PROGRAMMATION PYTHON

Chapitre 7: Exceptions

Sommaire

Introduction

Tous les programmes réalisés dans les chapitres précédents fonctionnent uniquement dans le cas où l'utilisateur entre une valeur attendu dans le cas où il entre une valeur inattendu le programme va générer une erreur.

Supposons le programme suivant qui demande à l'utilisateur de saisir un nombre et calcule le carré du nombre:

```
>>> # Fonction qui demande un nombre à l'utilisateur et retourne le carré
>>> def saisieEtCalculCarre():
...     nbre = input('Entrer un nombre:')
...     nbre = int(nbre)
...     return nbre * nbre
...
>>> # Exécution avec saisie d'un nombre
>>> saisieEtCalculCarre()
Entrer un nombre:2
4
>>> # Exécution avec saisie de tout sauf un nombre
>>> saisieEtCalculCarre()
Entrer un nombre:a
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    saisieEtCalculCarre()
  File "<pyshell#15>", line 3, in saisieEtCalculCarre
    nbre = int(nbre)
ValueError: invalid literal for int() with base 10: 'a'
```

On voit que Python génère une erreur quand on ne fournit pas un nombre comme attendu. Tout code écrit après ne sera pas traité et le programme s'arrête là dès qu'une erreur est rencontrée.

Introduction

Les erreurs comme celle vue précédemment peuvent être interceptées très facilement avec Python. Quand Python rencontre une erreur lors de l'exécution du code, il lève une **exception** comme ci-dessous:

```
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    saisieEtCalculCarre()
  File "<pyshell#15>", line 3, in saisieEtCalculCarre
    nbre = int(nbre)
ValueError: invalid literal for int() with base 10: 'a'
```

Le code précédent a échoué parce que nous n'avons pas préparé notre code à faire face à ce genre d'erreur. Quand Python tombe sur une exception, il exécute le code qui doit être exécuté lors d'une exception sinon il affiche une erreur comme précédemment.

Chaque exception est identifiée par deux informations:

- le type de l'exception dans notre cas ***ValueError***
- le message que renvoie Python pour aider à comprendre l'erreur qui vient de se produire.

Il faut noter que lors de la levée d'une exception, Python arrête l'exécution du programme si aucun code n'a été fourni en cas d'erreur. Quand un code est fourni en cas d'exception on parle d'**interception de l'exception**. Par exemple dans l'exemple précédent on aurait pu redemander le nombre à l'utilisateur quand il saisit autre chose qu'un nombre.

Syntaxe simple *try…except*

Pour intercepter des exceptions en Python, on dispose des mots clés ***try/except*** dont la syntaxe est la suivante:

```
try:
    # Bloc de code à exécuter qui peut générer ou non une exception

except:
    # Bloc de code qui sera exécuté en cas d'erreur
```

Dans la syntaxe nous avons dans l'ordre:

- Le mot clé ***try*** (essayer en anglais) suivi de deux points.
- Le bloc d'instructions à essayer.
- Le mot clé ***except*** suivi de deux points et au même niveau que le ***try*.**
- Le bloc d'instructions qui sera exécuté si une erreur est trouvée dans le premier bloc.

La syntaxe précédente intercepte n'importe quelle exception liée au bloc ***try***. Une manière plus élégante de procéder est d'intercepter des erreurs spécifiques afin d'afficher un message d'erreur plus indicatif. Par exemple dans le cas de la saisie d'un nombre positif pour division on pourrait intercepter le cas où l'utilisateur saisie une chaine de caractères au lieu d'un nombre, une erreur lors d'un dénominateur égal à 0 car la division par 0 est impossible.

Exemple avec *try…except*

Créer un programme qui demande un nombre à l'utilisateur et affiche un message d'erreur au cas où l'utilisateur saisi autre chose qu'un nombre.

```
nombre = input("Entrer un nombre:")
try:
    print("Vous avez saisi le nombre:", nombre)
except:
    print("Vous devez saisir un nombre svp !")
```

```
Entrer un nombre:12
Vous avez saisi le nombre: 12

Entrer un nombre:a
Vous devez saisir un nombre svp !

Entrer un nombre:12
Vous avez saisi le nombre: 12
```

On peut utiliser l'instruction ***pass*** si on veut laisser le bloc vide sans instructions.

