OFPPT

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle
et de la Promotion du Travail

manipulation de données (collections)

SOMMAIRE

- Namedtple
- Deque
- Chainmap
- OrderedDict
- DefaultDict

I-Namedtple :

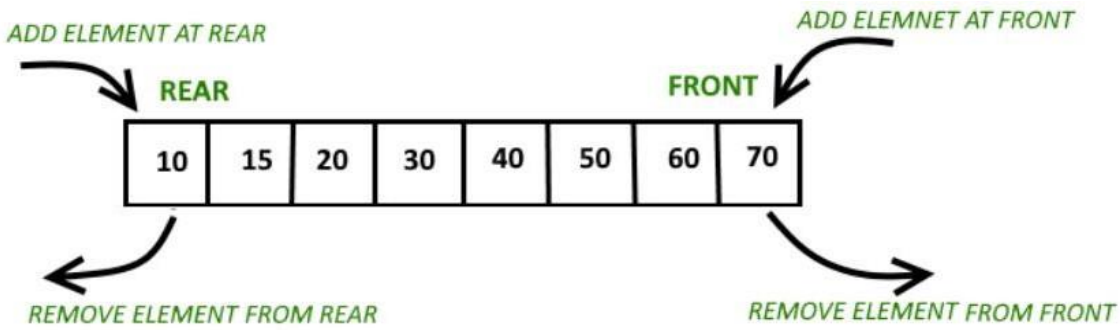
namedtuple : C'est une fonctionnalité très pratique de Python qui permet de créer des classes légères et immuables. Les namedtuple sont particulièrement utiles lorsque vous devez manipuler des enregistrements de données avec des champs nommés. Vous pouvez les utiliser pour représenter des objets simples mais structurés.

Le code :

```
from collections import namedtuple

# Définition d'une namedtuple nommée "Person"
Person = namedtuple('Person', ['name', 'age', 'city'])

# Création d'instances de Person
person1 = Person(name='Alice', age=30, city='New York')
person2 = Person(name='Bob', age=25, city='Paris')
person3 = Person(name='Charlie', age=35, city='London')

# Accès aux champs
print(person1.name)  # Affiche : Alice
print(person2.age)   # Affiche : 25
print(person3.city)  # Affiche : London

# Conversion en dictionnaire
person_dict = person1._asdict()
print(person_dict)   # Affiche : {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

II-Deque :

deque : Il s'agit d'une double-ended queue, une structure de données qui permet l'ajout et la suppression efficaces d'éléments à la fois au début et à la fin de la queue. Les deque sont souvent utilisées dans les algorithmes où vous avez besoin d'accéder rapidement aux éléments des deux extrémités de la file.



Le code :

```python
from collections import deque

# Création d'une deque avec quelques éléments initiaux
d = deque([1, 2, 3])

# Ajout d'un élément à la fin de la deque
d.append(4)

# Ajout d'un élément au début de la deque
d.appendleft(0)

# Affichage de la deque
print("Deque après ajout:", d)  # Affiche : Deque après ajout: deque([0, 1, 2, 3, 4])

# Suppression d'un élément au début
removed_item = d.popleft()
print("Élément supprimé:", removed_item)  # Affiche : Élément supprimé: 0
print("Deque après suppression:", d)       # Affiche : Deque après suppression: deque

# Suppression d'un élément à la fin
removed_item = d.pop()
print("Élément supprimé:", removed_item)  # Affiche : Élément supprimé: 4
print("Deque après suppression:", d)       # Affiche : Deque après suppression: deque
```

III-Chainemap :

chainmap : Cette structure de données permet de gérer plusieurs mappings (par exemple, des dictionnaires) comme une seule unité. Elle est utile lorsque vous avez plusieurs dictionnaires et que vous voulez les combiner de manière logique, tout en conservant la possibilité d'accéder à chacun d'eux individuellement.

Le code :

```
from collections import ChainMap

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

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

# Accès aux éléments de la ChainMap
print("Valeur de 'a' :", chain_map['a'])  # Affiche : Valeur de 'a' : 1
print("Valeur de 'b' :", chain_map['b'])  # Affiche : Valeur de 'b' : 2 (la valeur de dict1['b'] est utilisée car dict1 est prioritaire)
print("Valeur de 'c' :", chain_map['c'])  # Affiche : Valeur de 'c' : 4

