PYTHON – Exceptions

Introduction

En programmation, quel que soit le langage utilisé (et donc en Python), il existe plusieurs types d'erreurs pouvant survenir.

1. on connaît déjà les **erreurs de syntaxe** qui surviennent lorsque l'on fait une faute dans le code source. Ces erreurs sont faciles à corriger car le console peut les signaler.

```
>>> variable ='exemple"
  File "<stdin>", line 1
    variable ='exemple"
                      ^
SyntaxError: EOL while scanning string literal
```

- On voit le fichier et la ligne à laquelle s'est produite l'erreur
- Sur la dernière ligne, nous trouvons 2 informations :
    - SyntaxError : le type de l'erreur
    - EOL while scanning string literal : le message qu'envoie Python pour vous aider à comprendre l'erreur qui vient de se produire

2. Un autre type de problème peut survenir si le programme est écrit correctement mais qu'il **exécute une action interdite**. On peut citer comme exemple le cas où l'on essaye de lire le 10ème indice d'une liste de 8 éléments ou encore le calcul de la racine carrée d'un nombre négatif.
   On appelle ces erreurs les **erreurs d'implémentation** ou des ***exceptions***.

La gestion des exceptions permet de traiter les erreurs d'implémentation en les prévoyant à l'avance. Cela n'est pas toujours possible penser à toutes les erreurs susceptibles de survenir, mais on peut facilement en éviter une grande partie.

Exemples d'erreur d'implémentation

Imaginons que vous ayez décidé de réaliser une calculatrice. Vous auriez par exemple pu coder la division de deux nombres entiers de cette manière

```
def division(a,b): # Calcule a divisé par b.
    return a/b
a=int(input("Valeur pour a : "))
b=int(input("Valeur pour b : "))
print(a," / ",b," = ",division(a,b))
```

Ce code est tout à fait correct et fonctionne parfaitement, sauf dans un cas : si b vaut 0.
En effet, la division par 0 n'est pas une opération arithmétique valide.
Si on lance le programme avec b=0, on obtient une erreur et le message suivant s'affiche

```
Valeur pour a : 6
Valeur pour b : 0
Traceback (most recent call last):
  File "essai.py", line 6, in <module>
    print(a," / ",b," = ",division(a,b))
  File "essai.py", line 2, in division
    return a/b
ZeroDivisionError: division by zero
```

Autres exemples :

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

Les exceptions peuvent être de différents types, et ce type est indiqué dans le message :
les types indiqués dans l'exemple sont **ZeroDivisionError**, **NameError** et **TypeError**.

Principe général

Le principe général des exceptions est le suivant :

1. on crée une zone où l'ordinateur va *essayer* le code en sachant qu'une erreur peut survenir;
    - On utilise le mot-clé **try** suivi des deux points « : » (*try* signifie « essayer » en anglais).
2. si une erreur survient, on créé une zone où l'ordinateur va gérer les erreurs survenues
    - On utilise le mot-clé **except** suivi des deux points « : ». au même niveau d'indentation que le try.

voici la syntaxe :

```
try:
   # Bloc à essayer
except:
   # Bloc qui sera exécuté en cas d'erreur
```

Bloc try

Le mot-clé try permet d'indiquer qu'une certaine portion du code source pourrait générer des exceptions

le mot-clé except permet de créer un bloc de gestion d'une exception survenue.

Chaque bloc try doit obligatoirement être suivi d'un bloc except

Fonctionnement :

- Premièrement, les instruction(s) placée(s) entre les mots-clés try et except sont exécutée(s).
- Si aucune exception n'intervient, la *clause except* est sautée et l'exécution de l'instruction try est terminée.
- Si une exception intervient pendant l'exécution du bloc "try", le reste des instructions est sauté. Les instruction(s) dans "except" sont exécutée(s),
- puis l'exécution continue après le bloc try/except, s'il y a du code bien sûr.

