Script Python - Cherchez l'erreur

1. Les bonnes habitudes

Pour bien programmer avec Python, il est indispensable de structurer son script et de respecter certaines conventions pour que l'on puisse s'y repérer facilement et qu'il soit compréhensible sans effort par d'autres utilisateurs.

1.1. Sauvegarder son script

La première étape à réaliser est de sauvegarder le script dans un répertoire adapté (en général dans un dossier créé dans `mes documents` de votre espace personnel).

Quelques règles d'usage :

- Choisir un nom explicite
- Ne pas utiliser de caractères accentués, ni de caractères spéciaux à l'exception de _.
- Ne pas sauvegarder dans le répertoire par défaut.

Exemple

- `TD_dichotomie.py`
- `TD_avalanche.py`

1.2. Bien structurer son script

On va prendre l'habitude de structurer nos scripts en 4 parties qui peuvent être complétées de manière non linéaire.

- Bloc 1 : importation des bibliothèques,
- Bloc 2 : affectation des variables (données du problème),
- Bloc 3 : définition des fonctions,
- Bloc 4 : résolution du problème, affichage des résultats.

Un exemple est commenté à la page suivante.

```python
#Importation des bibliothèques
import numpy as np
import matplotlib.pyplot as plt

#Affectation des variables (données du problème)

m =1e-2    #masse en kg
k =4       #constante de raideur du ressort en N/m
w0 =np.sqrt(k/m)  #pulsation propre en rad/s
d = 1e-2   #position initiale en m
v0 =0      #vitesse initiale en m/s
#vecteur temps (2 périodes, 100 points)
vec_temps =np.linspace(0,4*np.pi/w0,100)


#Définition des fonctions
def position(t,w,x0):
    """renvoie la position (type float ou np.array)
    t : float ou liste ou np.array
    w : type float (pulsation propre)
    x0: type float (position initiale )"""
    return x0*np.cos(w*t)

def vitesse(t,w,x0):
    """renvoie la position (type float ou np.array)
    t : float ou liste ou np.array
    w : type float (pulsation propre)
    x0: type float (position initiale )"""
    return -x0*w*np.sin(w*t)

def maximum(vec):
    """renvoie le max d'une liste ou d'un np.array
    prend en argument une liste ou un np.array"""
    maxi =vec[0]
    for elem in vec :
        if elem >maxi:
            maxi= elem
    return maxi

#Résolution et appel des fonctions

vec_position = position(vec_temps,w0,d)
vec_vitesse = vitesse(vec_temps,w0,d)
vit_max =maximum(vec_vitesse) #vitesse maximale
print('vitesse maximum = ', vit_max)

plt.plot(vec_position,vec_vitesse,'*b')
plt.xlabel('position en m')
plt.ylabel('vitesse en m/s')
plt.title('Portrait de phase')
plt.show()  #affichage du graphique
```

Python

Bloc 1 :
On commence par importer les bibliothèques nécessaires à l'exécution du script.

Bloc 2 : Affectation des différentes variables :
comme en Chimie, en Physique et en Sciences e l'Ingénieur, on va utiliser au maximum des expressions littérales pour pouvoir faire varier les paramètres simplement sans modifier tout notre script.

On prendra soin de donner un nom explicite aux variables utilisées, et de préciser en commentaire leur signification et leur unité.

Ces différentes variables seront soit passées en argument des fonctions soit utilisées comme variables globales à l'intérieur des fonctions (peu recommandé dans le cadre du programme d'IPT).

Bloc 3 : Définition des fonctions :
On définit les fonctions sans se préoccuper de l'ordre dans lequel on aura besoin de les appeler.
On prend soin de préciser le type des arguments et le type des grandeurs renvoyées.

La documentation placée en début de fonction `"""..."""` peut être affichée dans une console avec l'aide : `help(maximum)`

Bloc 4 : Résolution du problème :
On appelle les différentes fonctions, et on stocke leur résultat (grandeur renvoyée) dans de nouvelles variables (affectation de `vec_temps` et `vec_vitesse` par exemple).

On peut tracer des courbes, afficher des résultats (commande `print()`).

Quand on teste un script, il est souvent plus pratique d'appeler les fonctions à l'intérieur de celui-ci plutôt que dans la console. Par contre, on n'hésitera pas à commenter # les parties du code devenues inutiles.
Par exemple, pour éviter d'ouvrir une fenêtre graphique à chaque exécution, on commentera la ligne `plt.show()`.

2. Anatomie d'une erreur Python

Python nous permet d'exécuter de nombreuses tâches identiques et sans effort, mais en contrepartie, nous faisons avec Python toujours les mêmes erreurs. Pour repérer ces erreurs et les corriger de manière très efficace, il est indispensable de savoir lire et d'être méthodique.
Une exception est un mécanisme d'interruption du programme utilisé pour signaler que quelque chose d'anormal est en train de se produire.
On les rencontre dans de nombreux cas, mais souvent, c'est dans le cadre d'erreurs. Par exemple, Python va lever une exception dans les cas suivant :
Lorsqu'un programme Python plante, c'est en général en raison d'une exception non gérée. Voici le genre d'affichage que cela produit dans la console :

Par exemple, en exécutant le script suivant, Python va lever une exception :

