Langage Python

2020-2021

Hamza

Exceptions

Erreurs syntaxiques et sémantiques

- En programmation, les erreurs sont par nature syntaxiques ou logiques.

- Syntaxiques, elles signalent un problème de construction et ne peuvent être ni compilées ni exécutées. Elles doivent être corrigées avant l'exécution

- Plusieurs erreurs de logique peuvent intervenir pendant l'exécution d'un programme :
  - insuffisance de mémoire, disque plein,
  - saisie non valide d'une valeur,
  - une fonction qui fonctionne mal, etc.

Exceptions

- Même si une instruction ou une expression est syntaxiquement correcte, elle peut générer une erreur lors de son exécution.

- Les erreurs détectées durant l'exécution sont appelées des *exceptions* et ne sont pas toujours fatales

- La plupart des exceptions toutefois ne sont pas prises en charge par les programmes, ce qui génère des messages d'erreurs comme celui-ci :

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

-***ZeroDivisionError***: le type de l'exception ;
-***division by zero***: le message qu'envoie Python pour vous aider à comprendre l'erreur qui vient de se produire.

Exceptions

- Une exception est dite levée lorsqu'une erreur apparait
- Une exception est dite capturée lorsqu'elle est gérée et traitée.
- Lorsqu'on utilise une opération, fonction, méthode, qui est susceptible de lever une exception :
  - Soit on capture l'exception et on exécute une suite d'instructions qui a pour objectif de gérer l'erreur
  - Soit on ne capture pas l'exception et dans ce cas, la fonction ou méthode en cours s'arrête et propage l'exception levée à la fonction appelante. Si cette propagation remonte jusqu'au programme principal sans que l'exception soit capturée, le programme s'arrête

https://docs.python.org/3/library/exceptions.html

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
```

```
BaseException
 +-- Exception
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
```

```
BaseException
 +-- Exception
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
+-- Warning
      +-- DeprecationWarning
      +-- PendingDeprecationWarning
      +-- RuntimeWarning
      +-- SyntaxWarning
      +-- UserWarning
      +-- FutureWarning
      +-- ImportWarning
      +-- UnicodeWarning
      +-- BytesWarning
      +-- ResourceWarning
```

Exceptions concrètes

Les exceptions suivantes sont celles qui sont habituellement levées.
exception **AssertionError :**Levée lorsqu'une instruction assert échoue.
exception **AttributeError :**Levée lorsqu'une référence ou une assignation d'attribut échoue.
exception **EOFError :** Levée lorsque la fonction input() atteint une condition de fin de fichier (EOF) sans lire aucune donnée
exception **ImportError :** Levée lorsque l'instruction import a des problèmes pour essayer de charger un module. Également levée dans le cas from ... import.
exception **IndexError:** Levée lorsqu'un indice de séquence est hors de la plage. (si un indice n'est pas un entier, TypeError est levée.)
exception **KeyError :** Levée lorsqu'une clef (de dictionnaire) n'est pas trouvée dans l'ensemble des clefs existantes.
exception **NameError :** Levée lorsqu'un nom local ou global n'est pas trouvé.
exception **NotImplementedError :** Cette exception est dérivée de RuntimeError. Dans les classes de base définies par l'utilisateur, les méthodes abstraites devraient lever cette exception lorsqu'elles nécessitent des classes dérivées pour remplacer la méthode, ou lorsque la classe est en cours de développement pour indiquer que l'implémentation concrète doit encore être ajoutée.
exception **OverflowError :** Levée lorsque le résultat d'une opération arithmétique est trop grand pour être représenté. Cela ne peut pas se produire pour les entiers (qui préfèrent lever MemoryError plutôt que d'abandonner). Cependant, pour des raisons historiques, OverflowError est parfois levée pour des entiers qui sont en dehors d'une plage requise.
exception **TypeError :** Levée lorsqu'une opération ou fonction est appliquée à un objet d'un type inapproprié.
exception **ValueError :**Levée lorsqu'une opération ou fonction native reçoit un argument qui possède le bon type mais une valeur inappropriée, et que la situation n'est pas décrite par une exception plus précise telle que IndexError.
exception **ZeroDivisionError :**Levée lorsque le second argument d'une opération de division ou d'un modulo est zéro. La valeur associée est une chaîne indiquant le type des opérandes et de l'opération.

https://docs.python.org/fr/3.7/tutorial/errors.html

Attraper une exception

try signifie "essayer" en anglais, ce mot clé permet d'essayer un bloc d'instructions

Si une exception se produit, le reste de la clause **try** est ignorée, et les instructions dans un bloc **except** sont exécutées.

Dans le cas contraire, on ignore les instructions du bloc **except**.

