Les listes

Les listes

En algorithmique, le terme **"tableau"** fait référence à une structure de données qui permet de stocker et d'organiser une collection ordonnée d'éléments sous un même nom.

Un tableau est une séquence d'éléments, où chaque élément est identifié par un index ou une clé. Voici une définition générale des tableaux en algorithmique :

En général, **les indices d'un tableau commencent à zéro**, c'est-à-dire que **le premier élément est à l'index 0**, **le deuxième à l'index 1**, et ainsi de suite.

Cette **convention est courante dans de nombreux langages de programmation** et en algorithmique.

INDEX	0	1	2	3	4
VALEURS	“H”	“E”	“L”	“L”	“O”

Les listes

En Python, **les listes sont des structures de données très flexibles et polyvalentes.**
C'est un des 4 types de données qui servent à regrouper un ensemble de données.

Voici quelques opérations courantes que vous pouvez effectuer sur les listes en Python.

Pour créer une liste, rien de plus simple:

```
>>> liste = []
```

Vous pouvez voir le contenu de la liste en l'appelant comme ceci:

```
>>> liste
    []
```

Les listes : ajouter une valeur à une liste python

On dit d'une liste qu'elle est ordonnée, car ses éléments ont un ordre. Chaque nouvel élément sera ajouté, par défaut, à la fin de la liste.
Vous pouvez **ajouter les valeurs** que vous voulez lors de la création de la liste python :

```
>>> liste = [1,2,3]
>>> liste
    [1, 2, 3]
```

Ou les ajouter après la création de la liste avec la méthode append (qui signifie "ajouter" en anglais):

```
>>> liste = []
>>> liste.append(1)
>>> liste.append("ok")
>>> liste
    [1, "ok"]
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

Les listes : Afficher un item d'une liste

Pour lire une liste, on peut demander à voir l'index de la valeur qui nous intéresse:

```
>>> liste = ["a","d","m"]
>>> liste[0]
    'a'
>>> liste[2]
    'm'
```

Le premier item commence toujours avec l'index 0. Pour lire le premier item on utilise la valeur 0, le deuxième on utilise la valeur 1, etc. Il est d'ailleurs possible de modifier une valeur avec son index

```
>>> liste = ["a","d","m"]
>>> liste[2] = "z"
>>> liste
    ['a', 'd', 'z']
```

Les listes : Supprimer une entrée avec un index

Il est parfois nécessaire de supprimer une entrée de la liste. Pour cela vous pouvez utiliser la fonction del qui va utiliser l'index d'une valeur pour la supprimer.

```
>>> liste = ["a", "b", "c"]
>>> del liste[1]
>>> liste
    ['a', 'c']
```

Les listes : Supprimer une entrée avec sa valeur

Il est possible de supprimer directement la valeur d'un index grâce à la méthode remove. Mais attention, si vous avez plusieurs occurrences de la valeur à retirer, seule la première sera supprimée.

```
>>> liste = ["a", "b", "c"]
>>> liste.remove("a")
>>> liste
    ['b', 'c']
```

Les listes : Inverser les valeurs d'une liste

Vous pouvez inverser les items d'une liste avec la méthode reverse .

```
>>> liste = ["a", "b", "c"]
>>> liste.reverse()
>>> liste
['c', 'b', 'a']
```

Essayez avec les connaissances apprises jusqu'ici, de prendre une chaîne de caractères, la transformer en liste, inverser l'ordre de la liste, puis de reformer une chaine avec.

Les listes : Compter le nombre d'items d'une liste

Il est possible de compter le nombre d'items d'une liste avec la fonction len .

