (float), une variable logique binaire (`True` ou `False`), un caractère, un mot ou une phrase (string, pour **chaine de caractères**) et bien plus.*

Remarque : pour les nombres réels, la virgule (le séparateur décimal) est représentée par un point.

La variable permet de garder dans la mémoire de l'ordinateur l'information voulue pour être utilisée plus tard dans le programme.

Pour créer une variable, il suffit de lui donner un nom qui ne commence pas par un chiffre et lui affecter une valeur. Pour affectrer une valeur à une variable on écrit le nom de la variable et la valeur souhaitée séparées d'un signe `=`. Par exemple:

```
message = "Hello world"    # Exemple de chaine de caractère (string)
pi = 3.141592              # Exemple de nombre réel à virgule (float)
```

définit une variable `message` qui contient une chaine de caractère délimintée par les guillements, et une variable `pi` qui contient un nombre réel à virgule. Une variable est un peu comme une boite, dans laquelle on peut ranger ce que l'on veut. Intuitivement, l'instruction

```
message = "Hello world"
```

peut se comprendre comme : dans la boite qui s'appelle `message`, je veux ranger la chaine de caractères `"Hello world"`.

Il faut donc bien comprendre que le signe = est utilisé d'une façon différente de son sens en mathématique. On parle de égal d'assignation.

En effet, en utilisant plusieurs fois ce symbole on peut changer ce qui est stocké dans une variable. Par exemple, avec le code

```
x=2
```

on crée une variable qui s'appelle simplement `x` *et on y range la valeur* `2`*. Si on écrit*

```
print("la valeur de x est ")
print(x)
```

On obtient

```
la valeur de x est
2
```

Mais maintenant, si on écrit

```
x=2
print("la valeur de x est ")
print(x)
x=3
print("la valeur de x est ")
print(x)
```

on obtient

```
la valeur de x est
2
la valeur de x est
3
```

En écrivant `x=2`*, dans la "boite" appellée* `x` *on vient stocker la valeur* `2` *mais si ensuite on écrit* `x=3`*, le programme comprends qu'on veut vider la boite et y mettre une nouvelle valeur (ici* `3`*). Il n'y a donc aucun soucis à écrire dans un code*

```
x=2
x=3
```

même si cela serait impossible en mathématique.

Comme tu l'auras compris, on peut afficher la valeur d'une variable à l'écran lors du lancement du code, grâce à l'instruction `print`*:*

```
print(message)
```

qui permet d'afficher à l'écran

```
Hello world
```

Remarque : *En python, pour définir que ce que l'on écrit est une chaine de caractères (un string en anglais), on l'écrit entre des guillemets ou des apostrophes simples.*

A ton tour! Crée une variable qui contient la chaine de caractère suivante: "Bonjour! Ceci est un programme qui calcule la vitesse de la station spatiale internationale." Ensuite, affiche ce message à l'écran grâce à l'instruction print.

```
# Ecris ton code ici
message = "Bonjour! Ceci est un programme qui calcule la vitesse de la station spatiale internationale."
print(message)
```

1. Déterminer la distance parcourue

Les étapes du calcul de la distance parcourue Δx grâce à deux photos sont les suivantes:

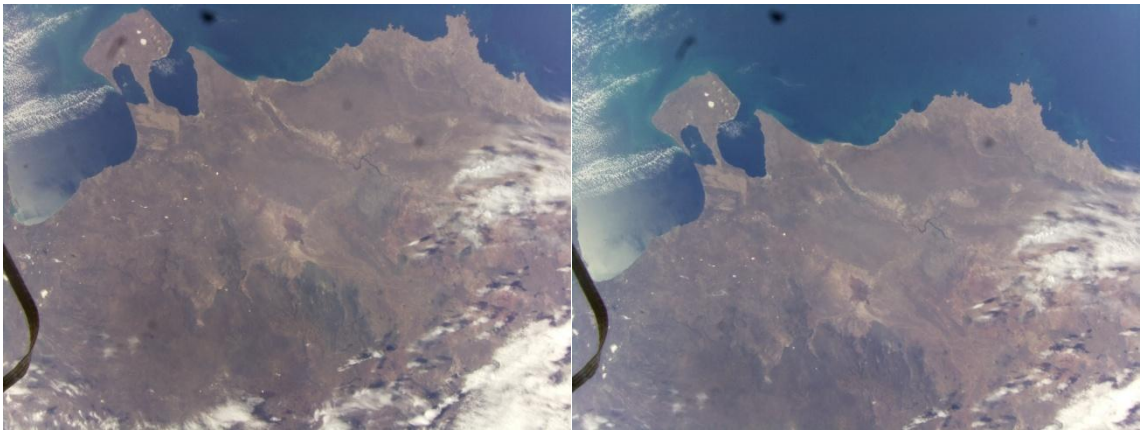
1. Des caractéristiques communes entre les images doivent être identifiées et leurs coordonnées sur l'image (exprimées en pixels) doivent être déduites,
2. La distance entre les coordonnées des points doit être calculée,
3. La distance obtenue sur l'image étant exprimée en pixels, il faudra la convertir en mètres.

La distance parcourue correspond à la différence de position en deux instants distincts. Pour deux photos prises successivement, le relief observé se déplace significativement. Pour l'oeil humain, il est facile de faire ce constat mais un ordinateur n'en est pas capable directement: il faut qu'on lui donne des instructions précises pour qu'il y arrive.

1.1. Caractéristiques uniques dans les photos

Comment sommes-nous capables de remarquer le déplacement du relief? On peut reconnaître des caractéristiques du relief qui n'ont pas changé d'une image à l'autre et observer que cette caractéristique a changé de place sur l'image, à cause du déplacement de la station. La caractéristique peut être un champ, un lac, la forme ondulée spécifique d'une rivière ou de la côte et qui est facilement reconnaissable, ou encore une montagne eneignée particulièrement visible, par exemple. Dans un ordinateur, un **algorithme** va imiter cette faculté humaine de retrouver des caractéristiques faciles à reconnaître et lister toutes celles qu'il peut trouver dans une image.

Les deux images ci-dessous sont deux photos de la Terre prisent de l'ISS à deux moments différents.



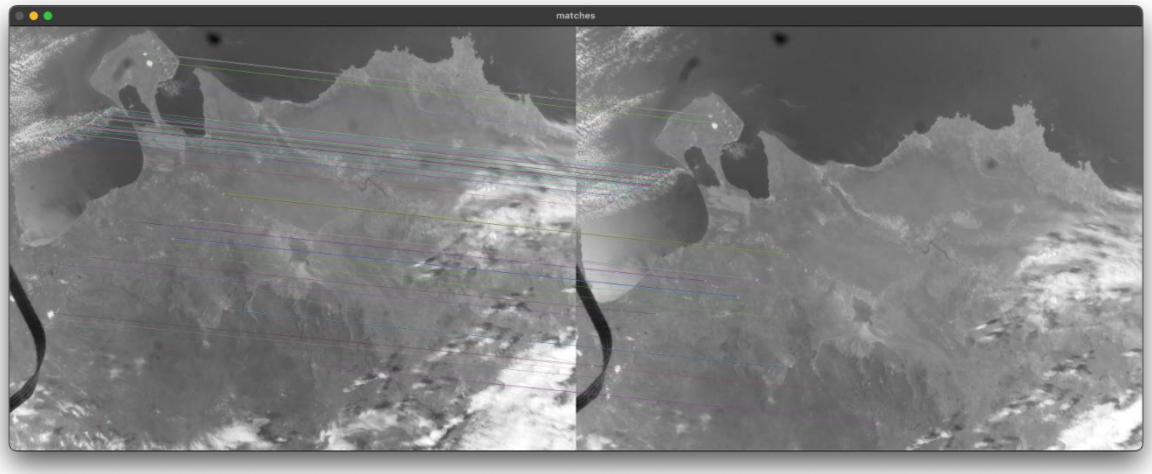
Le lapse de temps entre les deux photos, Δt, sera calculé dans une section ultérieure.

Pour déterminer la distance parcourue par l'ISS, on sélectionne des points caractéristiques, c'est à dire des points reconnaissables entre les deux images (les deux points marquées d'un cercle rouge sur les photos ci-dessous par expemple).

On associe à ces points des coordonnées x et y, localisant ainsi nos points d'intérêt sur les deux photos ; ces coordonnées sont obtenues en comptant le nombre de pixels à parcourir horizontalement et verticalement pour atteindre le point considéré depuis un coin fixé de l'image. La distance parcourue, Δx, est calculée en prenant la différence entre les coordonnées d'un point d'intérêt sur l'image 1 et celle du même point sur l'image 2.

Pour avoir un résultat plus précis, on applique cette méthode pour un nombre de points élevé. Ainsi, les incertitudes aléatoires sur la position des points caractéristiques se compenseront et le résultat sera plus précis. C'est une pratique courante en physique pour l'acquisition de données. Le Δx sera calculé pour chacun de ces points et la distance final parcourue par l'ISS sera la moyenne des tous ces Δx.

L'image ci-dessous te montre les deux images précédentes mises côte à côte, où les points identifiés comme similaire sont reliés par un fin trait de couleur :



Heureusement, tu disposes d'un algorithme qui effectue facilement cette tâche.

