Déboguer un programme

La phase de débogage d'un programme est très gourmande en temps.
Elle consiste à rechercher des erreurs de programmation.
C'est particulièrement vrai en Pyhton, dont le typage dynamique repousse la découverte des fautes, au moment de l'exécution.

1 – Le traceback de Python

Lorsque que votre interpréteur Python rencontre un problème, une **exception** est levé.
Si elle n'est pas capturée, cela provoque l'arrêt du programme et l'affichage d'un traceback.
Ce message permet de connaitre la nature et le contexte de l'erreur.

Lire les messages d'erreur en entier, et se familiariser avec, permet, à la longue, de gagner du temps.

Exo 1

Quel exception s'affiche si l'on tente d'exécuter le code suivant :

```
1  a = 1
2  for i in range(3):
3      print("i = {}, a = {}".format(i, a)
4      a = 2 * a
```

a. NameError b. SyntaxError c. IndexError d.IndentationError

A quel moment l'exception précédente sera t'elle levé ?

a. Avant l'exécution b. Prendant l'exécution

Les erreurs de syntaxe ou d'indentation sont repérées avant l'exécution du code, lors de la première lecture du texte du programme.

Cependant, il faut faire attention, le traceback indique la ligne où l'erreur à était detecté, pas forcément celle ou elle à était commise. Par exemple, lors du premier exercice, l'erreur est signalé ligne 4, mais elle est commise à la ligne 3.

2 – Erreur de syntaxe

Les erreurs de syntaxe vont empêcher l'interpréteur de comprendre le code écrit, et provoquerons la levée d'une exception avant même l'exécution du code.
Ces erreurs sont souvent facile à trouver :

- parenthèses, crochets ou guillemets mal fermé (syntaxError)
- mauvaise indentation (IndentationError)

```
for i in range(1, 10) :
    print("Quel est le carré de {} ?".format(1)
    print("C'est : {}".format(i**2))
```

L'erreur est levée ligne 3 mais se trouve ligne 2. Comme les instructions peuvent courir sur plusieurs ligne, elle n'est detecté que ligne 3.
Pour les erreurs de syntaxe, la petite flèche indique sur quel élèment porte l'erreur.

Attention, il ne faut pas mélanger les espaces et les tabulations pour l'indentation du code. Ce type d'erreur est difficile à repérer car la largeur d'une tabulation peut correspondre à celle de plusieurs espace.
Une tabulation doit insérer 4 espaces (il est possible de le configurer dans l'IDE).

Exo 2

Quel exeception va lever le code suivant :

```
1  v = 1
2  while v < 100:
3      if v % 7 == 0:
4          print(v, "est un multiple de 7")
5          else :
6          print(v, "n'est pas un multiple de 7")
7      v = v + 1
```

a. NameError b. SyntaxError c. IndexError d. IndentationError

A quel moment cette erreur est elle levée ?

a. avant l'exécution b. pendant l'exécution

3 - Erreurs à l'exécution

Les exceptions levées à l'exécution sont plus difficile à trouver, car elles nécessitent de comprendre l'exécution du code.
Elles sont aussi plus variées :

- NameError : un nom de variable a-t-il était mal orthographié ?
- IndexError : l'indice se trouve t-il en dehors de la liste ?
- TypeError : a-t-on essayé d'ajouter un nombre à une chaîne de caractère ?

S'en parler du type d'exception ni même de la ligne l'ayant levée, le traceback contient un historique des appels de fonction (la pile d'appel), qui permet de connaître le contexte d'exécution.
La recherche d'erreurs s'apparente alors a une enquête, il faut déterminer la cause (qui peut être à un tout autre endroit), c'est pour cette raison que le traceback se lit de bas en haut.

Exo 3

Le code suivant calcul les 10 premiers terme de la suite de Fibonacci, quel exception s'affiche si l'on tente de l'exécuter ?

```
f = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
for i in range(1, 10):
    f[i + 1] = f[i] + f[i - 1]
print(f)
```

a. NameError b. SyntaxError c. IndexError d.IndentationError

Lors du dernier tour de boucle, *i* vaut 9. On essaie alors de mettre à jour la valeur d'indice *i*+1 (c'est-à-dire 10) de *f*. Or *f* est une liste de longueur 10 dont les indices valides vont de 0 à 9 inclus. Tenter d'accéder à l'élément 10 provoque donc la levée d'une exception.

4 – Débogueur

Il permet de dérouler un programme pas à pas et de vérifier l'état de chaque variable.
C 'est un outil parfois indispensable pour comprendre des bugs complexes.

Le débogueur post-mortem permet d'inspecter l'état du programme, par exemple le contenu des variables, à partir du lieu où l'exception à était levée.
C'est souvent suffisant pour comprendre une erreur.

La plupart des IDE proposent un débogueur.

https://docs.python.org/fr/3/tutorial/errors.html