# Course: Python language

Ali ZAINOUL <ali.zainoul.az@gmail.com>

for IBM - Needemand
July 17, 2024

1. Exceptions
   - Principe de fonctionnement
   - Exceptions pré-définies et arbre d'héritage
   - Instructions `try ... except ... else ... finally`
   - Propagation des exceptions
   - Déclenchement d'exceptions
   - Définition d'une exception
2. Modules de la bibliothèque standard
   - Interaction avec l'interpréteur : module `sys`
   - Interaction avec le système d'exploitation : modules `os` et `pathlib`
   - Interaction avec le système de fichiers : module `os.path`
   - Expressions rationnelles : module `re`
   - Tests unitaires : instruction `assert`, module `unittest`
   - Tour d'horizon d'autres modules intéressants de la bibliothèque standard :
     - Module `datetime`
     - Module `math`

- Module `timeit`
- Module `urllib`
- Module `collections`
- Module `csv`
- Module `json`
- Module `unittest`
- Module `sqlite3`

# ** Exceptions **

# Principe de fonctionnement

- Les exceptions sont des événements qui se produisent lors de l'exécution d'un programme et qui interrompent le flux normal d'exécution.
- En Python, les exceptions sont des objets qui représentent des erreurs ou des conditions anormales.
- Lorsqu'une exception se produit, le programme s'arrête et affiche un message d'erreur à l'utilisateur.

# Exemple d'utilisation

```
try:
    # Code that may generate an exception
    result = 10 / 0
except ZeroDivisionError:
    # Handling of ZeroDivisionError exception
    print("Division by zero error!")
```

# Exceptions pré-définies et arbre d'héritage

- Python fournit un certain nombre d'exceptions pré-définies pour gérer différents types d'erreurs.
- Ces exceptions sont organisées dans une hiérarchie d'héritage, où les exceptions plus spécifiques héritent des exceptions plus générales.

# Exemple d'utilisation

```
try:
    # Code that may generate an exception
    file = open("example.txt", "r")
    content = file.read()
    file.close()
except FileNotFoundError:
    # Handling of FileNotFoundError exception
    print("File not found!")
```

# Instructions `try ... except ... else ... finally`

- L'instruction `try ... except ... else ... finally` est utilisée pour gérer les exceptions de manière plus complexe.
- La clause `else` est exécutée si aucune exception n'est levée dans la clause `try`.
- La clause `finally` est exécutée quel que soit le résultat de la clause `try`, que des exceptions aient été levées ou non.

# Exemple d'utilisation

```python
try:
    # Code that may generate an exception
    file = open("example.txt", "r")
except FileNotFoundError:
    # Handling of FileNotFoundError exception
    print("File not found!")
else:
    # Code executed if no exception is raised
    print("File read successfully!")
finally:
    # Code executed anyways
    file.close()
```

# Propagation des exceptions

- Lorsqu'une exception est levée dans une fonction, elle peut être propagée vers les fonctions appelantes jusqu'à ce qu'elle soit capturée.
- La propagation des exceptions permet de gérer les erreurs à différents niveaux de la hiérarchie d'appels de fonctions.

# Exemple d'utilisation

```python
def read_file(filename):
    try:
        file = open(filename, "r")
        content = file.read()
        file.close()
        return content
    except FileNotFoundError:
        print("File not found!")
        raise

try:
    data = read_file("example.txt")
except FileNotFoundError:
    print("Error reading file!")
```

# Déclenchement d'exceptions

- Les exceptions peuvent être déclenchées manuellement à l'aide de l'instruction `raise`.
- Cela permet de signaler des erreurs ou des conditions anormales dans le code.

# Exemple d'utilisation

```python
def validate_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative!")

try:
    validate_age(-5)
except ValueError as e:
    print("Invalid age:", e)
```

# Définition d'une exception

- En Python, il est possible de définir ses propres exceptions en créant de nouvelles classes dérivées de la classe de base `Exception`.
- Cela permet de personnaliser le comportement des exceptions et de les adapter aux besoins spécifiques de l'application.

