

COURS LICENCE II TECHNIQUE PYTHON

EUTG



LISTE :

- En Python, une liste est une collection ordonnée et modifiable (mutable) d'éléments,
- qui peuvent être de n'importe quel type de données (nombres, chaînes, autres listes, etc.).
- Les listes sont définies avec des crochets [] et leurs éléments sont séparés par des virgules.
- Chaque élément possède un indice commençant à 0, ce qui permet de les accéder et de les manipuler.

Séquences de données

Les listes

Création d'une liste

```
Liste_0=[ ]
```

```
Liste_1 = [1,3,5,7,9,11,13]
```

```
Liste_2 = [1, 'Clara' , 'Amine', 19]
```



```
1 #pour creer une liste en python
2 #on definit une liste vide
3 ma_liste = []
4 #on peut ajouter des elements de types entier a la liste
5 ma_liste=[1,2,3,4,5]
6 #on peut ajouter des elements de types chaine de caractere a la liste
7 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
8 #on peut ajouter des elements de types float a la liste
9 ma_liste=[1.5,2.5,3.5,4.5,5.5]
10 #on peut ajouter des elements de types booleen a la liste
11 ma_liste=[True,False,True,False,True]
12 #on peut ajouter des elements de types mixtes a la liste
13 ma_liste=[1,"ali",2.5,True,"bamako",3,False,4,"chicago",5.5,True]
14 #on peut ajouter des elements de types liste a la liste
15 ma_liste=[1,2,3,[4,5,6],7,8,9]
16 #on peut ajouter des elements de types dictionnaire a la liste
17 ma_liste=[1,2,3,{"nom":"ali","prenom":"bamako"},4,5,6]
18 #on peut ajouter des elements de types tuple a la liste
19 ma_liste=[1,2,3,(4,5,6),7,8,9]
20 #on peut ajouter des elements de types set a la liste
21 ma_liste=[1,2,3,{4,5,6},7,8,9]
22 #on peut ajouter des elements de types None a la liste
23 ma_liste=[1,2,3,None,4,5,6]
```

LISTE

- **Ordonnée** : Les éléments sont stockés dans un ordre spécifique, que l'on peut retrouver par un indice.
- **Modifiable (Mutable)** : Vous pouvez ajouter, supprimer ou changer des éléments après la création de la liste.
- **Séquence d'éléments** : Une liste peut contenir différents types de données au sein d'une même collection,
- comme des entiers, des chaînes de caractères ou des flottants. **Indexée** :
- Chaque élément a une position numérique,
- ou index, qui commence à 0.
- Par exemple, `ma_liste[0]` est le premier élément.
- **Taille variable** : Une liste n'a pas de taille fixe ; elle peut grandir ou diminuer.



Les listes

ma_liste =	[1,	2,	" Amine "	" Martin"	-5,	False]
Indices positifs	0	1	2	3	4	5
Indices négatifs	-6	-5	-4	-3	-2	-1



```
1 #travaillons maintenant sur les listes
2 #on peut acceder a un element de la liste en utilisant son index
3 ma_liste=[1,2,3,4,5]
4 print(ma_liste[0]) #affiche 1
5 print(ma_liste[1]) #affiche 2
6 print(ma_liste[2]) #affiche 3
7 print(ma_liste[3]) #affiche 4
8 print(ma_liste[4]) #affiche 5
9 #on peut modifier un element de la liste en utilisant son index
10 ma_liste=[1,2,3,4,5]
11 ma_liste[0]=10
12 ma_liste[1]=20
13 ma_liste[2]=30
14 ma_liste[3]=40
15 ma_liste[4]=50
16 print(ma_liste) #affiche [10,20,30,40,50]
17 #on peut ajouter un element a la fin de la liste en utilisant la methode append()
18 ma_liste=[1,2,3,4,5]
19 ma_liste.append(6)
20 print(ma_liste) #affiche [1,2,3,4,5,6]
```



```
1 #on peut ajouter un element a une position donnee de la liste en utilisant la methode insert()
2 ma_liste=[1,2,3,4,5]
3 ma_liste.insert(0,10)
4 ma_liste.insert(1,20)
5 ma_liste.insert(2,30)
6 print(ma_liste) #affiche [10,20,30,1,2,3,4,5]
7 #on peut supprimer un element de la liste en utilisant la methode remove()
8 ma_liste=[1,2,3,4,5]
9 ma_liste.remove(1)
10 ma_liste.remove(2)
11 print(ma_liste) #affiche [3,4,5]
12 #on peut supprimer un element de la liste en utilisant la methode pop()
13 ma_liste=[1,2,3,4,5]
14 ma_liste.pop()
15 ma_liste.pop()
16 print(ma_liste) #affiche [1,2,3]
17 #on peut supprimer un element de la liste en utilisant son index avec del
18 ma_liste=[1,2,3,4,5]
19 del ma_liste[0]
20 del ma_liste[1]
21 print(ma_liste) #affiche [3,4,5]
```



```
1 # essayer de comprendre le code suivant
2 # de recuperer la valeur d'une liste
3 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
4 print(ma_liste[0]) #affiche ali
5 print(ma_liste[:]) #affiche toute la liste
6 print(ma_liste[1:3]) #affiche bamako,chicago
7 print(ma_liste[-1]) #affiche Amadou
8 print(ma_liste[-2]) #affiche Boubacar
9 print(ma_liste[-3]) #affiche chicago
10 print(ma_liste[-4]) #affiche bamako
11 print(ma_liste[-5]) #affiche ali
12 print(ma_liste[::-2]) #affiche ali,chicago,Amadou
13 print(ma_liste[1::-2]) #affiche bamako,Boubacar
14 print(ma_liste[::-1]) #affiche Amadou,Boubacar,chicago,bamako,ali
15 print(ma_liste[::-3]) #affiche ali,Boubacar
16 print(ma_liste[1::-3]) #affiche bamako,Amadou
17 print(ma_liste[2:3]) #affiche chicago
18 print(ma_liste[1:4]) #affiche bamako,chicago,Boubacar
19 print(ma_liste[0:5]) #affiche ali,bamako,chicago,Boubacar,Amadou
20 print(ma_liste[0:4]) #affiche ali,bamako,chicago,Boubacar
21 print(ma_liste[0:3]) #affiche ali,bamako,chicago
22 print(ma_liste[0:2]) #affiche ali,bamako
23 print(ma_liste[0:1]) #affiche ali
24
```



```
1 # affiche la longueur de la liste
2 print(len(ma_liste)) #affiche 5
3 # affiche le type de la liste
4 print(type(ma_liste)) #affiche <class 'list'>
5 # affiche la liste
6 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'Boubacar', 'Amadou']
7 # affiche la liste en utilisant une boucle for
8 for i in ma_liste:
9     print(i)
10 # affiche la liste en utilisant une boucle while
11 i=0
12 while i<len(ma_liste):
13     print(ma_liste[i])
14     i+=1
15 # affiche la liste en utilisant une comprehension de liste
16 [print(i) for i in ma_liste]
17 # affiche la liste en utilisant la fonction map
18 list(map(print,ma_liste))
19 # affiche la liste en utilisant la fonction enumerate
20 for index, value in enumerate(ma_liste):
21     print(index, value)
22 # affiche la liste en utilisant la fonction zip
23 for i in zip(ma_liste, range(len(ma_liste))):
24     print(i)
```