Voici la liste des coordonnées x et y (en pixels) de 385 pixels remarquables sur la première image:

```
[[666.87, 629.55], [685.44, 639.36], [402.62, 770.69], [1594.52, 2036.44], …]
```

Liste des coordonnées des pixels équivalents sur la seconde image:

```
[[661.89, 1062.51], [679.68, 1071.36], [400.9, 1195.78], [1582.57, 2451.49],…]
```

NOTE : Si vous réalisez l'aventure depuis le simulateur Thonny, vous trouverez la liste complète des coordonnées à copier-coller à la fin de ce document. Si vous utilisez Google Colab, la liste complète apparaitra directement dans la cellule de code ci-dessus.

Python: Mini cours 2 - Les listes et les boucles

*Une **liste** est, comme son nom l'indique, une collection de données (on peut aussi la stocker dans une variable). Elle peut donc contenir des valeurs de tout type: des chiffres, des lettres et des chaînes de caractères, etc, et même d'autres listes! En Python, une liste est entourée par des crochets et ses éléments sont séparés par des virgules comme ceci:*

```
maListe = [] # Liste vide
monAutreListe = [3, 5, 9, 2] # Liste contenant uniquement des chiffres
mesLegumes = ["tomate", "patate", "chicon"] # Liste contenant uniquement des chaînes de caractères
maListeFourreTout = [mesLegumes, "Hello", 5/3, ["hey", 3.14, "oh!"]]# Liste contenant différent types d'objets
```

On peut afficher un élément précis de la liste en précisant l'indice de l'élément voulu (ATTENTION : On compte à partir de 0 !) entre crochets après le nom de la liste comme ceci:

```
print(monAutreListe)         # Affiche toute la liste
print(maListeFourreTout[1]) # Affiche le 2ème élément de la liste
print(maListeFourreTout[3]) # Affiche le 4ème élément de la liste
print(maListeFourreTout[3][1]) # Affiche le 2ème élément du 4ème élément de la liste
```

*Petite particularité de Python à ne jamais oublier: **les indices se comptent à partir de 0 et pas 1 !** Le résultat sera donc:*

```
[3, 5, 9, 2]
Hello
[hey, 3.14, oh!]
3.14
```

À ton tour :

Crée une variable qui contiendra la liste des coordonées x et y de chacun des pixels remarquables de la première image qui nous intéresse (utilise le copier-coller). Comme il y a deux coordonnées (x et y), une liste de deux éléments sera associée à chaque pixel. On aura donc une liste (les pixels) de liste (coordonées x et y).

Crée aussi une seconde variable contenant la liste des coordonées pour la seconde image.

```
# Ecris ton code ici
liste_coord_image1 = [[666.87, 629.55], [685.44, 639.36], [402.62, 770.69],
[1594.52, 2036.44], …]

liste_coord_image2 = [[661.89, 1062.51], [679.68, 1071.36], [400.9,
1195.78], [1582.57, 2451.49],…]
```

Une fois notre liste de pixels définie, nous souhaitons pouvoir effectuer des manipulations sur ces coordonnées. Pour cela, nous allons recourir à une technique très courante en programmation informatique, la ***boucle*** *(loop en anglais).*

Une boucle `for` *est destinée à effectuer un certain nombre de fois un nombre limité d'opérations répétitives, ce que l'on appelle un* ***processus itératif****.*

Après `for`*, on indique la condition sur le nombre d'opérations, par exemple* `i in range(10)` *signifie que l'on va répéter des opérations avec la variable* `i` *allant de 0 (inclus) à 10 (non inclus). On termine par* `:`*.*

Les lignes que l'on veut exécuter plusieurs fois avec la boucle doivent être décalées de quelques espaces pour définir le bloc d'opérations à réaliser à chaque itération.

Un exemple simple: si on souhaite calculer les 10 premiers multiples de 2 (c'est à dire les nombres pairs), on écrira:

```
for i in range(10): # Les instructions suivantes seront effectuées pour
toutes les valeurs de i allant de 0 (inclus) à 10 (non inclus)
  nombre_pair = i * 2  # on multiplie i par 2
  print(nombre_pair)    # on affiche le nombre obtenu à l'écran
```

le résultat sera

```
0
2
4
6
8
10
12
14
16
18
```

Ce qui vous intéresse, ce sont les différences de coordonnées selon x et y entre les listes de pixels définies ci-dessus.

Ecris une boucle `for` ***qui va effectuer cette opération pour les 385 éléments de chaque liste. Crée une variable*** `diff_pix_x` ***et une variable*** `diff_pix_y` ***qui contiendront la différence de pixels selon x, et selon y, entre l'image 2 et l'image 1, à chaque itération. Affiche leur valeur à l'écran à chaque itération.***

```
# Ecris ton code ici
for i in range(385):
  diff_pix_x = liste_coord_image2[i][0]-liste_coord_image1[i][0]
  diff_pix_y = liste_coord_image2[i][1]-liste_coord_image1[i][1]
  print(diff_pix_x)
  print(diff_pix_y)
```

Tu maitrises maintenant les processus itératifs! Tu es prêt.e pour le calcul de Δx !

REMARQUE : Il y a d'autres manières d'implémenter des processus itératifs dans un programme en python, par exemple en parcourant les éléments d'une liste avec `for i in nom_de_la_liste:` *ou avec une boucle* `while` *qui effectue un bloc d'opérations de manière répétitive, tant qu'une condition est satisfaite.*

Tu prendras le temps d'apprendre tout ça de retour sur Terre. Car maintenant, le temps presse…

1.2. Calcul de Δx

Nous avons obtenu la différence des coordonées pour un grand nombre de points correspondants sur les deux images analysées. Mais comment transformer ça en distance parcourue?

Ton idée est la suivante:

1. Tu vas convertir les différences en pixels en différences de distance en mètres, selon x et selon y,
2. Tu vas te servir du merveilleux théorème de Pythagore (de gros frissons apparaissent toujours partout sur ton corps à sa simple évocation) pour calculer la variation de distance parcourue par la station,
3. Tu vas effectuer la moyenne de toutes les valeurs obtenues pour un résultat plus précis.

C'est parti!

Commençons par la **première étape**! Tu dois convertir la différence en pixels selon x et y en distance en mètres. En regardant des photos d'objets de taille connue (largeur d'un lac, distance entre deux villages,…) tu as pu déterminer qu'un pixel sur une photo avec une

résolution de 4056x3040 pixels, correspond à 126,5 mètres. Ce facteur de conversion s'appelle la **Ground Sampling Distance** (GSD). Cela signifie que pour chaque pixel parcouru sur l'image, on a 126,5 m parcouru en réalité sur le sol.

Remarque: Si la résolution de l'image est différente, il faut adapter la valeur du GSD en utilisant un calculateur en ligne par exemple.

Par exemple, pour le premier élément des listes de pixels caractéristiques de chaque image, on peut calculer la différence de distance selon x et y:

```
diff_pixel_x = 661.89 - 666.87
diff_pixel_y = 1062.51 - 629.55
gsd = 126.5 # Ground Sampling Distance
diff_x = diff_pixel_x * gsd # Conversion en distance selon x, en mètres
diff_y = diff_pixel_y * gsd  # Conversion en distance selon y, en mètres
print(diff_x, "mètres")
print(diff_y, "mètres")
```

retournera

```
-629.9700000000023 mètres
54769.44 mètres
```

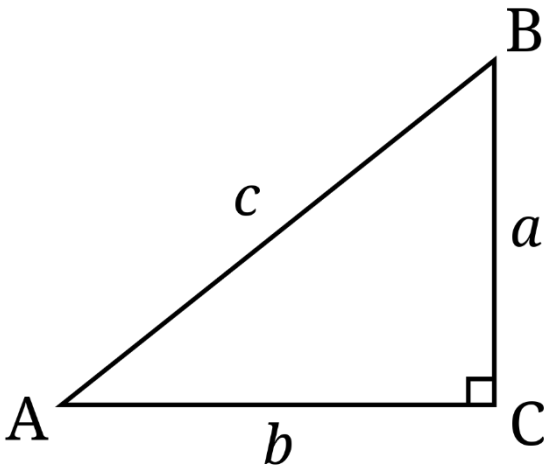
qui correspond à la distance (positive ou négative) parcourue par la station selon x et y, en mètres.

Reprend la boucle `for` effectuée auparavent, enlève les instructions `print` et, dans le bloc d'opérations à l'intérieur de la boucle, définis deux nouvelles variables `diff_x` et `diff_y`. Attribue leur le résultat de la conversion de la différence de pixels selon x et selon y en mètres. Affiche à l'écran le résultat obtenu pour chaque couple de pixels caractéristiques.

```
# Ecris ton code ici
gsd = 126.5

for i in range(385):
  diff_pix_x = liste_coord_image2[i][0]-liste_coord_image1[i][0]
  diff_pix_y = liste_coord_image2[i][1]-liste_coord_image1[i][1]
  diff_x = diff_pix_x * gsd
  diff_y = diff_pix_y * gsd
  print(diff_x)
  print(diff_y)
```

La **deuxième étape** consiste à utiliser le Théorème de Pythagore pour obtenir la distance parcourue.