```
>>> liste = [1,2,3,5,10]
>>> len(liste)
5
```

Les listes : Compter le nombre d'occurences d'une valeur

Pour connaître le nombre d'occurences d'une valeur dans une liste, vous pouvez utiliser la méthode count .

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.count("a")
3
>>> liste.count("c")
2
```

Les listes : Trouver l'index d'une valeur

La méthode index vous permet de connaître la position de l'item recherché.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

Les listes : manipuler une liste

Voici quelques astuces pour manipuler des listes:

```
>>> liste = [1, 10, 100, 250, 500]

>>> liste[0]
    1

>>> liste[-1] # Cherche la dernière occurence
    500

>>> liste[-4:] # Affiche les 4 dernières occurrences
    [500, 250, 100, 10]

>>> liste[:] # Affiche toutes les occurences
    [1, 10, 100, 250, 500]

>>> liste[2:4] = [69, 70]
    [1, 10, 69, 70, 500]

>>> liste[:] = [] # vide la liste
    []
```

Les listes : copier une liste

Beaucoup de débutants font l'erreur de copier une liste de cette manière

```
>>> x = [1,2,3]
>>> y = x
```

Or si vous changez une valeur de la liste y , la liste x sera elle aussi affectée par cette modification:

```
>>> x = [1,2,3]
>>> y = x
>>> y[0] = 4
>>> x
[4, 2, 3]
```

Les listes : copier une liste

Cette syntaxe permet de travailler sur un même élément nommé différemment. Alors comment copier une liste qui sera indépendante?

```
>>> x = [1,2,3]
>>> y = x[:]
>>> y[0] = 9
>>> x
    [1, 2, 3]
>>> y
    [9, 2, 3]
```

Pour des données plus complexes, vous pouvez utiliser la fonction deepcopy du module copy

```
>>> import copy
>>> x = [[1,2], 2]
>>> y = copy.deepcopy(x)
>>> y[1] = [1,2,3]
>>> x
[[1, 2], 2]
>>> y
[[1, 2], [1, 2, 3]]
```

Les listes : Transformer une liste en string

L'inverse est possible avec la méthode " join ".

```
>>> liste = ["Olivier","ENGEL","Strasbourg"]
>>> ":".join(liste)
'Olivier:ENGEL:Strasbourg'
```

Les listes : Transformer une string en liste

Parfois il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode split .

```
>>> ma_chaine = "Olivier:ENGEL:Strasbourg"
>>> ma_chaine.split(":")
['Olivier', 'ENGEL', 'Strasbourg']
```

Les listes : trouver un item dans une liste

Pour **savoir si un élément est dans une liste**, vous pouvez **utiliser le mot clé in** de cette manière:

```
>>> liste = [1,2,3,5,10]
>>> 3 in liste
True
>>> 11 in liste
False
```

Les listes : la fonction range

La fonction range génère une liste composée d'une simple suite arithmétique.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Les listes : agrandir une liste par une liste

Pour mettre bout à bout deux listes, ou une liste et un tuple, vous pouvez utiliser la méthode extend.

```
>>> x = [1, 2, 3, 4]
>>> y = [4, 5, 1, 0]
>>> x.extend(y)
>>> print x
[1, 2, 3, 4, 4, 5, 1, 0]
```

Les listes : permutations

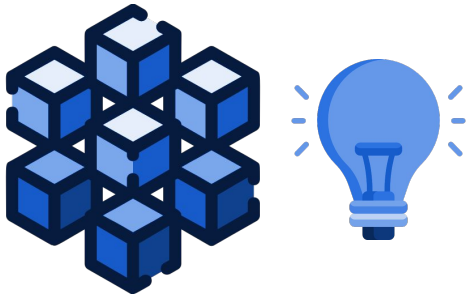
La permutation d'un ensemble d'éléments est **une liste de tous les cas possibles**. Si vous avez besoin de cette fonctionnalité, inutile de réinventer la roue, itertools s'en occupe pour vous.

```
>>> from itertools import permutations
>>> list(permutations(['a', 'b', 'c']))
[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'),
('c', 'a', 'b'), ('c', 'b', 'a')]
```

Les listes : Permutation d'une liste de liste

Comment afficher tous les cas possibles d'une liste elle-même composée de listes ?
Avec l'outil **product de itertools** :

```
>>> from itertools import product
>>> list(product(['a', 'b'], ['c', 'd']))
[('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd')]
```



Les listes : astuces

Les listes : Astuces

Afficher les 2 premiers éléments d'une liste

```
>>> liste = [1,2,3,4,5]
>>> liste[:2]
[1, 2]
```

Afficher le dernier item d'une liste:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-1]
6
```

Afficher le 3e élément en partant de la fin:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3]
4
```

Afficher les 3 derniers éléments d'une liste:

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3:]
[4, 5, 6]
```

Les listes : Astuces

Vous pouvez additionner deux listes pour les combiner ensemble en utilisant l'opérateur + :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y
[1, 2, 3, 4, 5, 6]
```

Vous pouvez même multiplier une liste:

```
>>> x = [1, 2]
>>> x*5
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

Ce qui peut être utile pour initialiser une liste:

```
>>> [0] * 5
[0, 0, 0, 0, 0]
```

Les tuples

Les Tuples : introduction

La principale différence entre les tuples et les listes est que **les tuples sont des objets immuables**, tandis que les listes sont mutables. Cela signifie que **les tuples ne peuvent pas être modifiés** alors que les listes peuvent l'être. Les tuples sont **plus efficaces en termes de mémoire que les listes**.

Déclaration d'un tuple :

```
>>> furniture = ('table', 'chair', 'rack', 'shelf')
```

Accéder à une valeur par son index :

```
>>> furniture[0]
# 'table'
```

Accéder à une tranche (slice) du tuple :

```
>>> furniture[1:3]
# ('chair', 'rack')
```

Récupérer le nombre d'éléments d'un tuple :

```
>>> len(furniture)
# 4
```

Les Tuples : Conversion

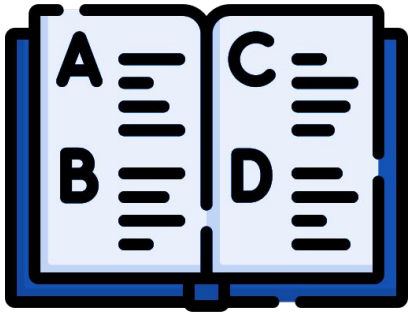
Conversion entre **list()** et **tuple()**

Tuple à partir d'une liste :

```
>>> tuple(['cat', 'dog', 5])
# ('cat', 'dog', 5)
```

Liste à partir d'un tuple :

```
>>> list(('cat', 'dog', 5))
# ['cat', 'dog', 5]
```



Les dictionnaires

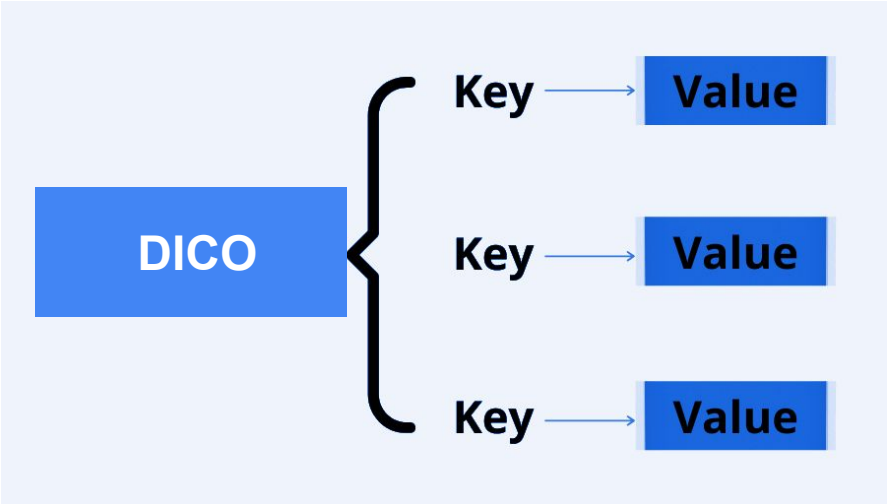
Les Dictionnaires : Introduction

Un dictionnaire est une structure de données qui permet de stocker des paires clé-valeur. Contrairement aux listes qui sont indexées par des nombres entiers, les dictionnaires utilisent des clés pour accéder à leurs valeurs associées.

Il est **défini par des accolades {}** et les **paires clé-valeur sont séparées par des virgules**.

Par exemple, un dictionnaire représentant des informations sur une personne pourrait ressembler à ceci :

```
personne = {
    'nom': 'Doe',
    'prénom': 'John',
    'âge': 30,
    'ville': 'Paris'
}
```



Les Dictionnaires : comment créer un dictionnaire ?

Pour initialiser un dictionnaire , on utilise la syntaxe suivante:

```
>>> a = {}
```

ou

```
>>> a = dict()
```

Les Dictionnaires : comment ajouter des valeurs ?

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur:

```
>>> a = {}
>>> a["nom"] = "Banner"
>>> a["prenom"] = "Bruce"
>>> a
{'nom': ‘Banner’, 'prenom': 'Bruce'}
```

Vous pouvez utiliser des clés numériques comme dans la logique des listes .

Les Dictionnaires : Comment récupérer une valeur ?

La méthode **get** vous permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, vous pouvez donner une valeur à retourner par défaut:

```
>>> data = {"name": "Banner", "age": 45}
>>> data.get("name")
'Banner'
>>> data.get("adresse", "Adresse inconnue")
'Adresse inconnue'
```

Les Dictionnaires : Vérifier la présence d'une clé ?

Vous pouvez utiliser la méthode **has_key()** pour vérifier la présence d'une clé que vous cherchez (python < v3.0) :

```
>>> a.has_key("name")
True
```

Sinon utiliser simplement **in** pour les versions à partir de 3.0 :

```
>>> "name" in data
True
```

Les Dictionnaires : supprimer une entrée ?

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes:

```
>>> del a["nom"]
>>> a
{'prenom': 'Bruce'}
```

Les Dictionnaires : récupérer les clés ?

Pour récupérer les clés on utilise la méthode keys .

```
>>> fiche = {"nom":"Banner", "prenom":"Bruce"}
>>> for cle in fiche.keys():
...     print cle
...
nom
prenom
```

Les Dictionnaires : créer une copie indépendante ?

Comme pour toute variable, vous ne pouvez pas copier un dictionnaire en faisant dic1 = dic2 :

```
>>> d = {"k1":"Bruce", "k2":"Banner"}
>>> e = d
>>> d["k1"] = "Hulk"
>>> e
{'k2': 'Banner', 'k1': 'Hulk'}
```

Pour créer une copie indépendante vous pouvez utiliser la méthode **copy** :

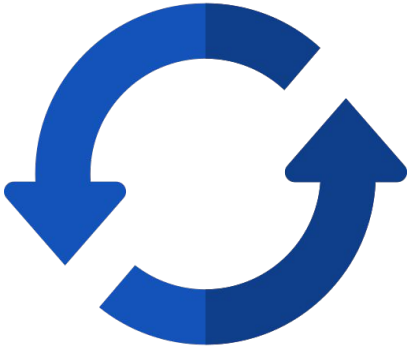
```
>>> d = {"k1":"Bruce", "k2":"Banner"}
>>> e = d.copy()
>>> d["k1"] = "Hulk"
>>> e
{'k2': 'Banner', 'k1': 'Bruce'}
```

Les Dictionnaires : comment fusionner ?

La méthode **update** permet de fusionner deux dictionnaires .

```
>>> a = {'nom': 'Banner'}
>>> b = {'prenom': 'bruce'}

>>> a.update(b)
>>> print(a)
{'nom': 'Banner', 'prenom': 'Bruce'}
```



Les boucles

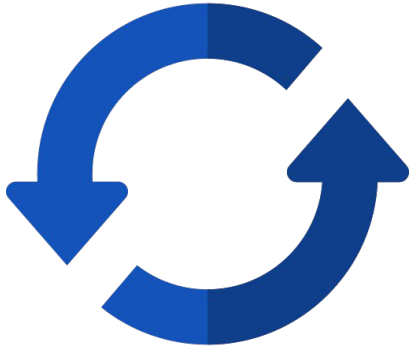
Les boucles : Introduction

En programmation, une boucle est une structure qui permet d'exécuter un ensemble d'instructions de manière répétée tant qu'une condition spécifiée est vraie. En Python, on utilise généralement deux types de boucles : la boucle "**for**" et la boucle "**while**".

Les boucles permettent de répéter des actions simplement et rapidement. Une boucle peut être vue comme une version informatique de **« faire X fois quelque chose »**.

Il y a **différents types de boucles** mais **elles se ressemblent toutes au sens où elles répètent une action** un certain nombre de fois.

Les différents types de boucles permettent d'utiliser différentes façon de commencer et de terminer une boucle. **Chaque type de boucle pourra être utilisé en fonction de la situation** et du problème que l'on cherche à résoudre.



Les boucles

Voici un exemple simple d'utilisation d'une boucle en pseudo-code :