Syntaxe complète *try…except*

Pour intercepter une erreur spécifique on utilisera la syntaxe suivante:

```
try:
    # Code qui peut générer une erreur
except nom_exception1:
    # Code en cas d'execption 1
except nom_exception2:
    # Code en cas d'exception 2
```

Chaque exception levée par Python dispose d'un type par exemple on peut avoir les exceptions suivantes dans le cas de la division entre deux nombres:

- ***NameError*:** générée lorsqu'on essaye d'accéder à une variable qui n'a pas été définie. Par exemple quand on essaye de lire une variable qui n'existe pas.
- ***ValueError*:** générée lors d'une erreur de conversion quand on essaye de convertir par exemple une chaine de caractères comme « aa » en nombre.
- ***TypeError***: générée lorsque l'un des nombres ne peut diviser ou être divisée comme les chaines de caractères.
- ***ZeroDivisionError*:** générée lorsqu'on essaye de diviser un nombre par 0.

Syntaxe complète *try…except*

On peut capturer l'exception et afficher son message grâce au mot clé ***as*** avec la syntaxe suivante:

```
try:
    # Code qui peut générer une erreur
except nom_exception as exception_retournée:
    # Traitement et affichage de l'exception
```

Par exemple pour capturer l'exception lorsqu'on entre une valeur qui ne peut être convertie en nombre donne comme programme:

```
nombre = input("Entrer un nombre:")
try:
    nombre = int(nombre)
    print("Vous avez saisi le nombre:", nombre)
except ValueError as exception:
    print("Voici l'exception:", exception)
```

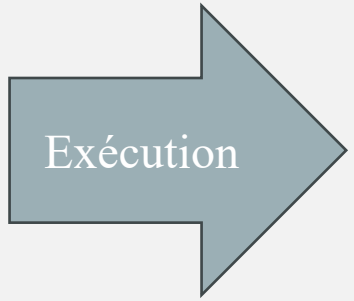
```
Entrer un nombre:a
Voici l'exception: invalid literal for int() with base 10: 'a'
```

Exemple complet *try…except*

Créer un programme qui demande deux nombres à l'utilisateur *nbre1* et *nbre2* et affiche le résultat de la division de *nbre1/nbre2*.

```
nbre1 = input("Entrer le nombre 1:")
nbre2 = input("Entrer le nombre 2:")

try:
    nbre1 = int(nbre1)
    nbre2 = int(nbre2)
    resultat = nbre1 / nbre2
    print(f"{nbre1} / {nbre2} = {resultat}")
except ValueError:
    print("Vous devez saisir des nombres svp !")
except ZeroDivisionError:
    print("La division par zéro est impossible !")
```




```
Entrer le nombre 1:a
Entrer le nombre 2:2
Vous devez saisir des nombres svp !


Entrer le nombre 1:2
Entrer le nombre 2:a
Vous devez saisir des nombres svp !


Entrer le nombre 1:2
Entrer le nombre 2:2
2 / 2 = 1.0


Entrer le nombre 1:sdssd
Entrer le nombre 2:0
Vous devez saisir des nombres svp !


Entrer le nombre 1:2
Entrer le nombre 2:0
La division par zéro est impossible !
```

Instructions *else* et *finally*

Il existe deux instructions qui peuvent être associées aux instructions ***try/except*** qui sont ***else*** et ***finally*** avec pour rôle:

- ***else*** **:** qui permet d'exécuter un code si aucune erreur n'est trouvée par exemple au lieu de calculer et afficher le résultat de la division dans le bloc ***try*** on aurait pu le faire dans le bloc ***else*.**
- ***finally*:** qui permet d'exécuter un code indépendamment du résultat de l'exécution. Le code dans le bloc ***finally*** sera exécuté en cas d'erreur ou pas.