`diff_x` correspond au côté b du triangle représenté ci-dessous, `diff_y` correspond au côté a, donc grâce à Pythagore, on obtient que la distance parcourue est donnée par l'hypothénuse $c = \sqrt{a^2 + b^2}$. Traduit en langage python, cela donne:

```
import numpy as np
distance = np.sqrt(diff_x**2+diff_y**2)
print("La distance parcourue est de", distance, "mètres")
```

qui donnera

```
La distance parcourue est de 54773.062906090076 mètres
```

Tu as probablement compris qu'en python `**` effectue l'opération exposant et que `np.sqrt()` permet d'effectuer la racine carrée (square root en anglais, d'où la dénomination sqrt) de ce qui est donné entre paranthèses. Comme cette fonctionalité n'existe pas dans le langage Python de base, nous avons importé une **librairie** qui permet de réaliser cette opération, `numpy`, grâce à la ligne `import numpy as np`. La partie `as np` permet d'attribuer un racourci pour l'appel de toutes les fonctionalités de numpy. On peut ainsi écrire `np.sqrt` au lieu de `numpy.sqrt`, ce qui est plus court.

Python: Mini cours 3 - Les librairies

Un des principes de base de la programmation informatique, c'est de pouvoir se servir de tout ce qui a été développé pendant les 50 dernières années (voir plus). Rien ne sert de réinventer la roue!

*De nombreuses **librairies** existent en Python, et dans d'autres langages, qui contiennent des parties de code permettant de réaliser des opérations diverses et variées: réaliser un graphique, afficher une image, créer un bouton, générer des nombres aléatoires, etc. Les possiblités sont innombrables. La librairie `numpy` est une des plus utilisées en programmation scientifique. Elle permet par exemple de calculer des exponentielles avec `np.exp`, des fonctions trigonométriques avec `np.sin`, `np.cos`, `np.tan`, etc, des racines carrées avec `np.sqrt`, des dérivées, intégrales, recherches de racines, etc.*

La librairie `matplotlib` *est utilisée pour les graphiques,* `tensorflow` *pour l'intelligence artificielle, juste pour ne citer que deux autres exemples très connus.*

Pour importer une librairie, au début de ton code, tu écris simplement:

```
import nom_de_la_librairie as raccourci
```

où la partie `as racourci` *est facutlative mais permet de définir un racourci pour ne pas devoir écrire le nom complet de la librairie (parfois long).*

Autre possibilité:

```
from numpy import sqrt
```

si tu ne souhaites qu'importer la fonctionalité `sqrt` *de la librairie* `numpy`*. Dans ce cas, tu pourras utiliser directement* `sqrt` *dans ton code, à la place de* `np.sqrt`*.*

Grâce aux librairies en Python, un nombre infini de possiblitiés s'ouvre à toi!

A ton tour! Reprends ton code écrit plus haut et rajoute dans le bloc d'opérations dans la boucle le calcul de la distance parcourue grâce au théorème de pythagore et à la méthode `sqrt()` **de la librairie** `numpy`**. N'hésite pas à imprimer la distance pour voir que ton code fonctionne.**

```
# Ecris ton code ici
import numpy as np

gsd = 126.5

for i in range(385):
  diff_pix_x = liste_coord_image2[i][0]-liste_coord_image1[i][0]
  diff_pix_y = liste_coord_image2[i][1]-liste_coord_image1[i][1]
  diff_x = diff_pix_x * gsd
  diff_y = diff_pix_y * gsd
  distance = np.sqrt(diff_x**2 + diff_y**2)
  print(distance)
```

Il nous reste à effectuer la **troisième étape**: réaliser la moyenne des distances obtenues. Pour réaliser une moyenne, tu dois sommer tous les éléments et ensuite diviser par leur nombre.

Imaginons que tu souhaites calculer en Python la moyenne des éléments de la liste `[1.20, 1.10, 1.30, 1.05, 1.15, 1.12]`. Tu pourrais faire ceci:

```
maliste = [1.20, 1.10, 1.30, 1.05, 1.15, 1.12]
nombre_elements = 6
moyenne = 0.  # on initialise la moyenne à 0

for i in range(nombre_elements):  # boucle sur les éléments de la liste
  element = maliste[i]   # on prend chaque élément de la liste
  moyenne = moyenne + element  # on l'ajoute à la moyenne, afin d'obtenir
la somme des éléments
```

```
moyenne = moyenne / nombre_elements # à la fin, on divise la somme des éléments par le nombre total d'éléments pour obtenir la moyenne
print("La moyenne est:", moyenne)  # On afficle la moyenne à l'écran
```

ce qui donnera:

```
La moyenne est: 1.153333333333333
```

qui est bien la moyenne des éléments de la liste (vérifie à la calculatrice si tu le souhaites).

Pour notre problème, tu dois donc modifier un peu ton code et:

- **créer une varialbe `moyenne` initialisée à `0` avant le début de la boucle `for`.**
- **à chaque itération de la boucle, additionner `moyenne` et la distance obtenue au point précédent.**
- **après la boucle `for`, diviser `moyenne` par le nombre d'éléments dans chaque liste, soit 385.**
- **affiche à l'écran le résultat obtenu, qui correspond à la valeur calculée de Δx.**

```
# Ecris ton code ici
moyenne = 0.

for i in range(385):
  diff_pix_x = liste_coord_image2[i][0]-liste_coord_image1[i][0]
  diff_pix_y = liste_coord_image2[i][1]-liste_coord_image1[i][1]
  diff_x = diff_pix_x * gsd
  diff_y = diff_pix_y * gsd
  distance = np.sqrt(diff_x**2 + diff_y**2)
  moyenne = moyenne + distance

moyenne = moyenne/385

print("Valeur estimée de Delta x [mètres] = ", moyenne)
```

Avant de passer à la suite et au calcul de Δt, nous allons aborder la notion de **fonction** en Python qui sera très utile par la suite…

Python: Mini cours 4 - Les fonctions

*Un peu comme au cours de math, tu peux définir une **fonction** en python qui te retournera en sortie (**output**) le résultat d'un calcul plus ou moins compliqué à partir d'une ou plusieurs variables données en entrée (**input**).*

La commande `print()` utilisée jusqu'à présent est un exemple de fonction dont le seul objectif est d'afficher le contenu de la variable indiquée entre ses parenthèses.

Par exemple, on peut définir une fonction qui va calculer et retourner le périmètre d'un cercle en fonction de son rayon. On pourrait écrire directement cette formule en Python comme ceci:

```
pi = 3.141592 # La constante mathématique (à peu près)
rayon = 5 # Le rayon du cercle
PerimetreCercle = 2 * pi * rayon # PerimetreCercle contient le résultat du calcul
```

Mais chaque fois que l'on voudra calculer le périmètre d'un cercle différent, il faudra réécrire ces lignes (exatement les mêmes !) en changeant la variable `rayon`*, ce qui n'est pas très pratique. C'est plus intelligent de créer une fonction qui pourra être appellée de nombreuses fois, à différents endroits d'un programme.*

Pour créer une fonction, il faut utiliser le mot-clé `def` *suivi du nom de la fonction que l'on va créer, de parenthèses qui peuvent contenir une ou plusieurs variables en* ***input****, et finalement d'un deux-points* `:`*.*

Ensuite, on écrira un ***bloc*** *d'opérations à effectuer, légèrement décalé vers la droite (comme pour les boucles).*

On terminera par l'instruction `return` *suivie de la variable (ou des variables) à retourner en* ***output****.*

Pour l'exemple du périmètre du cercle, on aura donc:

```
# Définition de ma fonction perimetrecercle
def perimetrecercle(rayon):
  # On écrit ici les étapes du calcul
  pi = 3.141592 # La constante mathématique (à peu près)
  perimetre = 2 * pi * rayon # perimetre contient le résultat du calcul
  return perimetre
```

Maintenant, lorsqu'on voudra calculer le périmètre d'un cercle, on appellera la fonction comme suit

```
# Méthode 1: On précise la valeur de Rayon entre parenthèses
methode1 = perimetrecercle(5)
print(methode1)
# Méthode 2:
# On crée une variable Rayon contenant la valeur
monRayon = 4
# Puis on appelle la fonction en indiquant cette variable entre parenthèses
methode2 = perimetrecercle(monRayon)
print(methode2)
```

A l'exécution, on aura

```
31.41592
25.132736
```

c'est bien le périmètre d'un cercle de rayon 5, puis celui d'un cercle de rayon 4.

Avant d'aller plus loin, remarquons que l'instruction `return` *est indispensable pour que le résultat du travail effectué par la fonction soit visible à la fin de son exécution.*

Contrairement aux fonctions en mathématique qui donnent toujours accès à leur résultat (Par exemple, la fonction définie par $f(x) = x^2$. On sait que $f(3) = 9$. Quand on écrit $f(3)$, c'est la même chose qu'écrire le nombre 9), en programmation, une fonction peut effectuer des opérations de manière "silencieuse", sans rien afficher après avoir terminé son travail. Cela est très pratique dans certains cas mais beaucoup moins quand le résultat est un nombre qu'on aimerait connaitre…

Dans le code précédent, si on enlève l'instruction `return`, et qu'on appelle la fonction, rien ne s'affiche. Autrement dit, la fonction calcul le périmètre du cercle, puis elle s'arrête mais sans l'afficher; et résultat est perdu !

Une dernière chose qu'il faut comprendre c'est que l'instruction `return` fonctionne différemment d'un `print`.

