FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Chapitre 3 : Gestion des exceptions

Licence Fondamentale : Tronc commun Informatique Appliquée

Pr: Y.Ouassit

Gestion des exceptions

1. Introduction aux exceptions
2. Gestion des exceptions
    a) Bloc try/except
    b) Gestion de plusieurs exceptions dans un seul bloc except
    c) Bloc finally
3. Lever une exception (ou lancer une exception)

Introduction

Dans la programmation, les erreurs sont fréquentes :

- Lorsqu'on développe un programme, il est normal de rencontrer des erreurs.
- Aucun programmeur n'y échappe, même les plus expérimentés.
- L'objectif n'est pas d'éviter toutes les erreurs, mais de savoir les identifier et les gérer correctement.

Trois grandes catégories d'erreurs :

1. Erreurs de syntaxe
2. Erreurs d'exécution
3. Erreurs logiques

Introduction

1 - Erreurs de syntaxe :

Les erreurs de syntaxe sont des erreurs dues au format incorrect d'une instruction Python.
Elles se produisent au moment où l'instruction est traduite (compilée) en langage machine, avant même que le programme ne soit exécuté.

```
>>> (3+4]
SyntaxError: invalid syntax
>>> if x == 5
SyntaxError: invalid syntax
>>> print 'hello'
SyntaxError: invalid syntax
>>> lst = [4;5;6]
SyntaxError: invalid syntax
>>> for i in range(10):
print(i)
SyntaxError: expected an indented block
```

Introduction

2 - Erreurs d'exécution:

Ce sont des erreurs qui se produisent pendant l'exécution du programme, c'est-à-dire après la phase de traduction/compilation.

Lorsqu'une erreur se produit, un objet « erreur » (exception) est créé.

- Cet objet possède un type lié à la nature de l'erreur.
- Il contient des informations sur l'erreur (message, origine, etc.).
- Par défaut, Python affiche ces informations puis interrompt l'exécution de l'instruction (et souvent du programme).

```
>>> 3/0
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    3/0
ZeroDivisionError: division by zero
```

```
>>> int('4.5')
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    int('4.5')
ValueError: invalid literal for int() with
base 10: '4.5'
```

Introduction

3 - Erreurs logiques:

Ce sont des erreurs dû à une mauvaise conception.

- Le programme s'exécute sans erreur
- Mais le résultat est faux

Exemple :

```
prix = 100
remise = 10
total = prix - remise / 100 #ça doit être total = prix*(1-remise/100)
print(total)
```

Introduction

L’importance de bien gérer les erreurs en programmation

- Les erreurs sont inévitables dans tout programme.
- Une mauvaise gestion peut provoquer l’arrêt du programme.
- La gestion des erreurs permet d’éviter les crashs et d’assurer la continuité de l’exécution.
- Elle améliore la fiabilité et la qualité du logiciel.
- Elle aide l’utilisateur à comprendre le problème grâce à des messages clairs.

Objectif :

Écrire des programmes robustes capables de détecter, traiter et contrôler les erreurs.

Introduction

Définition d'une exception :

Une exception est un événement qui se produit pendant l’exécution d’un programme et qui interrompt le déroulement normal des instructions.

En Python, une exception apparaît lorsqu'une erreur est détectée pendant l'exécution du programme (par exemple : division par zéro, conversion invalide, fichier introuvable, etc.).

Python génère alors un **objet exception** contenant des informations sur l’erreur.

