INF5190 - Introduction à Python (suite)

Jean-Philippe Caissy

11 septembre 2011

Erreurs et exceptions

Il y a généralement deux types d'erreurs en Python : les erreurs de syntaxe et les exceptions.

Erreurs de syntaxe

L'interpréteur va se plaindre d'erreur de syntaxe lorsque le *parser* de Python rencontre un erreur.

```
>>> while True a = 123:
  File "<stdin>", line 1
    while True a = 123:
               ^
SyntaxError: invalid syntax
```

L'interpréteur va également vous pointer avec une petite flèche l'erreur qu'il a trouvé.
Le nom du fichier et la ligne est également affiché. Dans cet exemple, le fichier est l'entrée standard car nous sommes dans l'interpréteur.

Erreurs et exceptions

Exceptions

- Même si une expression est syntaxiquement valide, il se peut qu'un erreur se produise lorsqu'il est exécuté.
- Les erreurs détectés durant l'exécution sont des exceptions en Python. Une exception n'est pas fatale, comme dans d'autres langages, il est possible de les gérer.

Erreurs et exceptions

Exceptions

Exemple

```
>>> 4 * variable_non_existante + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'variable_non_existante' is not defined
>>> '1' + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Erreurs et exceptions

Exceptions

```
>>> def methode_1():
        return method_2()
>>> def method_2():
        return 1 / 0
>>> methode_1()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in methode_1
  File "<stdin>", line 2, in method_2
ZeroDivisionError: integer division or modulo by zero
```

- La dernière ligne indique le type d'exception (`ZeroDivisionError`)
- La pile d'exécution est affichée

Erreurs et exceptions

Exceptions

- Les exceptions sont tous des objets

```
>>> ZeroDivisionError.__class__
<type 'type'>
>>> ZeroDivisionError
<type 'exceptions.ZeroDivisionError'>
>>> mon_exception = ZeroDivisionError
>>> import inspect
>>> inspect.getmro(mon_exception)
(<type 'exceptions.ZeroDivisionError'>,
  <type 'exceptions.ArithmeticError'>,
  <type 'exceptions.StandardError'>,
  <type 'exceptions.Exception'>,
  <type 'exceptions.BaseException'>,
  <type 'object'>)
>>>
```

Erreurs et exceptions

Exceptions

Gestion

La snytaxe `try/except` permet de gérer les exceptions lors de l'exécution

```
>>> def diviser_par_deux(n):
...     try:
...         print(n / 2)
...     except TypeError:
...         print("La valeur n'est pas un chiffre")
...
>>> diviser_par_deux(42)
21
>>> diviser_par_deux("test")
La valeur n'est pas un chiffre
```

Erreurs et exceptions

Exceptions

Gestion

On peut regrouper plusieurs exceptions dans le même bloc

```
....      except TypeError, ZeroDivisionError:
              [...]
```

Ou chainer des exceptions

```
....      except TypeError:
....          print("La veleur n'est pas un chiffre")
....      except ZeroDivisionError:
....          print("Impossible de diviser par zéro")
```

Erreurs et exceptions

Exceptions

Gestion

Il est possible de récupérer l'instance de l'exception.

```
>>> try:
...     f = open("fichier", 'r')
... except IOError as err:
...     print("Erreur : {0}".format(err))
...
Erreur : [Errno 2] No such file or directory: 'fichier'
```

Modules

Un module est un fichier contenant des définitions de classes, de méthodes et d'objets Python. Le nom du fichier sans l'extension `.py` représente le nom du module.

Modules standards

Python vient avec plusieurs modules dans la librairie standards. Pour les utiliser, il suffit de les importer.

```
>>> from datetime import date
>>> import os
>>> today = date.today()
>>> today
datetime.date(2019, 9, 10)
>>> os.path.isfile("/home/jpcaissy/src/vimrc/vimrc")
True
>>> os.curdir
'.'
```

Modules

Avec un fichier nommé `fibonacci.py` qui contient le code suivant :

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=" ")
        a, b = b, a+b
    print()
```

Il est possible d'importer le module et d'accéder aux méthodes :

```
>>> import fibonacci
>>> fibonacci.fib(10)
0 1 1 2 3 5 8
>>> fibonacci.fib(15)
0 1 1 2 3 5 8 13
>>> fibonacci.fib(100)
0 1 1 2 3 5 8 13 21 34 55 89
>>>
```

Modules

Dossiers

Un dossier peut également être représenté comme un module avec la présente du fichier spécial `__init__.py`
Voici le même fichier `fibonacci.py` à l'intérieur d'un dossier `modules`.

```
modules_python/
    __init__.py
    fibonacci.py
```

```
>>> from modules_python import fibonacci
>>> fibonacci.fib(15)
0 1 1 2 3 5 8 13

>>> from modules_python.fibonacci import fib
>>> fib(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

with

L'expression `with` en python permet de couvrir un bloc d'exécution avec un gestionnaire de contexte.

```
with open("file", "r") as f:
    print(f.readlines())
```

est équivalent à

```
f = open("file", "r")
print(f.readlines())
f.close()
```

Le gestionnaire de contexte de la méthode `open` s'assure du nettoyage des ressources après l'exécution du bloc.

Liens utiles

- Erreurs et exceptions
- Modules