Comprendre les erreurs en Python

1 Erreurs lors de l'envoi à l'interpréteur

▷ `Could not run code because it is incomplete.`

Le programme envoyé à Python est incomplet (cela peut être une parenthèse pas fermée, un `if`, un `for` ou un `while` sur la dernière ligne, et non suivi d'un bloc, etc.)

2 ImportError

▷ `ImportError: No module named '...'`

Il y a une erreur dans le nom du module (ou le module n'est pas installé sur l'ordinateur).

▷ `ImportError: cannot import name '...'`

La fonction indiquée n'a pas été trouvée dans le module (soit il y a une erreur dans le nom de la fonction, soit ce n'est pas le bon module)

3 IndentationError

▷ `IndentationError: expected an indented block`

Python attendait une ligne décalée vers la droite par rapport à la ligne précédente, car elle contenait un `for`, un `while`, un `if`, un `else`, un `def`...

▷ `IndentationError: unexpected indent`

Le début de la ligne se situe trop à gauche ou trop à droite. Ce peut simplement être un espace en trop en début de ligne.

4 IndexError

▷ `IndexError: list index out of range`

On a tenté d'accéder à un élément dans une liste, un tuple... en fournissant un index trop grand ou trop petit.

Exemple : « `L[len(L)]` » au lieu de « `L[len(L)-1]` »

Il peut arriver que cette erreur apparaisse suite à une omission de « `range` » dans une boucle for : « `for i in (0, len(L)) :` » au lieu de « `for i in range(0, len(L)) :` »

5 NameError / UnboundLocalError

▷ `NameError: name '...' is not defined`

Le nom (rappelé par Python) n'est pas défini. Pensez à vérifier les majuscules/minuscules (et évitez les accents dans la mesure du possible dans les noms).

▷ `UnboundLocalError: local variable '...' referenced before assignment`

Le nom indiqué par Python a été utilisé dans une expression dans la fonction préalablement à la première affectation de ce nom.

6 MemoryError / RecursionError / RuntimeError

▷ `MemoryError: ...`

Une liste ou un tableau sont trop grands pour tenir dans la mémoire de l'ordinateur.

▷ `RecursionError: maximum recursion depth exceeded`

▷ `RuntimeError: maximum recursion depth exceeded`

Une fonction récursive ne termine pas, revoir la terminaison.

7 OverflowError / ValueError / ZeroDivisionError

▷ `OverflowError: ...`

Le résultat du calcul excède les capacités de représentation des nombres flottants ou complexes.

▷ `ValueError: math domain error`

L'argument de la fonction n'est pas dans le domaine de définition.

▷ `ValueError: ... arg is an empty sequence`

La fonction indiquée attendait une liste, un tableau ou une séquence non vide, et l'argument ne contient aucun élément.

▷ `ZeroDivisionError: division by zero`

On a tenté d'effectuer une opération liée à une division (division `/`, division entière `//`, opérateur `%`...) avec un diviseur nul.

8 SyntaxError

▷ `SyntaxError: can't assign to function call`

À gauche du signe égal se trouve une fonction, quand Python y attends un nom. Cela peut être dû à une inversion des éléments à gauche et à droite du signe égal : « `sin(2) = x` » au lieu de « `x = sin(2)` ».

▷ `SyntaxError: can't assign to literal`

À gauche du signe égal se trouve autre chose qu'un nom.

Cela peut notamment arriver si l'on écrit « `a=1, b=2` » au lieu de « `a, b = 1, 2` ».

▷ `SyntaxError: EOL while scanning string literal`

Une chaîne de caractère (ouverte par des guillemets) n'est pas/mal fermée.

▷ `SyntaxError: invalid syntax`

C'est probablement l'erreur la plus générale : quelque chose ne respecte pas la syntaxe Python. Un marqueur `^` indique le caractère que Python a échoué à interpréter, mais l'erreur peut être située avant.

En toute fin de ligne commençant par **`for`**, **`while`**, **`if`**, **`else`** ou **`elif`** (entre autres), c'est possiblement un oubli des deux points « : ». Si Python désigne une parenthèse ou un crochet fermant, c'est possiblement qu'il manque le pendant ouvrant. Entre deux arguments d'une fonction, cela peut être une virgule oubliée. Dans un test, cela peut être le mauvais opérateur (« `=` » au lieu de « `==` », « `=!` » au lieu de « `!=` », « `=>` » au lieu de « `>=` »...) Si la ligne ne semble contenir aucune erreur, c'est probablement une parenthèse ou un crochet mal fermé sur la ligne du dessus.