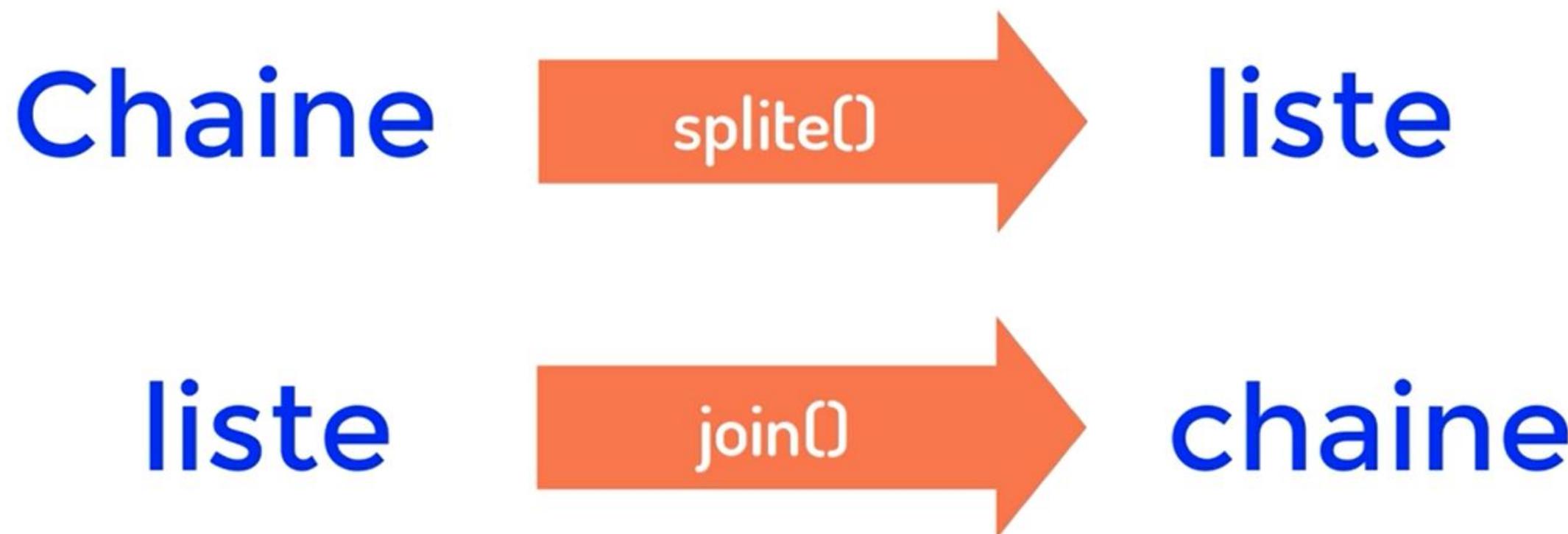
```
1 # affiche la liste en utilisant la fonction filter
2 list(filter(Lambda x: print(x), ma_liste))
3 # affiche la liste en utilisant la fonction sorted
4 print(sorted(ma_liste))
5 # affiche la liste en utilisant la fonction reversed
6 print(list(reversed(ma_liste)))
7 # affiche la liste en utilisant la fonction sum
8 # cette fonction ne marche que pour les listes d'entiers ou de float
9 ma_liste=[1,2,3,4,5]
10 print(sum(ma_liste)) #affiche 15
11 ma_liste=[1.5,2.5,3.5,4.5,5.5]
12 print(sum(ma_liste)) #affiche 17.5
13 # affiche la liste en utilisant la fonction min
14 ma_liste=[1,2,3,4,5]
15 print(min(ma_liste)) #affiche 1
16 ma_liste=[1.5,2.5,3.5,4.5,5.5]
17 print(min(ma_liste)) #affiche 1.5
18 # affiche la liste en utilisant la fonction max
19 ma_liste=[1,2,3,4,5]
20 print(max(ma_liste)) #affiche 5
21 ma_liste=[1.5,2.5,3.5,4.5,5.5]
22 print(max(ma_liste)) #affiche 5.5
```



```
1 #remplacer une valeur d'une liste
2 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
3 ma_liste[0]="Aminata"
4 ma_liste[1]="Moussa"
5 ma_liste[2]="Fatoumata"
6 ma_liste[3]="Seydou"
7 ma_liste[4]="Awa"
8 print(ma_liste) #affiche ['Aminata', 'Moussa', 'Fatoumata', 'Seydou', 'Awa']
9 #supprimer une valeur d'une liste sans utiliser la methode remove(),del () ou pop()
10 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
11 ma_liste=ma_liste[:2]+ma_liste[3:]
12 print(ma_liste) #affiche ['ali', 'bamako', 'Boubacar', 'Amadou']
```

```
1 #concatenation de liste
2 ma_liste1=["ali","bamako","chicago"]
3 ma_liste2=["Boubacar","Amadou"]
4 ma_liste=ma_liste1+ma_liste2
5 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'Boubacar', 'Amadou']
6 #repetition de liste
7 ma_liste=["ali","bamako","chicago"]
8 ma_liste=ma_liste*3
9 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'ali', 'bamako', 'chicago', 'ali', 'bamako', 'chicago']
10 #verifier si un element est dans une liste
11 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
12 print("ali" in ma_liste) #affiche True
13 print("Aminata" in ma_liste) #affiche False
14 print("bamako" in ma_liste) #affiche True
15 print("Moussa" in ma_liste) #affiche False
16 print("chicago" in ma_liste) #affiche True
17 print("Fatoumata" in ma_liste) #affiche False
18 print("Boubacar" in ma_liste) #affiche True
19 #verifier si un element n'est pas dans une liste
20 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
21 print("Aminata" not in ma_liste) #affiche True
22 print("ali" not in ma_liste) #affiche False
23 print("Moussa" not in ma_liste) #affiche True
24 print("bamako" not in ma_liste) #affiche False
25 print("Fatoumata" not in ma_liste) #affiche True
26 print("chicago" not in ma_liste) #affiche False
```

