

Les structures de données en Python

Table des matières

I. Listes	3
II. Exercice : Quiz	6
III. Tuples	7
IV. Exercice : Quiz	8
V. Dictionnaires	9
VI. Exercice : Quiz	12
VII. Ensembles	13
VIII. Exercice : Quiz	16
IX. Essentiel	17
X. Auto-évaluation	17
A. Exercice	17
B. Test	18
Solutions des exercices	19

I. Listes

Durée : 1 h

Environnement de travail : PyCharm

Contexte

De nos jours, il existe de nombreux langages de programmation que l'on peut utiliser pour réaliser un programme quand on développe une application. Python est l'un des plus connus et est le plus utilisé par les développeurs et programmeurs.

Dans ce cours, on va parler de structure des données dans ce langage. Afin de consulter et travailler de manière efficace, on a besoin des structures de données qui permettent d'organiser et de stocker des données tout en définissant également la relation entre ces données et toutes les opérations pouvant être effectuées dessus. Le choix d'une structure de données dépend du contexte, du problème à résoudre et surtout du type de données dont vous disposez car une structure de données inadaptée entraîne des durées de fonctionnement inutilement longues, des pertes de mémoire et le gaspillage de stockage.

Certains types de structure de données sont mutables ou immuables. Pour le cas spécifique de Python, il dispose de ces deux types de données qui sont fréquemment utilisés quand on fait une programmation

En Python, on compte quatre instructions pour déclarer une structure de données. On va parler dans ce cours de ces 4 instructions : la liste, le tuple, le dictionnaire et l'ensemble.

Définition

Une liste, c'est une structure de données contenant une série de valeurs. En Python cette structure peut se présenter sous différents types pour construire une liste ; cela pourrait être un entier ou une chaîne de caractères.

Encore un sujet que vous devez comprendre avant de commencer à écrire des programmes sur les données de type liste : les listes peuvent contenir plusieurs valeurs, ce qui facilite l'écriture de programmes pour gérer de grandes quantités de données.

Et puisque les listes peuvent contenir d'autres listes, vous pouvez les utiliser pour organiser les données en structure hiérarchisée.

Une liste contient plusieurs valeurs dans une séquence ordonnée. Le terme « *valeur de liste* » fait référence à la liste elle-même (qui est une valeur pouvant être stockée dans une variable ou passée à une fonction comme toute autre valeur) ce qui n'est pas le cas des valeurs à l'intérieur de la liste.

Tout comme les valeurs de chaînes de caractères qui sont saisies avec des guillemets pour marquer le début et la fin, une liste commence par un crochet d'ouverture et se termine par un crochet de fermeture. Les valeurs sont séparées par des virgules.

La liste `[]` est une liste vide qui ne contient aucune valeur, similaire à `""` pour la chaîne de caractères vide.

Exemple Récupérer les valeurs individuelles de liste avec l'indice

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
```

Supposons que la liste soit stockée dans une variable nommée « *spam* ». Python évaluera `spam[0]` comme `"cat"` et `spam[1]` comme `"bat"`.

- `"cat"` équivaut à `spam[0]`
- `"bat"` équivaut à `spam[1]`
- `"rat"` équivaut à `spam[2]`
- `"elephant"` équivaut à `spam[3]`

Le nombre entier entre crochets qui suit la liste est appelé un indice. La première valeur de la liste est d'indice 0, la deuxième valeur est à l'index 1, la troisième valeur est à l'index 2, et ainsi de suite.

Le schéma montre comment une valeur de liste est attribuée à spam, ainsi que ce que la position d'indice évalue.

Remarque

Python vous donnera un message d'erreur `IndexError` si vous utilisez un indice qui dépasse le nombre de valeurs dans votre valeur de liste.

Les index ne peuvent être que des valeurs entières, pas de chiffre à virgule.

Les listes peuvent également contenir d'autres valeurs de liste. Les valeurs dans cette liste de listes sont accessibles à l'aide de plusieurs index.

Le premier indice dicte la valeur de liste à utiliser et le second indique la valeur dans la valeur de liste. Si vous n'utilisez qu'un seul indice, le programme imprimera la valeur complète de la liste à cet indice.

```
1 liste1=[1, 2, 3]
2 liste2=[4, 5, 6]
3 liste3=[7, 8, 9]
4 spam = [liste1, liste2, liste3]
5 print(spam[1][1])
```

La sortie de ce code est : 5

On déclare 3 variables : liste1, liste2, liste3 ayant chacune des valeurs ; on stocke toutes ces listes dans une variable spam, et pour accéder à la valeur 5 de la liste2 on a mis un code spam [1] [1], le premier est l'indice de la liste spam qui est liste2, car on commence toujours par l'indice zéro dans la liste ; et le deuxième argument est l'indice de la liste2 ici, l'indice 1 est la valeur 5 car c'est le deuxième élément.

Prenons un exemple pour mieux le comprendre.

Pour le premier test, ouvrez l'IDE PyCharm et créez un projet, donnez juste le nom qui vous convient, et créez un fichier avec l'extension .py. C'est dans ce fichier Python qu'on va faire le test.

La méthode print() :

Pour mieux le comprendre, on déclare une variable spam pour mettre les listes que nous voulons créer et manipuler. Quand on lance le projet, il va nous lister toutes les listes qui existent dans la variable spam. Après la déclaration, on affiche les éléments qui se trouvent dans la liste par la méthode `print()`.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 print(spam)
```

La sortie de ce code est : ['cat', 'bat', 'rat', 'elephant']

En utilisant les indices, voici un exemple :

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 print(spam[0])
```

La sortie de ce code est : cat

On peut aussi utiliser le négatif, mais cette fois-ci le négatif compte depuis le dernier élément qu'on a mis dans la liste : ici par exemple le « elephant » prend l'indice - 1 et le « rat » le - 2 et ainsi de suite.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 print(spam [-1])
```

La méthode len() :

Pour savoir quel est le dernier élément dans une liste, on peut utiliser la méthode `len()` comme le montre l'exemple suivant :

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 print(spam[len(spam)-1])
```

Contrairement aux chaînes de caractères, on peut altérer la structure dans les listes, ou en modifier ses contenus (remplacer, ajouter et supprimer) ; comme les exemples qu'on va voir ci-dessous, en prenant toujours la variable et les contenus qu'on a déjà créés :

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 print(spam[2])
3 spam[2] = 'lion'
4 print(spam[2])
5 print(spam)
```

