R308 : Consolidation de la programmation
Chapitre 3 : Les exceptions et la gestion des erreurs

1 Présentation :

- 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).

2 Lien :

https://www.youtube.com/watch?v=sTZeuRdptc0

3 Exemples d'exceptions à partir d'un programme :

```
"""division de deux nombres
"""
# declaration des variables
a: float = None
b: float = None
q: float = None
# saisie
a = float(input("a : "))
b = float(input("b : "))
# traitement
q = a / b
# affichage
print(f"{a} / {b} = {q}")
print("fin du programme")
```

1. Exécution sans erreur :

```
a : 1
b : 2
1 / 2 = 0.5
fin du programme
```

2. Division par 0 :

```
a : 1
b : 0
Traceback (most recent call last):
  File "main_exceptions.py", line 11, in <module>
    q = a / b
ZeroDivisionError: division by zero
```

3. Saisie incorrecte :

```
a : 1
b : azerty
Traceback (most recent call last):
  File "ch3_exceptions1_cm.py", line 9, in <module>
    b = int(input("b : "))
ValueError: invalid literal for int() with base 10: 'azerty'
```

3.2 Traitement de l'erreur en utilisant la structure try except :

- Le bloc try englobe le code susceptible de provoquer des erreur.
- Si le traitement est interrompu par une exception, alors l'interpréteur recherche un bloc `except` correspondant au type de l'exception.

```
try:
    # saisie
    a = float(input("a : "))
    b = float(input("b : "))
    # traitement
    q = a / b
    # affichage
    print(f"{a} / {b} = {q}")
except ZeroDivisionError:
    print("la division est impossible ...")
print("suite du programme")
```

1. Division par 0 :

```
a : 1
b : 0
la division est impossible ... (1)
fin du programme (2)
```

(1) : l'erreur a été détectée et l'affichage du bloc except a été effectué.
Toutefois, l'erreur **ValueError** n'a pas été traitée.
(2) : le programme s'est terminé normalement.

3.3 Version plus aboutie :

- Il est possible de mettre plusieurs bloc except

```
try:
    # saisie
    a = float(input("a : "))
    b = float(input("b : "))
    # traitement
    q = a / b
    # affichage
    print(f"{a} / {b} = {q}")
except ZeroDivisionError:
    print("la division est impossible ...")
except ValueError:
    print("attention, vous avez fait une erreur de saisie")
print("fin du programme")
```

- une même clause `except` peut citer plusieurs exceptions sous la forme d'un tuple

```
except ( ZeroDivisionError, ValueError):
    print("une erreur s'est produite")
```

3.4 Récupération du message d'exception (aide au débogage) :

- Il est possible d'afficher l'information correspondant à l'exception sous la forme d'une chaine de caractères.
- Par héritage, toutes les exceptions sont de type **Exception**

```
try:
    # saisie
    a = float(input("a : "))
    b = float(input("b : "))
    # traitement
    q = a / b
    # affichage
    print(f"{a} / {b} = {q}")
except Exception as ex:
    print(ex)
print("fin du programme")
```

```
a : 1
b : A
invalid literal for int() with base 10: 'A'
fin du programme
```

3.5 Clause else :

- Il est possible d'ajouter une clause **else** après les blocs **try except**.
- Le bloc **else** est exécutée uniquement si le bloc **try** se termine normalement.
- Elle doit se placer après toutes les clauses **except**.

L'utilisation de la clause **else** permet une très bonne lisibilité du programme :

```
# saisie avec gestion des erreurs
try:
    a = float(input("a : "))
    b = float(input("b : "))
    q = a / b # l'instruction peut generer une erreur
except Exception as ex:
    print(ex)
else: # affichage si pas d erreur
    print(f"{a} / {b} = {q}")
print("fin du programme")
```

3.6 Post-traitement

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

```
try:
      ... bloc susceptible de produire des erreurs
except Exception as ex:
      ... traitement si erreurs
else: # affichage si pas d erreur
      ... traitement si pas d'erreurs
finally :
      ... bloc exécuté dans tous les cas
```

Modifier le programme de telle sorte que :

- uniquement la saisie soit faite dans la clause try.
- un message apparaît en cas d'erreur pour indiquer la nature de l'erreur.
- un message de félicitation apparaît si las saisies sont correctes.

3.7 Lever ses propres exceptions :

- Le mot clé **raise** (soulever) permet de lever une exception qui sera traiter normalement.
- Il sera toutefois nécessaire d'utiliser les exceptons existantes.

Exemple de division entre deux nombres compris entre 0 et 100 :

```
try:
      # saisie
      a = float(input("a : "))
      b = float(input("b : "))
      # controle de la saisie
      if a <= 0 or 100 < a:
            raise ValueError # levée de l'exception
      if b <= 0 or 100 < b:
            raise ValueError # levée de l'exception
      # gestion des erreurs
except ValueError:
      print("attention, vous avez fait une erreur de saisie")
else :
      # traitement et affichage
      q = a / b
      # affichage
      print(f"{a} / {b} = {q}")
print("fin du programme")
```

3.8 Gérer ses propres exceptions :

- Il est possible de créer sa propre classe qui hérite de la classe le base Exception.
- Les principes d'héritage et de gestion des exceptions doivent être respectés

```
class MonException(Exception): # (1)
      def __init__(self):
            Exception.__init__(self)
      def __str__(self) -> str: # (2)
      return "vous ne respectez pas les contraintes de saisie"

if __name__ == "__main__":
      """division de deux nombres compris entre 0 et 100
      """
      # declaration des variables
      a: float = None
      b: float = None
      q: float = None
      try:
            # saisie
            a = float(input("a : "))
            b = float(input("b : "))
            # controle de la saisie
            if a < 0 or 100 < a:
                  raise MonException #(3)
            if b <= 0 or 100 < b:
                  raise MonException #(3)
      except ValueError:
            print("attention, vous avez fait une erreur de saisie")
      except MonException as mex: #(4)
            print(mex)
      else :
            # traitement et affichage
            q = a / b
            # affichage
            print(f"{a} / {b} = {q}")
      print("fin du programme")
```

(1) : MonException hérite de la classe Exception
(2) : surdéfinition de la méthode __str_()
(3] : levée de l'exception en cas d'erreur
(4) : gestion de l'exception
Une autre possibilité :

```
except (MonException, ValueError) as ex:
      print(ex)
```

3.9 Passage de paramètre à une exception :

Il est également possible de passer des paramètres au constructeur d'une exception :

```
class MonException(Exception): # (1)
    def __init__(self, param : objet):
        Exception.__init__(self)
        ...
    def __str__(self) -> str: # (2)
        return "vous ne respectez pas les contraintes de saisie"
```

Lors de la levée de l'exception, il faudra alors préciser la valeur du paramètre :

```
raise MonException(le_parametre)
```

3.10 Les exceptions à connaître :

ValueError	Signale que la valeur n'est pas correcte.
TypeError	Signale que le type de la donnée n'est pas correct pour l'instruction à exécuter.
IndexError	Signale que l'on veut accéder à un élément d'une liste, d'un n-uplet ou d'une chaîne de caractères avec un index invalide.
KeyError	Signale que la clé n'existe pas dans un dictionnaire.