Introduction

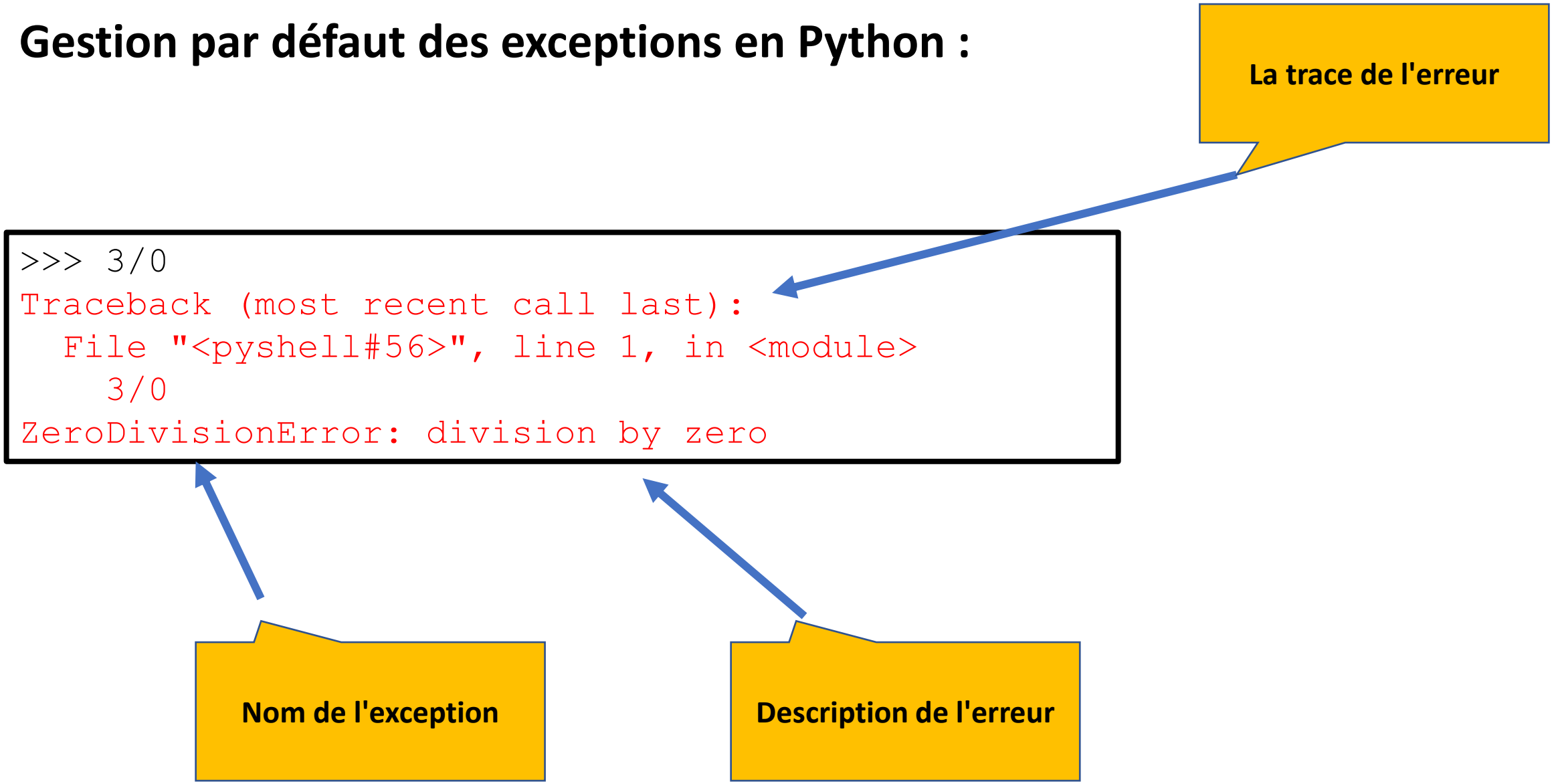
Gestion par défaut des exceptions en Python :

La gestion offre une gestion par défaut des exceptions, et le comportement de Python Lorsqu'une exception se produit est :

1. Le programme s'arrête immédiatement
2. Python affiche un message appelé traceback
3. Le type d'erreur est indiqué (ex : TypeError, IndexError…)

Introduction

Gestion par défaut des exceptions en Python :



Introduction

Gestion par défaut des exceptions en Python :

```
Traceback (most recent call last):
  File "<tmp 1>", line 11, in <module>
    test(4, 3)
  File "<tmp 1>", line 8, in test
    print(factoriel(a))
  File "<tmp 1>", line 4, in factoriel
    f = a * i
NameError: name 'a' is not defined
```

TRACE DE L'EXCEPTION

Gestion des exceptions

Un programme simple : Version 0

```
x = int(input("Entrez un nombre : "))
print(10 / x)
print("Fin du programme")
```

- Que se passe-t-il si on tape 0 ?
- Si on tape du texte ?

<tmp 1> (unsaved) - Interactive Editor for Python

File Edit View Settings Shell Run Tools Help

<tmp 1>

```
1 x = int(input("Entrez un nombre : "))
2 print(10 / x)
3 print("Fin du programme")
4
```

Shells

Python

```
>>> (executing file "<tmp 1>")
Entrez un nombre :
```

Gestion des exceptions

Afin d'éviter que le programme s'arrête brutalement lorsqu'une erreur se produit, on peut placer le bloc de code susceptible de générer une erreur dans une structure de gestion d'exceptions :

```
try:
    # code susceptible de provoquer une erreur
except:
    # code exécuté si une erreur se produit
```

Try : ensemble des instructions susceptibles de provoquer une exception.
Handler : ensemble des instructions dans le bloc except qui sont exécutées lorsqu'une exception se produit.