*`print` affiche une chaine de caractère (du texte) à l'écran. Avec `return`, on peut retourner le résultat sous forme d'un nombre **ET** on pourra stocker ce résultat dans une variable pour l'utiliser plus tard (ce qui ne serait pas possible si c'était un `print` qu'on avait utilisé un à l'intérieur de la fonction).*

A ton tour! Teste sur l'exemple précédent du calcul du périmètre pour voir ce qu'il se passe si on enlève le return ou si on le remplace par un print. Essaye de stocker le résultat dans une variable.

```
# Ecris ton code ici

# Premier essai
def perimetrecercle(rayon):
  # On écrit ici les étapes du calcul
  pi = 3.141592 # La constante mathématique (à peu près)
  perimetre = 2 * pi * rayon # perimetre contient le résultat du calcul

MaVariable = perimetrecercle(5)
print(MaVariable)

# Deuxieme essai
def perimetrecercle(rayon):
  # On écrit ici les étapes du calcul
  pi = 3.141592 # La constante mathématique (à peu près)
  perimetre = 2 * pi * rayon # perimetre contient le résultat du calcul
  print(perimetre)

MaVariable = perimetrecercle(5)
print(MaVariable)
```

2. Déterminer le laps de temps entre les photos

Maintenant que tu sais de quelle distance la station spatiale s'est déplacée, il te reste à déterminer combien de temps s'est écoulé entre les deux images que tu as utilisées. Ces deux images sont accessibles sous les noms `'photo_0683.jpg'` et `'photo_0684.jpg'`.

Pour connaitre le laps de temps écoulé, tu vas devoir accéder aux **méta-données** des photos.

Dans le passé, si l'on voulait se rappeler de la date, l'heure ou la position exacte de la prise d'une photo, il fallait le noter au dos de la photo. De nos jours, les photos numériques permettent d'intégrer dans les images ces informations supplémentaires (et bien d'autres). C'est ce que l'on appelle des méta-données.

En accédant à ces données, tu pourras savoir à quel moment chaque photo a été prise et il sera facile alors de connaitre le laps de temps.

Mais le temps presse !!! L'ISS pourrait quitter sa trajectoire d'un moment à l'autre et tu n'as pas le temps de coder toute une fonction permettant d'accéder aux méta-données, de localiser quelle méta-donnée correspond au temps, …

Heureusement, tu as de bonnes habitudes de programmation et tu sais que les programmeurs et programmeuses aguéris ne codent jamais inutilement quelque chose que d'autres ont surement déjà mis au point. Ils/elles réutilisent les outils existants.

Pour ça, il suffit d'importer des **librairie** ou d'utiliser des **fonctions** écrites par d'autres.

Alors, vite ! Execute le code suivant, déniché dans un dossier de l'ordinateur, pour disposer de la fonction obtenir_temps. Tu pourras alors l'appeler dans ton propre code pour calculer le laps de temps écoulé.

```
#############################################################################
# Petit code écrit par Raphaël Liégeois lors de son séjour à bord d'ISS.
# La fonction optenir_temps demande une image en input et retourne le
moment
# de la journée, en secondes, auquel une image de l'AstroPi a été prise
#############################################################################

from exif import Image          # Import de Image de la librairie exif
from datetime import datetime   # Import de datetime de la librairie
datetime
def obtenir_temps(image):
  #Retourne en seconde, le moment de la journée auquel une image a été
prise à partir des métadonnées
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    temps_str = img.get('datetime_original')
    temps = datetime.strptime(temps_str, '%Y:%m:%d %H:%M:%S')
    temps = 3600*temps.hour + 60*temps.minute + temps.second
  return temps
```

Le module `exif` contient `Image`, une fonction qui va nous permettre d'ouvrir l'image mais aussi d'accéder à ses méta-données. Le module `datetime` permet d'interpréter les informations de date et d'horaire et de les transformer. Prenons maintenant une photo déjà prise.

La fonction `obtenir_temps`, qui utilise ces deux outils intelligemment, fournit directement en secondes le moment de la journée auquel une photo passée en argument a été prise.

Par exemple, en entrant

```
print(obtenir_temps('photo_0683.jpg'))
```

le résultat suivant devrait s'afficher:

```
55917
```

(Ce qui veut dire que la photo a été prise à 15 heures 31 minutes 57 secondes).

Fais le test pour vérifier que le code que tu as importé fonctionne correctement.

```
# Ecris ton code ici
print(obtenir_temps('photo_0683.jpg'))
```

A toi de jouer à présent!

Pour calculer la différence de temps entre deux photos prises successivement, il faut créer une fonction `obtenir_laps_temps()`.

On la définit comme suit:

```
def obtenir_laps_temps(image1, image2):
```

Pour chaque image, on va extraire le temps grâce à la fonction `obtenir_temps` que l'on a importée :

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)
```

Ensuite, il suffit de calculer la différence des deux temps; en n'oubliant pas de renvoyer le résultat pour pouvoir l'utiliser ensuite :

```
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)

  difference_temps = temps2 - temps1
  return difference_temps
```

Pas une seconde à perdre ! Tu peux à présent déterminer le laps de temps (en secondes) qui sépare la prise des deux photos `'photo_0683.jpg'` et `'photo_0684.jpg'`.

N'oublie pas de stocker le résultat dans une variable pour le réutiliser ensuite.

```
# Ecris ton code ici
def obtenir_laps_temps(image1, image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)

  difference_temps = temps2 - temps1
  return difference_temps

deltaT = obtenir_laps_temps('photo_0683.jpg', 'photo_0684.jpg')
print(deltaT)
```

3. Calcul de la vitesse moyenne

Maintenant que tu as calculé Δx et Δt, tu as toutes les données pour calculer la vitesse de l'ISS.

Créons une fonction qui calcule la vitesse de l'ISS à partir de la distance moyenne parcourue (disons en kilomètres) et du laps de temps écoulé (en secondes) :

```
def calculer_vitesse(distance_moyenne, laps_temps):
```

Après tous les efforts que tu as fait, la vitesse est très facile à obtenir : il suffit de diviser la distance parcourue par le laps de temps écoulé entre les deux images. On obtient alors une valeur, que l'on renvoie:

```
def calculer_vitesse(distance_moyenne, laps_temps):
  vitesse = distance_moyenne / laps_temps
  return vitesse
```

Pour obtenir une valeur en km/s, n'oublie pas de convertir la distance parcourue en kilomètres avant d'appeler ta fonction.

```
# Ecris ton code ici
def calculer_vitesse_en_kmps(distance_moyenne, laps_temps):
  vitesse = distance_moyenne / laps_temps
  return vitesse

Deltaxkm = moyenne/1000.
vitesse = calculer_vitesse_en_kmps (Deltaxkm,deltaT)
print("Vitesse de l'ISS estimée:", vitesse, "km/s")
```

Est-ce la bonne vitesse pour rester en orbite?

Humm… Tu retiens bien l'altitude de l'ISS, ça c'est environ 400 km. Et le temps pour une orbite complète autour de la Terre ? Ça c'est aussi facile en fait ! Après tout ce temps passé à bord de l'ISS tu le connais par cœur : c'est le temps entre deux levers de Soleil depuis l'ISS, 1 heure et 33 minutes. Mais la vitesse, c'est pas quelque chose que l'on retient…

Avec ces informations et le rayon de la Terre $R_T = 6378$ km, tu dois pouvoir calculer en Python la vitesse que doit avoir habituellement la station spatiale internationale. Définis une variable pour chaque donnée du problème, puis effectue le calcul. Et attention aux unités, comme le rapellait sans cesse Monsieur Ducobu!

```
# Ecris ton code ici
pi = 3.141592
rT = 6378. * 1000. # en mètres
altitude = 400. * 1000. # en mètres
perimetre = 2. * pi * (rT + altitude)
periode = 93.* 60. # en secondes
vitesse_m_par_s = perimetre/periode
vitesse_km_par_s = vitesse_m_par_s / 1000.
print("Vitesse habituelle de ISS estimée:", vitesse_km_par_s, "km/s")


#Estimation de combien la vitesse estimée dévie de la vitesse attendue (en
pourcents)
print("Différence relative :", 100*(vitesse-vitesse_km_par_s)/vitesse, "%")
```

Bravo! Tu es sauvé.e, tu sais maintenant si tu dois ou non rallumer les moteurs (une marge de 10% dans la réponse est acceptable).

Réponse : la vitesse obtenue est dans la marge des 10% par rapport à la vitesse théorique calculée. Il n'est donc pas nécessaire d'allumer les moteurs.

Pour aller plus loin…

a) Estimation de la vitesse à partir de la loi de la gravitation universelle

Si tu souhaites être plus précis.e et vérifier ton calcul par une autre méthode (vu l'enjeu, mieux vaux vérifier sa réponse deux fois plutôt qu'une), tu peux également calculer la valeur théorique de la vitesse de la station en orbite à peu près circulaire, grâce à la **loi de gravitation universelle de Newton** et aux caractéristiques du **mouvement circulaire uniforme**.

L'avantage de ce calcul c'est qu'il est fait indépendemment du précédent (que tu avais fait en te rappellant l'altitude de l'ISS et le temps entre deux levers de Soleil à bord de l'ISS). Si les deux résultats sont compatibles, ça te donne une bonne raison en plus de faire confiance à ton estimation !

Selon Newton, la force exercée par la Terre sur la station est donnée par:

$$F_G = G\frac{m_{ISS}\, M_T}{r^2}$$

où $G = 6.67 \times 10^{-11}\, m^3\, kg^{-1}\, s^{-2}$ est la constante gravitationnelle de Newton, m_{ISS} la masse de la station spatiale, $M_T = 5.97 \times 10^{24}$ kg est la masse de la Terre et r la distance entre le centre de la Terre et l'ISS, donc $r = R_T + 400$ km.

Pour un mouvement circulaire, cette force est égale à la force centripète, qui se calcule comme:

$$F_c = \frac{m_{ISS}\, v_{ISS}^2}{r}$$

où v_{ISS} est la vitesse de la station spatiale.

En égalant les deux expressions et en isolant v, déduis en que

$$v = \sqrt{G \frac{M_T}{r}}$$

qui ne dépend plus de la masse de la station spatiale.

Définis une fonction `vitesse_theorique` qui calculera et retournera la vitesse en kilomètres par seconde, en fonction de l'altitude de la station h. Appelle cette fonction avec h correspondant à 400 km et vérifie si la vitesse est bien très proche de celle calculée au point précédent

À toi de jouer !