Conversion d'une chaine en une liste et inversion



```
1 #conversions entre liste et autres types de donnees
2 #conversion d'une chaine de caractere en liste
3 ma_chaine="ali,bamako,chicago,Boubacar,Amadou"
4 ma_liste=ma_chaine.split(",")
5 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'Boubacar', 'Amadou']
6 #conversion d'une liste en chaine de caractere
7 ma_liste=["ali","bamako","chicago","Boubacar","Amadou"]
8 ma_chaine=",".join(ma_liste)
9 print(ma_chaine) #affiche ali,bamako,chicago,Boubacar,Amadou
10 #conversion d'un tuple en liste
11 ma_tuple=("ali","bamako","chicago","Boubacar","Amadou")
12 ma_liste=list(ma_tuple)
13 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'Boubacar', 'Amadou']
14 #conversion d'un set en liste
15 ma_set={"ali","bamako","chicago","Boubacar","Amadou"}
16 ma_liste=list(ma_set)
17 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'Boubacar', 'Amadou']
18 #conversion d'un dictionnaire en liste
19 ma_dict={"nom":"ali","prenom":"bamako","ville":"chicago","pays":"USA"}
20 ma_liste=list(ma_dict)
21 print(ma_liste) #affiche ['nom', 'prenom', 'ville', 'pays']
22 ma_liste=list(ma_dict.values())
23 print(ma_liste) #affiche ['ali', 'bamako', 'chicago', 'USA']
24 ma_liste=list(ma_dict.items())
25 print(ma_liste) #affiche [('nom', 'ali'), ('prenom', 'bamako'), ('ville', 'chicago'), ('pays', 'USA')]
26 #utilisation de fonctions predefinies avec les listes
27
```

Les tuples

Les tuples en Python sont des séquences ordonnées et immuables d'éléments. Ils sont similaires aux listes, mais une fois créés, on ne peut pas modifier, ajouter ou supprimer d'éléments d'un tuple. Ils sont définis en utilisant des parenthèses (). Les tuples sont utiles pour représenter des collections de données qui ne doivent pas être modifiées, comme les coordonnées géographiques ou les enregistrements de base de données.

L'immutabilité des tuples permet d'optimiser le code en termes de performance et de sécurité, car les données restent constantes. Ils peuvent contenir différents types de données.

Les tuples

T1 = (1,2,3)

T2 = ("Amine" , 34, 5, "Amandine")

T3 = ("Amine", 34, 5, "Lina", ("Mastafi", 40), 56)

T4 = (32, -36, "Alain", "Said", [1,3,5,7,9], 10000)



```
1 #les tuples maintenant
2 tuple1 = (1, 2, 3)
3 tuple2 = ('a', 'b', 'c')
4 #concatener deux tuples
5 tuple3 = tuple1 + tuple2
6 print(tuple3) # affiche (1, 2, 3, 'a', 'b', 'c')
7 #repetition d'un tuple
8 tuple4 = tuple1 * 3
9 print(tuple4) # affiche (1, 2, 3, 1, 2, 3, 1, 2, 3)
10 #accéder à un élément d'un tuple
11 print(tuple1[0]) # affiche 1
12 print(tuple1[1]) # affiche 2
13 print(tuple1[2]) # affiche 3
14 #trouver la longueur d'un tuple
15 print(len(tuple1)) # affiche 3
16 #vérifier si un élément est dans un tuple
17 print(2 in tuple1) # affiche True
18 print(5 in tuple1) # affiche False
19 #trouver l'index d'un élément dans un tuple
20 print(tuple1.index(2)) # affiche 1
21 #compter le nombre d'occurrences d'un élément dans un tuple
22 tuple5 = (1, 2, 2, 3, 3, 3)
23 print(tuple5.count(2)) # affiche 2
24 print(tuple5.count(3)) # affiche 3
```



```
1 #déballer un tuple
2 a, b, c = tuple1
3 print(a) # affiche 1
4 print(b) # affiche 2
5 print(c) # affiche 3
6 #utiliser un tuple comme clé dans un dictionnaire
7 dict1 = {tuple1: 'valeur1', tuple2: 'valeur2'}
8 print(dict1) # affiche {(1, 2, 3): 'valeur1', ('a', 'b', 'c'): 'valeur2'}
9 #itérer sur les éléments d'un tuple
10 for element in tuple1:
11     print(element) # affiche 1 2 3
12 #utiliser la fonction zip avec des tuples
13 tuple8 = ('x', 'y', 'z')
14 zipped = zip(tuple1, tuple8)
15 print(list(zipped)) # affiche [(1, 'x'), (2, 'y'), (3, 'z')]
16 #trier un tuple
17 tuple9 = (3, 1, 2)
18 sorted_tuple = tuple(sorted(tuple9))
19 print(sorted_tuple) # affiche (1, 2, 3)
```



```
1 #determiner le type d'un tuple
2 print(type(tuple1)) # affiche <class 'tuple'>
3 #trouver le maximum et le minimum dans un tuple
4 print(max(tuple1)) # affiche 3
5 print(min(tuple1)) # affiche 1
6 #trouver la somme des éléments dans un tuple
7 print(sum(tuple1)) # affiche 6
8 #trouver la moyenne des éléments dans un tuple
9 print(sum(tuple1) / len(tuple1)) # affiche 2.0
10 #trouver la longueur d'un tuple
11 print(len(tuple1)) # affiche 3
12 #trouver le type d'un élément dans un tuple
13 print(type(tuple1[0])) # affiche <class 'int'>
14 print(type(tuple2[0])) # affiche <class 'str'>
15 #utiliser la fonction enumerate avec un tuple
16 for index, value in enumerate(tuple1):
17     print(index, value) # affiche 0 1, 1 2, 2 3
18 #utiliser la fonction map avec un tuple
19 squared = tuple(map(lambda x: x**2, tuple1))
20 print(squared) # affiche (1, 4, 9)
21 #utiliser la fonction filter avec un tuple
22 even = tuple(filter(lambda x: x % 2 == 0, tuple1))
23 print(even) # affiche (2,)
24 #utiliser la fonction reversed avec un tuple
25 reversed_tuple = tuple(reversed(tuple1))
26 print(reversed_tuple) # affiche (3, 2, 1)
```