9 TypeError / AttributeError

Si une `TypeError` ou une `AttributeError` fait mention de `NoneType`, voir aussi la section dédiée plus bas.

▷ `AttributeError: '...' object has no attribute '...'`

L'objet désigné par le nom situé à gauche d'un point « `.` » n'est pas du bon type pour que l'on puisse utiliser ce qui suit le point (par exemple `L.append(1)` lorsque `L` n'est pas une liste, `x.real` lorsque `x` n'est pas un complexe, etc.).

▷ `TypeError: '...' object is not subscriptable`

Un crochet ouvrant est précédé par quelque chose qui n'est ni une liste, ni un dictionnaire, ni une séquence... Python vous indique le type de ce qui précède le crochet ouvrant incriminé.

Si le type indiqué est une fonction, peut-être avez-vous mis des crochets au lieu de parenthèses : « `abs[i]` » au lieu de « `abs(i)` ».

▷ `TypeError: '...' object is not callable`

Une parenthèse ouvrante est précédée par quelque chose qui n'est ni un nom de fonction, ni un opérateur. Python vous indique le type de ce qui précède la parenthèse ouvrante incriminée. Ce peut être un signe `*` oublié, « `2(x+1)` » au lieu de « `2*(x+1)` ».

Si le type indiqué est une liste, peut-être avez-vous mis des parenthèses au lieu de crochets : « `L(i)` » au lieu de « `L[i]` ».

▷ `TypeError: '...' object is not iterable`

L'objet qui trouve à droite d'un « **`in`** », qui est l'unique argument de `max`, `min` ou `sum`, à droite d'un `+=` suivant un nom de liste, ou bien à droite d'un signe égal à gauche duquel il y a plusieurs noms n'est pas, comme Python l'attendait une liste, un `range`...

Exemple : « **`for`** `i` **`in`** `len(L) :` » au lieu de « **`for`** `i` **`in`** `range(len(L)) :` », « `L += 1` » au lieu de « `L += [ 1 ]` », etc.

▷ `TypeError: unsupported operand type(s) for +: 'int' and 'list'`

Vous avez essayé d'utiliser un opérateur binaire de façon incorrecte. Python vous précise l'opérateur, et le type des deux arguments : ici, on a essayé d'additionner (`+`) un entier (`int`) et une liste (`list`). Par exemple parce que l'on a écrit « `s = s + L` » au lieu de « `s = s + L[i]` »

Si l'un des deux types est une fonction, peut-être est-ce un oubli des arguments, ou des parenthèses pour une fonction sans arguments. Si c'est un nom, il faut possiblement remonter à son affectation (« `x = random` » au lieu de « `x = random()` » par exemple)

▷ `TypeError: can only concatenate list (not "int/float...") to list`

Même chose que l'erreur précédente, lorsque l'on utilise l'opérateur `+`, une liste à gauche, et autre chose qu'une liste (un entier, un flottant...) à droite.

Si le type est une fonction, voir l'erreur précédente.

▷ `TypeError: unorderable types: bool() < str()`

Vous avez essayé d'utiliser un opérateur de comparaison de façon incorrecte. Python vous précise l'opérateur, et le type des deux arguments : ici, on a essayé de comparer (`<`) un booléen (`bool`) et une chaîne de caractères (`str`).

Si l'un des deux types est une fonction, voir deux erreurs au-dessus.

10 Erreurs impliquant le type « NoneType »

Les erreurs « `NoneType` » surviennent lorsqu'un nom ne désigne rien. Cela arrive fréquemment parce que le nom désigne le résultat d'une fonction et que l'on a oublié un **`return`** dans cette fonction (ou que l'on a utilisé `print` plutôt que **`return`**). Cela arrive aussi lorsque l'on écrit « `L = L.append(0)` » au lieu de « L.append(0) », ou bien « L = L.sort() » au lieu de « L.sort() ». On rappelle que de nombreuses fonctions sur les listes agissent sur la liste, mais *ne retournent rien.*