```
VARIABLES
 tab EST_DU_TYPE LISTE
 i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
 tab[1] PREND_LA_VALEUR 1
 tab[2] PREND_LA_VALEUR 5
 tab[3] PREND_LA_VALEUR 2
 POUR i ALLANT_DE 1 A 3
   DEBUT_POUR
   AFFICHER* tab[i]
   FIN_POUR
FIN_ALGORITHME
```

Les boucles : While

En anglais " while " signifie "Tant que". Pour créer une boucle , il faut donc utiliser ce mot clé suivi d'une indication qui dit quand la boucle s'arrête.
Un exemple sera plus parlant:

On désire **écrire 100 fois** cette phrase:

"Je ne dois pas poser une question sans lever la main"

Ecrire à la main prend beaucoup de temps et beaucoup de temps x 100 c'est vraiment beaucoup de temps, et peu fiable, même pour les chanceux qui connaissent le copier-coller. Et un bon programmeur est toujours un peu fainéant et perfectionniste, il cherchera la manière la plus élégante de ne pas répéter du code.

```
>>> i = 0
>>> while i < 100:
...     print("Je ne dois pas poser une question sans lever la main")
...     i = i +1
...
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
```

Les boucles : For

La boucle for permet de **faire des itérations sur un élément**, comme une chaîne de caractères par exemple ou une liste, un **tuple**, un **dictionnaire**, un **ensemble** ou une **chaîne** :

```
>>> pets = ['Bella', 'Milo', 'Loki']
>>> for pet in pets:
...     print(pet)
...
# Bella
# Milo
# Loki


>>> for i in range(5):
...     print(f'La boucle s'arrête à 5! ou 4? ({i})')
```

```
>>> v = "Bonjour toi"
>>> for lettre in v:
...     print(lettre)
...
B
o
n
j
o
u
r

t
o
i
```

Les boucles : Range

Il est possible de créer une boucle facilement avec **range**, qui renvoie une séquence de nombre :

```
for i in range(0,100):
  print(i)
```

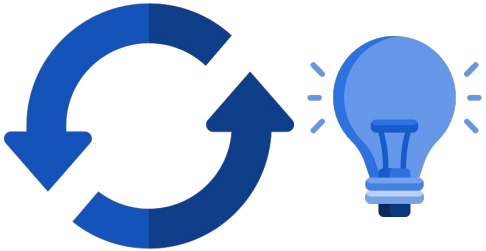
On peut aussi vouloir itérer non pas par unité mais par incrément de 10 par exemple :

```
for i in range(0,100,10):
  print(i)
```

Les boucles : Stopper une boucle avec break

Pour stopper immédiatement une boucle on peut utiliser le mot clé break :

```
>>> liste = [1,5,10,15,20,25]
>>> for i in liste:
...     if i > 15:
...             print("On stoppe la boucle")
...             break
...     print(i)
...
1
5
10
15
On stoppe la boucle
```



Les boucles & listes : astuces

Les boucles : utiliser des boucles **for** avec des **listes**

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> for item in furniture:
...     print(item)

# table
# chair
# rack
# shelf
```

Les boucles : boucler dans plusieurs **listes** avec **zip**

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> price = [100, 50, 80, 40]

>>> for item, amount in zip(furniture, price):
...     print(f'The {item} costs ${amount}')

# The table costs $100
# The chair costs $50
# The rack costs $80
# The shelf costs $40
```

[Listes, Tuples & Boucles]
Workshop

Réaliser les exercices

Réalisez les exercices suivants (vous n'avez pas le droit d'utiliser la création de fonction ou toutes autres possibilités algorithmique de python hormis les variables, conditions, les listes, les tuples et les boucles et tout ce qui peut s'y associer).

L'intégralité de vos exercices seront à faire avec **Visual Studio Code** dans un dossier **boucles** avec à l'intérieur un fichier par exercice par exemple :

/boucles
—--- exo1.py
—--- exo2.py
....

Exercices disponible ici.

Récupérer le Cours

Scannez moi