Gestion des exceptions

Un programme simple : Version 1

```
try :

    x = int(input("Entrez un nombre : "))
    print(10 / x)

except :
    print("Une erreur s'est produite!")

print("Fin du programme")
```

- Si un erreur est survenue, le programme ne plante mais un message d'erreur est affiché.
- **Mais : Le message n'indique pas quelle instruction a généré l'erreur.**

<tmp 1> (unsaved) - Interactive Editor for Python

File Edit View Settings Shell Run Tools Help

<tmp 1>

```
1 try :
2     x = int(input("Entrez un nombre : "))
3     print(10 / x)
4 except :
5     print("Une erreur s'est produite!")
6 print("Fin du programme")
```

Shells

Python

```
>>>
```

Gestion des exceptions

Un programme simple : Version 2

```
try :

    x = int(input("Entrez un nombre : "))
    print(10 / x)

except  Exception as e   :
    print(f"Erreur : {e}")

print("Fin du programme")
```

- Si un erreur est survenue, le programme affiche le message d'erreur sans planter.
- **MAIS : Si nous voulons personnaliser les messages affiché ???**

<tmp 1> (unsaved) - Interactive Editor for Python

File Edit View Settings Shell Run Tools Help

<tmp 1>

```
1 try :
2     x = int(input("Entrez un nombre : "))
3     print(10 / x)
4 except Exception as e:
5     print(f"Erreur : {e}")
6 print("Fin du programme")
```

Shells

Python

```
>>> (executing file "<tmp 1>")
Entrez un nombre :
```

Gestion des exceptions multiples

Types des exceptions built-in :

Exception	Description
`NameError`	Lancé lors de la tentative d'évaluation d'un identifiant non attribué (nom)
`TypeError`	Une exception est levée lorsqu'une opération ou une fonction est appliquée à un objet de type incorrect.
`ValueError`	Une exception est levée lorsqu'une opération ou une fonction reçoit un argument du type attendu mais d'une valeur incorrecte.
`IndexError`	Une exception est levée lorsqu'un index de séquence est en dehors de la plage des index valides.
`FileNotFoundError`	Une exception est levée on essaie d'ouvrir un fichier inexistant
`ZeroDivisionError`	Erreur lors de la tentative de division par 0
`KeyError`	Déclenché lorsqu'on veut accéder à une clé non existante d'un dictionnaire.
`KeyboardInterrupt`	Lancé lorsque l'utilisateur appuie sur Ctrl-C, la touche d'interruption

Gestion des exceptions multiples

Un programme simple : Version 3

```
try :

    x = int(input("Entrez un nombre : "))
    print(10 / x)

except ValueError as e :
    print("Erreur : Valeur invalide!")
except ZeroDivisionError as e :
    print("Erreur : Division par zéro!")
```

- Un message d'erreur personnalisé par type d'erreur.
- **Comment affiché un message particulier si le bloc s'est exécuté sans aucune exception ?**

Gestion des exceptions multiples

Le bloc else :

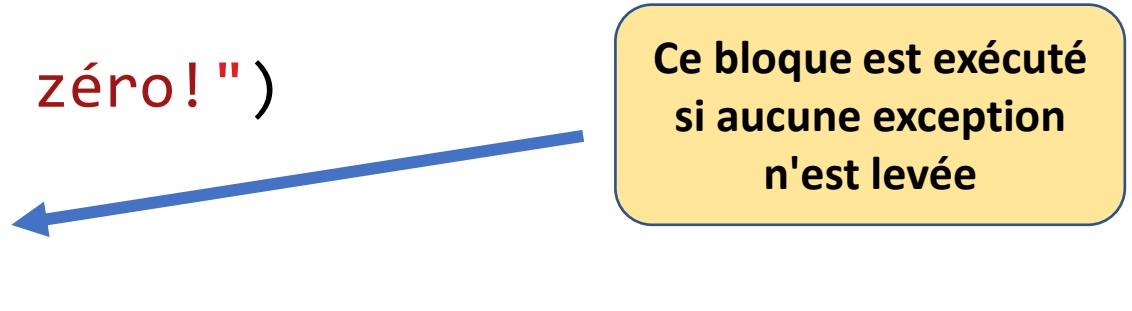
Le bloc else s'exécute si aucune exception n'est levée dans le bloc try. Il est utile pour exécuter du code qui doit seulement s'exécuter si tout s'est bien passé dans le try.

```
try:
    # bloc à exécuter
except Exception:
    # bloc pour gérer les exceptions
else:
    # bloc à exécuter si aucune exception n'est produite
```

