Cours Introduction à Python
Gestionnaire d'erreurs et d'exceptions

Il y a au moins deux types d'erreurs à distinguer : les *erreurs de syntaxe* et les *exceptions*.

1. Les erreurs de syntaxe :

Les erreurs de syntaxe, qui sont des erreurs d'analyse du code. Le nom de fichier et le numéro de ligne sont affichés pour vous permettre de localiser facilement l'erreur lorsque le code provient d'un script.
Message d'erreur :

```
SyntaxError: invalid syntax
```

2. 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*.
Exemples des messages d'erreurs :

```
>>> 10 * (1/0)
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
NameError: name 'spam' is not defined
>>> '2' + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

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**.

Vous trouverez dans la section ***Built-in Exceptions*** de la documentation officielle de python la liste des exceptions natives et leur signification.

2.1. Gestion des exceptions :

Il est possible d'écrire des programmes qui prennent en charge certaines exceptions.
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
        except ValueError:
            print("Erreur : Conversion en entier impossible !")
```

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

```
except (RuntimeError, TypeError, NameError):
...     pass
```

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.

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

2.2.Déclencher une exception :

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

```
try:
...raise NameError
except NameError:
...print 'Une exception est déclenchée!'
```

2.3.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 zero!")
else:
    print ("résultat est", result)
finally:
    print ("Exécution de la clause finally ")
```

Essayer ce code avec les valeurs suivantes : x,y=1,0 | x,y=1,2 | x,y="2","4"

Exercices :

1. Créer un script qui permet de créer une liste L contenant les 100 premiers entiers premiers et demande de l'utilisateur un indice n pour afficher le $n^{ème}$ 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.
2. Ecrire un script python qui ajoute un élément à une liste dans un dictionnaire des listes. Utiliser un 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 script doit inclure les clauses *else* et *finally* dans votre bloc try-except.