LES DICTIONNAIRES

En Python, les dictionnaires sont des structures de données flexibles et puissantes qui permettent de stocker des informations sous forme de paires clé-valeur. Ils sont non ordonnés et les clés doivent être uniques et immuables (chaînes de caractères, nombres, etc.). Les valeurs peuvent être de n'importe quel type de données. On accède aux valeurs via leurs clés. Les dictionnaires sont essentiels pour organiser et manipuler des données associatives, similaires aux objets en JavaScript ou aux hash maps dans d'autres langages. Ils offrent des méthodes intégrées pour l'ajout, la suppression, et la mise à jour des éléments, ainsi que pour l'itération sur leurs clés, valeurs ou paires clé-valeur.



```
1 # les dictionnaires maintenant
2 # creer un dictionnaire
3 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
4 print(mon_dict) # affiche {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
5 # acceder a une valeur dans un dictionnaire
6 print(mon_dict['nom']) # affiche 'Alice'
7 print(mon_dict.get('age')) # affiche 25
8 # ajouter une paire clé-valeur dans un dictionnaire
9 mon_dict['pays'] = 'France'
10 print(mon_dict) # affiche {'nom': 'Alice', 'age': 25, 'ville': 'Paris', 'pays': 'France'}
11 # modifier une valeur dans un dictionnaire
12 mon_dict['age'] = 26
13 print(mon_dict) # affiche {'nom': 'Alice', 'age': 26, 'ville': 'Paris', 'pays': 'France'}
14 # supprimer une paire clé-valeur dans un dictionnaire
15 del mon_dict['ville']
16 print(mon_dict) # affiche {'nom': 'Alice', 'age': 26, 'pays': 'France'}
17 # supprimer une paire clé-valeur dans un dictionnaire en utilisant pop()
18 mon_dict.pop('pays')
19 print(mon_dict) # affiche {'nom': 'Alice', 'age': 26}
20 # obtenir toutes les clés d'un dictionnaire
21 cles = mon_dict.keys()
22 print(cles) # affiche dict_keys(['nom', 'age'])
23 # obtenir toutes les valeurs d'un dictionnaire
24 valeurs = mon_dict.values()
25 print(valeurs) # affiche dict_values(['Alice', 26])
26 # obtenir toutes les paires clé-valeur d'un dictionnaire
27 paires = mon_dict.items()
28 print(paires) # affiche dict_items([('nom', 'Alice'), ('age',
```



```
1 # iterer sur les clés d'un dictionnaire
2 for cle in mon_dict:
3     print(cle) # affiche 'nom' puis 'age'
4 # iterer sur les valeurs d'un dictionnaire
5 for valeur in mon_dict.values():
6     print(valeur) # affiche 'Alice' puis 26
7 # iterer sur les paires clé-valeur d'un dictionnaire
8 for cle, valeur in mon_dict.items():
9     print(cle, valeur) # affiche 'nom Alice' puis 'age 26'
10 # verifier si une clé existe dans un dictionnaire
11 print('nom' in mon_dict) # affiche True
12 print('ville' in mon_dict) # affiche False
13 # obtenir la longueur d'un dictionnaire
14 print(len(mon_dict)) # affiche 2
15 # vider un dictionnaire
16 mon_dict.clear()
17 print(mon_dict) # affiche {}
18 # creer un dictionnaire a partir de deux listes
19 cles = ['nom', 'age', 'ville']
20 valeurs = ['Bob', 30, 'New York']
21 mon_dict = dict(zip(cles, valeurs))
22 print(mon_dict) # affiche {'nom': 'Bob', 'age': 30, 'ville': 'New York'}
23 # copier un dictionnaire
24 mon_dict2 = mon_dict.copy()
25 print(mon_dict2) # affiche {'nom': 'Bob', 'age': 30, 'ville': 'New York'}
26 # obtenir la valeur par defaut d'une clé dans un dictionnaire
27 print(mon_dict.get('pays', 'Inconnu')) # affiche 'Inconnu'
28 # mettre a jour un dictionnaire avec un autre dictionnaire
29 mon_dict3 = {'pays': 'USA', 'age': 31}
30 mon_dict.update(mon_dict3)
31 print(mon_dict) # affiche {'nom': 'Bob', 'age': 31, 'ville': 'New York', 'pays': 'USA'}
```



```
1 # trier un dictionnaire
2 mon_dict_sorted = dict(sorted(mon_dict.items()))
3 print(mon_dict_sorted) # affiche {'age': 31, 'nom': 'Bob', 'pays': 'USA', 'ville': 'New York'}
4 # determiner le type d'un dictionnaire
5 print(type(mon_dict)) # affiche <class 'dict'>
6 # utiliser la fonction enumerate avec un dictionnaire
7 for index, (cle, valeur) in enumerate(mon_dict.items()):
8     print(index, cle, valeur) # affiche 0 nom Bob, 1 age 31, etc.
9 # utiliser la fonction map avec un dictionnaire
10 valeurs_maj = dict(map(lambda item: (item[0], str(item[1]).upper()), mon_dict.items()))
11 print(valeurs_maj) # affiche {'nom': 'BOB', 'age': '31', 'ville': 'NEW YORK', 'pays': 'USA'}
12 # utiliser la fonction filter avec un dictionnaire
13 mon_dict_filtered = dict(filter(lambda item: item[1] != 31, mon_dict.items()))
14 print(mon_dict_filtered) # affiche {'nom': 'Bob', 'ville': 'New York', 'pays': 'USA'}
15 # utiliser la fonction zip avec un dictionnaire
16 cles = ['nom', 'age', 'ville']
17 valeurs = ['Charlie', 28, 'Los Angeles']
18 mon_dict_zipped = dict(zip(cles, valeurs))
19 print(mon_dict_zipped) # affiche {'nom': 'Charlie', 'age': 28, 'ville': 'Los Angeles'}
20 # obtenir le maximum et le minimum des valeurs dans un dictionnaire
21 print(max(mon_dict.values())) # affiche 'USA'
22 print(min(mon_dict.values())) # affiche 'Bob'
23 # obtenir la somme des valeurs dans un dictionnaire
24 # cette operation n'est possible que si les valeurs sont des nombres
25 mon_dict_numbers = {'a': 1, 'b': 2, 'c': 3}
26 print(sum(mon_dict_numbers.values())) # affiche 6
27 # obtenir la moyenne des valeurs dans un dictionnaire
28 print(sum(mon_dict_numbers.values()) / len(mon_dict_numbers)) # affiche 2.0
29 # trier un dictionnaire par ses valeurs
30 mon_dict_sorted_by_values = dict(sorted(mon_dict_numbers.items(), key=lambda item: item[1]))
31 print(mon_dict_sorted_by_values) # affiche {'a': 1, 'b': 2, 'c': 3}
32 # trier un dictionnaire par ses clés
33 mon_dict_sorted_by_keys = dict(sorted(mon_dict_numbers.items()))
34 print(mon_dict_sorted_by_keys) # affiche {'a': 1, 'b': 2, 'c': 3}
35 # obtenir la longueur d'un dictionnaire
36 print(len(mon_dict)) # affiche 4
37 # determiner le type d'un dictionnaire
38 print(type(mon_dict)) # affiche <class 'dict'>
```