Gestion des exceptions multiples

Un programme simple : Version 4

```
try :
    x = int(input("Entrez un nombre : "))
    print(10 / x)
except ValueError as e :
    print("Erreur : Valeur invalide!")
except ZeroDivisionError as e :
    print("Erreur : Division par zéro!")
else :
    print("Pas d'exception!")
```



- Un message d'erreur personnalisé par type d'erreur.
- **Comment affiché un message dans tous les cas de fin d'exécution ?**

Gestion des exceptions multiples

Le bloc finally :

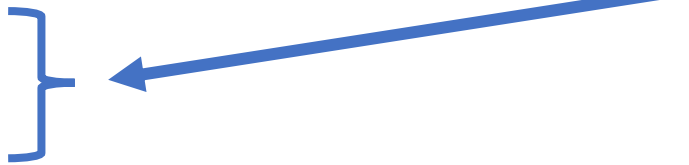
Le bloc finally est exécuté qu'une exception ait été levée ou non. Il est utile pour effectuer des opérations de nettoyage ou libérer des ressources, comme fermer un fichier ou une connexion.

```
try:
    # bloc à exécuter
except Exception:
    # bloc pour gérer les exceptions
else:
    # bloc à exécuter si aucune exception n'est produite
finally:
    # bloc à exécuter dans tous les cas
```

Gestion des exceptions multiples

Un programme simple : Version 4

```
try :
    x = int(input("Entrez un nombre : "))
    print(10 / x)
except ValueError as e :
    print("Erreur : Valeur invalide!")
except ZeroDivisionError as e :
    print("Erreur : Division par zéro!")
else :
    print("Pas d'exception!")
finally :
    print("Fin du programme!")
```

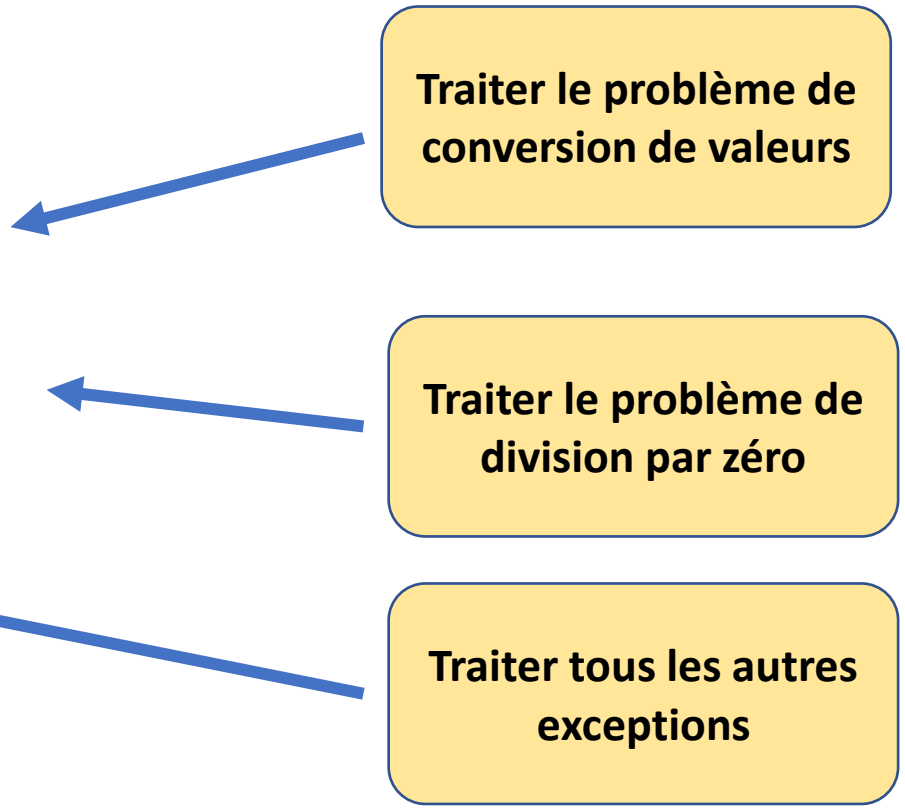


Ce bloque est toujours exécuté

Gestion des exceptions multiples

Un programme simple : Version 5

```
try :
    x = int(input("Entrez un nombre : "))
    print(10 / x)
except ValueError as e :
    print("Erreur : Valeur invalide!")
except ZeroDivisionError as e :
    print("Erreur : Division par zéro!")
except Exception as e :
    print(f"Erreur : {e}")
else :
    print("Pas d'exception!")
finally :
    print("Fin du programme!")
```