Les sorties de ces codes sont :

rat

lion

['cat', 'bat', 'lion', 'elephant']

La méthode insert() :

On peut également mettre un élément dans la liste ou bien même préciser où nous voulons le mettre en utilisant la méthode `insert()` ; on va prendre un exemple en mettant un élément *"tiger"* entre cat et bat.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 spam.insert(1, 'tiger')
3 print(spam)
```

Le résultat sera : ['cat', 'tiger', 'bat', 'rat', 'elephant']

La fonction enumerate() :

Pour connaître les éléments existants dans une liste et récupérer l'index, on peut utiliser l'instruction `for` et la fonction `enumerate()`.

Voici donc un exemple : ici on va déclarer une variable appelée « *element* » :

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 for element in enumerate(spam) :
3     print(element)
```

La sortie de ce code est :

(0, 'cat')

(1, 'bat')

(2, 'rat')

(3, 'elephant')

La fonction del() :

Suppression d'élément dans la liste.

On va voir deux possibilités pour la suppression :

Premièrement on va essayer de supprimer un élément dans la liste par l'indice, pour cela on va utiliser la fonction `del()`.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 del spam[0]
3 print(spam)
```

La sortie de ce code est : ['bat', 'rat', 'elephant']

La méthode remove :

Deuxièmement, on peut utiliser la méthode `remove()`, mais cette fois-ci, on doit mentionner la valeur qu'on va supprimer.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 spam.remove('cat')
3 print(spam)
```

La sortie de ce code est : `['bat', 'rat', 'elephant']`

La condition if...else :

Si nous voulons savoir qu'un élément existe dans la liste en précisant aussi la position ou l'index, on peut utiliser la condition `if...else` comme le montre l'exemple suivant :

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 if 'bat' in spam :
3     print('Cet élément est à la position {} de la liste.'.format(spam.index(bat)))
4 else :
5     print('l'élément bat n'existe pas dans la liste spam')
```

La sortie de ce code est : `Cet élément est à la position 1 de la liste spam.`

Si jamais l'élément demandé n'est pas dans la liste, il va afficher la deuxième condition.

La fonction sort() :

Pour afficher les données en ordre croissant, on peut utiliser la fonction `sort()` ou au contraire en ordre décroissant, on utilise la fonction `sort()` avec `reverse=True` à l'intérieur des parenthèses comme ceci : `sort(reverse=True)`.

Exemple

Voici donc un exemple pour mieux expliquer ces fonctions.

On prend toujours la valeur `spam` qu'on a déjà créée et on va afficher les valeurs d'abord par ordre croissant, puis par ordre décroissant.

```
1 spam = ['cat', 'bat', 'rat', 'elephant']
2 spam.sort()
3 print(spam)
4 spam.sort(reverse=True)
5 print(spam)
```

La sortie de ces codes est :

```
['bat', 'cat', 'elephant', 'rat']
['rat', 'elephant', 'cat', 'bat']
```

Exercice : Quiz

[solution n°1 p.21]

Question 1

Laquelle de ces déclarations est vraie pour la déclaration sur la liste :

- ☐ Lettres = {'pommes', 'banane', 'orange'}
- ☐ Lettres = ['pommes', 'banane', 'orange']
- ☐ Lettres = ('pommes', 'banane', 'orange')

Question 2

Laquelle de ces réponses est vraie si on déclare une variable liste = [5, 10, 15, 20, 30] et qu'on fait le print(liste[2]) ?

- ☐ 5
- ☐ 10
- ☐ 15
- ☐ 20
- ☐ 30

Question 3

Supposons qu'on a des valeurs entières dans une liste ; laquelle de ces déclarations est vraie :

- ☐ Nombre= ["15";"20"]
- ☐ Nombre= (15 , 20)
- ☐ Nombre= [15, 20]
- ☐ Nombre= int ([15, 20])

Question 4

Si nous voulons supprimer un élément dans une liste en pointant avec sa valeur, quelle fonction doit-on utiliser pour la suppression ?

- ☐ del()
- ☐ remove()

Question 5

Pour insérer un élément entre les indices 0 et 1 de la liste spam= ["blanc", "rouge", "vert", "bleue"], quelle est la bonne manière de l'écrire ?

- ☐ spam= ["*blanc*", "*rouge*", "*vert*", "*bleue*"]
- ☐ a- spam.insert(0, "*noir*")
- ☐ b-spam.insert(1, "*noir*")
- ☐ c-spam.insert(2, "*noir*")

III. Tuples

Définition

Un **tuple** est une **liste** qui ne peut plus être modifiée. Il permet de créer une collection ordonnée contenant plusieurs éléments. En mathématiques, on parlera de n-uplet. Par exemple, un triplé est constitué de 3 éléments.

Les tuples ressemblent aux listes. Toutefois, on ne peut pas les modifier une fois qu'ils ont été créés et ils sont entourés de parenthèses au lieu de crochets.

Si vous n'avez qu'une seule valeur dans votre tuple, vous pouvez l'indiquer en plaçant une virgule après la valeur entre parenthèses. Sinon, Python pensera qu'il s'agit d'une chaîne de caractères, la virgule lui permet de l'identifier comme un tuple.

On peut utiliser la fonction `type()` pour déterminer le type de données.

Outre les différents types de crochets utilisés pour les délimiter, la principale différence entre un tuple et une liste est que l'objet tuple est immuable. Une fois que nous avons déclaré le contenu d'un tuple, nous ne pouvons pas le modifier. Python peut implémenter certaines optimisations pour rendre le code légèrement plus rapide qu'avec des listes.

Méthode

On va voir comment on déclare une variable en tuples. Ici on a déclaré trois tuples : `tuple1`, `tuple2`, `tuple3`. On va afficher d'abord les valeurs dans le `tuple1`, et puis celui qui porte l'index 0, le dernier élément dans le `tuple2` et de l'index 2 dans le `tuple3`.

```
1 tuple1= 1, 3, 5, 7, 9
2 tuple2=(20, 'Jean', 'Canada')
3 tuple3=('Victor', 690, ['United-Stade', 'Washington'])
4 print(tuple1)
5 print(tuple1[0])
6 print(tuple2[-1])
7 print(tuple3[2])
```

Les résultats sont donc :

```
(1, 3, 5, 7, 9)
```

```
1
```

```
Canada
```

```
['United-Stade', 'Washington']
```

On peut également déclarer une variable avec les valeurs qui ne sont pas dans la parenthèse en tuple comme le cas de la première variable.

Dans les tuples, il est nécessaire de connaître la fonctionnalité de « *déballage de séquence* ». C'est une affectation des valeurs différentes d'un tuple dans les variables séparées. On va prendre un exemple pour le `tuple3` qu'on a déjà déclaré avant.

