DÉCOUVREZ
LES FORMATIONS
DE L'OFPPT

OFPPT

FILIÈRES
CONDITIONS D'ACCÈS
PROGRAMME
DIPLÔMES

# Module : programmation orienté objet

# Projet : manipulation des donnes


❖ ***Realise par :***

- ✓ *Ilham barakat*
- ✓ *Hatim harchane*
- ✓ *Adam el mosaoui*
- ✓ *Assiya hamilou*


***Annee scolaire*** ***2023 2024***

***Formateur*** *othmane haqay*

***Filiere*** *devloppement digital 104*

***Centre de formation*** *ntic 2 sidi maarouf*

OFPPT

# PLAN :



# 1-INTRODUCTION :

- La programmation orientée objet est l'une des méthodologies récentes de programmation, couramment utilisée par les langages de programmation les plus répandus (python, JavaScript, C#.Net, ...). Cette méthodologie succède à la programmation impérative en lui ajoutant les notions d'objets et de classes.

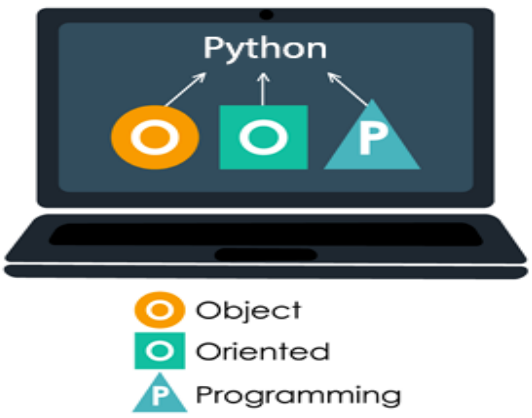
L'orientation objet présente de nombreux avantages, sinon elle ne serait pas le paradigme dominant du développement logiciel moderne. L'un des principaux avantages est l'encapsulation de la logique et des données dans des classes individuelles. Cela améliore la maintenabilité et rend le code plus facile à étendre.

# 2-DÉFINITION COLLECTION :

-En programmation orientée objet (POO), une collection fait référence à une structure de données qui permet de stocker et de manipuler un groupe d'éléments connexes. Une collection peut contenir des objets de différents types et peut fournir des opérations pour ajouter, supprimer, rechercher et parcourir ces éléments

## - Voici quelques caractéristiques

- **Stockage d'objets**
- **Manipulation d'objets**
- **Itération**
- **Redimensionnement dynamique**
- **Types de collections**



# 3-DÉFINITION NAMEDTUPLE :

- Une namedtuple est une structure de données immuable similaire à un tuple, mais avec des champs nommés. Elle est définie dans le module collections de Python et permet de créer des types de données simples avec des champs nommés, rendant le code plus lisible et évitant la confusion due à l'ordre des éléments dans un tuple standard.

## - exemple :

```
t=(3,28)
print(t) #affiche le contenu de t(3, 28)
print(t[0]) #affiche 1ere element  3
print(t[1]) #affiche 2eme element 28
```

- La classe namedtuple du module collections permet d'ajouter des noms explicites à chaque élément d'un tuple pour rendre ces significations claires dans un programme Python

```
from collections import namedtuple

# Définition d'une namedtuple nommée 'Point' avec deux champs 'x' et 'y'
Point = namedtuple('Point', ['x', 'y'])
# Création d'une namedtuple 'p1' avec des valeurs pour les champs 'x' et 'y'
p1 = Point(x=10, y=20)
# Accès aux champs 'x' et 'y' de la namedtuple 'p1'
print("Valeur de x:", p1.x)
print("Valeur de y:", p1.y)
# Conversion de la namedtuple 'p1' en tuple
tuple_p1 = p1
print("Tuple équivalent:", tuple_p1)  # Affiche: (10, 20)
# Indexation et tranche sur une namedtuple
print("Première valeur:", p1[0])
print("Deuxième valeur:", p1[1])
print("Tranche:", p1[:2])
# Décomposition d'une namedtuple
x, y = p1
print("Valeur de x:", x)
print("Valeur de y:", y)
```

