
Analyse du traceback

Laurent Signac – CC-BY-SA – 02-12-23 1528 631aee9aedcd60ae9f0d

There are two ways to write error-free programs; only the third one works. – *Alan Perlis*

💡 Le document Web Traquer les erreurs^a est complémentaire de celui-ci.

^ahttps://ensip.gitlab.io/pages-info/ressources/transverse/traquer_erreurs/

Le traceback est l'ensemble des lignes affichées par l'interpréteur Python lorsqu'une exception est levée. Comprendre le traceback est **très utile** pour trouver rapidement une faute et l'expliquer.

Voici un exemple de traceback affiché lors de l'exécution d'un code un peu obscur :

```
1 def truc(c):
2     a = [1, 2, 3]
3     a[c] = (1 / c)
4
5 def machin(a, b):
6     d = min(a, b) * 2
7     truc(d)
8
9 for i in range(0, 5):
10    machin(5 - i, i)
```

Le traceback est :

```
Traceback (most recent call last):
  File "/home/signac/obscur.py", line 10, in <module>
    machin(5 - i, i)
  File "/home/signac/obscur.py", line 7, in machin
    truc(d)
  File "/home/signac/obscur.py", line 3, in truc
    a[c] = (1 / c)
ZeroDivisionError: division by zero
```

Ce traceback est constitué d'une liste de lignes en défaut (il y en a 3 ici), puis d'un message qui précise le type d'exception (c'est la dernière ligne).

En lisant le traceback de haut en bas, on avance dans le temps. Ce qui s'est produit en dernier est donc à **la fin** du traceback.

En le lisant de haut en bas, on note (notez la correspondance entre ce qui est indiqué ci-dessous et le contenu du traceback) :

- exécution de la ligne 10 «`machin(5 - i, i)`», donc on entre dans la fonction `machin` qui a provoqué
- l'exécution de la ligne 7 «`truc(d)`», donc on entre dans la fonction `truc` qui a provoqué
- l'exécution de la ligne 3 «`a[c] = (1 / c)`» qui a provoqué
- la levée d'exception `ZeroDivisionError: division by zero`

La plupart du temps, la lecture des **trois dernières lignes** du traceback permet de cerner le problème :

```
File "/home/signac/obscur.py", line 3, in truc
    a[c] = (1 / c)
ZeroDivisionError: division by zero
```

ce qui signifie : Exécution de la ligne 3 «`a[c] = (1 / c)`» qui a provoqué l'exception `ZeroDivisionError`.

La toute dernière ligne du traceback est le message associé à l'exception qui a été levée. Ce message indique quel genre d'erreur s'est produite. Est-ce que c'est un problème d'indentation, de noms de variable non défini ou autre ? Ici, il s'agit d'une division par 0.

Les deux lignes précédentes indiquent à quel moment de l'exécution cette exception a été levée. Ici, en calculant «`a[c] = (1 / c)`».

Il est donc très facile de deviner que le problème est dû au calcul de $1 / c$ (c'est la seule division de la ligne), alors que `c` est nul.

Cette analyse (les trois dernières lignes du traceback) peut être suffisante pour corriger l'erreur ou non.

Si on a besoin de plus de contexte, on remonte dans le traceback, pour savoir **pourquoi** on est arrivé jusqu'à la ligne «`a[c] = (1 / c)`» avec une valeur de `c` manifestement nulle.

💡 À partir de maintenant, lorsque vous obtenez une erreur lors de l'exécution de code Python, vous devez (dans cet ordre) :

1. regarder la dernière ligne du traceback pour connaître la nature de l'exception
2. regarder les deux lignes qui précèdent la dernière pour avoir le numéro de ligne et son contenu
3. et ensuite uniquement, regarder votre code, à l'endroit indiqué, pour comprendre la faute

Pour corriger la faute, se contenter de ces trois points peut suffire, ou il peut être nécessaire de remonter dans le traceback. Si on veut par exemple, empêcher que `c` soit nul, c'est peut-être la ligne 9 qu'il faut corriger :

```

1 def truc(c):
2     a = [1, 2, 3]
3     a[c] = (1 / c)
4
5 def machin(a, b):
6     d = min(a, b) * 2
7     truc(d)
8
9 for i in range(1, 5):
10    machin(5 - i, i)

```

Cette fois-ci, l'exécution donne :

```

Traceback (most recent call last):
  File "/home/signac/obscur.py", line 10, in <module>
    machin(5 - i, i)
  File "/home/signac/obscur.py", line 7, in machin
    truc(d)
  File "/home/signac/obscur.py", line 3, in truc
    a[c] = (1 / c)
IndexError: list assignment index out of range

```

Le message d'exception n'est plus le même. Il y a une autre faute. Peut-être que cette faute a été ajoutée par notre modification, ou peut-être que cette faute était déjà présente, mais qu'auparavant, nous ne l'avions pas atteinte, le programme ayant stoppé son exécution avant : la correction effectuée a peut-être permis d'aller plus loin dans l'exécution.

Saurez-vous répondre à ces questions :

- est-ce que la nouvelle faute a été ajoutée ou bien était-elle déjà présente ?
- quelle ligne de code a provoqué la nouvelle exception ?
- quelle est la raison immédiate de la levée d'exception ?
- quelle était la valeur de `i` lorsque l'exception a été levée ?
- il est peu probable que la ligne qui a provoqué la levée de l'exception soit celle à corriger. Vous pouvez proposer des corrections minimales (tout effacer, par exemple, est une correction qui enlève généralement les erreurs... mais ne présente pas un grand intérêt par ailleurs) qui permettent de ne plus avoir ce message (même si ici, ça n'a pas trop de sens puisqu'on ne sait pas à quoi peut servir cette portion de code)

💡 Depuis Python 3.11, le positionnement de l'exception est mieux décrit dans le Traceback. Certaines améliorations sont déjà présentes dans la version 3.10 (messages plus explicites par exemple).
Pour une erreur de ce type :

```
>>> x = {'a': 1, 'b': {'c': None}}
>>> v = x['b']['c'][4]
TypeError: 'NoneType' object is not subscriptable
```

Le traceback avant Python 3.11 ne permettait pas de savoir si la valeur à None était x, x['b'] ou x['b']['c'].

Depuis Python 3.11, la portion de la ligne en défaut est indiquée par des ^ :

```
v = x['b']['c'][4]
      ~~~~~^~~~~~
TypeError: 'NoneType' object is not subscriptable
```

Certaines erreurs sont détectées à la lecture du code, et d'autres à l'exécution. Les erreurs de syntaxe et d'indentation sont détectées à la lecture du code. Il n'y a donc pas de traceback. Les erreurs de nom ou de type (et les autres) sont détectées à l'exécution. Il y a un traceback. Ça signifie aussi que ces erreurs peuvent passer inaperçues lors d'une exécution (si le code concerné n'est pas exécuté ou que les valeurs en entrée ne conduisent pas à l'erreur).

💡 Le document [Web Traquer les erreurs^a](https://ensip.gitlab.io/pages-info/ressources/transverse/traquer_erreurs/) est complémentaire de celui-ci.

^ahttps://ensip.gitlab.io/pages-info/ressources/transverse/traquer_erreurs/