```
1 tuple3=('Victor', 690, ['United-Stade', 'Washington'])
2 nom, adresse, lieu = tuple3
3 # et maintenant on va afficher la valeur de nom et le lieu comme ceci :
4 print(nom)
5 print(lieu)
```

Les résultats sont :

```
Victor
```

```
['United-Stade', 'Washington']
```

Exercice : Quiz

[solution n°2 p.22]

Question 1

Dans les déclarations suivantes, laquelle est une déclaration de type tuple ?

- ☐ tuple1=["rouge", "bleu", "noir"]
- ☐ tuple1=("rouge", "bleu", "noir")
- ☐ tuple1={"rouge", "bleu", "noir"}

Question 2

Après une déclaration de tuple, on a comme donnée : tuple=(1, 5, 20, 4). Si on fait le print(tuple[- 1 :: - 2]), laquelle de ces propositions s'affichera ?

- ☐ Index error
- ☐ 5
- ☐ 4
- ☐ (4, 5)

Question 3

Supposons qu'on a une liste de type tuple : tuple1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). On lance la commande suivante : print(tuple1[2 : 4], tuple1[: 3], tuple1[5 :]), laquelle de ces propositions s'affichera ?

- ☐ (2, 4) (1, 3) (5,10)
- ☐ (3, 4) (1, 2, 3) (6, 7, 8, 9, 10)

Question 4

Choisissez la bonne réponse : laquelle va afficher la valeur 2 de cette déclaration tuple : tuple1 = ([1, 2, 3], 'chiffre', (4, 5, 6)).

- ☐ print(tuple1[0 :3][1])
- ☐ print(tuple1[0 :3](1))
- ☐ print(tuple1[0][1])

Question 5

Quel est le résultat de ce script :

```
1 tuple = 'noir', 2, 'blanc'
2 a, b, c = tuple
3 print(a)
4
```

- ☐ ('noir',2,'blanc')
- ☐ SyntaxError
- ☐ noir

V. Dictionnaires

Définition

Comme une liste, un dictionnaire est une collection de valeurs. Mais contrairement aux indices pour les listes, les indices des dictionnaires peuvent utiliser de nombreux types de données différents, pas seulement les entiers.

Les index des dictionnaires sont appelés des **keys**, ils sont associés aux valeurs appelées la paire **keys-value**.

Les dictionnaires peuvent utiliser des entiers comme clés ou des chaînes de caractères tout comme les listes utilisent l'indice, mais ils ne doivent pas commencer à 0.

Dictionnaire et liste

Les listes n'ont pas besoin d'être toujours homogènes, ce qui en font un outil puissant. Une seule liste peut contenir des données comme des entiers, des chaînes et des objets. Les listes sont modifiables et peuvent donc être modifiées même après leur création.

Dans le dictionnaire, les valeurs de données ne sont pas ordonnées. Contrairement aux autres types de données qui contiennent une seule valeur, le dictionnaire contient la paire clé-valeur. Chaque paire clé-valeur dans un dictionnaire est séparée par deux points « : », tandis que chaque clé est séparée par une virgule « , ».

Contrairement aux listes, les éléments des dictionnaires ne sont pas classés. Le premier élément de la liste spam serait du spam [0]. Mais il n'y a pas de premier élément dans un dictionnaire.

Bien que l'ordre des éléments soit important pour déterminer si deux listes sont identiques, l'ordre importe peu dans la paire clé-valeur d'un dictionnaire.

Étant donné que les dictionnaires ne sont pas classés, ils ne peuvent pas être découpés comme des listes. Essayer d'accéder à une clé qui n'existe pas dans un dictionnaire entraînera un message d'erreur `KeyError`, un peu comme l'erreur `IndexError` « hors de portée » dans une liste.

Bien que les dictionnaires ne soient pas ordonnés, le fait que vous puissiez avoir des valeurs pour les clés vous permet d'organiser vos données.

Les méthodes `keys()`, `values()` et `items()`

Il existe trois méthodes sur les dictionnaires qui renvoient des valeurs de type liste des clés, valeurs ou les deux :

- `Keys()`
- `Values()`
- `Items()`

Les valeurs renvoyées par ces méthodes ne sont pas des vraies listes : elles ne peuvent pas être modifiées et ne disposent pas de la méthode `append()`.

Mais ces types de données (clés, valeurs et éléments) peuvent être utilisés dans des boucles `for`.

Utiliser les méthodes `keys()`, `values()` et `items()` dans une boucle `for` permet d'itérer sur les clés, les valeurs ou la paire clé-valeur dans un dictionnaire.

Remarquez que les valeurs retournées par la méthode `items()` sont des tuples de la clé et de la valeur.

Si vous voulez une vraie liste à partir de l'une de ces méthodes, passez la valeur avec la fonction `list()`.

Voici quelques exemples qui vont éclaircir les explications ci-dessus.

D'abord, l'initialisation d'un dictionnaire :

```
var = { } ou bien var = dict( )
```

Pour ajouter des valeurs dans le dictionnaire, il faut mettre une clé et une valeur comme ceci :

```
1 var = {}  
2 var['nom'] = 'Jacques'  
3 var['âge'] = 35  
4 print(var)
```

La sortie de ce code est : {'nom': 'Jacques', 'âge': 35}

Ou on peut directement déclarer la clé et la valeur avec :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}  
2 print(var['âge'])
```

La sortie de ce code est : 35

Remarque

On note qu'on ne peut pas créer la même clé dans un dictionnaire mais qu'on peut utiliser la même variable plusieurs fois.

var.values et var.keys

Si on veut afficher simplement les variables et ou les clés, on peut utiliser `var.values` et `var.keys` comme ceci :

```
1 print(var.values())  
2 print(var.keys())
```

Le résultat est :

```
dict_values(['Jacques', 35])
```

```
dict_keys(['nom', 'âge'])
```

La méthode dict :