```
# Ecris ton code ici
def vitesse_theorique(altitude_en_km):
  rT = 6378. * 1000. #en metres
  r = rT + altitude_en_km * 1000. #en mètres
  G = 6.67 * 10.**(-11)
  mT = 5.97 * 10.**(24.)
  v = np.sqrt(G * mT/ r) #en mètres par seconde
  return v

vitesse_grav_km_par_s = vitesse_theorique(400.)/1000.
print("Vitesse estimée à partir du MCU et gravitation de Newton:",
vitesse_grav_km_par_s, "km/s")
```

b) Correction due à la rotation de la terre

Cette estimation de la vitesse théorique faite grâce à la loi de gravitation universelle et au mouvement circulaire uniforme doit te permettre de gagner en confiance quant à la trajectoire suivie par l'ISS.

Il y a cependant un petit détail qui te tracasse...

Le calcul que tu viens de faire te permet d'estimer à la vitesse de l'ISS sur son orbite autours de la terre (c'est le `vitesse_theorique` que tu as calculé) mais ce que tu as pu déterminer

avec les photos prises par l'AstroPi c'est la vitesse de déplacement de l'ISS par rapport au sol (ce que tu as obtenu grâce à ta fonction `calculer_vitesse`) !

Le problème est donc que ces deux vitesse ne sont a priori pas les mêmes puisque, pendant que l'ISS se déplace autour de la terre, la terre, elle, tourne sur elle-même !

Comment relier ces deux informations l'une à l'autre ?

Dans l'idéal, il faudrait savoir de combien le sol s'est déplacé, dû à la rotation de la terre, pendant le laps de temps où tu as fait ton expérience. Et aussi, dans quelle direction.

Difficile d'avoir ces informations sans connaitre avec précision la trajectoire suivie par l'ISS...

Mais tu as plus d'un tour dans ton sac !

Ton idée est la suivante :

Si on ne peut pas savoir exactement comment le sol s'est déplacé, essayons de faire le calcul dans le pire des cas. Si la différénce entre la vitesse de rotation (que tu as calculée théoriquement) et la vitesse par rapport au sol (que tu as pu mesurer) n'est pas trop importante dans ce cas là, tu seras sur.e que ton estimation est pertinente dans tous les cas.

Quel est "le pire des cas" ? Autrement dit, dans quel cas le déplacement du sol en dessous de l'ISS serait-il le plus important ?

Comme le déplacement est dû à la rotation de la terre, demandons-nous quels sont les points qui se déplacent le plus à cause de cet effet.

La réponse t'apparait rapidement : ce sont les points à l'équateur qui parcourent la plus grande distance (car ce sont les points à la surface de la terre qui sont les plus éloignés de son axe de rotation).

Le pire des cas possibles (celui où la vitesse de rotation de l'ISS autours de la terre différerait le plus de sa vitesse par rapport au sol) serait donc le cas où l'ISS se déplace le long de l'équateur; soit dans le même sense que la rotation de la terre (et là `calculer_vitesse` donnera un résulat plus petit que `vitesse_theorique`) soit dans le sens opposé (et là `calculer_vitesse` donnera un résulat plus grand que `vitesse_theorique`).

Dans ce "pire des cas", la différence entre la vitesse théorique et la vitesse par rapport au sol est donc donnée (en valeur absolue) par la vitesse de déplacement des points de l'équateur.

Et ça, tu n'as aucun mal à le calculer : *On sait que la terre fait un tour complet sur elle-même en 24 heures. Sur ce laps de temps, un point à l'équateur a donc parcouru une distance qui correspond à la circonférence d'un cercle dont le rayon n'est autre que le rayon de la terre* R_T *!*

À partir de ces informations, calcule la vitesse de déplacement des points à l'équateur en km/s, appellons-là `vitesse_equateur`.

Compare-là ensuite à la `vitesse_theorique` que tu as obtenue au point précédent; si `vitesse_equateur` vaut moins de 10% de la vitesse théorique, tu peux considérer qu'elle est négligeable (elle tombera dans la barre d'erreur de ton estimation).

```
# Ecris ton code ici

pi=3.141592
rT=6378. * 1000. #en mètres
vitesse_equateur = (2*pi*rT)/(24*3600.) #en mètres par seconde
vitesse_equateur_km_par_s = vitesse_equateur/1000. #en kilomètres par
seconde

print(vitesse_equateur_km_par_s)

#Comparaison avec la vitesse theorique :
print(vitesse_equateur_km_par_s/vitesse_grav_km_par_s)
```

Petit jeu en attendant la mission de secours

Pour terminer, en attendant une fusée de secours pour vous redescendre sur Terre, ta collègue astronaute sur l'ISS décide de créer un petit jeu en Python à propos de la station spatiale internationale, de la physique et du langage Python.

Voici un exemple de code qui utilise une notion de Python que nous n'avons pas abordé: les expressions conditionnelles `if`.

Peux-tu le modifier pour ajouter de nouvelles questions avec propositions? N'hésite pas à jouer aussi.

```
import numpy as np

# Définition de la liste de question:
liste_de_questions = ["Pourquoi flotte-t-on dans ISS?",
                      "Si la vitesse d'ISS est réduite et induit sa chute,
que ressentent les astronautes jusqu'à la rentrée dans l'atmosphère?",
                      "Quelle instruction est utilisée créer une boucle en
Python"]
# Définition des choix multiples (liste de liste)
liste_de_choix_multiples = [["1) La gravité est plus faible loin de la
Terre",
                             "2) L'accélération centrifuge dans le
référentiel de l'ISS compense l'attraction gravitationnelle de la Terre",
```

```
                                "3) A cause de la poussée d'Archimède qui
compense l'attraction gravitationnelle de la Terre"],
                               ["1) Ils continuent à flotter",
                                "2) Ils sont attirés vers le bas (en direction
de la Terre)",
                                "3) Ils sont attirés vers le haut"],
                               ["1) def",
                                "2) import",
                                "3) for"]]
# Définition des bonnes réponses
liste_des_bonnes_reponses = [2, 1, 3]
nombre_de_questions = len(liste_de_questions)  # len() permet d'obtenir la
longueur d'une liste
# print(nombre_de_questions)