Faisons un test de conversion d'année en entier, en enfermant dans un bloc try l'instruction susceptible de lever une exception. :

```
annee = input("saisir une année")
try: # On essaye de convertir l'année en entier
   annee = int(annee)
except:
   print("Erreur lors de la conversion de l'année.")
print("Aurevoir")
```

Vous pouvez tester ce code en précisant plusieurs valeurs différentes pour la variable annee, comme « 2010 » ou « annee2010 ».

```
saisir une année : annee2010
Erreur lors de la conversion de l'année.
Aurevoir
```

```
saisir une année : 2010
Aurevoir
```

Notre code est maintenant « sécurisé » et correct mais attention :

- ici, si l'utilisateur saisit une année impossible à convertir, le système affiche certes une erreur mais finit par planter (**puisque l'année, au final, n'a pas été convertie**).
  Une des solutions envisageables est :
  - d'attribuer une valeur par défaut à l'année, en cas d'erreur
  - ou de redemander à l'utilisateur de saisir l'année (gérer avec une boucle)
- Ensuite cette méthode essaye une instruction et **intercepte *n'importe quelle* exception liée à cette instruction**. Ici, c'est acceptable car nous n'avons pas énormément d'erreurs possibles sur cette instruction. Mais c'est une mauvaise habitude à prendre.

Voici un autre exemple :

```
try:
   resultat = numerateur / denominateur
except:
   print("Une erreur est survenue... laquelle ?")
```

Ici, plusieurs erreurs sont susceptibles d'intervenir, chacune levant une exception différente.

- **NameError**: l'une des **variables** numerateur ou denominateur n'a **pas** été **définie** (elle n'existe pas).
  Si vous essayez dans l'interpréteur l'instruction print(numerateur) alors que vous n'avez pas défini la variable numerateur, vous aurez la même erreur.
- **TypeError**: l'une des **variables** numerateur ou denominateur **ne peut diviser ou être divisée** (les chaînes de caractères ne peuvent être divisées, ni diviser d'autres types, par exemple).
- **ZeroDivisionError**: Si denominateur vaut 0, cette exception sera levée.

Cette énumération est là pour montrer que

- plusieurs erreurs peuvent se produire sur une instruction
- la forme minimale intercepte toutes ces erreurs sans les distinguer, ce qui peut être problématique parfois.

Exécuter le bloc except pour un type d'exception précis

Entre le mot-clé except et les deux points, vous pouvez préciser le type de l'exception que vous souhaitez traiter.

```
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur n'a pas été définie.")
```

Ce code ne traite que le cas où une exception NameError est levée.

On peut intercepter les autres types d'exceptions en créant d'autres blocs except à la suite :

```
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur n'a pas été définie.")
except TypeError:
    print("La variable numerateur ou denominateur a un type incompatible avec la division.")
except ZeroDivisionError:
    print("La variable de nominateur est égale à 0.")
```

Je vous conseille de *toujours* préciser un type d'exceptions après except.

Cela signifie que, si une erreur se produit, vous devez être capable de l'anticiper.

On peut capturer l'exception et **afficher son message** grâce au **mot-clé as**.

```
try:
   # Bloc de test
except type_de_l_exception as exception_retournee:
   print("Voici l'erreur :", exception_retournee)
```

Dans ce cas, une variable **exception_retournee** est créée par Python si une exception du type précisé est levée dans le bloc try.

Exemple :

```
annee = input("saisir une année : ")
try: # On essaye de convertir l'année en entier
   annee = int(anneeannee)
Except NameError as exception_retournee:
   print("Voici l'erreur :", exception_retournee)
```

```
saisir une année : aa
Voici l'erreur : name 'anneeannee' is not defined
```

J'ai bien affiché le **message** de l'exception NameError

Le mot-clé else

Dans un bloc try, else va permettre d'exécuter une action si aucune erreur ne survient dans le bloc.

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

Dans les faits, on utilise assez peu else.

La plupart des développeurs préfère mettre la ligne contenant le print directement dans le bloc try.

Le mot-clé finally

Finally permet d'exécuter du code après un bloc try, *quel que soit le résultat de l'exécution dudit bloc*.

```
try:
   # Test d'instruction(s)
except type_de_l_exception:
   # Traitement en cas d'erreur
finally:
   # Instruction(s) exécutée(s) qu'il y ait eu des erreurs ou non
```

Le bloc finally est exécuté dans tous les cas de figures.