La syntaxe pour chacune des instructions est la suivante:

```
try:
    # Code pouvant léver une exception
except exception1:
    # Code à exécuter en cas d'exception 1
except exception2:
    # Code à exécuter en cas d'exception 2
else:
    # Code à exécuter en si aucune erreur dans le try
```

```
try:
    # Code pouvant léver une exception
except exception1:
    # Code à exécuter en cas d'exception 1
except exception2:
    # Code à exécuter en cas d'exception 2
finally:
    # Code à exécuter dans tous les cas de figures
```

Exemple de *try/except* avec *else*

Le programme précédent avec la division de *nbre1* par *nbre2* pourrait donner comme résultat avec *else*:

```
nbre1 = input("Entrer le nombre 1:")
nbre2 = input("Entrer le nombre 2:")

try:
    # Code pouvant générer une erreur
    nbre1 = int(nbre1)
    nbre2 = int(nbre2)
    resultat = nbre1 / nbre2
except ValueError:
    print("Vous devez saisir des nombres svp !")
except ZeroDivisionError:
    print("La division par zéro est impossible !")
else:
    # Afficher le résultat de la division si aucune erreur
    print(f"{nbre1} / {nbre2} = {resultat}")
```

Exécution

```
Entrer le nombre 1:2
Entrer le nombre 2:a
Vous devez saisir des nombres svp !


Entrer le nombre 1:2
Entrer le nombre 2:a
Vous devez saisir des nombres svp !


Entrer le nombre 1:2
Entrer le nombre 2:2
2 / 2 = 1.0


Entrer le nombre 1:2
Entrer le nombre 2:0
La division par zéro est impossible !
```

Le programme avec le ***else*** est identique à celui sans ***else***. Certains programmeurs préfèrent mettre tout le code dans le bloc ***try*** et d'autres préfèrent séparer les deux sujets en deux blocs:

- Dans le bloc ***try*** pour les instructions qui peuvent être source d'erreur.
- Dans le bloc ***else***, le code qui peut doit être exécuté si aucune erreur.

Libre à chacun d'utiliser le modèle de programmation qu'il préfère.

Exemple de *try/except* avec *finally*

Ecrivons le programme précédent mais celle fois-ci on va afficher le message « Ce programme a été réalisé pour tester les exceptions » à la fin de l'exécution.

```
nbre1 = input("Entrer le nombre 1:")
nbre2 = input("Entrer le nombre 2:")

try:
    # Code pouvant générer une erreur
    nbre1 = int(nbre1)
    nbre2 = int(nbre2)
    resultat = nbre1 / nbre2
    # Afficher le résultat de la division si aucune erreur
    print(f"{nbre1} / {nbre2} = {resultat}")
except ValueError:
    print("Vous devez saisir des nombres svp !")
except ZeroDivisionError:
    print("La division par zéro est impossible !")
finally:
    # Ce message va s'afficher en cas d'erreur ou non
    print("Ce programme a été réalisé pour tester les exceptions.")
```

Exécution

```
Entrer le nombre 1:a
Entrer le nombre 2:3
Vous devez saisir des nombres svp !
Ce programme a été réalisé pour tester les exceptions.


Entrer le nombre 1:3
Entrer le nombre 2:0
La division par zéro est impossible !
Ce programme a été réalisé pour tester les exceptions.


Entrer le nombre 1:6
Entrer le nombre 2:3
6 / 3 = 2.0
Ce programme a été réalisé pour tester les exceptions.
```

Comme on le remarque le code dans le ***finally*** est exécuté en cas d'exception ou non.
La principale différence entre le ***finally*** et le ***else*** est que le ***finally*** est exécuté en cas d'exception ou non alors que le ***else*** est exécuté dans le cas où on n'a pas d'exception.

Les assertions

En Python, il existe un mot clé ***assert*** qui permet de tester une condition et lève l'exception ***AssertionError*** si elle est fausse. La syntaxe est la suivante:

assert test_a_effectuer

```
nbre=5
assert nbre == 5
assert nbre == 8
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    assert nbre == 8
AssertionError
```

On peut utiliser ***assert*** dans les blocs ***try/except*** pour lever des exceptions dans le cas où une condition n'est pas vérifiée.