```
try:
        <Blocs à essayer>
except :
        <Bloc qui sera exécuté en cas d'erreur>
```

Attraper une exception

Une instruction **try** peut comporter plusieurs clauses **except** pour permettre la prise en charge de différentes exceptions. Mais un seul gestionnaire, au plus, sera exécuté.
Mais une même clause **except** peut citer plusieurs exceptions sous la forme d'un tuple entre parenthèses.

```
try:
        < blocs d'instructions >
except nomException1 :
        <blocs d'instructions >
except nomException2 as donnee:
        <blocs d'instructions>
except (nomException3, nomException4):
        <blocs d'instructions>
except:
        <blocs d'instructions>
```

Attraper une exception

Une clause **else**: peut suivre toutes les clauses **except** : elle est exécutée lorsque aucune exception n'a été attrapée par les clauses **except** précédentes (il ne peut pas y avoir de clause **except** après un **else**)

```
try:
        < blocs d'instructions >
except nomException1:
        <blocs d'instructions>
except nomException2:
        <blocs d'instructions>
else:
        <blocs d'instructions>
```

Attraper une exception

Une clause **`finally`**: peut terminer l'instruction **`try`**.
Son code est toujours exécuté (qu'il y ait eu une exception ou pas).

```
try:
        < blocs d'instructions >
except nomException1:
        <blocs d'instructions>
except :
        <blocs d'instructions>
finally:
        <blocs d'instructions>
```

Exemple

```
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur n'a pas été définie.")
except TypeError:
    print("L'un des deux variables possède un type incompatible avec la division.")
except ZeroDivisionError:
    print("La variable denominateur est égale à 0.")
else:
    print("Le résultat obtenu est", resultat)
```

Lever une exception

On lève une exception à l'aide l'instruction **raise** suivie de la création d'une instance d'une sous classe de Exception

Le constructeur prend au moins en paramètre une chaîne de caractères explicitant l'erreur

```
raise TypeDeLException("message à afficher")
```

```
class classe:
    def __init__(self,para1,...):
        if para1<0:
            raise ValueError("un argument inappropriée")
        self.para1=para1
        ...
```

Lever une exception

Dans ce programme, on s'assure que l'utilisateur ne saisit pas une année inférieure ou égale à 0 par exemple

```
annee = input("Saisissez une année supérieure à 0 :")
try:
    annee = int(annee)  # Conversion de l'année
    if annee <= 0:
            raise ValueError("l'année saisie est négative ou nulle")
except ValueError:
    print("Vous n'avez pas saisi un nombre.")
```

Assert

`assert` est un moyen de s'assurer, avant de continuer, qu'une condition est respectée. En général, on l'utilise dans des blocs **`try`**

```
assert test
```

Si le test renvoie ***True***, l'exécution se poursuit normalement. Sinon, une exception **`AssertionError`** est levée.

Assert

Dans ce programme, on s'assure que l'utilisateur ne saisit pas une année inférieure ou égale à 0 par exemple

```
annee = input("Saisissez une année supérieure à 0 :")
try:
    annee = int(annee) # Conversion de l'année
    assert annee > 0
except ValueError:
    print("Vous n'avez pas saisi un nombre.")
except AssertionError:
    print("L'année saisie est inférieure ou égale à 0.")
```

Création d'un type d'exception

Pour créer un type d'exception il faut créer une sous classe (ou sous-sous classe) de la classe Exception

On peut la configurer comme toute classe (redéfinir init , ajouter des attributs, ajouter des méthodes) mais elles sont souvent très simples (voire vides, utilisation de pass)

La bonne pratique veut que le nom d'une exception se termine par Error.

```
class MemePointInterditError(Exception):
    pass
```