Chap. 1 Introduction à la programmation Python Thème 1

Erreurs et « bugs »

Sommaire du chapitre 1

Plusieurs types d'erreurs, ou « bugs », peuvent se produire lorsque l'on exécute un programme Python. Un message d'erreur indique parfois la cause de l'erreur.

Un peu d'histoire

L'usage du terme « bug » (insecte) vient du fait qu'un papillon de nuit qui s'était glissé dans les circuits d'un des premiers ordinateurs était à l'origine d'un dysfonctionnement. La traduction française officielle est « bogue ».

I Erreurs de syntaxe

Les **erreurs de syntaxe** sont dues au non-respect des règles d'écriture de Python, comme une indentation incorrecte ou, ici, l'oubli d'une parenthèse :

```
>>> for i in range(10:
  File "<stdin>", line 1
    for i in range(10:
                   ^
SyntaxError: invalid syntax
```

II Erreurs de définition

Les **erreurs de définition** sont dues à l'usage d'un nom qui n'a pas encore été défini, par exemple une fonction ou une variable qui n'existent pas encore, comme ici la fonction `avance` :

```
>>> avance()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'avance' is not defined
```

III Erreurs de type

Les **erreurs de type** sont dues à des valeurs dont le type n'est pas compatible avec l'expression dans laquelle elles apparaissent, comme ici la tentative d'ajouter un entier et une chaîne de caractères :

```
>>> print(3 + "cm")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

IV Erreurs d'exécution

Les **erreurs d'exécution** sont dues à des instructions que l'ordinateur ne sait pas exécuter, comme une division par zéro :

```
>>> 10/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

V Erreurs de logique

Les **erreurs de logique** n'occasionnent en général pas de message d'erreur car il s'agit d'erreurs dans la logique du programme, qui n'enfreignent pas les règles de Python. Ce sont souvent les erreurs les plus difficiles à résoudre. Une erreur de logique courante est la **boucle infinie** (cf. leçon sur Instructions) : le programme ne s'arrête jamais car la condition d'une boucle non bornée ne devient jamais fausse :

```
1 while x > 0 :
2     x = x + 1
```

Dans ce cas, il faut en général interrompre le programme « de force » en tapant `Control-C`.
Il n'existe pas de méthode miracle pour « débugger » mais il est souvent utile d'ajouter des **traces** dans son programme pour vérifier qu'il fonctionne comme prévu, ou pour comprendre la cause de l'erreur, notamment à l'aide de la fonction `print()`.

Les environnements de programmation tels que Spyder ou EduPython permettent d'exécuter un programme *pas à pas*, c'est-à-dire une instruction à la fois, et d'explorer les valeurs des variables à chaque pas.