LES ENSEMBLES:

En Python, un ensemble (ou set) est une collection non ordonnée d'éléments uniques, semblable à une liste mais sans doublons. Les éléments d'un ensemble sont immutables (c'est-à-dire non modifiables, comme les chaînes de caractères ou les tuples) mais l'ensemble lui-même peut être modifié. Les ensembles sont très utiles pour supprimer les doublons d'une liste et pour effectuer des opérations mathématiques comme l'union, l'intersection ou la différence.



```
1 # les sets maintenant
2 # creer un set
3 mon_set = {1, 2, 3, 4, 5}
4 print(mon_set) # affiche {1, 2, 3, 4, 5}
5 # ajouter un element a un set
6 mon_set.add(6)
7 print(mon_set) # affiche {1, 2, 3, 4, 5, 6}
8 # supprimer un element d'un set
9 mon_set.remove(3)
10 print(mon_set) # affiche {1, 2, 4, 5, 6}
11 # supprimer un element d'un set sans erreur si l'element n'existe pas
12 mon_set.discard(10)
13 print(mon_set) # affiche {1, 2, 4, 5, 6}
14 # verifier si un element est dans un set
15 print(2 in mon_set) # affiche True
16 print(3 in mon_set) # affiche False
17 # obtenir la longueur d'un set
18 print(len(mon_set)) # affiche 5
19 # vider un set
20 mon_set.clear()
21 print(mon_set) # affiche set()
22 # creer un set a partir d'une liste
23 liste = [1, 2, 2, 3, 4, 4, 5]
24 mon_set = set(liste)
25 print(mon_set) # affiche {1, 2, 3, 4, 5}
```



```
1 # creer un set a partir d'une chaine de caractere
2 chaîne = "hello"
3 mon_set = set(chaîne)
4 print(mon_set) # affiche {'h', 'e', 'l', 'o'}
5 # operations ensemblistes
6 set1 = {1, 2, 3}
7 set2 = {3, 4, 5}
8 # union de deux sets
9 set_union = set1 | set2
10 print(set_union) # affiche {1, 2, 3, 4, 5}
11 # intersection de deux sets
12 set_intersection = set1 & set2
13 print(set_intersection) # affiche {3}
14 # difference de deux sets
15 set_difference = set1 - set2
16 print(set_difference) # affiche {1, 2}
17 # difference symetrique de deux sets
18 set_sym_diff = set1 ^ set2
19 print(set_sym_diff) # affiche {1, 2, 4, 5}
20 # verifier si un set est un sous-ensemble d'un autre set
21 print(set1 <= set_union) # affiche True
22 print(set2 <= set1) # affiche False
23 # verifier si un set est un super-ensemble d'un autre set
24 print(set_union >= set1) # affiche True
25 print(set1 >= set2) # affiche False
26 # determiner le type d'un set
27 print(type(mon_set)) # affiche <class 'set'>
28 # iterer sur les elements d'un set
29 for element in mon_set:
30     print(element) # affiche les elements du set
```



```
1 # utiliser la fonction enumerate avec un set
2 for index, value in enumerate(mon_set):
3     print(index, value) # affiche l'index et l'element du set
4 # utiliser la fonction map avec un set
5 squared_set = set(map(lambda x: x**2, mon_set))
6 print(squared_set) # affiche le set avec les elements au carre
7 # utiliser la fonction filter avec un set
8 even_set = set(filter(lambda x: x % 2 == 0, mon_set))
9 print(even_set) # affiche le set avec les elements pairs
10 # utiliser la fonction zip avec un set
11 set_a = {1, 2, 3}
12 set_b = {'a', 'b', 'c'}
13 zipped_set = zip(set_a, set_b)
14 print(list(zipped_set)) # affiche [(1, 'a'), (2, 'b'), (3, 'c')]
15 # obtenir le maximum et le minimum dans un set
16 print(max(mon_set)) # affiche le maximum du set
17 print(min(mon_set)) # affiche le minimum du set
18 # obtenir la somme des elements dans un set
19 print(sum(mon_set)) # affiche la somme des elements du set
20 # obtenir la moyenne des elements dans un set
21 print(sum(mon_set) / len(mon_set)) # affiche la moyenne des elements du set
22 # obtenir la longueur d'un set
23 print(len(mon_set)) # affiche la longueur du set
24 # determiner le type d'un set
25 print(type(mon_set)) # affiche <class 'set'>
```



```
1 # supprimer un element
2 mon_set = {1, 2, 3, 4, 5}
3 mon_set.pop()
4 print(mon_set) # affiche le set apres suppression d'un element
5 # supprimer un element aleatoirement
6 import random
7 mon_set.remove(random.choice(list(mon_set)))
8 print(mon_set) # affiche le set apres suppression d'un element aleatoire
9 # convertir un set en liste
10 liste_from_set = list(mon_set)
11 print(liste_from_set) # affiche la liste convertie du set
12 # convertir une liste en set
13 liste = [1, 2, 2, 3, 4, 4, 5]
14 set_from_list = set(liste)
15 print(set_from_list) # affiche le set converti de la liste
16 # convertir un set en tuple
17 tuple_from_set = tuple(mon_set)
18 print(tuple_from_set) # affiche le tuple converti du set
19 # convertir un tuple en set
20 tuple_data = (1, 2, 2, 3, 4, 4, 5)
21 set_from_tuple = set(tuple_data)
22 print(set_from_tuple) # affiche le set converti du tuple
23 # convertir un set en dictionnaire
24 # cette operation n'est possible que si les elements du set sont des tuples de deux elements (cle, valeur)
25 set_of_tuples = {('a', 1), ('b', 2), ('c', 3)}
26 dict_from_set = dict(set_of_tuples)
27 print(dict_from_set) # affiche le dictionnaire converti du set
28 # convertir un dictionnaire en set
29 dict_data = {'a': 1, 'b': 2, 'c': 3}
30 set_from_dict = set(dict_data.items())
31 print(set_from_dict) # affiche le set converti du dictionnaire
32
```



```
1 #modifier une valeur dans un dictionnaire
2 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
3 mon_dict['age'] = 26
4 print(mon_dict) # affiche {'nom': 'Alice', 'age': 26, 'ville': 'Paris'}
5 #supprimer une paire clé-valeur dans un dictionnaire
6 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
7 del mon_dict['ville']
8 print(mon_dict) # affiche {'nom': 'Alice', 'age': 25}
9 #supprimer une paire clé-valeur dans un dictionnaire en utilisant pop()
10 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
11 mon_dict.pop('age')
12 print(mon_dict) # affiche {'nom': 'Alice', 'ville': 'Paris'}
13 #vider un dictionnaire
14 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
15 mon_dict.clear()
16 print(mon_dict) # affiche {}
17 #copier un dictionnaire
18 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
19 mon_dict2 = mon_dict.copy()
20 print(mon_dict2) # affiche {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
21 #mettre a jour un dictionnaire avec un autre dictionnaire
22 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
23 mon_dict2 = {'age': 26, 'pays': 'France'}
24 mon_dict.update(mon_dict2)
25 print(mon_dict) # affiche {'nom': 'Alice', 'age': 26, 'ville': 'Paris', 'pays': 'France'}
26 #trier un dictionnaire
27 mon_dict = {'nom': 'Alice', 'age': 25, 'ville': 'Paris'}
28 mon_dict_sorted = dict(sorted(mon_dict.items()))
29 print(mon_dict_sorted) # affiche {'age': 25, 'nom': 'Alice', 'ville': 'Paris'}
```