bonne_reponse = True
numero = 0

while(bonne_reponse and numero<nombre_de_questions ):
  print(liste_de_questions[numero])
  print(liste_de_choix_multiples[numero][0])
  print(liste_de_choix_multiples[numero][1])
  print(liste_de_choix_multiples[numero][2])
  reponse = int(input("Quelle est votre réponse? "))
  if(reponse == liste_des_bonnes_reponses[numero]):
    print("C'est gagné, tu peux continuer !")
    print("\n") #\n dans une chaine de caractère permet d'ajouter un retour
à la ligne
    numero = numero+1
  else:
    print("C'est faux, tu as perdu la partie... Essaye encore ! ;)")
    print("\n") #\n dans une chaine de caractère permet d'ajouter un retour
à la ligne
    bonne_reponse = False
print("C'est terminé, il n'y a plus de questions...  Inventes-en et ajoute
les dans le programme!")
```

Si tu souhaites participer au concours AstroPi-Mission Space Lab de l'agence spatiale européenne…

La vitesse obtenue est satisfaisante mais l'estimation peut encore être améliorée. En particulier, pour participer au concours Mission Space Lab, il faudra aller plus en profondeur dans la caractérisation des points semblables d'une image à une autre. Si cela t'intéresse, n'hésite pas à consulter nos autres ressources pédagogiques consacrées à Mission Space Lab sur la page suivante : https://esero.be/index.php/secondaire/.

Voici déjà quelques pistes à explorer pour améliorer l'estimation :

- La correspondance des caractéristiques a été faite par un algorithme de Force Brute puis triée du moins correspondant au plus correspondant mais la recherche des coordonnées correspondantes a utilisé toutes les correspondances sans tenir compte du triage. Il conviendrait d'utiliser les meilleures correspondances uniquement pour améliorer l'estimation.

- Lors de la détection des caractéristiques, seul 1000 caractéristiques ont été cherchées avec algorithme ORB. Qu'en est-il si on change ce paramètre?
- Est-ce qu'un autre algorithme donnerait de meilleurs résultats? Une liste d'algorithme est disponible dans la documentation OpenCV: https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html
- Si des nuages apparaissent, la détection des caractéristiques va être biaisée par leur présence. Comment les ignorer? Sinon, comment minimiser leur impact en sachant qu'il peuvent être en mouvement eux aussi?
- Comment utiliser un modèle de Deep Learning avec l'accélérateur Google Coral peut aider dans ce projet? Voici une ressource à ce sujet: https://esero.fr/wp-content/uploads/2022/12/ImageClassificationCoral_RaspPiProj_FR.pdf
- Au lieu d'utiliser que deux images, peut-on en utiliser plus?
- Au lieu d'utiliser la méta-donnée du temps de prise de l'image, mesurer directement le temps plus précisément au moment de prendre l'image.
- La distance moyenne estimée correspond à la distance au sol et non au niveau de l'orbite de l'ISS. Comment adapter la distance mesurée?
- La Terre n'est pas parfaitement sphérique donc sa courbure varie en fonction du lieu où se situe l'ISS. Comment cela impacte la distance moyenne estimée au sol?
- La Terre tourne aussi à sa propre vitesse durant l'orbite de l'ISS. Comment cela impacte-t-il l'estimation de la vitesse?
- La hauteur de l'ISS varie au cours de son orbite mais aussi au cours de la journée. Quel impact cela a-t-il et comment en tenir compte?
- Peut-on trouver une expression pour le GSD en fonction de la hauteur à partir du sol, les dimension du capteur et de la distance focale? Est-ce que ça peut améliorer l'estimation de la distance au sol?

Liste complète des coordonnées des points d'intérêts pour copier-coller avec Thonny

Voici la liste des coordonnées x et y (en pixels) de 385 pixels remarquables sur la première image:

[[666.87, 629.55], [685.44, 639.36], [402.62, 770.69], [1594.52, 2036.44], [960.77, 190.08], [673.92, 628.99], [510.0, 676.8], [2169.82, 2431.09], [676.8, 630.0], [1011.6, 1449.6], [240.54, 787.97], [1532.4, 1214.4], [667.01, 630.72], [532.22, 725.76], [613.44, 654.91], [1531.2, 1201.2], [960.08, 190.77], [568.17, 717.47], [2171.41, 2429.4], [1592.4, 2037.6], [1173.6, 1520.64], [442.08, 763.2], [1197.5, 1565.57], [1159.49, 1517.18], [2170.81, 2430.59], [531.51, 647.96], [664.0, 655.0], [961.2, 190.8], [674.4, 630.0], [1176.0, 1520.4], [532.8, 725.76], [571.68, 1657.44], [333.5, 800.06], [231.41, 701.71], [980.64, 198.72], [1592.64, 2037.6], [464.83, 741.31], [1295.92, 1388.48], [1157.0, 1516.0], [682.8, 619.2], [468.0, 741.6], [407.52, 768.96], [549.5, 707.1], [1293.93, 1389.31], [530.01, 724.1], [510.11, 676.82], [921.6, 1406.4], [407.81, 768.96], [404.35, 769.31], [715.39, 611.71], [559.87,

711.66], [288.65, 2095.17], [549.42, 722.61], [1017.62, 254.41], [1176.0, 1521.0], [266.4, 2140.8], [568.51, 717.12], [1522.02, 1208.91], [269.57, 675.99], [1410.05, 1412.12], [662.89, 653.93], [992.54, 250.82], [1304.28, 1386.69], [960.29, 193.49], [1075.0, 420.0], [582.0, 699.6], [1310.4, 1387.2], [470.4, 712.8], [1536.48, 1212.48], [1293.12, 1389.6], [535.68, 715.68], [1444.32, 1664.64], [1198.08, 1566.72], [607.68, 660.96], [532.22, 715.39], [570.24, 1657.15], [532.92, 715.39], [331.78, 800.41], [1596.67, 2038.35], [980.81, 199.07], [922.75, 1405.9], [532.92, 725.76], [664.8, 656.4], [1172.4, 1521.6], [550.08, 725.76], [326.59, 812.16], [1760.83, 2191.1], [195.26, 884.74], [684.29, 637.63], [770.69, 599.62], [582.27, 701.71], [664.38, 654.43], [333.43, 1704.5], [313.53, 2057.34], [1480.8, 1812.0], [291.6, 704.4], [1314.0, 1388.4], [603.6, 739.2], [1247.04, 1444.32], [653.76, 686.88], [392.26, 1567.3], [1159.14, 1515.8], [549.92, 706.68], [1530.32, 1204.35], [582.27, 701.71], [1294.56, 1540.8], [1531.01, 1213.06], [84.67, 1588.03], [671.85, 630.37], [390.67, 1833.89], [289.64, 2093.18], [528.0, 678.0], [1411.2, 1412.4], [548.64, 708.48], [561.6, 709.92], [715.68, 612.0], [632.16, 655.2], [1444.61, 1664.06], [1171.58, 1522.02], [665.63, 655.26], [931.05, 1405.9], [684.29, 637.01], [231.41, 801.24], [388.18, 1833.39], [591.23, 695.14], [530.31, 716.64], [1075.2, 420.0], [404.64, 763.2], [274.75, 812.16], [1529.28, 1202.69], [632.45, 649.73], [324.86, 850.18], [67.39, 540.86], [232.24, 702.95], [930.63, 1405.9], [923.17, 1405.9], [1021.21, 1448.2], [1530.02, 1203.95], [1444.0, 1665.0], [1444.0, 1670.0], [1202.4, 1566.0], [1522.8, 1210.8], [108.0, 406.8], [1025.28, 1450.08], [675.36, 629.28], [86.4, 328.32], [203.04, 686.88], [574.56, 698.4], [200.45, 314.5], [1159.14, 1524.1], [595.12, 692.58], [80.87, 288.23], [684.29, 638.67], [465.32, 741.52], [1158.56, 1516.88], [1292.4, 1389.6], [1311.6, 1384.8], [477.6, 686.4], [178.8, 694.8], [665.28, 655.2], [459.65, 748.22], [510.11, 675.99], [1011.92, 1449.45], [104.51, 361.3], [551.0, 726.0], [1951.2, 2637.6], [270.72, 676.8], [260.64, 1916.64], [259.2, 832.32], [201.6, 315.36], [528.77, 715.39], [317.95, 1755.65], [921.02, 1406.59], [441.68, 761.01], [530.84, 649.04], [460.34, 746.5], [532.5, 649.45], [790.8, 1398.0], [84.0, 446.4], [328.8, 1708.8], [509.76, 676.8], [1530.32, 1204.76], [273.72, 810.78], [360.81, 796.26], [1306.37, 1388.48], [1530.32, 1211.81], [271.23, 676.82], [558.98, 709.47], [1761.6, 2192.4], [1759.2, 2186.4], [795.6, 1400.4], [213.6, 913.2], [453.6, 691.2], [233.28, 846.72], [696.96, 597.6], [260.93, 2153.09], [1159.49, 1524.1], [464.49, 740.28], [375.32, 800.41], [1198.54, 1565.57], [768.89, 597.2], [441.93, 749.48], [1528.82, 1206.34], [440.73, 756.05], [336.82, 791.88], [3547.35, 2307.57], [1246.0, 1445.0], [319.2, 1758.0], [613.2, 655.2], [228.0, 838.8], [1251.36, 1444.32], [388.8, 740.16], [658.37, 651.46], [1480.9, 1812.67], [551.23, 725.76], [157.25, 551.23], [103.68, 362.88], [85.02, 327.63], [1448.2, 1152.09], [2620.2, 2525.65], [1204.0, 1566.0], [235.0, 1770.0], [699.6, 2833.2], [1251.6, 1444.8], [361.2, 717.6], [1761.12, 2191.68], [391.68, 774.72], [547.78, 2899.58], [1308.44, 1387.24], [329.7, 1708.65], [1022.7, 1448.2], [558.38, 710.66], [580.48, 702.3], [662.89, 652.14], [409.0, 1836.0], [550.8, 726.0], [194.4, 884.4], [325.44, 851.04], [194.4, 884.16], [770.4, 599.04], [1022.28, 1449.45], [330.95, 801.24], [438.94, 758.44], [610.8, 667.2], [606.53, 660.1], [404.35, 762.05], [333.5, 1703.81], [392.26, 774.14], [159.67, 860.54], [922.67, 1406.4], [702.3, 2830.71], [349.0, 806.0], [350.4, 806.4], [666.72, 630.72], [192.96, 855.36], [316.8, 1755.36], [160.7, 862.27], [361.3, 797.26], [327.6, 850.8], [571.2, 718.8], [632.16, 649.44], [292.32, 704.16], [319.68, 1759.68], [403.11, 768.89], [349.36, 794.27], [462.83, 740.52], [552.41, 707.68], [256.79, 683.79], [240.48, 790.56], [240.19, 791.42], [57.02, 414.72], [317.26, 1756.34], [390.67, 776.36], [1159.56, 1517.88], [1021.21, 1447.61], [347.57, 791.88], [509.0, 680.0], [222.0, 736.8], [1452.96, 1152.0], [768.96, 594.43], [437.94, 758.94], [558.38, 719.62], [329.65, 799.05], [525.6, 714.0], [377.28, 802.08], [561.6, 710.21], [376.7, 801.79], [931.39, 1404.86], [670.05, 630.64], [595.0, 610.0], [465.12, 740.16], [72.0, 616.32], [349.06, 796.61], [557.38, 719.12], [1307.86, 1388.48], [1012.25, 1448.2], [624.0, 664.0], [350.85, 793.77], [632.03, 651.94], [634.0, 649.0], [468.0, 738.72], [931.68, 1405.44], [613.44, 655.2], [191.81, 651.46], [454.12, 690.51], [261.27, 1907.71], [266.25, 2142.44], [361.9, 799.05], [462.23, 741.72], [100.8, 842.4], [1759.1,