# Modification des valeurs dans la ChainMap
chain_map['b'] = 5
chain_map['d'] = 6

# Affichage des dictionnaires originaux après modification
print("dict1 après modification :", dict1)  # Affiche : dict1 après modification : {'a': 1, 'b': 5}
print("dict2 après modification :", dict2)  # Affiche : dict2 après modification : {'b': 3, 'c': 4, 'd': 6}

# Affichage de la ChainMap après modification
print("ChainMap après modification :", chain_map)  # Affiche : ChainMap après modification : ChainMap({'a': 1, 'b': 5}, {'b': 3, 'c': 4, 'd': 6})
```

IV-OrderedDict :

ordereddict : C'est un dictionnaire ordonné, introduit dans Python 3.1. Contrairement à un dictionnaire standard, il conserve l'ordre d'insertion des éléments. Cela peut être utile lorsque vous avez besoin de parcourir les éléments d'un dictionnaire dans un ordre spécifique.

Le code :

```python
from collections import OrderedDict

# Création d'un OrderedDict avec des éléments non triés
unordered_dict = {'banana': 3, 'apple': 4, 'orange': 2, 'grape': 1}

# Affichage de l'OrderedDict non trié
print("OrderedDict non trié :", unordered_dict)
# Affiche : OrderedDict non trié : {'banana': 3, 'apple': 4, 'orange': 2, 'grape': 1}

# Création d'un OrderedDict à partir du dictionnaire non trié
ordered_dict = OrderedDict(unordered_dict)

# Affichage de l'OrderedDict trié
print("OrderedDict trié :", ordered_dict)
# Affiche : OrderedDict trié : OrderedDict([('banana', 3), ('apple', 4), ('orange', 2), ('grape', 1)])

# Ajout d'un nouvel élément à l'OrderedDict
ordered_dict['pear'] = 5

# Affichage de l'OrderedDict après l'ajout
print("OrderedDict après ajout :", ordered_dict)
# Affiche : OrderedDict après ajout : OrderedDict([('banana', 3), ('apple', 4), ('orange', 2), ('grape', 1), ('pear', 5)])

# Parcours des éléments de l'OrderedDict dans l'ordre d'insertion
print("Parcours des éléments dans l'ordre d'insertion :")
for key, value in ordered_dict.items():
    print(key, value)
# Affiche :
# banana 3
# apple 4
# orange 2
# grape 1
# pear 5
```

V-DefaultDict :

defaultdict : Cette structure de données est une sous-classe de dict qui fournit des valeurs par défaut pour les clés absentes lors de l'accès au dictionnaire. Cela peut être très utile pour éviter les erreurs de clé manquante lors de l'accès à un dictionnaire.

Le code :

```python
from collections import defaultdict

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

# Incrémentation d'une clé existante
d['a'] += 1

# Accès à une clé inexistante
print("Valeur de 'b' :", d['b'])  # Affiche : Valeur de 'b' : 0 (la valeur par défaut pour int est 0)

# Affichage du defaultdict après les opérations
print("defaultdict après les opérations :", d)  # Affiche : defaultdict après les opérations : defaultdict(<class 'int'>, {'a': 1, 'b': 0})

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

# Ajout d'éléments à une liste pour une clé inexistante
d_list['colors'].append('red')
d_list['colors'].append('blue')

# Affichage du defaultdict avec une liste
print("defaultdict avec une liste :", d_list)  # Affiche : defaultdict avec une liste : defaultdict(<class 'list'>, {'colors': ['red', 'blue']})

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

# Ajout d'éléments à un ensemble pour une clé inexistante
d_set['letters'].add('a')
d_set['letters'].add('b')
d_set['letters'].add('a')  # L'ensemble ne contiendra qu'une seule occurrence de 'a'

# Affichage du defaultdict avec un ensemble
print("defaultdict avec un ensemble :", d_set)  # Affiche : defaultdict avec un ensemble : defaultdict(<class 'set'>, {'letters': {'a', 'b'}})
```