TP N°10: Les exceptions

Les 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*. Lorsqu'une *erreur d'exécution* survient, Python lève une exception : il stoppe le programme et affiche le message d'erreur (la pile d'appels et le type d'*exception*). Exemple :

```
>>> 10 * (1/0)
ZeroDivisionError: integer division or modulo by zero
```

Les exceptions peuvent être de différents types. Les exceptions les plus courantes sont :

- Accéder à une clé non-existante d'un dictionnaire déclenche une exception `KeyError`.
- Un problème de conversion déclenche une exception `ValueError`.
- Appeler une méthode non-existante déclenche une exception `AttributeError`.
- Référencer une variable non-existante déclenche une exception `NameError`.
- Mélanger les types de données sans conversion déclenche une exception `TypeError`.

La liste complète des exceptions : https://docs.python.org/3/library/exceptions.htmlexception-hierarchy

Déclencher une exception

L'instruction `raise` permet au programmeur de déclencher une exception spécifique. L'instruction `raise` prend deux arguments : le type de l'exception et une chaîne de caractère qui décrit l'exception. Exemple :

```
age = input('Votre age : ')
if age < 0 :
    raise ValueError, "age invalide" # lever une exception de type ValueError
```

L'instruction `raise` est bien pratique lorsque vous souhaitez stopper l'exécution d'un programme si une variable ne contient pas la bonne valeur ou si une condition non vérifiée.

Gestion des exceptions

Il est possible d'écrire des programmes qui prennent en charge certaines exceptions. Gérer une exception permet d'intercepter une erreur pour éviter un arrêt du programme.

La bloc try-except

L'exemple suivant demande une saisie à l'utilisateur jusqu'à ce qu'un entier valide ait été entré.

```
while True:
    try:
        x = int(input("Saisir un entier: "))
        break # arrêt de la boucle while
    except ValueError:
        print("Erreur : Conversion en entier impossible !")
```

Remarques :

- 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. Exemple :

```
L = [1,2,3]
try:
    i = int(input(" i = "))
    x,y = L[i],L[i+1]
    print(x / y)
except ValueError:
```

```
        print ("Indice saisi non numérique")
    except IndexError:
        print ("Indice saisi dépasse la taille de la liste")
    except ZeroDivisionError:
        print ("Division par zéro!")
```

- Une même clause `except` peut citer plusieurs exceptions sous la forme d'un tuple entre parenthèses. Exemple :

```
except ValueError, IndexError, ZeroDivisionError :
    print ("Erreur")
```

- La dernière clause `except` peut omettre le(s) nom(s) d'exception(s), pour servir de joker. Elle peut aussi être utilisée pour afficher un message d'erreur avant de relever l'exception. Exemple :

```
except:
    print("Erreur inattendue")
    raise # relever l'exception : Autoriser python à gérer lui-même l'exception
```

Les clauses optionnelles pour l'instruction try

L'instruction `try - except` a également deux clauses optionnelles :

1. La clause `else` qui, lorsqu'elle est présente, doit suivre toutes les clauses `except`. Elle est utile pour le code qui doit être exécuté lorsqu'aucune exception n'a été levée par la clause `try`.
2. La clause `finally` est toujours exécutée avant de quitter l'instruction `try`, qu'une exception ait été déclenchée ou non.

Exemple:

```
try:
    result = x / y
except ZeroDivisionError:
    print ("Division par zéro!")
else:  # exécutée si aucune exception n'a été levée
    print ("Le résultat est = ", result)
finally:  # exécutée toujours, qu'une exception ait été déclenchée ou non
    print ("Exécution de la clause finally ")
```

Exercice 1

1. Créer une fonction Python `saisie()` qui saisie et retourne un entier strictement positif. Ajouter un bloc `try-except` qui gère une exception de type `ValueError`, ce qui pourrait se produire si l'utilisateur saisie une valeur non numérique. Imprimer un message d'erreur si cela se produit.
2. Créer une fonction Python `creer_liste(n)` qui crée et retourne une liste contenant les n premiers entiers premiers et demande de l'utilisateur un indice i pour afficher le i-ème nombre premier. Ajouter un bloc try-except qui gère une exception de type `IndexError`, ce qui pourrait se produire si l'index fourni dépasse la longueur de la liste. Imprimer un message d'erreur si cela se produit.
3. Écrire une fonction Python `ajouter_elt(d,L,x)` qui ajoute un élément x à une liste L dans un dictionnaire d des listes. Utiliser un bloc `try-except` qui gère une éventuelle exception de type `KeyError` possible si la liste avec le nom fourni n'existe pas encore dans le dictionnaire. Votre fonction doit inclure les clauses `else` et `finally` dans le bloc `try-except`.