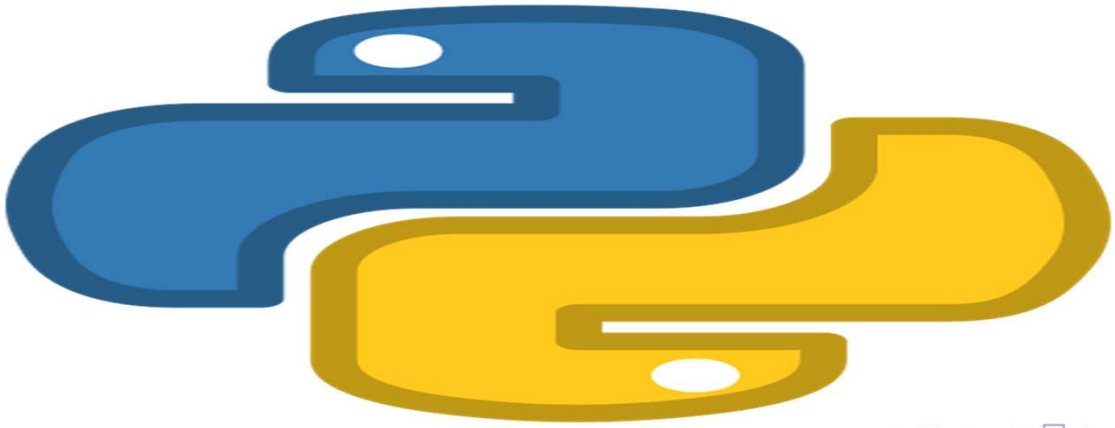
Python : exceptions

Mohammed OUANAN

m.ouanan@umi.ac.ma



Plan

Plan

Python

Une exception ?

- C'est une erreur qui se produit pendant l'exécution de notre programme
- Une exception dans un programme implique généralement son arrêt d'exécution

Considérons la fonction suivante

```
def division (a: int, b: int) -> int:
    return a / b
```

Considérons la fonction suivante

```
def division (a: int, b: int) -> int:
    return a / b
```

Appeler la fonction avec une valeur pour le deuxième paramètre égal à zéro déclenche une exception

```
print(division (2, 0))
print('fin')
# affiche ZeroDivisionError: division by zero
```

Considérons la fonction suivante

```
def division (a: int, b: int) -> int:
    return a / b
```

Appeler la fonction avec une valeur pour le deuxième paramètre égal à zéro déclenche une exception

```
print(division (2, 0))
print('fin')
# affiche ZeroDivisionError: division by zero
```

Remarque

Le message `fin` n'a pas été affiché.

Python

Comment faire pour poursuivre l'exécution ?

- Repérer les blocs pouvant générer une exception
- Capturer l'exception correspondante
- Afficher un message relatif à cette exception
- Continuer l'exécution

Python

Comment faire pour capturer une exception ?

- Utiliser un bloc `try: ... except: ...`
- Le `try` pour entourer une instruction susceptible de déclencher une exception
- Le `except` pour capturer l'exception et afficher un message qui lui correspond

Python

Utilisons le bloc `try except` pour capturer l'exception et continuer l'exécution

```
try:
    print(division (2, 0))
except:
    print('division par zéro')
print('fin')

# affiche division par zéro
# affiche fin
```

Python

On peut aussi préciser le type de l'exception

```
try:
    print(division (2, 0))
except ZeroDivisionError:
    print('division par zéro')
print('fin')

# affiche division par zéro
# affiche fin
```

Python

Et si on voulait récupérer et afficher le message par défaut de l'exception

```
try:
    print(division (2, 0))
except ZeroDivisionError as e:
    print(e)
print('fin')

# affiche division by zero
# affiche fin
```

Python

Pour avoir une description plus détaillée, on utilise `traceback`

```
import traceback

def division (a: int, b: int) -> int:
    return a / b
try:
    print(division (2, 0))
except ZeroDivisionError as e:
    traceback.print_exc()
print('fin')

# affiche
# Traceback (most recent call last):
#   File "C:/Users/x/cours-exception/main.py", line 7, in <module>
#     print(division (2, 0))
#   File "C:/Users/x/cours-exception/main.py", line 5, in division
#     return a / b
# ZeroDivisionError: division by zero
# fin
```