# Exemple d'utilisation

```python
class CustomError(Exception):
    pass

def process_data(data):
    if not data:
        raise CustomError("Empty data!")

try:
    process_data([])
except CustomError as e:
    print("Error:", e)
```

## ** Modules de la bibliothèque standard **

# Interaction avec l'interpréteur : module `sys`

- Le module `sys` fournit des fonctionnalités permettant d'interagir avec l'interpréteur Python.
- Il permet d'accéder à des informations sur l'environnement d'exécution, de manipuler le chemin de recherche des modules, et plus encore.

# Exemple d'utilisation

```python
import sys

# Affichage de la version de Python
print("Python version:", sys.version)

# Affichage du chemin de recherche des modules
print("Module search path:", sys.path)
```

# Interaction avec le système d'exploitation : modules `os` et `pathlib`

- Les modules `os` et `pathlib` fournissent des fonctionnalités pour interagir avec le système d'exploitation et manipuler les chemins de fichiers de manière efficace.
- Le module `os` permet d'exécuter des opérations système telles que la création de répertoires, la navigation dans le système de fichiers, etc.
- Le module `pathlib` fournit une interface orientée objet pour manipuler les chemins de fichiers et de répertoires de manière portable.

# Exemple d'utilisation

```python
import os
from pathlib import Path

# Création d'un répertoire
os.makedirs("mydir")

# Récupération du répertoire courant
current_dir = Path.cwd()
print("Current directory:", current_dir)

# Vérification de l'existence d'un fichier
file_path = Path("myfile.txt")
if file_path.exists():
    print("File exists!")
else:
    print("File does not exist.")
```

# Interaction avec le système de fichiers : module `os.path`

- Le module `os.path` fournit des fonctionnalités pour manipuler les chemins de fichiers de manière efficace.
- Il permet de vérifier l'existence de fichiers, de répertoires, de manipuler les extensions de fichier, etc.

# Exemple d'utilisation

```python
import os.path

# Vérifier l'existence d'un fichier
file_path = "example.txt"
if os.path.exists(file_path):
    print("Le fichier existe:", file_path)
else:
    print("Le fichier n'existe pas:", file_path)

# Obtenir le répertoire parent d'un fichier
parent_directory = os.path.dirname(file_path)
print("Répertoire parent:", parent_directory)

# Obtenir l'extension d'un fichier
file_extension = os.path.splitext(file_path)[1]
print("Extension de fichier:", file_extension)
```

# Expressions rationnelles : module `re`

- Le module `re` permet de travailler avec des expressions rationnelles en Python.
- Il fournit des fonctionnalités pour rechercher, extraire et manipuler des motifs de texte basés sur des modèles définis.

# Tests unitaires : instruction `assert`, module `unittest`

- L'instruction `assert` est utilisée pour vérifier si une expression est vraie. Si l'expression est fausse, une erreur AssertionError est levée.
- Le module `unittest` fournit un framework pour écrire et exécuter des tests unitaires en Python.

# Exemple d'utilisation

```python
import unittest

def add(a, b):
    return a + b

class TestAddFunction(unittest.TestCase):

    def test_add_positive_numbers(self):
        self.assertEqual(add(1, 2), 3)

    def test_add_negative_numbers(self):
        self.assertEqual(add(-1, -2), -3)

    def test_add_mixed_numbers(self):
        self.assertEqual(add(1, -2), -1)
        self.assertEqual(add(-1, 2), 1)

if __name__ == "__main__":
    unittest.main()
```

# Module `datetime`

- Le module `datetime` fournit des classes pour manipuler des dates et des heures en Python.
- Il permet de créer, manipuler et formater des objets de date et d'heure.

# Exemple d'utilisation

```python
from datetime import datetime

# Création d'un objet de date et d'heure
now = datetime.now()
print("Date and time:", now)

# Formatage de la date et de l'heure
formatted_date = now.strftime("%Y-%m-%d %H:%M:%S")
print("Formatted date and time:", formatted_date)

# Extraction des composants de la date et de l'heure
year = now.year
month = now.month
day = now.day
hour = now.hour
minute = now.minute
second = now.second
print("Year:", year)
print("Month:", month)
print("Day:", day)
print("Hour:", hour)
print("Minute:", minute)
print("Second:", second)
```

# Module `math`

- Le module `math` fournit des fonctions mathématiques standard en Python.
- Il inclut des fonctions trigonométriques, logarithmiques, de puissance, etc.