GESTION DES ERREURS EN PYTHON

En Python, la gestion des erreurs, ou gestion des exceptions, se fait principalement avec les blocs Try, except, else, et finally pour anticiper et réagir aux erreurs d'exécution.

**On utilise le mot-clé raise pour signaler une erreur intentionnellement,
ou pour relancer une exception déjà gérée.**

**Les erreurs de syntaxe sont signalées lors de la phase de développement,
tandis que les exceptions surviennent pendant l'exécution du code**

Les blocs Try-except Ce mécanisme permet d'exécuter un code potentiellement problématique (try), puis de le traiter si une erreur se produit (except).

Try : Le code susceptible de générer une erreur est placé ici.

except : Si une exception survient dans le bloc Try, le code dans le bloc except est exécuté.

On peut spécifier le type d'exception à intercepter pour une gestion plus fine.

else : Le code dans ce bloc s'exécute uniquement si aucune exception n'a été levée dans le bloc Try.

finally : Le code dans ce bloc est exécuté quoi qu'il arrive, qu'une erreur soit survenue ou non.

Les différents types des erreurs courants en langage de programmation

Les erreurs courantes en Python se répartissent en trois catégories :

les erreurs de syntaxe (comme `IndentationError`),

qui surviennent lorsque le code n'est pas correctement écrit ;

les erreurs d'exécution (appelées aussi exceptions, comme `TypeError`, `ZeroDivisionError`, `KeyError`),

qui se produisent pendant que le programme s'exécute ; et les erreurs logiques,

où le code ne contient pas d'erreurs de syntaxe ou d'exécution mais ne produit pas le résultat attendu.

1. Erreurs de syntaxe (SyntaxError)

Ces erreurs signalent un problème dans la structure du code qui empêche l'interpréteur Python de le comprendre.

Exemple : Oublier des parenthèses ou des deux-points, ou utiliser une indentation incorrecte (l'exemple le plus courant est IndentationError).

Comment les éviter : Vérifiez attentivement la ponctuation, les mots-clés et l'indentation du code.

2. Erreurs d'exécution (Exceptions)

Ces erreurs surviennent lorsque le code est syntaxiquement correct mais que quelque chose d'inattendu se produit pendant l'exécution.

TypeError : Une opération est effectuée sur des objets de types incompatibles, par exemple, additionner un nombre et une chaîne de caractères.

ZeroDivisionError : Tentative de diviser un nombre par zéro.

KeyError : Essai d'accéder à une clé qui n'existe pas dans un dictionnaire.

IndexError : Accès à un index d'une liste ou d'une séquence qui est hors de portée.

AttributeError : Tentative d'accéder à un attribut ou une méthode qui n'existe pas sur un objet.

ImportError : Échec de l'importation d'un module, soit parce qu'il n'est pas installé, soit à cause d'une faute de frappe dans son nom.

3. Erreurs logiques

Ces erreurs ne provoquent pas l'arrêt du programme, mais le font fonctionner de manière incorrecte ou inattendue.

Exemple : Utiliser un opérateur de comparaison erroné, ou une formule de calcul incorrecte, qui donne un résultat faux sans pour autant produire une erreur d'exécution.

Comment les résoudre : Cette catégorie d'erreurs est souvent plus difficile à trouver, car le programme s'exécute. Il est utile de vérifier étape par étape la logique du programme, d'afficher les valeurs des variables avec `print()`, ou d'utiliser un débogueur.