Python
```
liste =[0,1,2]

print( liste [3])
```

On obtient alors l'affichage (simplifié) suivant dans la console :

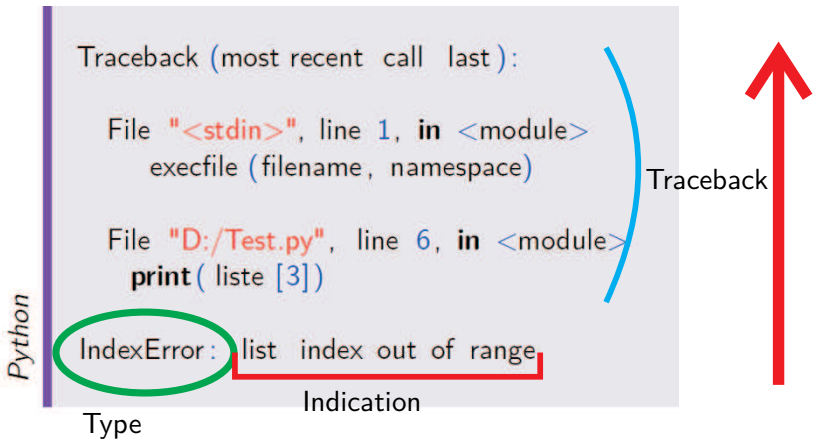
Python
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    execfile (filename, namespace)
  File "D:/Test.py", line 6, in <module>
    print( liste [3])
IndexError: list index out of range
```

Analyse du message d'erreur :

Les erreurs affichées par Python se lisent toujours **de bas en haut**. La ligne la plus importante est **la dernière** sur laquelle on trouve 2 informations :

→ La première, c'est le type d'erreur qui s'est produit. Dans cet exemple, il s'agit d'une `IndexError`.

→ La seconde, c'est le message contenu dans cette exception. Au travers de ce message, Python donne des indications sur ce qui a provoqué l'erreur. Ici, il s'agit d'une erreur d'indice qui ne se trouve pas dans l'intervalle attendu.



Les lignes qui précèdent la dernière sont ce que l'on appelle le "traceback". C'est ce qui permet de trouver rapidement où s'est produite l'erreur.

Ces lignes vont en général 2 par 2 :

→ La première ligne indique, le fichier dans lequel on se trouve (ici `D:/Test.py`), la ligne de ce fichier à laquelle s'est produite l'erreur, et le nom de la fonction que l'on est en train d'exécuter (ici `print(3)`).

→ La seconde ligne présente l'instruction qui a fait planter le programme.

Les paires de lignes qui précèdent celle-ci sont les appels de fonctions successifs qui ont mené l'interpréteur jusqu'à cette instruction. Cela permet de savoir à peu près où en était votre programme dans son exécution avant qu'il ne plante. Il arrive parfois que l'erreur se trouve avant la ligne indiquée (par exemple lorsqu'on oublie de fermer une parenthèse).

Rq

Il est possible d'anticiper d'éventuelles erreurs en utilisant la construction **try/except** :

```
try:
    int(input("Donne moi au hasard un nombre entier"))
except ValueError:
    print("Rentre un entier, pas un mot")
```

Enfin, si votre programme tourne de façon anormalement longue, cela suggère une boucle infinie. Interrompez l'exécution du programme sur la console en tapant au clavier `ctrl+c (KeyboardInterrupt)` ou en utilisant l'icône dédié de votre logiciel. Puis identifiez la boucle **while** fautive.

3. Résumé des erreurs classiques

Voici une liste non exhaustive des erreurs rencontrées :

Nom de l'erreur	Explications de l'erreur
`ImportError`	Erreur sur le nom du module importé.
`IndentationError`	Problème d'indentation. Mettre des indentations après " :". Ne jamais mélanger des tabulations avec des espaces. Aligner les indentations.
`IndexError`	Erreur dans l'indice d'une liste.
`IOError`	Erreur avec un fichier. Vérifier que le chemin d'accès est correct, que le fichier est fermé après utilisation, que vous avez bien préciser les droits ('r', 'w', 'r+').
`NameError`	Variable ou fonction non définies. Vérifier l'orthographe, l'importation des modules, les guillemets pour une chaîne de caractères.
`SyntaxError`	Problème de syntaxe. Relire très attentivement le code.
`RuntimeError`	Appel trop important d'une fonction récursive ou suspicion de boucle infinie.
`TypeError`	Confusion dans le type de la variable, erreur de syntaxe. Vérifier votre syntaxe et vérifier que que le type des variables est cohérent avec les fonctions utilisées.
`UnboundLocalError`	Variable locale non définie. Initialiser vos variables locales.
`ValueError`	Problème de type (confusion chaîne de caractères, nombre entier ou flottant).
`ZeroDivisionError`	Division ou modulo par 0.
`AttributeError`	On essaie d'appliquer une méthode ou fonction non définie pour la classe d'un objet

Les bons réflexes en cas d'erreur inextricable :

- ◇ Relancer la console.
- ◇ Vérifier le nom du fichier, et le dossier dans lequel il est enregistré.
- ◇ Faire des print pour suivre l'interprétation de notre script en mode pas à pas.
- ◇ Commenter une partie de programme si celui-ci ne s'exécute pas et décommenter petit à petit.