Même si Python trouvait une instruction return dans votre bloc except par exemple, il exécutera le bloc finally.

Utilité : finally est destinée à définir des actions de nettoyage devant être exécutées dans certaines circonstances, notamment pour libérer des ressources externes (telles que des fichiers ou des connections réseau), que l'utilisation de ces ressources ait réussi ou non

Déclencher une exception

L'instruction raise permet au programmeur de **déclencher** une exception spécifique

```
raise TypeDeLException
```

Exemple

Nous allons déclencher une exception de type ValueError si l'utilisateur saisit une année négative ou nulle

```
annee = input("saisir une année : ") # L'utilisateur saisit l'année
try:
    annee = int(annee) # On tente de convertir l'année
    if annee<=0:
        raise ValueError
except ValueError:
    print("La valeur saisie est invalide (l'année est peut-être négative).")
```

Ici, on déclenche une exception avec **raise** quand la saisie est <=0, que l'on intercepte immédiatement avec **except**.

except **ValueError** va donc gérer à la fois une erreur de conversion en int()(avec le try…except classique) et la saisie <=0 (avec le raise)

Liste des exceptions

IndentationError ou TabError	Le fichier mélange des tabs et des espaces ou n'utilisent pas le même nombre de tabs ou d'espaces partout.	Activez l'affichage des tabs et espaces dans votre éditeur de texte, et assurez-vous d'utiliser 4 espaces partout comme valeur d'indentation.
ImportError	Python ne peut pas importer un module.	Vérifier que le nom du module ne comporte pas de fautes (Python est sensible à la casse).
AttributeError	Vous demandez à un objet de vous fournir un attribut qu'il ne possède pas.	Vérifiez que le nom de l'attribut et le nom de l'objet ne contiennent pas de faute. Vérifier que l'attribut a bien été créé avant son accès et pas supprimé entre temps
NameError	Vous tentez d'utiliser un nom (de variable, fonction, classe, etc) qui n'existe pas.	Vérifiez que ce nom ne contient pas de faute (Python est sensible à la casse). Assurez-vous que ce que vous nommez a bien été créé avant cette ligne.
IndexError	Vous tentez d'accéder à une partie d'une indexable (souvent une liste ou un tuple) qui n'existe pas.	Assurez vous que l'indexable contient assez d'éléments. Si le cas d'un indexable trop court est normal, vous pouvez vérifier la longueur de l'indexable avec len().
KeyError	Vous tentez d'accéder à une clé d'un dictionnaire qui n'existe pas.	Assurez vous que la clé existe.
TypeError	Vous tentez une opération incompatible avec ce type.	Vous pouvez vérifier le type d'un objet avec type(). Vérifiez également que vous n'utilisez pas un opérateur incompatible avec un type (par exemple, & ne fonctionne pas sur les strings) ou entre deux types incompatibles (par exemple, il n'est pas possible d'additionner une string avec un entier).
ValueError	Vous passez une valeur à une fonction qui n'a aucun sens dans ce contexte ou dont le sens est ambiguë.	Assurez-vous de ne pas passer une valeur aberrante et que le résultat attendu soit évident. Par exemple, si vous essayez de faire int('é'), la conversion de la lettre "é" en entier n'a pas de résultat évident.
OverflowError	Vous faites des calculs trop gros pour les types numériques des base	Utilisez le module decimal

ZeroDivisionError	Vous faites une division par zéro	Assurez-vous que sous aucune condition aucun dénominateur n'est égal à 0
IOError	Erreur d'entrée / sortie	Vérifiez que vous pouvez lire / écrire depuis et vers la ressource que vous utilisez. Parmi les problèmes récurrents : disque dur plein, système corrompu, absence de permissions, fichier inexistant, etc.
OSError	L'OS retourne une erreur	Les causes peuvent être très variées, mais concernent souvent le système de fichier. Vérifier les lignes où vous utiliser les modules os, shutils, popen, subprocess, multiprocessing, etc.
MemoryError	Vous utilisez trop de mémoire	Vérifiez vos remplissages de listes et dictionnaires.

Toute la liste

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