Exemples des cas pratiques

Try:

```
numérateur = int(input("Entrez le numérateur :"))
dénominateur = int(input("Entrez le dénominateur :"))
résultat = numérateur / dénominateur
print(f"Le résultat est :{résultat}")
except ZeroDivisionError:
    print("Erreur : Division par zéro ! ")
except ValueError:
    print("Erreur :Veuillez entrer des nombres valides.")
except Exception as e:
# Pour capturer d'autres exceptions
    print(f"Une erreur inattendue est survenue :{e}")
```



```
1 # gestion de l'erreur IndexError dans une liste
2 ma_liste = [1, 2, 3]
3 try:
4     print(ma_liste[5]) # tente d'accéder à un index inexistant
5 except IndexError:
6     print("L'index 5 n'existe pas dans la liste.")
7
8 # gestion de l'erreur ValueError lors de la conversion de types
9 try:
10    nombre = int("abc") # tente de convertir une chaîne non numérique en entier
11 except ValueError:
12    print("Impossible de convertir 'abc' en entier.")
13
14 # gestion de l'erreur TypeError lors d'une opération invalide
15 try:
16    résultat = '2' + 2 # tente d'ajouter une chaîne et un entier
17 except TypeError:
18    print("Impossible d'ajouter une chaîne et un entier.")
19
20 # gestion de l'erreur ZeroDivisionError lors d'une division par zéro
21 try:
22    résultat = 10 / 0 # tente de diviser par zéro
23 except ZeroDivisionError:
24    print("Division par zéro n'est pas permise.")
```



```
1 # gestion de l'erreur AttributeError lors de l'accès à un attribut inexistant
2 try:
3     ma_liste = [1, 2, 3]
4     ma_liste.appendd(4) # tente d'appeler une méthode inexistante
5 except AttributeError:
6     print("La méthode 'appendd' n'existe pas pour les listes.")
7
8 # gestion de l'erreur FileNotFoundError lors de l'ouverture d'un fichier inexistant
9 try:
10    with open('fichier_inexistant.txt', 'r') as f:
11        contenu = f.read()
12 except FileNotFoundError:
13     print("Le fichier 'fichier_inexistant.txt' n'a pas été trouvé.")
14 # gestion de l'erreur ImportError lors de l'importation d'un module inexistant
15 try:
16    import module_inexistant # tente d'importer un module inexistant
17 except ImportError:
18    print("Le module 'module_inexistant' n'a pas été trouvé.")
19
20 # gestion de l'erreur SyntaxError lors de l'exécution d'un code avec une erreur de syntaxe
21 try:
22    eval('x === x') # tente d'exécuter un code avec une erreur de syntaxe
23 except SyntaxError:
24    print("Il y a une erreur de syntaxe dans le code.")
25 # gestion de l'erreur MemoryError lors d'une opération consommant trop de mémoire
26 try:
27    grosse_liste = [1] * (10**8) # tente de créer une liste très grande
28 except MemoryError:
29    print("Opération échouée en raison d'une consommation excessive de mémoire.")
30
```



```
1 # avec raise
2 def verifier_age(age):
3     if age < 0:
4         raise ValueError("L'age ne peut pas etre negatif.")
5     elif age < 18:
6         raise ValueError("L'age doit etre au moins 18 ans.")
7     else:
8         return "Age valide."
9 try:
10     print(verifier_age(-5)) # tente de verifier un age negatif
11 except ValueError as e:
12     print(e) # affiche le message d'erreur
13
14 try:
15     print(verifier_age(15)) # tente de verifier un age inferieur a 18
16 except ValueError as e:
17     print(e) # affiche le message d'erreur
```

Les fichiers

La gestion de fichiers en Python permet de lire, écrire, modifier et supprimer des fichiers texte ou binaires, en utilisant principalement la fonction `open()` et les méthodes associées (`read()`, `write()`, `close()`). L'utilisation de la syntaxe `with open(...)` est une bonne pratique essentielle, car elle garantit que le fichier est automatiquement fermé, même en cas d'erreur. Pour des opérations plus complexes sur les systèmes de fichiers, des modules comme `os` (pour supprimer, renommer, vérifier l'existence) et `shutil` (pour copier, déplacer) sont utilisés.

Fonctions de base

`open(chemin, mode)` : Ouvre un fichier.

chemin : Le nom ou le chemin complet du fichier.

mode : Spécifie l'opération, par exemple :

'r' : Lecture seule.

'w' : Écriture (écrase le contenu existant).

'a' : Ajout à la fin du fichier.

'r+' : Lecture et écriture.

`close()` : Ferme le fichier, libérant les ressources système. La syntaxe with le fait automatiquement.

`read()` : Lit tout le contenu du fichier sous forme de chaîne de caractères.

`readlines()` : Lit tout le contenu du fichier et le retourne sous forme d'une liste de chaînes, une par ligne.

`write(texte)` : Écrit du texte dans le fichier.

`writelines(liste_de_lignes)` : Écrit une liste de chaînes de caractères dans le fichier.

```
# Ouvrir et lire un fichier
with open("mon_fichier.txt", "r") as f:
    contenu = f.read()
    print(contenu)
# Écrire dans un fichier (écrasant le contenu existant)
lignes = ["Première ligne.\n", "Deuxième ligne.\n"]
with open("mon_fichier.txt", "w") as f:
    f.writelines(lignes)
# Ajouter du contenu à la fin d'un fichier
with open("mon_fichier.txt", "a") as f:
    f.write("Troisième ligne ajoutée.\n")
```

Modules supplémentaires

Module os : Permet d'interagir avec le système d'exploitation pour des opérations sur les fichiers et dossiers.

os.path.exists(chemin) : Vérifie si un fichier existe.

os.remove(chemin) : Supprime un fichier de manière permanente.

os.rename(ancien_nom, nouveau_nom) : Renomme un fichier.

Module shutil : Pour des opérations de copie et de déplacement de fichiers et dossiers.

Module pathlib : Une approche moderne et orientée objet pour manipuler les chemins de fichiers.