Python

Pour exécuter une suite d'instructions si aucune exception n'a été levée

```
try:
    print(division (4, 2))
except ZeroDivisionError as e:
    print(e)
else:
    print('calcul terminé avec succès')
print('fin')

# affiche calcul terminé avec succès
# affiche fin
```

Python

Pour exécuter une suite d'instructions qu'une exception soit levée ou non

```
try:
    print(division (4, 0))
except ZeroDivisionError as e:
    print(e)
finally:
    print('tout le temps exécuté')
print('fin')

# affiche tout le temps exécuté
# affiche fin
```

Python

Pour exécuter une suite d'instructions qu'une exception soit levée ou non

```
try:
    print(division (4, 0))
except ZeroDivisionError as e:
    print(e)
finally:
    print('tout le temps exécuté')
print('fin')

# affiche tout le temps exécuté
# affiche fin
```

Retestez avec les valeurs 4 et 2 pour les paramètres de la fonction `division` et vérifiez que le le message `tout le temps exécuté` est toujours affiché.

Python

Et si on appelle la fonction avec un type autre que `int`, une exception sera lancée (et non capturée, le message `fin` ne sera donc pas affiché)

```
try:
    print(division (2, 'a'))
except ZeroDivisionError:
    print('division par zéro')
print('fin')

# affiche TypeError: unsupported operand type(s) for
  /: 'int' and 'str'
```

Python

On peut utiliser plusieurs fois le bloc except

```
try:
    print(division (2, 0))

except ZeroDivisionError:
    print('division par zéro')

except TypeError:
    print('Les deux paramètres doivent être de type
       entier')

print('fin')

# affiche division par zéro
# affiche fin
```

Python

On peut aussi définir un bloc `except` général

```
try:
    print(division (4, 2))

except ZeroDivisionError:
    print('division par zéro')

except TypeError:
    print('Les deux paramètres doivent être de type entier')

except:
    print('erreur inconnue')

print('fin')

# affiche 2.0
# affiche fin
```

Python

On peut fusionner les blocs `except`

```
try:
    print(division (2, 'a'))

except (ZeroDivisionError, TypeError):
    print('Problème avec les paramètres')

except:
    print('erreur inconnue')

print('fin')

# affiche Problème avec les paramètres
# affiche fin
```

Python

Malgré le typage des paramètres de la fonction, les types numériques (`float` par exemple) peuvent passer

```
try:
    print(division (4.0, 2))

except (ZeroDivisionError, TypeError):
    print('Problème avec les paramètres')

except:
    print('erreur inconnue')

print('fin')

# affiche 2.0
# affiche fin
```

Python

On peut utiliser `raise` pour lancer une exception si le type ne correspond pas

```
def division (a: int, b: int) -> int:
    if type(a) == int and type(b) == int:
        return a / b
    else:
        raise Exception("Cette fonction n'accepte que les
           entiers")

try:
    print(division (4.0, 2))
except (ZeroDivisionError, TypeError):
    print('Problème avec les paramètres')
except Exception as e:
    print(e)
print('fin')

# affiche Cette fonction n'accepte que les entiers
# affiche fin
```

Python

Une exception peut aussi lancée en utilisant `assert`

```
def division (a: int, b: int) -> int:
    assert type(a) == int and type(b) == int, Exception("Cette
       fonction n'accepte que les entiers")
    return a / b

try:
    print(division (4.0, 2))
except (ZeroDivisionError, TypeError):
    print('Problème avec les paramètres')
except Exception as e:
    print(e)
print('fin')

# affiche Cette fonction n'accepte que les entiers
# affiche fin
```

Python

On a utilisé (ou vu) des exceptions prédéfinies

- `ZeroDivisionError`
- `ValueError`

Python

On a utilisé (ou vu) des exceptions prédéfinies

- `ZeroDivisionError`
- `ValueError`

On peut aussi définir nos exceptions personnalisées

Python

Considérons la classe `Adresse` suivante

```
class Adresse:

    def __init__(self,  rue: str = '', ville: str = '',
       code_postal: str = ''):
         self.__rue = rue
         self.__ville = ville
         self.__code_postal = code_postal

# +  getters + setters + __str__
```

Python

Supposons que

`code_postal` doit contenir exactement 5 chiffres

Python

Supposons que

`code_postal` doit contenir exactement 5 chiffres

Démarche à faire

- Créer notre propre exception (qui doit étendre la classe `Exception`)
- Dans le constructeur de `Adresse`, on lance une exception si `code_postal` ne contient pas 5 chiffres.