On peut également créer un dictionnaire Python à partir de la liste (des clés, des valeurs) en utilisant la méthode `dict` suivant le `zip`. Supposons ici qu'on a deux listes :

```
1 var1 = {'nom','âge'}  
2 var2 = {'Jacques',35}  
3 res= dict(zip(var1, var2))  
4 print(res)
```

Le résultat sera : {'nom' : 'Jacques', 'âge'=35}

La fonction len() :

De plus si on veut savoir combien d'associations sont présentes dans la variable « *var* », on peut utiliser la fonction `len()` :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}  
2 print(len(var))
```

La sortie de ce code est : 2

Maintenant, on va ajouter une association dans notre dictionnaire :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}  
2 var['lieu']= 'France'  
3 print(var)
```

La sortie de ce code est : {'nom': 'Jacques', 'age': 35, 'lieu': 'France'}

La méthode get()

Pour vérifier si une clé existe dans le dictionnaire : on peut utiliser la méthode get() :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}
2 print(var.get('prenom'))
```

La sortie de ce code est : None

Le résultat est None parce que la clé 'prenom' n'est pas dans le dictionnaire.

Si la « clé » existe dans le dictionnaire, il va mettre en résultat la valeur de cette clé :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}
2 print(var.get('nom'))
```

La sortie de ce code est : Jacques

La fonction update()

On peut aussi fusionner les dictionnaires en Python en utilisant la fonction update() .

```
1 var = {'nom' : 'Jacques', 'age' : 35}
2 var1= {'prenom' : 'Michel', 'lieu' : 'France'}
3 var.update(var1)
4 print(var)
```

La sortie de ce code est : {'nom': 'Jacques', 'age': 35, 'prenom': 'Michel', 'lieu': 'France'}

La fonction del() :

Et pour finir la fonction del () qui sert à supprimer une clé et sa valeur :

```
1 var = {'nom' : 'Jacques', 'âge' : 35}
2 var['sexe'] = 'fille ou garçon'
3 print(var)
4 del var['sexe']
5 print(var)
```

Les résultats sont :

```
{'nom': 'Jacques', 'age': 35, 'sexe': 'fille ou garçon'}
{'nom': 'Jacques', 'age': 35}
```

Exercice : Quiz

[solution n°3 p.23]

Question 1

Quel est le retour de ce script :

```
1 dictionnaire = dict([('pomme', 20), ('banane', 50), ('mangue', 30)])
2 print(dictionnaire)
```

- ☐ SyntaxError
- ☐ {"pomme": 20, "banane": 50, "mangue": 30}
- ☐ ("pomme": 20, "banane": 50, "mangue": 30)

Question 2

Quelle fonction vide le dictionnaire suivant :

```
etudiant = {"nom": "Maria", "couleur": "blanche", "pointure": 35}
```

- ☐ On ne peut pas le vider
- ☐ del etudiant
- ☐ etudiant.clear()

Question 3

Lesquels de ces dictionnaires renvoient des valeurs de type liste des clés, valeurs ou les deux :

- ☐ Keys ()
- ☐ Values ()
- ☐ for ()
- ☐ Items ()

Question 4

Quelle est la sortie de ce dictionnaire :

```
1 diction1 = {'nom' : 'Valérie' , 'travail' : 'Ingénieur'}
2 diction2 = {'âge' : 37, 'salaire' : 50000}
3 diction1.update(diction2)
```

- ☐ {"nom", "Valérie", "travail", "Ingénieur", "âge", 37, 50000}
- ☐ {'nom': 'Valérie', 'travail': 'Ingénieur', 'âge': 37, 'salaire': 50000}
- ☐ TypeError

Question 5

Dans ce code de suppression, laquelle des propositions peut être la sortie ?

```
1 diction3 = {'Valérie' , 'travail' : 'Ingénieur', 'âge' : 37, 'salaire' : 50000}
2 del diction3['âge']
```

- ☐ {"Valérie", "travail": "Ingénieur", "âge": 37, "salaire": 50000}
- ☐ {"Valérie", "travail": "Ingénieur", "salaire": 50000}
- ☐ {"Valérie", "travail": "Ingénieur", 37, "salaire": 50000}

VII. Ensembles

Définition

Les ensembles, ou sets, sont l'un des types de données en Python. Un ensemble permet de collecter les éléments et il n'est pas ordonné ; il n'a pas d'index et on ne peut pas avoir d'élément dupliqué. Il est utilisé pour supprimer les valeurs doublons surtout lorsqu'on combine deux données.

Pour l'utilisation, on emploie les accolades {} pour y mettre les éléments et les séparer par une virgule.

On peut ajouter n'importe quel nombre d'éléments et même de types différents, s'il s'agit de : int, float, string, tuple, etc.

On utilise la fonction `set()` si on veut créer un ensemble vide, car la syntaxe `{}` crée un dictionnaire vide mais pas un ensemble vide.

Pour mieux le comprendre, on va prendre quelques exemples.

Créons d'abord un ensemble.

```
1 ex1 = {5, 12.5, 'Paul'}
2 ex2 = {'Julien', 2, 3, 2, 5, 9, 7, 3}
3 print('le type est :', type(ex1))
4 print(ex1)
5 print(ex2)
```

Les sorties de ces codes sont :

```
le type est : <class 'set'>
{'Paul', 12.5, 5}
{2, 3, 5, 7, 'Julien', 9}
```

On voit ici qu'il ne peut pas y avoir d'éléments mutables.

On va créer un ensemble à partir d'une chaîne, ici les éléments ne seront pas ordonnés comme on les met en premier.

```
1 ex3 = set('45614894')
2 print(ex3)
```

La sortie de ce code est : `{'9', '5', '6', '4', '1', '8'}`

Les méthodes `add()` et `update()` :

L'exemple suivant va nous montrer la différence entre une liste et un ensemble sur l'ajout d'un élément. On ne peut pas accéder à un élément d'un ensemble, surtout on ne peut pas le modifier à l'aide de l'index. Si on veut ajouter un seul élément, on utilise la méthode `add()`. En revanche s'il s'agit de plusieurs éléments, on utilise la méthode `update()`.

```
1 ex1 = {5, 12.5, 'Paul'}
2 ex1.add(10)
3 ex1.update([20, 'Marc'], {40, 'Luc'})
4 print(ex1)
```

La sortie de ce code est : `{5, 40, 'Marc', 'Luc', 12.5, 'Paul', 20, 10}`

Les méthodes `discard()` et `remove()` :

Et maintenant, parlons de la suppression d'un élément d'un ensemble.