Module csv : Offre des fonctions spécifiques p



```
1 # gestion des fichiers
2 # ouvrir et lire un fichier
3 try:
4     with open('fichier.txt', 'r') as fichier:
5         contenu = fichier.read()
6         print(contenu) # affiche le contenu du fichier
7 except FileNotFoundError:
8     print("Le fichier n'a pas ete trouve.")
9 except IOError:
10    print("Erreur lors de la lecture du fichier.")
11
12 # ouvrir et ecrire dans un fichier
13 try:
14     with open('fichier.txt', 'w') as fichier:
15         fichier.write("Bonjour, ceci est une ligne ecrise dans le fichier.\n")
16         print("Ecriture reussie dans le fichier.")
17 except IOError:
18     print("Erreur lors de l'ecriture dans le fichier.")
19
20 # ouvrir et ajouter du texte a un fichier
21 try:
22     with open('fichier.txt', 'a') as fichier:
23         fichier.write("Ceci est une ligne ajoutee au fichier.\n")
24         print("Ajout reussi dans le fichier.")
25 except IOError:
26     print("Erreur lors de l'ajout dans le fichier.")
27
```



```
1 # lire un fichier ligne par ligne sans traiter les erreurs
2 with open('fichier.txt', 'r') as fichier:
3     for ligne in fichier:
4         print(ligne.strip()) # affiche chaque ligne du fichier sans les espaces inutiles
5
6 # lire un fichier et compter le nombre de lignes
7 try:
8     with open('fichier.txt', 'r') as fichier:
9         lignes = fichier.readlines()
10        print(f"Le fichier contient {len(lignes)} lignes.")
11 except FileNotFoundError:
12     print("Le fichier n'a pas ete trouve.")
13 except IOError:
14     print("Erreur lors de la lecture du fichier.")
15
16 # autre methode de lire un fichier
17 try:
18     fichier = open('fichier.txt', 'r')
19     contenu = fichier.read()
20     print(contenu) # affiche le contenu du fichier
21 except FileNotFoundError:
22     print("Le fichier n'a pas ete trouve.")
23 except IOError:
24     print("Erreur lors de la lecture du fichier.")
25 finally:
26     fichier.close() # ferme le fichier pour liberer les ressources
```

Les modules

En Python, un module est un fichier qui contient du code Python (fonctions, classes, variables) et qui peut être réutilisé dans d'autres scripts pour organiser le code et éviter la duplication. Les modules sont importés pour utiliser leurs définitions, et Python offre des modules standards intégrés ainsi que la possibilité de créer ses propres modules personnalisés ou d'utiliser ceux développés par d'autres.

À quoi servent les modules ?

Organisation du code : Ils permettent de découper un grand programme en fichiers plus petits et gérables, ce qui facilite la compréhension du code.

Réutilisation du code : Une fois définies dans un module, des fonctions, classes et variables peuvent être utilisées dans différents projets sans avoir à les copier.

Maintenance : En regroupant le code par fonctionnalité (par exemple, un module pour les calculs mathématiques), il est plus facile de le maintenir et de le modifier.

Éviter les collisions de noms : L'utilisation de modules permet de séparer les définitions, évitant ainsi les conflits de noms entre des fonctions ou variables identiques utilisées dans différents contextes.

Comment utiliser un module ?

1. Importer le module : Dans votre script, vous utilisez la commande import suivie du nom du module (sans l'extension .py).

```
import mon_module
```

2. Accéder aux définitions : Pour utiliser une fonction, une classe ou une variable du module, vous devez préfixer son nom avec le nom du module, suivi d'un point.

```
mon_module.ma_fonction()
```

3. Importer des éléments spécifiques : Vous pouvez également importer des éléments spécifiques d'un module avec la syntaxe from, ce qui vous permet de les utiliser directement sans le préfixe du module.

```
from mon_module import ma_fonction
```

Types de modules

Modules standards : Ils sont intégrés automatiquement avec Python et font partie de sa "bibliothèque standard".

Modules externes : Ce sont des modules développés par des tiers, souvent hébergés dans des dépôts et téléchargeables.

Modules personnalisés : Ce sont les modules que vous créez vous-même, en créant vos propres fichiers .py.



```
1 # les modules en python
2 # importer un module standard
3 import math
4 print(math.sqrt(16)) # affiche 4.0
5 print(math.pi) # affiche 3.141592653589793
6 # importer un module avec un alias
7 import numpy as np
8 array = np.array([1, 2, 3])
9 print(array) # affiche [1 2 3]
10 # importer une fonction spécifique d'un module
11 from datetime import datetime
12 now = datetime.now()
13 print(now) # affiche la date et l'heure actuelles
14 # importer toutes les fonctions d'un module
15 from random import *
16 print(randint(1, 10)) # affiche un entier aléatoire entre 1 et 10
17 # créer un module personnel
18 # créer un fichier mon_module.py avec le contenu suivant:
19 ''' def saluer(nom):
20     return f"Bonjour, {nom}!" '''
21
22 # importer et utiliser le module personnel
23 import mon_module
24 print(mon_module.saluer("Alice")) # affiche "Bonjour, Alice!"
25 # utiliser la fonction dir() pour lister les attributs et méthodes d'un module
26 print(dir(math)) # affiche la liste des attributs et méthodes du module math
27 # utiliser la fonction help() pour obtenir de l'aide sur un module
28 help(math) # affiche la documentation du module math
```



```
1 # installer et utiliser un module externe avec pip
2 # dans le terminal, executer la commande: pip install requests
3 import requests
4 response = requests.get('https://api.github.com')
5 print(response.status_code) # affiche le code de statut de la reponse
6 print(response.json()) # affiche le contenu JSON de la reponse
7 # mettre a jour un module externe avec pip
8 # dans le terminal, executer la commande: pip install --upgrade requests
9 # desinstaller un module externe avec pip
10 # dans le terminal, executer la commande: pip uninstall requests
11 # determiner le type d'un module
12 print(type(math)) # affiche <class 'module'>
13 print(type(np)) # affiche <class 'module'>
14 print(type(mon_module)) # affiche <class 'module'>
15 print(type(requests)) # affiche <class 'module'>
16 # utiliser la fonction enumerate avec un module
17 # d'autre bibliotheques
18 import calendar
19 for index, month in enumerate(calendar.month_name):
20     print(index, month) # affiche l'index et le nom du mois
21 # utiliser la fonction map avec un module
22 import string
23 uppercase_letters = list(map(str.upper, string.ascii_lowercase))
24 print(uppercase_letters) # affiche la liste des lettres majuscules
25 # utiliser la fonction filter avec un module
26 import string
27 vowels = list(filter(lambda x: x in 'aeiou', string.ascii_lowercase))
28 print(vowels) # affiche la liste des voyelles
29 # utiliser la fonction zip avec un module
30 import string
31 letters = string.ascii_lowercase[:5]
32 numbers = range(1, 6)
33 zipped = zip(letters, numbers)
34 print(list(zipped)) # affiche [('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
35 # obtenir le maximum et le minimum dans un module
36 import numpy as np
37 array = np.array([1, 2, 3, 4, 5])
38 print(np.max(array)) # affiche 5
39 print(np.min(array)) # affiche 1
40 # obtenir la somme des elements dans un module
```

FIN DE COURS

LICENCE 2

CONTACT : 93-01-25-82

EMAIL: mahamadousacko599@gmail.com

Mr SACKO