**fields** : Cette méthode retourne une liste des noms de champs de la namedtuple

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
print(Point._fields)  # Affiche: ['x', 'y']
```

**asdict()** : Cette méthode retourne un dictionnaire contenant les champs nommés et leurs valeurs

```
p = Point(x=10, y=20)
print(p._asdict())  # Affiche: {'x': 10, 'y': 20}
```

**replace()** : Cette méthode retourne une nouvelle namedtuple avec les valeurs de champs spécifiées remplacées

```
p2 = p._replace(x=100)
print(p2)  # Affiche: Point(x=100, y=20)
```

# 4-DÉFINITION DEQUE:

- Un deque, abréviation de "double-ended queue"est une structure de données en Python.
Les déques sont implémentés dans le module collections de Python et sont généralement utilisés dans des scénarios où il est nécessaire d'accéder rapidement aux éléments à partir des deux extrémités de la séquence.

## Exemple :

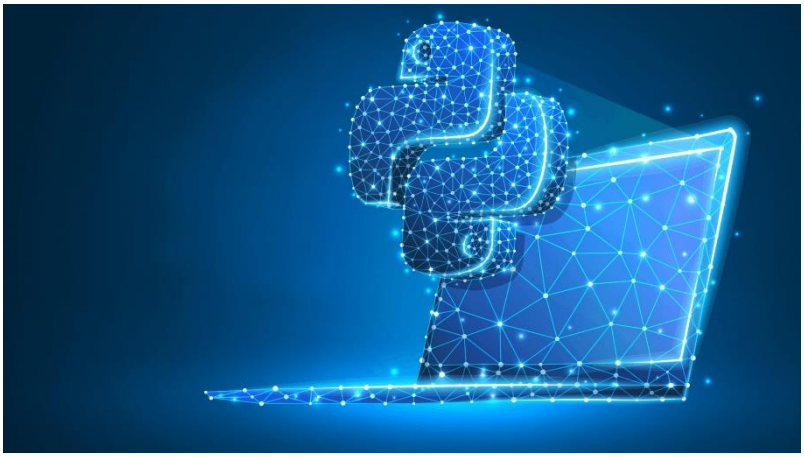
```
from collections import deque

# Création d'un deque vide
d = deque()

# Ajout d'éléments au début et à la fin du deque
d.append(1)       # Ajout de 1 à la fin du deque
d.appendleft(2)   # Ajout de 2 au début du deque

# Suppression d'éléments au début et à la fin du deque
x = d.pop()       # Suppression et retour du dernier élément du deque
y = d.popleft()   # Suppression et retour du premier élément du deque

print("Élément retiré de la fin du deque :", x)  # Affiche: Élément retiré de la fin du
deque : 1
print("Élément retiré du début du deque :", y)  # Affiche: Élément retiré du début du
deque : 2
```



# 5-DÉFINITION CHAINMAP:

-Est une classe de la bibliothèque standard Python qui permet de gérer plusieurs mappings (dictionnaires) comme une seule unité. Elle est utile lorsque vous avez plusieurs dictionnaires et que vous souhaitez les traiter comme une seule entité.

## Exemple :

```
from collections import ChainMap

# Création de deux dictionnaires
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}

# Création d'un ChainMap à partir des deux dictionnaires
chain_map = ChainMap(dict1, dict2)

# Accès aux éléments du ChainMap
print(chain_map['a'])  # Affiche: 1
print(chain_map['b'])  # Affiche: 2 (de dict1, car dict1 a la priorité)
print(chain_map['c'])  # Affiche: 4