On utilise les méthodes `discard()` et `remove()`. On va voir dans l'exemple suivant ce qui différencie aussi les deux quand l'élément qu'on aimerait supprimer n'existe pas dans l'ensemble. Pour la méthode `discard()` le résultat est toujours le même, il reste inchangé. C'est à dire que si l'élément à supprimer n'est pas dans l'ensemble, le résultat sera l'ensemble déclaré initialement.

Prenons comme exemple l'ensemble qu'on a créé au-dessus :

```
1 ex1 = {5, 40, 'Marc', 'Luc', 12.5, 'Paul', 20, 10}
2 ex1.discard(40)
3 ex1.discard(100)
4 print(ex1)
```

Les résultats de ces codes sont :

```
{5, 'Paul', 'Marc', 10, 12.5, 20, 'Luc'}
{5, 40, 10, 12.5, 'Marc', 'Paul', 'Luc', 20}, puisque l'élément indiqué n'est pas dans le « ensemble », alors il n'y a pas de changement.
```

Mais lorsqu'on emploie la méthode `remove()`, il affichera une erreur comme ceci si l'élément n'existe pas dans l'ensemble.

```
1 ex1 = {5, 40, 'Marc', 'Luc', 12.5, 'Paul', 20, 10}
2 ex1.remove(100)
```

La sortie de ce code est : `KeyError: 100`

La méthode `union()` :

Maintenant, parlons de l'union dans les ensembles, ici par exemple on va déclarer deux ensembles, et on va afficher leur union en utilisant la méthode `union()`. On a « *ens1* » comme premier ensemble et « *ens2* » pour le deuxième, l'union de « *ens1* » et « *ens2* » est l'ensemble de tous les éléments de « *ens1* » et de tous les éléments de « *ens2* ». On note toujours que les éléments doublés ne seront inclus qu'une seule fois dans le résultat.

```
1 ens1= {10, 20, 30, 40, 50, 'dix', 'cinquante'}
2 ens2= {10, 20, 30, 'trente', 'zéro'}
3 ens3= ens1.union(ens2)
4 print(ens3)
```

La sortie de ce code est : `{'trente', 'dix', 'cinquante', 40, 10, 'zéro', 50, 20, 30}`

La méthode `intersection()` :

Abordons maintenant l'intersection, qui n'est que l'ensemble composé de tous les éléments communs de « *ens1* » et « *ens2* ». Cette fois-ci, on utilise la méthode `intersection()`.

```
1 ens1= {10, 20, 30, 40, 50, 'dix', 'cinquante'}
2 ens2= {10, 20, 30, 'trente', 'zéro'}
3 ens4 = ens1.intersection(ens2)
4 print(ens4)
```

La sortie de ce code est : `{10, 20, 30}`

Les méthodes `difference()` et `symmetric_difference()` :

Pour la nouvelle méthode, on utilise le `difference()` qui affiche les éléments de l'ensemble « *ens1* » qu'on ne trouve pas dans l'ensemble « *ens2* ».

```
1 ens1= {10, 20, 30, 40, 50, 'dix', 'cinquante'}
2 ens2= {10, 20, 30, 'trente', 'zéro'}
3 ens5= ens1.difference(ens2)
4 print(ens5)
```

La sortie de ce code est : `{40, 50, 'cinquante', 'dix'}`

Et si on veut afficher les éléments qui n'existent pas dans l'« *ens2* » dans un ensemble « *ens1* » et les éléments qui n'existent pas dans l'« *ens1* » dans un ensemble « *ens2* », on utilise la méthode `symmetric_difference()`.

```
1 ens1= {10, 20, 30, 40, 50, 'dix', 'cinquante'}
2 ens2= {10, 20, 30, 'trente', 'zéro'}
3 ens6 = ens1.symmetric_difference(ens2)
4 print(ens6)
```

Le résultat sera :

`{'zéro', 'trente', 40, 'cinquante', 50, 'dix'}`

La méthode `clear()` :

La méthode `clear()` est une méthode en Python qui permet de supprimer tous les éléments d'un ensemble. Voici un exemple simple pour illustrer son utilisation :

```
1 #Créer un ensemble
2 ens1= {1, 2, 3, 4, 5}
3
4 #Afficher l'ensemble avant l'utilisation de la méthode clear()
5 print(ens1)
6
7 #Supprimer tous les éléments de l'ensemble
8 ens1.clear()
```

```

9
10 #Afficher l'ensemble après l'utilisation de la méthode clear()
11 print(ens1)

```

Comme on peut le voir, la méthode `clear()` a supprimé tous les éléments de l'ensemble `ens1`. Avant l'utilisation de la méthode, l'ensemble contenait les éléments 1, 2, 3, 4 et 5. Après l'utilisation de la méthode, l'ensemble est vide comme le montre le résultat `{}` qui représente un ensemble vide en Python.

Exercice : Quiz

[solution n°4 p.24]

Question 1

Lesquelles de ces déclarations sont correctes pour le type ensemble :

- ☐ `ens1 = set('64821568464648')`
- ☐ `ex1.discard(40)`
- ☐ `ens2=set(['Sarah','Odile'])`
- ☐ `ens3=set(('Jean','Robert'))`

Question 2

Quelle est la sortie de ce code ?

```

1 ens1 = {5, 6, 'jean'}
2 ens1.add(10, 15)

```

- ☐ `{5, 6, 'jean', 10, 15}`
- ☐ `{5, 6, 'jean', 10}`
- ☐ `TypeError`

Question 3

Quel est le résultat de ce code ?

```

1 ens1 = {1, 5, 10, 'jaune', 'bleue'}
2 ens2 = {3, 4, 5, 6, 'vert', 'jaune', 'rouge', 'bleue'}
3 ens3= ens1.union(ens2)
4 print(ens3)

```

- ☐ `{1, 5, 6, 'vert', 10, 'jaune', 'bleue', 'rouge', 4, 5, 3, 'bleue', 'jaune'}`
- ☐ `{1, 5, 6, 'vert', 10, 'jaune', 'bleue', 'rouge', 4, 3, 'bleue'}`
- ☐ `{1, 5, 10, 'jaune', 'bleue'} {3, 4, 5, 6, 'vert', 'jaune', 'rouge', 'bleue'}`