Par exemple dans un programme qui demande de saisir un nombre positif, on peut avoir deux types d'exception le ***ValueError*** si on saisit une chaine de caractères au lieu d'un nombre et le ***AssertionError*** si on saisit un nombre qui n'est pas positif.

Certains programmeurs utilisent une condition avec le *if* au lieu de *assert*. Tout dépend de votre choix.

Exemple avec les assertions

Ecrire un programme qui va demander à de saisir un nombre positif.

```
nbre = input("Entrer un nombre positif:")

try:
    # Convertion en nombre
    nbre = int(nbre)
    # Verification si le nombre est positif ou non
    assert nbre > 0
except ValueError:
    # Si l'on ne saisit pas un nombre
    print("Vous devez saisir un nombre svp !")
except AssertionError:
    # Si l'on ne saisit pas un nombre strictement positif
    print("Vous devez saisir un nombre strictement positif svp !")
else:
    # Affichage du message en cas de non erreur
    print("Merci vous avez saisi le nombre positif:", nbre)
```

Exécution

```
Entrer un nombre positif:a
Vous devez saisir un nombre svp !


Entrer un nombre positif:-5
Vous devez saisir un nombre strictement positif svp !


Entrer un nombre positif:5
Merci vous avez saisi le nombre positif: 5
```

```
# Programme équivalent mais avec if/else
nbre = input("Entrer un nombre positif:")
try:
    # Convertion en nombre
    nbre = int(nbre)
    # Verification si le nombre est positif ou non
    if nbre <= 0:
        # Si l'on ne saisit pas un nombre strictement positif
        print("Vous devez saisir un nombre strictement positif svp !")
except ValueError:
    # Si l'on ne saisit pas un nombre
    print("Vous devez saisir un nombre svp !")
else:
    # Affichage du message en cas de non erreur
    print("Merci vous avez saisi le nombre positif:", nbre)
```

```
# Programme équivalent mais avec if/else
nbre = input("Entrer un nombre positif:")
try:
    # Convertion en nombre
    nbre = int(nbre)
    # Verification si le nombre est positif ou non
    if nbre <= 0:
        # Si l'on ne saisit pas un nombre strictement positif
        print("Vous devez saisir un nombre strictement positif svp !")
    else:
        # Affichage du message en cas de non erreur
        print("Merci vous avez saisi le nombre positif:", nbre)
except ValueError:
    # Si l'on ne saisit pas un nombre
    print("Vous devez saisir un nombre svp !")
```

Générer une exception

Jusqu'à présent on a vu comment intercepter les exceptions levées par Python, dans le cas où on écrit un module qui sera utilisé par d'autres développeurs on peut générer des exceptions spécifiques en cas d'erreur.

Python fournit un mot clé ***raise*** qui permet de lever des exceptions spécifiques avec la syntaxe suivante:

raise TypeDeLException("message à afficher")

Pour lever une exception *NombreNonPositif* dans le programme précédent on aurait comme résultat:

```
nbre = input("Entrer un nombre positif:")
try:
    # Convertion en nombre
    nbre = int(nbre)
    # Verification si le nombre est positif ou non
    if nbre <= 0:
        # Si l'on ne saisit pas un nombre strictement positif on lève l'exception
        raise NombreNonPositif("Nombre strictement positif attendu !")
except ValueError:
    # Si l'on ne saisit pas un nombre
    print("Vous devez saisir un nombre svp !")
else:
    # Affichage du message en cas de non erreur
    print("Merci vous avez saisi le nombre positif:", nbre)
```

L'exécution du programme précèdent lèvera une exception dans le cas où on saisit un nombre négatif ou nulle.

Les autres développeurs qui utilisent notre module devrons traiter l'exception NombreNonPositif dans leur except.