# Exemple d'utilisation

```python
import math

# Exemples de fonctions mathématiques
print("Sin(30 degrees):", math.sin(math.radians(30)))  # Output: 0.49999999999999994
print("Cos(45 degrees):", math.cos(math.radians(45)))  # Output: 0.7071067811865476
print("Tan(60 degrees):", math.tan(math.radians(60)))  # Output: 1.7320508075688767
print("Logarithm base 2 of 16:", math.log2(16))          # Output: 4.0
print("Square root of 25:", math.sqrt(25))               # Output: 5.0
print("Factorial of 5:", math.factorial(5))              # Output: 120
```

# Module `timeit`

- Le module `timeit` est utilisé pour mesurer le temps d'exécution de petits fragments de code Python.
- Il fournit un moyen simple de comparer les performances de différentes implémentations.

# Exemple d'utilisation

```python
import timeit

# Exemple de mesure du temps d'exécution
time_taken = timeit.timeit(stmt='[i**2 for i in range(1000)]', number=10000)
print("Time taken:", time_taken)
```

# Module `urllib`

- Le module `urllib` est utilisé pour récupérer des données à partir d'URLs en Python.
- Il fournit des fonctionnalités pour ouvrir, lire et analyser des pages web.

# Exemple d'utilisation

```python
import urllib.request

# Exemple de récupération de données à partir d'une URL
response = urllib.request.urlopen('https://www.example.com')
html = response.read()
print(html)
```

# Module `collections`

- Le module `collections` fournit des classes spécialisées de conteneurs de données en Python.
- Il inclut des classes telles que `Counter`, `deque`, `namedtuple`, etc., qui étendent les fonctionnalités des types de données intégrés.

# Exemple d'utilisation

```python
import collections

# Exemple d'utilisation de Counter
word_counts = collections.Counter(['apple', 'banana', 'apple', 'orange', 'banana'])
print("Word counts:", word_counts)
```

# Module `csv`

- Le module `csv` est utilisé pour lire et écrire des fichiers CSV (Comma-Separated Values) en Python.
- Il fournit des fonctionnalités pour travailler avec des données tabulaires stockées dans des fichiers CSV.

# Exemple d'utilisation

```python
import csv

# Exemple de lecture d'un fichier CSV
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

# Module `json`

- Le module `json` est utilisé pour travailler avec des données JSON (JavaScript Object Notation) en Python.
- Il fournit des fonctionnalités pour sérialiser et désérialiser des objets Python en JSON et vice versa.

# Exemple d'utilisation

```python
import json

# Exemple de sérialisation et désérialisation JSON
data = {'name': 'John', 'age': 30}
json_string = json.dumps(data)
print("JSON string:", json_string)

parsed_data = json.loads(json_string)
print("Parsed data:", parsed_data)
```

# Module `unittest`

- Le module `unittest` fournit un framework de test unitaire intégré à Python.
- Il permet de définir et d'exécuter des tests unitaires pour vérifier le bon fonctionnement des modules et des fonctions.

# Exemple d'utilisation

```python
import unittest

# Exemple de test unitaire
class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('hello'.upper(), 'HELLO')

    def test_isupper(self):
        self.assertTrue('HELLO'.isupper())
        self.assertFalse('Hello'.isupper())

if __name__ == '__main__':
    unittest.main()
```

# Module `sqlite3`

- Le module `sqlite3` est utilisé pour travailler avec des bases de données SQLite en Python.
- Il fournit des fonctionnalités pour exécuter des requêtes SQL, récupérer des données et gérer les transactions.

# Exemple d'utilisation

```python
import sqlite3

# Exemple de création d'une base de données SQLite
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Création d'une table
cursor.execute('''CREATE TABLE IF NOT EXISTS stocks
                  (date text, trans text, symbol text, qty real, price real)''')

# Insertion de données
cursor.execute("INSERT INTO stocks VALUES ('2024-04-30', 'BUY', 'GOOG', 100, 1225.12)")

# Sauvegarde des modifications
conn.commit()

# Fermeture de la connexion
conn.close()
```