Question 4

Quelle est la sortie de ce script ?

```

1 ens1 = {2, 4, 9, 'jaune', 'rouge'}
2 ens2 = {3, 5, 9, 'vert', 'rouge', 'bleue'}
3 ens3= ens2.difference(ens1)
4 print(ens3)

```

- ☐ `{9, 'rouge'}`
- ☐ `{'jaune', 2, 4}`
- ☐ `{3, 5, 'vert', 'bleue'}`

Question 5

Choisissez la bonne réponse pour le résultat de ce code :

```
1 ens1 = {2, 4, 9, 'bleue', 'rouge'}
2 ens2 = {4, 5, 9, 'vert', 'noir', 'bleue'}
3 ens3 = ens1.intersection(ens2)
```

- ☐ {2, 5, 'vert', 'noir', 'rouge'}
- ☐ {4, 9, 'bleue'}
- ☐ {2, 4, 5, 9, 'bleue', 'vert', 'rouge', 'vert', 'noir'}

IX. Essentiel

Nous avons évoqué quatre types de structures de données qui sont la liste, le tuple, le dictionnaire et l'ensemble en Python. Ces structures de données nous permettent de manipuler des données pendant une série de codage du programme en Python. La liste et le dictionnaire sont mutables car, au fil du temps, on peut modifier, ajouter des éléments. L'ensemble permet de collecter les éléments qui ne sont pas ordonnés ; il n'a pas d'index et on ne peut pas avoir d'élément dupliqué.

Donc, avant de coder ou de faire un programme, il est important d'analyser les données, ensuite, de faire le bon choix pour le type de structure de données qui correspond aux besoins et, enfin, d'appliquer ce que les types nous offrent.

Pour le cas de la liste, elle est fréquemment utilisée quand il s'agit de faire un programme en Python. Elle est riche en matière de méthodes ou fonctions, comme `append()`, `sort()`, `extend()`, `clear()`, `pop()`, `reverse()`, etc. Il est très important de les connaître par cœur afin de ne pas perdre de temps pendant le codage.

Concernant les tuples, même s'ils sont immuables, ils ont une très grande souplesse car ils peuvent contenir des éléments mutables afin qu'ils puissent évoluer pendant des exécutions.

Quant au dictionnaire, la plupart du temps, la clé de ce dernier est toujours en str ou chaîne de caractères mais elle n'est pas limitée car elle peut aussi être un tuple, ce qui est très pratique, par exemple, pendant la manipulation des coordonnées géométriques. Ce type de structure est également très utile quand on a des données très complexes à décrire.

Enfin, l'ensemble est utile pour supprimer les valeurs doublons surtout lorsqu'on combine deux données.

En Python, on dit que ces quatre types sont des containers, c'est-à-dire qu'il s'agit d'objets qui peuvent contenir une collection d'autres objets. On peut faire ou construire des listes qui contiennent des tuples, des dictionnaires ou même d'autres listes, mais également des dictionnaires contenant des listes, des tuples ; on peut déclarer des variables en ensemble à partir de ces trois types. On dit que les combinaisons sont infinies.

X. Auto-évaluation

A. Exercice

Supposons qu'on a une liste appelée `nombre` [], et qui a des valeurs : 12, 5, 6, 45, 1, 18.5, 18.1 et une autre avec des valeurs : Mario, Lydia, Marie, Franc, Michel, Andreas.

Question 1

[solution n°5 p.26]

Créez un script qui va mettre la première liste par ordre croissant.

Question 2

[solution n°6 p.26]

Créez un script pour que la deuxième liste sorte par ordre décroissant.

Supposons que nous voulons avoir le propos d'un étudiant qui a eu les félicitations pour être le premier de la classe. Créons donc une variable nommée « *etudiant* », et insérons les valeurs suivantes : « *Nom* », « *Prénom* », « *âge* », « *Num_matriculation* », les valeurs sont : « *Patricia* », « *Joséphine* », « *18* », « *78459* ».

Question 3

[solution n°7 p.26]

Créez un dictionnaire à partir de ces données.

Question 4

[solution n°8 p.26]

Créez des listes à partir de ces données.

Question 5

[solution n°9 p.27]

Créez un ensemble à partir du dictionnaire qu'on a créé au-dessus.

B. Test

Exercice 1 : Quiz

[solution n°10 p.27]

Question 1

Quel est le résultat de ce code ?

```
1 code= {'Nom' : 'David', 'sexe' : 'Garçon', 'âge' : 35}
2 print('type de ce code est :', type(code))
```

- ☐ type de ce code est : <class "set">
- ☐ type de ce code est : <class "dict">
- ☐ type de ce code est : <class "list">

Question 2

De quel type de déclaration s'agit-il : « `variable = {7, "Robert", "Francine", 15}` » ?

- ☐ Déclaration de type liste
- ☐ Déclaration de type dictionnaire
- ☐ Déclaration de type tuple
- ☐ Déclaration de type ensemble

Question 3

Laquelle est la réponse exacte pour cette déclaration : « `code= 5, 8, "Mark"` » ?

- ☐ Cette déclaration ne fonctionne pas, il faut avoir des `[]`, `()`, `{ }` qui englobent les valeurs
- ☐ Cette déclaration fonctionne et c'est un type tuple
- ☐ C'est une déclaration de type ensemble

Question 4

Quelle est la sortie d'une code de type `list()` ?

- ☐ `["var1", "var2"]`
- ☐ `("var1", "var2")`
- ☐ `{"var1", "var2"}`

Question 5


Quelle est la principale différence entre les listes et les tuples ?

- ☐ Les listes sont muables, les tuples sont immuables
- ☐ Les listes sont immuables, les tuples sont muables
- ☐ Il n'y a pas de différence

Solutions des exercices


Exercice p. 6 Solution n°1**Question 1**

Laquelle de ces déclarations est vraie pour la déclaration sur la liste :

- ☐ Lettres = {"pommes", "banane", "orange"}
- ☒ Lettres = ["pommes", "banane", "orange"]
- ☐ Lettres = ("pommes", "banane", "orange")
-  Pour déclarer une variable avec les listes, il faut les mettre entre les crochets.


Question 2

Laquelle de ces réponses est vraie si on déclare une variable liste = [5, 10, 15, 20, 30] et qu'on fait le print(liste[2]) ?