Python

Cre´ons l'exception `CodePostalError`

```
class CodePostalError(Exception):

    def __init__(self):
        super().__init__("Le code postal doit contenir 5 caract`eres")
```

Python

Modifions le constructeur de la classe `Adresse`

```
from code_postal_error import CodePostalError


class Adresse:

    def __init__(self,  rue: str = '', ville: str = '',
       code_postal: str = ''):
         self.__rue = rue
         self.__ville = ville
         if len(code_postal) != 5:
             raise CodePostalError()
         self.__code_postal = code_postal

# + getters + setters + __str__
```

Python

Testons tout cela dans le `main`

```
from adresse import Adresse

try:
    adresse = Adresse("paradis", "Marseille", "1301")
except Exception as e:
    print(e)
print("fin")
```

Python

Testons tout cela dans le `main`

```
from adresse import Adresse

try:
    adresse = Adresse("paradis", "Marseille", "1301")
except Exception as e:
    print(e)
print("fin")
```

Le message affiché est :

```
Le code postal doit contenir exactement 5 chiffres
fin
```

Python

Question

Comment faire si on veut afficher les valeurs qui ont déclenché l'exception dans le message ?

Python

Modifions la première exception `CodePostalError`

```
class CodePostalError(Exception):

    def __init__(self, value):
        super().__init__(f"Le code postal {value} doit contenir
            exactement 5 caractères")
```

Python

Modifions le constructeur de la classe `Adresse`

```
from code_postal_error import CodePostalError


class Adresse:

    def __init__(self,  rue: str = '', ville: str = '',
       code_postal: str = ''):
         self.__rue = rue
         self.__ville = ville
         if len(code_postal) != 5:
             raise CodePostalError(code_postal)
         self.__code_postal = code_postal

# getters + setters + __str__
```

Python

Testons tout cela dans le `main`

```
from adresse import Adresse

try:
    adresse = Adresse("paradis", "Marseille", "1301")
except Exception as e:
    print(e)
print("fin")
```

Python

Testons tout cela dans le `main`

```
from adresse import Adresse

try:
    adresse = Adresse("paradis", "Marseille", "1301")
except Exception as e:
    print(e)
print("fin")
```

Le message affiché est :

```
Le code postal 1301 doit contenir exactement 5 chiffres
fin
```