# Modification d'un mapping sous-jacent
dict1['a'] = 10

# Les modifications sont reflétées dans le ChainMap
print(chain_map['a'])  # Affiche: 10
```

# 6-DÉFINITION COUNTER :

-est une classe de la bibliothèque standard Python, disponible dans le module `collections`, qui est utilisée pour compter les occurrences des éléments dans une séquence ou un ensemble d'éléments. Il s'agit essentiellement d'un dictionnaire spécialisé où les clés sont les éléments de la séquence et les valeurs sont les comptes correspondants de chaque élément

## Exemple :

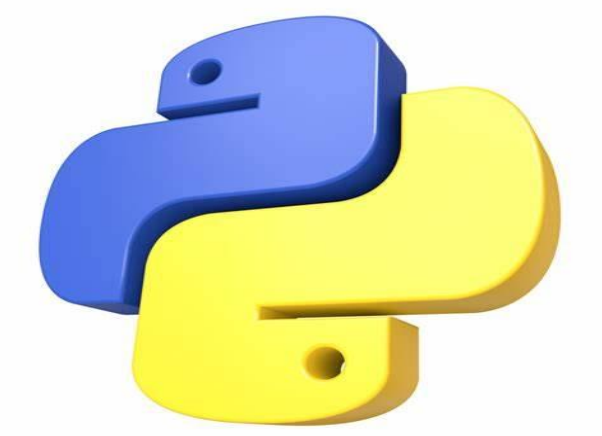
```
from collections import Counter

# Création d'un Counter à partir d'une liste
liste = ['a', 'b', 'c', 'a', 'b', 'a']
counter_liste = Counter(liste)
print(counter_liste)  # Affiche: Counter({'a': 3, 'b': 2, 'c': 1})

# Accès aux éléments les plus courants
print(counter_liste.most_common(2))  # Affiche: [('a', 3), ('b', 2)]

# Opérations arithmétiques
counter1 = Counter({'a': 2, 'b': 1})
counter2 = Counter({'a': 1, 'b': 2})
print(counter1 + counter2)  # Affiche: Counter({'a': 3, 'b': 3})

# Création d'un Counter à partir d'une chaîne de caractères
chaine = 'abracadabra'
counter_chaine = Counter(chaine)
print(counter_chaine)  # Affiche: Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
```



# 6-DÉFINITION ORDEREDDICT:

-`OrderedDict` garde une trace de l'ordre dans lequel les éléments ont été ajoutés, et cet ordre est préservé lors de l'itération ou de l'accès aux éléments.

## Exemple :

```
from collections import OrderedDict

# Création d'un OrderedDict
ordered_dict = OrderedDict()

# Ajout d'éléments à l'OrderedDict
ordered_dict['a'] = 1
ordered_dict['b'] = 2
ordered_dict['c'] = 3

# Affichage des éléments dans l'ordre d'insertion
for key, value in ordered_dict.items():
    print(key, value)

# Affichage de la clé 'b' avant la clé 'a' (l'ordre d'insertion est préservé)
print(list(ordered_dict.keys()))  # Affiche: ['a', 'b', 'c']
```

# 7-DÉFINITION DEFAULTDICT:

-Permet de créer des dictionnaires avec des valeurs par défaut pour les clés absentes. Lorsque vous accédez à une clé qui n'existe pas dans un `defaultdict`

## Exemple :

```
from collections import defaultdict

# Création d'un defaultdict avec une valeur par défaut de type int (0 par défaut)
d = defaultdict(int)

# Ajout de quelques éléments
d['a'] = 1
d['b'] = 2

# Accès à une clé absente
print(d['c'])  # Affiche: 0 (la valeur par défaut pour une clé absente est 0)

# Affichage du defaultdict
print(d)  # Affiche: defaultdict(<class 'int'>, {'a': 1, 'b': 2, 'c': 0})
```

# 8-CONCLUSION :

Ces méthodes pour manipulation des donnes sont toutes conçues pour simplifier la manipulation et l'analyse de données en Python, en offrant des fonctionnalités supplémentaires par rapport aux types de données de base comme les listes et les dictionnaires. En utilisant ces méthodes de manière appropriée, vous pouvez accélérer le développement de vos programmes et rendre votre code plus lisible et plus efficace.