- ☐ 5
- ☐ 10
- ☒ 15
- ☐ 20
- ☐ 30
-  Le résultat est « 15 » parce que dans la liste on compte les éléments dans le tableau par l'index 0, l'index 1 et ainsi de suite, ici donc on a la valeur 5 de l'index 0, le 10 de l'index 1 et le 15 de l'index 2.


Question 3

Supposons qu'on a des valeurs entières dans une liste ; laquelle de ces déclarations est vraie :

- ☐ Nombre= ["15","20"]
- ☐ Nombre= (15 , 20)
- ☒ Nombre= [15, 20]
- ☐ Nombre= int ([15, 20])
-  La réponse correcte est « Nombre = [15,20] » car les valeurs sont dans les crochets et les valeurs ne sont pas entre des guillemets ce qui fait d'elles des entiers.


Question 4

Si nous voulons supprimer un élément dans une liste en pointant avec sa valeur, quelle fonction doit-on utiliser pour la suppression ?

- ☐ del()
- ☒ remove()
-  « del() » supprime l'élément de la liste avec l'indice.

Question 5


Pour insérer un élément entre les indices 0 et 1 de la liste spam= ["blanc", "rouge", "vert", "bleue"], quelle est la bonne manière de l'écrire ?

- ☐ spam= ["blanc", "rouge", "vert", "bleue"]
- ☐ a- spam.insert(0, "noir")
- ☒ b-spam.insert(1, "noir")
- ☐ c-spam.insert(2, "noir")
-  La réponse est b-spam.insert(1, "noir") car le noir va être inséré dans l'indice 1.

Exercice p. 8 Solution n°2


Question 1

Dans les déclarations suivantes, laquelle est une déclaration de type tuple ?

- ☐ tuple1=["rouge", "bleu", "noir"]
- ☒ tuple1=("rouge", "bleu", "noir")
- ☐ tuple1={"rouge", "bleu", "noir"}
-  Dans le tuple, on déclare une variable en mettant des crochets et guillemets si c'est une variable de type chaînes, et simplement des crochets si c'est un entier ; chaque valeur doit se séparer par une virgule.


Question 2

Après une déclaration de tuple, on a comme donnée : tuple=(1, 5, 20, 4). Si on fait le print(tuple[- 1 :: - 2]), laquelle de ces propositions s'affichera ?

- ☐ Index error
- ☐ 5
- ☐ 4
- ☒ (4, 5)
-  Ce script veut qu'on affiche le dernier élément du tableau et le deuxième car l'index « - » (moins) compte depuis le dernier élément et on compte par l'index 1 depuis le dernier, index 2 pour l'avant dernier et ainsi de suite.

Question 3


Supposons qu'on a une liste de type tuple : tuple1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). On lance la commande suivante : print(tuple1[2 : 4], tuple1[: 3], tuple1[5 :]), laquelle de ces propositions s'affichera ?

- ☐ (2, 4) (1, 3) (5,10)
- ☒ (3, 4) (1, 2, 3) (6, 7, 8, 9, 10)
-  Le tuple[2 : 4] veut qu'on affiche la valeur de l'index 2, ici c'est 3 et la valeur de l'index 4, c'est 4 car on compte par index zéro la valeur 1. Et la case vide indique tous les éléments avant ou à l'arrière, cela dépend de sa place.

Question 4

Choisissez la bonne réponse : laquelle va afficher la valeur 2 de cette déclaration tuple : tuple1 = ([1, 2, 3], 'chiffre', (4, 5, 6)).

- ☐ print(tuple1[0:3][1])
- ☐ print(tuple1[0:3](1))
- ☒ print(tuple1[0][1])


 On a comme premier indice la valeur qu'on a déclarée sur la première variable, et dans cette variable il y a encore des chaînes : on compte par l'indice 0 la première variable dedans.

Question 5

Quel est le résultat de ce script :

```
1 tuple = 'noir', 2, 'blanc'
2 a, b, c = tuple
3 print(a)
4
```

- ☐ ("noir",2,"blanc")
- ☐ SyntaxError
- ☒ noir


 Le « a » ici récupère la valeur « noir » et l'affiche.

Exercice p. 12 Solution n°3**Question 1**

Quel est le retour de ce script :

```
1 dictionnaire = dict([('pomme', 20), ('banane', 50), ('mangue', 30)])
2 print(dictionnaire)
```

- ☐ SyntaxError
- ☒ {"pomme": 20, "banane": 50, "mangue": 30}
- ☐ ("pomme": 20, "banane": 50, "mangue": 30)

 Le retour en dictionnaire est entre accolades.

Question 2

Quelle fonction vide le dictionnaire suivant :

```
etudiant = {'nom': 'Maria', 'couleur': 'blanche', 'pointure': 35}
```

- ☐ On ne peut pas le vider
- ☒ del etudiant
- ☐ etudiant.clear()

Q La bonne réponse est « *del etudiant* » car la fonction *del* existe vraiment dans le dictionnaire.

Question 3

Lesquels de ces dictionnaires renvoient des valeurs de type liste des clés, valeurs ou les deux :

- ☒ Keys ()
- ☒ Values ()
- ☐ for ()
- ☒ Items ()

Q Les valeurs renvoyées par ces méthodes ne sont pas de vraies listes parce qu'elles ne peuvent pas être modifiées et ne disposent pas de la méthode *append()*.

Question 4

Quelle est la sortie de ce dictionnaire :

```
1 diction1 = {'nom' : 'Valérie' , 'travail' : 'Ingénieur'}
2 diction2 = {'âge' : 37, 'salaire' : 50000}
3 diction1.update(diction2)
```

- ☐ {"nom", "Valérie", "travail", "Ingénieur", "âge", 37, 50000}
- ☒ {'nom': 'Valérie', 'travail': 'Ingénieur', 'âge': 37, 'salaire': 50000}
- ☐ TypeError

Q La méthode *update()* sert à combiner les deux dictionnaires.

Question 5

Dans ce code de suppression, laquelle des propositions peut être la sortie ?

```
1 diction3 = {'Valérie' , 'travail' : 'Ingénieur', 'âge' : 37, 'salaire' : 50000}
2 del diction3['âge']
```

- ☐ {"Valérie", "travail" : "Ingénieur", "âge" : 37, "salaire" : 50000}
- ☒ {"Valérie", "travail" : "Ingénieur", "salaire" : 50000}
- ☐ {"Valérie", "travail" : "Ingénieur", 37, "salaire" : 50000}


Q La réponse est « {"Valérie", "travail" : "Ingénieur", "salaire" : 50000} » car la méthode permet de supprimer le « clé » et sa valeur avec.