Python

Exercice

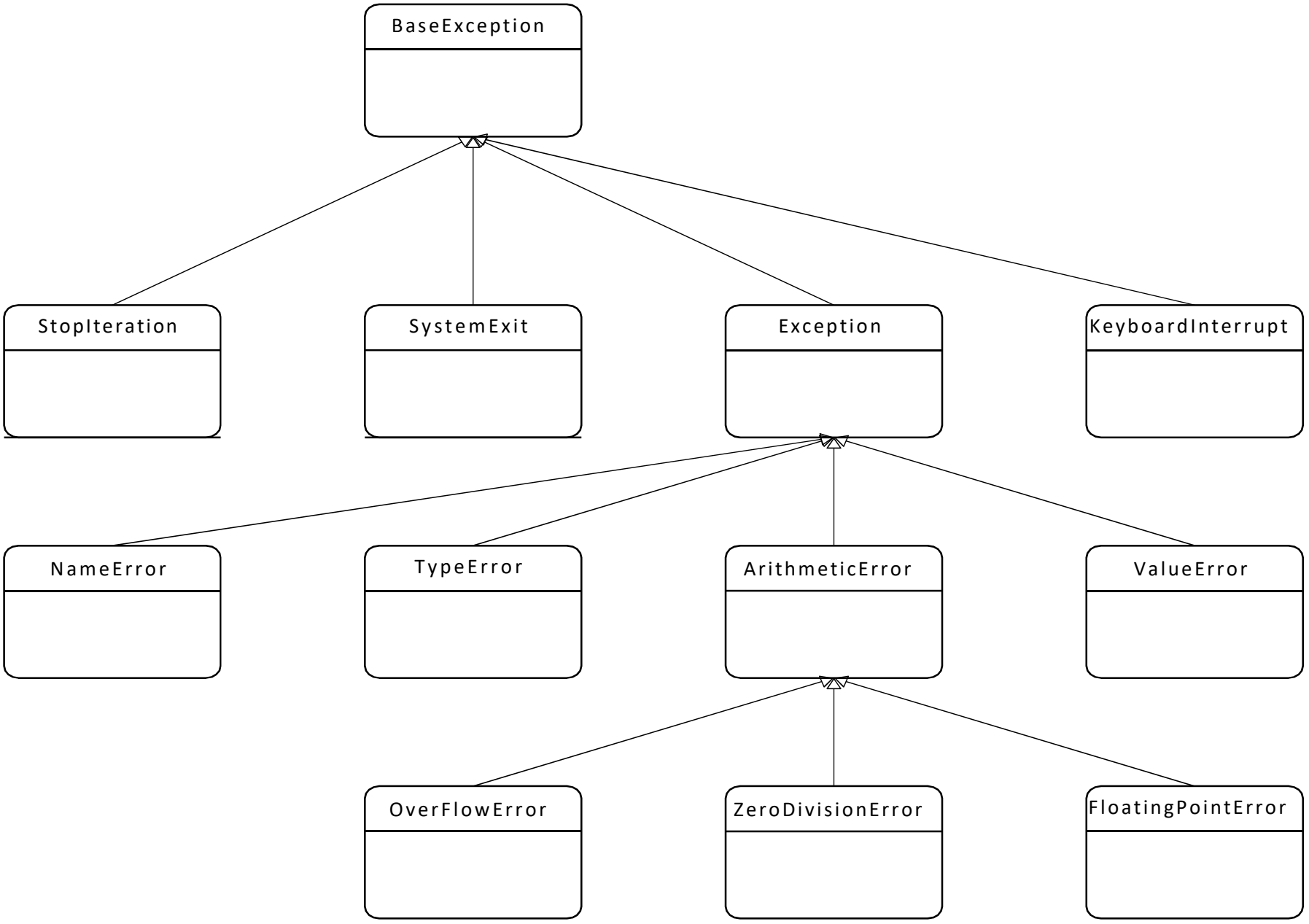
Nous voulons modifier l'exception précédente et la renommer en `AdresseError` pour qu'elle lève une exception si le code postal ne contient pas 5 caractère ou si la rue n'est pas entièrement en majuscule. L'exception doit rester paramétrée

Python

Quelques conventions **PEP 8** sur les exceptions

- Une exception est une classe donc elle doit respecter le **Pascal case**
- Le nom d'une exception doit se terminer par le mot `Error`

Hiérarchie des classes d'exception



Python

`BaseException`

- La classe `BaseException` est au sommet de la hiérarchie des exceptions en **Python**.
- Elle n'est pas destinée à être directement levée par les développeurs intégrées.
- Elle est plutôt utilisée comme classe mère pour toutes les autres exceptions intégrées.

Python

BaseException

- La classe `BaseException` est au sommet de la hiérarchie des exceptions en **Python**.
- Elle n'est pas destinée à être directement levée par les développeurs intégrées.
- Elle est plutôt utilisée comme classe mère pour toutes les autres exceptions intégrées.

Exception

- Généralement utilisée pour définir les exceptions personnalisées
- Classe de base pour toutes les exceptions intégrées, à l'exception de `SystemExit`, `KeyboardInterrupt`, `GeneratorExit` et `StopIteration`.

Python

Exceptions intégrées ou prédéfinies (Built-in exceptions)

- Elles sont des types d'exceptions qui sont incluses dans le langage Python par défaut.
- Elles sont disponibles pour être utilisées dans vos programmes sans nécessiter d'importation supplémentaire.
- Exemples
  - `TypeError` : Levée lorsque l'opération ou la fonction est appliquée à un objet d'un type inapproprié.
  - `ValueError` : Levée lorsque la fonction reçoit un argument d'une valeur incorrecte.
  - `NameError` : Levée lorsque le nom d'une variable ou d'une fonction n'est pas trouvé dans l'espace de noms local ou global.
  - `ZeroDivisionError` : Levée lorsque la division ou le modulo par zéro est tenté.
  - `ImportError` : Levée lorsqu'un import échoue
  - `IndentationError` : Levée lorsque l'indentation est incorrect.
  - ...