LES EXCEPTION PYTHON

Option : DEVELOPPEMENT DIGITAL

ETABLIS PAR : Yasser KHIAT, Saad AIT ALLOUCH, Souhaib EDDAHMANI, Mohammed Reda MENYANI

Les exceptions et la gestion des erreurs

- Toutes les erreurs qui se produisent lors de l'exécution d'un programme Python sont représentées par une **exception**. Une exception est un objet qui contient des informations sur le contexte de l'erreur. Lorsqu'une exception survient et qu'elle n'est pas traitée alors elle produit une interruption du programme et elle affiche sur la sortie standard un message ainsi que la pile des appels (*stacktrace*). La pile des appels présente dans l'ordre la liste des fonctions et des méthodes qui étaient en cours d'appel au moment où exception est survenue.
- Si nous prenons le programme suivant :

```
1  def do_something_wrong():
2      1 / 0
3
4  def do_something():
5      do_something_wrong()
6
7  do_something()
```

L’exécution de ce programme affiche sur la sortie d’erreur :

```
$ python3 programme.py

Traceback (most recent call last):
  File "test.py", line 7, in <module>
    do_something()
  File "test.py", line 5, in do_something
    do_something_wrong()
  File "test.py", line 2, in do_something_wrong
    1 / 0
ZeroDivisionError: division by zero
```

Nous voyons que le programme a échoué à cause d’une exception de type ZeroDivisionError et nous avons la pile d’erreur qui nous indique que le programme s’est interrompu à la ligne 2 suite à l’instruction `1 / 0` dans la fonction `do_something_wrong()` qui a été appelée par la fonction `do_something()` à la ligne 5, elle-même appelée par le programme à la ligne 7.

Traiter une exception

Parfois un programme est capable de traiter le problème à l'origine de l'exception. Par exemple si le programme demande à l'utilisateur de saisir un nombre et que l'utilisateur saisit une valeur erronée, le programme peut simplement demander à l'utilisateur de saisir une autre valeur. Plutôt que de faire échouer le programme, il est possible d'essayer de réaliser un traitement et, s'il échoue de proposer un traitement adapté :

```
1  nombre = input("Entrez un nombre : ")
2  try:
3      nombre = int(nombre)
4  except ValueError:
5      print("Désolé la valeur saisie n'est pas un nombre.")
```

À la ligne 2, on démarre un bloc try pour indiquer le traitement que l'on désire réaliser. Si ce traitement est interrompu par une exception, alors l'interpréteur recherche un bloc except correspondant au type (ou à un type parent) de l'exception. Dans notre exemple, si la fonction int ne peut pas créer un entier à partir du paramètre, elle produit une exception de type ValueError. Donc après le bloc try, on ajoute une instruction except pour le type ValueError (lignes 4 et 5). Ce bloc n'est exécuté que si la conversion en entier n'est pas possible.

Lorsqu'une exception survient dans un bloc try, elle interrompt immédiatement l'exécution du bloc et l'interpréteur recherche un bloc except pouvant traiter l'exception. Il est possible d'ajouter plusieurs blocs except à la suite d'un bloc try. Chacun d'entre-eux permet de coder un traitement particulier pour chaque type d'erreur :

```python
try:
    numerateur = int(input("Entrez un numérateur : "))
    denominateur = int(input("Entrez un dénominateur : "))
    resultat = numerateur / denominateur
    print("Le resultat de la division est", resultat)
except ValueError:
    print("Désolé, les valeurs saisies ne sont pas des nombres.")
except ZeroDivisionError:
    print("Désolé, la division par zéro n'est pas permise.")
```

L'intérêt de la structure du try except est qu'elle permet de dissocier dans le code la partie du traitement normal de la partie du traitement des erreurs.