Exercices

Mini Projet 1

Créer un module ***multiplication*** qui contiendra les fonctions suivantes:

- *addition(nbre1, nbre2):* qui retourne le résultat de *nbre1 + nbre2*
- *soustraction(nbre1, nbre2):* qui retourne le résultat de *nbre1 - nbre2*
- *multiplication(nbre1, nbre2):* qui retourne le résultat de *nbre1 * nbre2*
- *division(nbre1, nbre2):* qui retourne le résultat de *nbre1 / nbre2*

Créer un deuxième module ***helper*** qui contiendra les fonctions suivantes:

- *afficheMenu():* qui affiche le menu des actions (addition, soustraction, multiplication, division)
- *saisirAction(minAction, maxAction):* qui demande le numéro de l'action à réaliser numéro compris entre *minAction* et *maxAction* par exemple dans notre cas çà sera entre 1 et 4
- *saisirNombre(typeNbre):* qui va demander à l'utilisateur de saisir un nombre *typeNbre* permet de dire si c'est le nombre 1 ou le nombre 2 qui doit être renseigné.

Créer un fichier main.py qui va utiliser l'ensemble des fonctions des modules précédents pour faire réaliser une calculatrice simple.

Cette fois-ci il faut gérer les cas d'erreurs possibles avec les blocs ***try/catch*** et si besoin ***finally/else/assert.***

Exercices

Mini Projet 2

Coder un jeu de nombre magique où l'utilisateur doit deviner un nombre aléatoire généré en 10 tentatives maximum.
Le jeu doit être constitué de trois modes, le mode facile (nombre magique entre 0 et 100), le mode moyen (nombre magique entre 0 et 1000) et le mode difficile (nombre magique entre 0 et 10000).
Après choix du mode faire deviner le nombre magique à l'utilisateur quand l'utilisateur entre un nombre inférieur au nombre magique afficher « Le nombre magique est plus grand que nombreSaisi ».
Quand l'utilisateur entre un nombre supérieur au nombre magique afficher « Le nombre magique est plus petit que nombreSaisi ».
Sinon afficher « Bingo vous avez trouvé le nombre magique en x essais ».
En cas d'échec afficher un message d'échec et demander si l'utilisateur veut recommencer si oui réafficher le menu.
Cette fois-ci il faut gérer les cas d'erreurs possibles avec les blocs ***try/catch*** et si besoin ***finally/else/assert.***

Exercices

Mini Projet 3

Le rôle de ce programme est d'importer la fonction ***drawRosace*** du module ***rosace*** ensuite votre rôle est de comprendre comment utiliser cette fonction. Ensuite demander à l'utilisateur les paramètres d'entrées et appeler la fonction avec la fonction avec les paramètres fournis par l'utilisateur.
Votre travail ici et de savoir utiliser un module externe et comment demander des valeurs correctes à l'utilisateur.
Il faut gérer les cas d'erreurs possibles lors de la saisie des paramètres avec les blocs ***try/catch*** et si besoin ***finally/else/assert.***

Mini Projet 4

L'objectif de ce mini projet est de créer une application de quiz sur les tables de multiplication.
Le but est de tirer au hasard un nombre qui sera mis dans une variable par exemple nbre1 et un autre nombre dans une variable nbre2 ensuite afficher la question suivante à l'écran: « Combien vaut le produit de nbre1 x nbre2 ? » en remplaçant nbre1 et nbre2 par leur valeur.
Si l'utilisateur(trice) saisit une réponse correcte afficher « Bravo ! » sinon afficher « Perdu ! La bonne réponse était x » x étant le résultat de la multiplication.

Exercices

Mini Projet 5

Refaire le mini projet 4 mais cette fois-ci l'on doit faire le choix entre trois modes:

- Mode facile (table de 0 à 10 composée de 10 questions uniquement les nombres positifs)
- Mode moyen (table de 0 à 20 composée 20 questions uniquement les nombres positifs)
- Mode difficile (table de 0 à 30 composée de 30 questions et multiplication de nombres positifs et négatifs)

L'utilisateur devra faire un choix du mode au départ ensuite on affiche les questions l'une à la suite de l'autre.

A la fin on devra afficher le score de l'utilisateur.

FIN CHAPITRE 7