2185.92], [622.08, 660.1], [193.54, 857.09], [426.82, 708.48], [790.04, 1397.61], [549.5, 1992.73], [1759.24, 2187.23], [531.6, 648.0], [2563.2, 2547.36], [361.2, 715.2], [505.2, 747.6], [595.2, 609.6], [233.28, 800.64], [1010.88, 1449.79], [404.9, 763.22], [685.2, 638.4], [686.4, 630.0], [258.0, 724.8], [1480.55, 1814.4], [313.53, 808.7], [329.65, 809.8], [228.96, 838.08], [672.48, 635.04], [453.6, 686.88], [527.04, 677.38], [1758.74, 2188.73], [78.0, 841.2], [82.94, 286.85], [553.0, 653.0], [358.0, 718.0], [548.4, 1992.0], [57.6, 416.16], [328.46, 812.19], [350.44, 794.19], [746.4, 586.8], [160.7, 820.8], [136.8, 645.12], [86.4, 328.32], [268.74, 2146.33], [309.31, 698.11], [104.51, 363.29], [311.74, 806.22], [328.46, 811.19], [615.11, 653.93], [1171.7, 1519.27], [1160.95, 1519.27], [1852.51, 2762.63], [1084.0, 956.0], [103.68, 364.32], [794.27, 1400.43], [166.8, 354.0], [3245.76, 2818.08], [259.78, 1914.02], [67.68, 675.36], [388.8, 777.6], [204.0, 735.6], [361.15, 796.61], [228.1, 837.73], [1492.99, 2718.49], [229.32, 838.46], [229.82, 698.11], [271.2, 774.0], [692.4, 634.8], [594.71, 694.24], [408.96, 776.16], [653.93, 2917.31], [229.0, 876.0], [117.6, 549.6], [3941.5, 2257.4], [3271.45, 2769.8], [394.8, 850.8], [333.6, 801.6], [67.39, 675.65], [630.04, 650.94], [447.6, 703.2]]

Liste des coordonnées des pixels équivalents sur la seconde image:

[[661.89, 1062.51], [679.68, 1071.36], [400.9, 1195.78], [1582.57, 2451.49], [950.4, 635.9], [668.74, 1060.99], [506.4, 1105.2], [2149.91, 2844.15], [672.0, 1062.0], [1003.0, 1869.0], [240.54, 1208.91], [1518.0, 1648.8], [661.82, 1062.72], [528.77, 1154.3], [608.26, 1085.18], [1516.8, 1635.6], [951.78, 638.67], [564.02, 1146.7], [2149.91, 2841.46], [1579.68, 2452.32], [1164.96, 1941.12], [439.2, 1188.0], [1188.86, 1985.47], [1150.56, 1938.24], [2149.91, 2842.66], [528.52, 1077.94], [659.0, 1087.0], [951.6, 638.4], [669.6, 1062.0], [1167.6, 1940.4], [528.48, 1153.44], [570.24, 2063.52], [333.5, 1223.42], [231.41, 1124.72], [969.12, 646.56], [1580.4, 2451.6], [463.1, 1168.13], [1283.97, 1815.48], [1148.4, 1936.8], [678.0, 1052.4], [465.12, 1167.84], [406.08, 1193.76], [545.36, 1136.33], [1285.63, 1816.47], [527.52, 1152.09], [505.13, 1104.81], [915.6, 1825.2], [406.08, 1194.05], [402.28, 1194.39], [709.17, 1045.09], [557.38, 1139.65], [293.62, 2485.83], [546.44, 1149.6], [1006.87, 702.3], [1168.0, 1941.0], [272.4, 2529.6], [565.06, 1145.66], [1509.58, 1644.36], [269.57, 1101.08], [1397.61, 1839.28], [656.92, 1083.91], [981.79, 695.14], [1293.53, 1813.09], [949.54, 637.81], [1064.0, 864.0], [578.0, 1129.0], [1302.0, 1813.0], [466.8, 1138.8], [1520.64, 1645.92], [1283.9, 1816.13], [531.36, 1144.8], [1432.8, 2086.56], [1189.44, 1985.76], [603.36, 1091.52], [530.5, 1143.94], [570.24, 2063.23], [530.84, 1144.63], [331.78, 1223.42], [1584.23, 2453.07], [970.44, 644.89], [916.53, 1824.77], [528.77, 1152.92], [660.0, 1088.4], [1165.2, 1941.6], [547.2, 1153.44], [326.59, 1233.79], [1746.72, 2604.96], [196.99, 1302.91], [679.1, 1069.63], [763.78, 1033.34], [577.29, 1129.7], [659.4, 1087.4], [335.92, 2105.12], [319.5, 2448.51], [114.46, 2441.04], [290.4, 1128.0], [1302.0, 1813.2], [598.8, 1167.6], [1236.96, 1867.68], [649.44, 1117.44], [392.26, 1973.38], [1150.85, 1936.74], [547.43, 1137.16], [1515.39, 1639.8], [576.3, 1131.69], [1284.48, 1962.72], [1517.18, 1648.51], [1575.94, 2452.03], [667.7, 1061.68], [226.08, 2584.8], [292.63, 2481.35], [524.4, 1107.6], [1399.2, 1837.2], [545.76, 1136.16], [558.72, 1139.04], [709.92, 1046.88], [626.4, 1087.2], [1432.51, 2085.7], [1508.54, 1645.06], [659.4, 1086.57], [924.83, 1824.77], [679.31, 1069.98], [231.41, 1221.77], [388.18, 2239.49], [587.64, 1125.12], [523.14, 1143.04], [1064.4, 864.0], [401.76, 1186.56], [274.75, 1233.79], [1513.73, 1636.42], [627.26, 1080.0], [324.86, 1271.81], [67.39, 964.22], [232.24, 1123.89], [923.17, 1823.94], [915.7, 1823.94], [1015.23, 1869.23], [1515.69, 1637.51], [1432.8, 2086.8], [1582.85, 2448.58], [1195.2, 1986.0], [1508.4, 1645.2], [108.0, 832.8], [1016.64, 1869.12], [671.04, 1061.28], [84.96, 756.0], [203.04, 1110.24], [570.24, 1127.52], [198.72, 744.77], [1150.85, 1945.04], [588.9,

1121.82], [80.87, 715.39], [680.14, 1072.05], [462.83, 1169.51], [1152.59, 1937.9], [1283.04, 1817.28], [1298.4, 1809.6], [475.2, 1114.8], [180.0, 1117.2], [659.52, 1087.2], [456.19, 1175.04], [505.96, 1105.23], [1005.7, 1868.31], [104.51, 788.3], [548.0, 1156.0], [792.0, 1946.4], [267.84, 1101.6], [1586.3, 2453.76], [259.2, 1252.8], [198.72, 744.48], [525.31, 1142.21], [321.41, 2156.54], [915.84, 1824.77], [438.91, 1187.14], [526.69, 1078.27], [457.85, 1174.49], [527.52, 1079.93], [786.0, 1813.2], [84.0, 871.2], [332.4, 2109.6], [506.88, 1105.92], [1515.8, 1638.14], [273.72, 1231.72], [360.81, 1221.77], [1296.42, 1813.99], [1515.39, 1644.78], [268.74, 1099.84], [555.39, 1139.45], [1747.2, 2607.6], [1743.84, 2599.2], [1165.36, 1940.89], [213.6, 1330.8], [450.72, 1117.44], [233.28, 1265.76], [689.76, 1031.04], [266.11, 2540.16], [1150.85, 1945.73], [462.41, 1167.44], [373.25, 1225.5], [1192.32, 1984.44], [763.91, 1032.65], [438.94, 1176.48], [1513.89, 1639.31], [437.15, 1182.45], [336.82, 1214.7], [3504.35, 2744.72], [226.02, 2583.71], [321.6, 2156.4], [607.2, 1084.8], [226.8, 1258.8], [1239.84, 1867.68], [387.36, 1166.4], [653.18, 1081.73], [1468.8, 2230.56], [546.05, 1152.58], [157.25, 976.32], [103.68, 790.04], [85.02, 754.79], [1432.86, 1586.3], [2592.83, 2948.66], [1003.68, 1869.12], [223.95, 2663.5], [1746.0, 2604.0], [1240.8, 1868.4], [360.0, 1142.4], [265.42, 2540.16], [390.24, 1199.52], [1431.0, 1585.0], [1297.44, 1810.08], [331.78, 2108.85], [1015.23, 1868.73], [555.39, 1137.66], [573.31, 1135.87], [655.72, 1082.12], [223.95, 2603.78], [547.2, 1155.6], [195.6, 1302.0], [325.44, 1271.52], [195.84, 1303.2], [763.2, 1035.36], [1016.06, 1868.31], [330.95, 1224.25], [435.95, 1185.44], [606.24, 1097.28], [603.07, 1090.37], [402.62, 1185.41], [335.23, 2104.7], [390.53, 1199.23], [161.74, 1279.41], [916.7, 1824.44], [967.46, 644.97], [350.0, 1230.0], [350.4, 1228.8], [662.4, 1062.72], [192.96, 1275.84], [322.56, 2152.8], [162.43, 1280.45], [358.32, 1221.27], [325.2, 1272.0], [566.0, 1148.0], [626.4, 1081.44], [290.88, 1128.96], [321.12, 2157.12], [403.11, 1194.39], [349.36, 1215.3], [459.84, 1167.52], [546.44, 1134.67], [256.79, 1107.8], [240.48, 1209.6], [240.19, 1209.6], [57.02, 841.54], [321.41, 2156.54], [389.84, 1200.61], [1152.09, 1938.4], [1014.04, 1866.84], [347.57, 1214.7], [506.0, 1106.0], [220.8, 1156.8], [1441.2, 1585.2], [763.2, 1029.6], [435.46, 1184.44], [555.39, 1149.6], [329.65, 1221.86], [523.2, 1142.4], [375.84, 1226.88], [557.8, 1138.41], [373.25, 1225.15], [924.48, 1824.77], [662.89, 1060.62], [570.32, 2063.32], [462.24, 1167.84], [72.0, 1038.24], [349.06, 1218.24], [554.9, 1149.6], [1295.92, 1812.49], [1006.28, 1869.23], [620.0, 1093.0], [348.36, 1216.79], [627.06, 1082.42], [1432.8, 1585.44], [466.8, 1166.4], [924.48, 1824.48], [606.24, 1084.32], [191.81, 1073.09], [452.04, 1117.67], [791.42, 1816.13], [273.72, 2530.62], [358.32, 1225.45], [462.23, 1168.12], [100.8, 1260.0], [1743.55, 2598.91], [668.16, 1061.28], [193.54, 1273.54], [425.09, 1133.57], [785.89, 1812.33], [549.5, 2389.82], [1741.82, 2602.78], [528.0, 1078.8], [218.57, 2644.39], [357.6, 1140.0], [502.8, 1173.6], [410.4, 1076.4], [231.6, 1222.8], [1003.97, 1867.97], [401.32, 1189.62], [681.0, 1073.0], [680.4, 1064.4], [256.8, 1147.2], [1468.8, 2230.85], [313.53, 1230.23], [329.65, 1232.61], [227.52, 1258.56], [667.01, 1066.18], [450.72, 1114.56], [523.58, 1102.46], [1743.82, 2600.79], [709.2, 1112.4], [82.08, 715.68], [351.36, 1231.2], [216.0, 686.4], [332.64, 2109.6], [57.6, 839.52], [328.46, 1233.21], [348.36, 1219.28], [740.16, 1020.96], [159.84, 1239.84], [136.8, 1066.8], [84.67, 755.14], [252.0, 2148.48], [308.16, 1123.2], [104.51, 788.8], [311.74, 1229.03], [325.56, 1233.79], [607.15, 1084.91], [1164.53, 1938.5], [1150.2, 1938.5], [70.85, 623.81], [1280.45, 1963.01], [100.8, 792.0], [788.3, 1815.48], [403.2, 1207.2], [159.6, 1239.6], [115.2, 715.68], [109.49, 221.46], [387.36, 1202.4], [202.8, 1156.8], [297.6, 1174.8], [228.1, 1258.68], [187.2, 1090.8], [229.32, 1257.7], [230.4, 1121.76], [272.16, 1195.2], [686.88, 1067.04], [639.6, 1066.8], [696.96, 1029.6], [549.92, 2388.79], [111.6, 706.8], [379.2, 52.8], [252.0, 1262.4], [269.57, 622.08], [393.6, 1270.8], [334.08, 1224.0], [117.6, 1027.2], [627.06, 1083.91], [56.4, 840.0]]

Version complète du code exécutable en une fois