Exercice p. 16 Solution n°4

Question 1

Lesquelles de ces déclarations sont correctes pour le type ensemble :

- ☒ `ens1 = set("64821568464648")`
- ☐ `ex1.discard(40)`
- ☒ `ens2=set(["Sarah", "Odile"])`
- ☒ `ens3=set(("Jean", "Robert"))`


 La première proposition est tout simplement un ensemble, il y a une présence de set, la troisième proposition est la même mais à partir d'une liste, et la quatrième proposition est une déclaration d'un ensemble à partir d'un tuple.

Question 2

Quelle est la sortie de ce code ?

```
1 ens1 = {5, 6, 'jean'}
2 ens1.add(10, 15)
```

- ☐ `{5, 6, "jean", 10, 15}`
- ☐ `{5, 6, "jean", 10}`
- ☒ `TypeError`


 La méthode `add()` sert à insérer un seul élément, dans cette commande, si on veut insérer ces deux éléments, on emploie la méthode `update()`.

Question 3

Quel est le résultat de ce code ?

```
1 ens1 = {1, 5, 10, 'jaune', 'bleue'}
2 ens2 = {3, 4, 5, 6, 'vert', 'jaune', 'rouge', 'bleue'}
3 ens3= ens1.union(ens2)
4 print(ens3)
```

- ☐ `{1, 5, 6, "vert", 10, "jaune", "bleue", "rouge", 4, 5, 3, "bleue", "jaune"}`
- ☒ `{1, 5, 6, "vert", 10, "jaune", "bleue", "rouge", 4, 3, "bleue"}`
- ☐ `{1, 5, 10, "jaune", "bleue"} {3, 4, 5, 6, "vert", "jaune", "rouge", "bleue"}`

 La méthode `union()` sert à combiner les deux ensembles et ne prend qu'une seule fois la valeur doublante.

Question 4

Quelle est la sortie de ce script ?

```
1 ens1 = {2, 4, 9, 'jaune', 'rouge'}
2 ens2 = {3, 5, 9, 'vert', 'rouge', 'bleue'}
3 ens3= ens2.difference(ens1)
4 print(ens3)
```

- ☐ `{9, 'rouge'}`
- ☐ `{"jaune", 2, 4}`
- ☒ `{3, 5, "vert", "bleue"}`

Q La méthode `difference()` sert à détecter les éléments qui ne sont pas identiques. Ici, on demande d'afficher les éléments dans « *ens2* » qui ne sont pas dans « *ens1* ».

Question 5

Choisissez la bonne réponse pour le résultat de ce code :

```
1 ens1 = {2, 4, 9, 'bleue', 'rouge'}
2 ens2 = {4, 5, 9, 'vert', 'noir', 'bleue'}
3 ens3 = ens1.intersection(ens2)
```

- ☐ {2, 5, "vert", "noir", "rouge"}
- ☒ {4, 9, 'bleue'}
- ☐ {2, 4, 5, 9, "bleue", "vert", "rouge", "vert", "noir"}

Q La méthode `intersection()` sert à connaître les éléments identiques dans les ensembles.

p. 17 Solution n°5

On doit utiliser la méthode `sort()`.

```
1 nombres = [12, 5, 6, 45, 1, 18.5, 18.1]
2 nombres.sort()
3 print(nombres)
```

Résultat :

```
[1, 5, 6, 12, 18.1, 18.5, 45]
```

p. 17 Solution n°6

Utilisez la méthode `sort (reverse=True)`

```
lettres = [ 'Mario', 'Lydia', 'Marie', 'Franc', 'Michel', 'Andreas' ]
1 lettres.sort(reverse=True)
2 print(lettre)
```

Résultat :

```
['Michel', 'Mario', 'Marie', 'Lydia', 'Franc', 'Andreas']
```

p. 18 Solution n°7

```
1 etudiant = {'Nom' : 'Patricia', 'Prénom' : 'Joséphine', 'âge' :18, 'Num_matriculation'=78459}
```

p. 18 Solution n°8

```
1 etudiant1= ['Nom', 'Prénom', 'âge', 'Num_matriculation']
2 etudiant2=['Patricia', 'Joséphine', 18, 78459]
```

p. 18 Solution n°9


```
1 etudiant = set({'Nom' : 'Patricia', 'Prénom' : 'Joséphine', 'âge' : 18,  
  'Num_matriculation' : 78459})
```

Exercice p. 18 Solution n°10

Question 1


Quel est le résultat de ce code ?

```
1 code= {'Nom' : 'David', 'sexe' : 'Garçon', 'âge' : 35}  
2 print('type de ce code est :', type(code))
```

- ☐ type de ce code est : <class "set">
- ☒ type de ce code est : <class "dict">
- ☐ type de ce code est : <class "list">
-  Il s'agit d'un dictionnaire, il y a des accolades et des « : » qui séparent les clés et les valeurs.


Question 2

De quel type de déclaration s'agit-il : « *variable* = {7, "Robert", "Francine", 15} » ?

- ☐ Déclaration de type liste
- ☐ Déclaration de type dictionnaire
- ☐ Déclaration de type tuple
- ☒ Déclaration de type ensemble
-  Il y a des accolades qui englobent les valeurs, elles sont séparées par des virgules et on ne voit pas de « : » qui séparent les clés et les valeurs.

Question 3

Laquelle est la réponse exacte pour cette déclaration : « *code* = 5, 8, "Mark" » ?

- ☐ Cette déclaration ne fonctionne pas, il faut avoir des [], (), { } qui englobent les valeurs
- ☒ Cette déclaration fonctionne et c'est un type tuple
- ☐ C'est une déclaration de type ensemble
-  Il s'agit d'une déclaration tuple même s'il n'y a pas de parenthèses.

Question 4

Quelle est la sortie d'une code de type list() ?

- ☒ ["var1", "var2"]
- ☐ ("var1", "var2")
- ☐ {"var1", "var2"}

Q Quand on exécute un code de type `list()`, le résultat se met entre crochets.

Question 5

Quelle est la principale différence entre les listes et les tuples ?

- ☒ Les listes sont muables, les tuples sont immuables
- ☐ Les listes sont immuables, les tuples sont muables
- ☐ Il n'y a pas de différence
- Q Le tuple n'est pas modifiable alors que la liste est modifiable.