Exercices :

Exercice 1 : Complétez ce code pour gérer toutes les exceptions possibles :

```
liste = [10, 20, 30]

i = int(input("Entrez un indice : "))
print(liste[i])
```

Exercices :

Exercice 1 : Complétez ce code pour gérer toutes les exceptions possibles :

```
liste = [10, 20, 30]

try:
    i = int(input("Entrez un indice : "))
    print(liste[i])
except ValueError as e:
    print("Erreur : Valeur invalide!")
except IndexError as e:
    print("Erreur : indice hors de la liste!")
except Exception as e :
    print(f"Erreur : {e}")
```

Exercices :

Exercice 2 : Donner le type d'exception généré par chacune de ces instructions :

```
print(10 / 0)   # ZeroDivisionError

print('5' + 3)  # ValueError

int('abc')  # ValueError

liste = [1, 2, 3]
print(liste[5])   # IndexError

dictionnaire = {'a': 1, 'b': 2}
print(dictionnaire['c'])  # KeyError

f = open("c:/Users/test.txt" , "r") # FileNotFoundError
```

Exercices :

Exercice 3 : Quel est le résultat d'exécution du script suivant :

```
try:
    a = "5"
    b = 2
    print(a + b)
except TypeError:
    print("Erreur de type")
print("Fin")
```

```
try:
    liste = [1, 2, 3]
    print(liste[2])
except IndexError:
    print("Erreur d'indice")
else:
    print("Pas d'erreur")
```

Exercices :

Exercice 3 : Quel est le résultat d'exécution du script suivant :

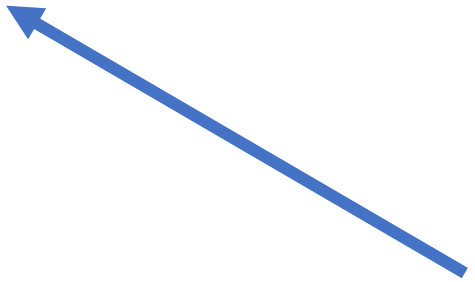
```
try:
    x = int("abc")
except ValueError:
    print("Erreur de conversion")
finally:
    print("Fin du programme")
```

```
try:
    print(valeur)
except NameError:
    print("Variable non définie")
else:
    print("OK")
finally:
    print("Terminé")
```

Capturer plusieurs exceptions

Il est possible de gérer plusieurs exceptions dans un même bloc **except** en les regroupant dans un tuple. Cela permet de traiter de la même manière plusieurs types d'exceptions.

```
try:
    num = int(input("Entrez un nombre : "))
    resultat = 10 / num
except (ZeroDivisionError, ValueError) as e:
    print(f"Erreur : {e}")
except Exception as e :
    print("Erreur inconnue!")
```



Même traitement pour ces deux exceptions

Lever une exception

Il est possible de lever explicitement une exception en utilisant le mot-clé **raise**. Cela permet de signaler qu'une condition anormale s'est produite et de transférer le contrôle au bloc except approprié.

Syntaxe :

```
raise ExceptionType("message")
```

Lever une exception

Exemple :

```
def verifier_age(age):
    if age < 0:
        raise ValueError("L'âge ne peut pas être négatif.")
    elif age < 18:
        print("Vous êtes mineur.")
    else:
        print("Vous êtes majeur.")

try:
    verifier_age(-5)
except ValueError as e:
    print(f"Erreur : {e}")
```

Les assertions

Les assertions permettent de vérifier qu'une condition est vraie pendant l'exécution du programme.

Si la condition est fausse, une AssertionError est levée. Elles sont principalement utilisées pendant le développement pour détecter des erreurs de logique.

Syntaxe :

```
assert condition, "message d'erreur"
```

Les assertions

Exemple :

```
def verifier_age(age):
    assert age >= 0, "L'âge ne peut pas être négatif."

    if age < 18:
        print("Vous êtes mineur.")
    else:
        print("Vous êtes majeur.")


try:
    verifier_age(-5)
except AssertionError as e:
    print(f"Erreur : {e}")
```

Les assertions

Exercice : Compléter la fonction ci-dessous pour vérifier la validité des paramètres.

```
def montant(prix, quantite):

    return prix * quantite
```

```
def montant(prix, quantite):

    assert isinstance(prix, (int, float)), "Le prix doit être un nombre"
    assert isinstance(quantite, int), "La quantité doit être un entier"
    assert prix >= 0, "Le prix doit être positif"
    assert quantite >= 0, "La quantité doit être positive"

    return prix * quantite
```