Si le même traitement est applicable pour des exceptions de types différents, il est possible de fournir un seul bloc except avec le tuple des exceptions concernées :

```python
try:
    numerateur = int(input("Entrez un numérateur : "))
    denominateur = int(input("Entrez un dénominateur : "))
    resultat = numerateur / denominateur
    print("Le resultat de la division est", resultat)
except (ValueError, ZeroDivisionError):
    print("Désolé, quelque chose ne s'est pas bien passé.")
```

Récupérer le message d'une exception :

Les exceptions possèdent une représentation sous forme de chaîne de caractères pour fournir un message. Pour avoir accès à l'exception, on utilise la syntaxe suivante :

```
1  nombre = input("Entrez nombre : ")
2  try:
3      nombre = int(nombre)
4  except ValueError as e:
5      print(e)
```

À la ligne 4, on précise que l'exception de type ValueError est accessible dans le bloc except sous le nom `e` ce qui permet d'afficher l'exception (c'est-à-dire le message d'erreur).

Clause *else:*

Il est possible d'ajouter une clause else après les blocs try except. Le bloc else est exécuté uniquement si le bloc try se termine normalement, c'est-à-dire sans qu'une exception ne survienne.

```
1  try:
2      numerateur = int(input("Entrez un numérateur : "))
3      denominateur = int(input("Entrez un dénominateur : "))
4      resultat = numerateur / denominateur
5  except (ValueError, ZeroDivisionError):
6      print("Désolé, quelque chose ne s'est pas bien passé.")
7  else:
8      print("Le resultat de la division est", resultat)
```

Note

Le bloc `else` permet de distinguer la partie du code qui est susceptible de produire une exception de celle qui fait partie du comportement nominal du code mais qui ne produit pas d'exception.

Post-traitement :

Dans certain cas, on souhaite réaliser un traitement après le bloc try que ce dernier se termine correctement ou bien qu'une exception soit survenue. Dans cas, on place le code dans un bloc finally.

```
1  try:
2      numerateur = int(input("Entrez un numérateur : "))
3      denominateur = int(input("Entrez un dénominateur : "))
4      resultat = numerateur / denominateur
5      print("Le resultat de la division est", resultat)
6  except (ValueError, ZeroDivisionError):
7      print("Désolé, quelque chose ne s'est pas bien passé.")
8  finally:
9      print("afficher ceci quel que soit le résultat")
```

Un bloc finally est systématique appelé même si le bloc try est interrompu par une instruction return.

Lever une exception :

Il est possible de signaler une exception grâce au mot-clé raise.

```
if x < 0:
    raise ValueError
```

Pour la plupart des exceptions, il est possible de passer en paramètre un message pour décrire le cas exceptionnel :

```
if x < 0:
    raise ValueError("La valeur ne doit pas être négative")
```

Le mot-clé raise est également utilisé pour relancer une exception dans un bloc mot-clé except.

```
1  try:
2      numerateur = int(input("Entrez un numérateur : "))
3      denominateur = int(input("Entrez un dénominateur : "))
4      resultat = numerateur / denominateur
5      print("Le resultat de la division est", resultat)
6  except (ValueError, ZeroDivisionError):
7      print("Désolé, quelque chose ne s'est pas bien passé.")
8      raise
```

Dans l'exemple ci-dessus, l'exception traitée dans le bloc except est relancée à la ligne 8.

```
1  try:
2      numerateur = int(input("Entrez un numérateur : "))
3      denominateur = int(input("Entrez un dénominateur : "))
4      resultat = numerateur / denominateur
5      print("Le resultat de la division est", resultat)
6  except (ValueError, ZeroDivisionError) as e:
7      print("Désolé, quelque chose ne s'est pas bien passé.")
8      raise Exception from e
```

Conclusion :

Python est un langage qui a été créé avec l'objectif d'être facilement compréhensible. Pour cette raison, et pour la structure du langage en général qui pousse les développeurs à coder plus proprement et à adopter de bonne pratiques, il est l'un des langages les plus recommandés aux personnes souhaitant s'initier à la programmation. Si Python est simple à apprendre, il n'en est pas moins puissant et versatile : ce langage peut être utilisé pour effectuer des tâches complexes et pour mener à bien différents projets très différents, que ce soit la création d'applications Web ou de programmes divers.