```
############################
# PARTIE CALCUL DE DELTA X #
############################

import numpy as np

liste_coord_image1 = [[666.87, 629.55], [685.44, 639.36], [402.62,
770.69],…]
liste_coord_image2 = [[661.89, 1062.51], [679.68, 1071.36], [400.9,
1195.78],…]

gsd = 126.5

moyenne = 0.

for i in range(385):
  diff_pix_x = liste_coord_image2[i][0]-liste_coord_image1[i][0]
  diff_pix_y = liste_coord_image2[i][1]-liste_coord_image1[i][1]
  diff_x = diff_pix_x * gsd
  diff_y = diff_pix_y * gsd
  distance = np.sqrt(diff_x**2 + diff_y**2)
  moyenne = moyenne + distance

moyenne = moyenne / 385

print("Valeur estimée de Delta x [mètres] = ", moyenne)

############################
# PARTIE CALCUL DE DELTA T #
############################

#########################################################################
# Petit code écrit par Raphaël Liégeois lors de son séjour à bord d'ISS.
# La fonction optenir_temps demande une image en input et retourne le
moment
# de la journée, en secondes, aurquel une image de l'AstroPi a été prise
#########################################################################

from exif import Image          # Import de Image de la librairie exif
from datetime import datetime   # Import de datetime de la librairie
datetime
def obtenir_temps(image):
  #Retourne en seconde, le moment de la journée auquel une image a été
prise à partir des métadonnées
  with open(image, 'rb') as fichier_image:
    img = Image(fichier_image)
    temps_str = img.get('datetime_original')
    temps = datetime.strptime(temps_str, '%Y:%m:%d %H:%M:%S')
    temps = 3600*temps.hour + 60*temps.minute + temps.second
  return temps


print(obtenir_temps('photo_0683.jpg'))


def obtenir_laps_temps(image1,image2):
  temps1 = obtenir_temps(image1)
  temps2 = obtenir_temps(image2)
```

```
  difference_temps = temps2 - temps1
  return difference_temps

deltaT = obtenir_laps_temps("photo_0683.jpg","photo_0684.jpg")
print(deltaT)


################################
# PARTIE CALCUL DE LA VITESSE #
################################


#Calcul de la vitesse sur base des donnée qu'on a
def calculer_vitesse_en_kmps(distance_moyenne, laps_temps):
  vitesse = distance_moyenne / laps_temps
  return vitesse

Deltaxkm = moyenne/1000.
vitesse =calculer_vitesse_en_kmps(Deltaxkm,deltaT)
print("Vitesse de l'ISS estimée:", vitesse, "km/s")



#Estimation de ce que devrait normalement être la vitesse
pi = 3.141592
rT = 6378. * 1000. # en mètres
altitude = 400. * 1000. # en mètres
perimetre = 2. * pi * (rT+altitude)
periode = 93.*60. # en secondes
vitesse_m_par_s = perimetre/periode
vitesse_km_par_s = vitesse_m_par_s / 1000.
print("Vitesse habituelle de ISS estimée:", vitesse_km_par_s, "km/s")


#Estimation de combien la vitesse estimée dévie de la vitesse attendue (en
pourcents)
print("Différence relative :", 100*(vitesse-vitesse_km_par_s)/vitesse, "%")


################################
# PARTIE POUR ALLER PLUS LOIN #
################################

### CALCUL AVEC LA LOI DE LA GRAVITATION UNIVERSELLE

def vitesse_grav(altitude_en_km):
  rT = 6378. * 1000. #en metres
  r = rT + altitude_en_km * 1000. #en mètres
  G = 6.67 * 10.**(-11)
  mT = 5.97 * 10.**(24.)
  v = np.sqrt(G * mT/ r) #en mètres par seconde
  return v

vitesse_grav_km_par_s = vitesse_grav(400.)/1000.
print("Vitesse estimée à partir du MCU et gravitation de Newton:",
vitesse_grav_km_par_s, "km/s")


### CORRECTION DUE A LA ROTATION DE LA TERRE
```

```
vitesse_equateur = (2*pi*rT)/(24*3600.) #en mètres par seconde
vitesse_equateur_km_par_s = vitesse_equateur/1000. #en kilomètres par
seconde

print("Vitesse estimée des points à l'équateur:",
vitesse_equateur_km_par_s, "km/s")

#Comparaison avec la vitesse theorique :
print("Rapport avec la vitesse estimée à partir du MCU et gravitation de
Newton:", 100.*vitesse_equateur_km_par_s/vitesse_grav_km_par_s, "%")
```

LIENS

Projet Mission Space Lab

La plupart de ces liens sont uniquement disponible en anglais.

Page de présentation du projet
https://astro-pi.org/mission-space-lab/

Règlementation relative au projet
https://astro-pi.org/mission-space-lab/rulebook

Guide du créateur
https://projects.raspberrypi.org/en/projects/mission-space-lab-creator-guide/0

Guide de l'estimation de la vitesse de l'ISS
https://projects.raspberrypi.org/en/projects/astropi-iss-speed

Guide du mentor
https://astro-pi.org/mission-space-lab/

IDE Thonny

https://thonny.org/

Conclusion/Pour aller plus loin

À l'issue de cette activité, vos élèves auront acquis les bases essentielles de la programmation en Python, tout en mobilisant des notions clés de leur cours de physique. Ce parcours introductif constitue ainsi une excellente préparation au **challenge officiel Mission Space Lab**, qui leur permettra d'aller plus loin en explorant des techniques avancées, telles que le traitement d'images pour estimer la vitesse de l'ISS de manière plus réaliste.

Pour prolonger l'expérience ou vous lancer dans la version complète du challenge, rendez-vous dans le **dossier AstroPi Mission Space Lab** disponible sur esero.be.