# Les messages d'erreurs en Python

## Présentation d'un message d'erreur

En général, un message d'erreur (on dit aussi **exception**) se présente de la façon suivante :

```
1 Traceback (most recent call last):
2   File "nom-fichier.py", line ??, in <module>
3     instruction
4               ^
5 TypeError: description
```

On peut distinguer plusieurs parties :

- en ligne 2, le nom du fichier accompagné du numéro de ligne qui a généré l'erreur ;
- en ligne 3, l'instruction qui a généré l'erreur ;
- en ligne 4, un éventuel curseur prenant la forme d'un accent circonflexe et qui est une tentative de Python de vous indiquer où se situe l'erreur ;
- en ligne 5, le type d'erreur accompagné de sa description.

Cette présentation peut varier notamment si l'erreur a lieu dans un appel de fonction auquel cas Python affichera des lignes supplémentaires indiquant sur quelle ligne se situe l'erreur dans le corps de la fonction. Pour un liste de toutes les exceptions pouvant être levées par Python : https://docs.python.org/fr/3.8/library/exceptions.html

## Les messages d'erreurs fréquents

### 1) Les erreurs de syntaxe

Ce sont des erreurs d'écriture ou plus précisément de non respect des règles grammaticales du langage Python. Elles peuvent être provoquées par l'oubli des deux points après un `if`, un `for`, un `while`, une définition de fonction ou bien encore par l'utilisation du = symbole d'affectation en lieu et place du double égal == symbole de comparaison.

```
>>> if i == 1 # oubli des :
            ^
SyntaxError: invalid syntax
>>> if i = 1 # oubli du ==
         ^
SyntaxError: invalid syntax
```

Dans l'exemple suivant, c'est l'oubli du guillemet fermant la chaîne qui provoque une erreur.

```
>>> print("Hello world !!!)
  File "<stdin>", line 1
    print("Hello world !!!)
                          ^
SyntaxError: EOL while scanning string literal
```

## 2) Les erreurs de type

Elles sont la plupart du temps dues à un oubli de transtypage. Dans l'exemple ci-dessous, on tente de concaténer une chaîne avec un entier, ce qui est impossible.

```
>>> i = 10
>>> print("i = " + i) # pensez à transtyper i avec str(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

On a le même type d'erreur dans l'exemple suivant sauf que dans ce cas c'est un type numérique que l'on tente d'additioner à une chaîne.

```
>>> 2 + "deux"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Autre cas provoquant une erreur de type, lorsqu'on réalise un appel de fonction en fournissant plus ou pas assez de paramètres que nécessaire. Ici, la fonction `f` ne prend qu'un seul paramètre.

```
Traceback (most recent call last):
  File "script.py", line 4, in <module>
    print(f(5,2))
TypeError: f() takes 1 positional argument but 2 were given
```

Il est possible que vous tentiez de modifier une donnée non mutable comme un tuple par exemple, auquel cas vous obtiendrez ce genre de message :

```
t = (1, 5)
t[0] = 10
# A l'exécution
Traceback (most recent call last):
  File "script.py", line 2, in <module>
    t[0] = 10
TypeError: 'tuple' object does not support item assignment
```

## 3) Les erreurs d'indentation et de tabulation

Une erreur d'indentation est levée lorsque les règles d'indentation du langage Python ne sont pas respectées. Vérifiez vos blocs de code et l'alignement de vos instructions.

```
IndentationError: expected an indented block
IndentationError: unexpected indent
IndentationError: unindent does not match any outer indentation level
```

Les erreurs de tabulation quant à elles sont pénibles à régler. Elles sont dues à une mauvaise gestion des espaces et des tabulations par votre éditeur de code. Celui-ci possède souvent une option qui

permet de résoudre le problème via une commande de menu du style `Convert tabs to spaces` ou `Format -> Untabify region`. N'hésitez pas à réécrire le bloc de code mis en cause si l'erreur persiste.

```
TabError: inconsistent use of tabs and spaces in indentation
```

## 4) Les erreurs d'indexation

Vous tentez d'accéder à un élément se trouvant en dehors d'une séquence. Rappelez-vous que Python indexe les séquences en partant de 0.

```
voyelles = ['a', 'e', 'i', 'o', 'u', 'y']
print(voyelles[6]) # le dernier élément de la séquence est voyelle[5]
# A l'exécution
Traceback (most recent call last):
  File "script.py", line 2, in <module>
    print(voyelles[6])
IndexError: list index out of range
```

## 5) Autres erreurs

Une erreur d'évaluation est levée lorque vous tentez de transtyper une chaîne de caractères alphabétiques en un entier ou un flottant. Dans la très grande majorité des cas, ce sera lors d'une saisie utilisateur via `input`.

```
Traceback (most recent call last):
  File "script.py", line 1, in <module>
    int(input("Entrer un nombre : "))
ValueError: invalid literal for int() with base 10: 'deux'
```

Les erreurs de nommage apparaissent lorque vous manipulez une variable qui n'existe pas. Peut-être vous trompez-vous dans l'écriture de la variable. Par exemple, on écrit `adrese` au lieu d'`adresse`.

```
NameError: name 'adrese' is not defined
```

En informatique comme en mathématiques, il est interdit de diviser par 0. Le message est alors suffisamment explicite.

```
ZeroDivisionError: integer division or modulo by zero
```

 