PROGRAMMATION PYTHON

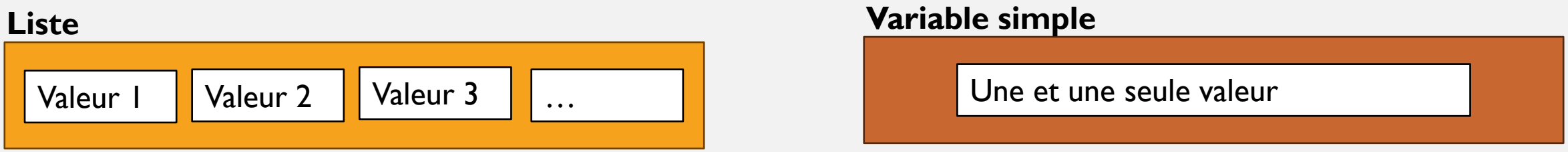
Chapitre 9: Les listes

Sommaire

Définition

Les listes sont des séquences, ce sont des objets spéciaux capable de contenir des valeurs de n'importe quel type.

Contrairement aux variables simples qui ne peuvent contenir qu'une seule valeur, les listes peuvent contenir plusieurs valeurs.



Une liste est un type de variable qui permet d'associer plusieurs valeurs différentes ayant la même étiquette. Les valeurs 1, 2 et 3 précédentes seront représentées par le même nom de variable.

On peut imaginer une liste comme une étagère avec plusieurs tiroirs numérotés, chaque tiroir peut contenir un élément différent, le tiroir 1 peut contenir une brosse et le tiroir 2 un stylo. Ces deux éléments totalement différents sont stockés dans un même objet qui est l'étagère.

Une liste peut contenir plusieurs nombres entiers ou uniquement des nombres à virgule ou uniquement des chaines de caractères ou un mélange de tout. Une liste peut contenir une autre liste.

Création de listes

Python nous offre deux possibilités pour créer des listes soit on utilise le mot clé **list** ou soit on crée la liste avec une liste vide.

```
>>> # Méthode avec le mot clé list
>>> maListe = list()
>>> type(maListe)
    <class 'list'>
>>> maListe
    []
```

Equivalent à

```
>>> # Méthode avec une liste vide
>>> maListe = []
>>> type(maListe)
    <class 'list'>
>>> maListe
    []
```

Dans les deux exemples précédents, on crée une liste vide. On peut créer une liste non vide en spécifiant des éléments de départ lors de la création.

```
>>> # Liste avec des éléménts de départ
>>> maListe = [1, 'a', True, 2.0]
>>> type(maListe)
    <class 'list'>
>>> maListe
    [1, 'a', True, 2.0]
```

Création de listes

Chaque élément d'une liste est identifié par un numéro appelé indice et celui-ci est assigné à chaque élément lors de l'insertion dans la liste. Le premier élément aura pour indice 0, le suivant indice 1….

Pour accéder à un élément d'une liste, on utilise le format de code suivant **nom_liste[indice].**

```
>>> # Création d'une liste
>>> maListe = [1, 'a', True, [2, 3], 2.0]
>>> # Affichage du premier élément de la liste et son type
>>> maListe[0]
1
>>> type(maListe[0])
<class 'int'>
>>> # Affichage du deuxième élément de la liste et son type
>>> maListe[1]
'a'
>>> type(maListe[1])
<class 'str'>
>>> # Affichage du troisième élément de la liste et son type
>>> maListe[2]
True
>>> type(maListe[2])
<class 'bool'>
>>> # Affichage du quatrième élément de la liste et son type
>>> maListe[3]
[2, 3]
>>> type(maListe[3])
<class 'list'>
>>> # Affichage du dernier élément de la liste et son type
>>> maListe[4]
2.0
>>> type(maListe[4])
<class 'float'>
```

On peut assigner à une autre variable la valeur d'un élément de la liste.

Insertion d'éléments dans une liste

Pour insérer un élément dans une liste, Python nous offre plusieurs fonctions dont les principales sont:

- **insert:** qui permet d'insérer un élément à un endroit précis dans la liste. Cette technique entraine un décalage des indices des autres éléments.
- **append:** qui permet d'insérer un élément à la fin de la liste.

```
>>> # Création d'une liste avec des éléments
>>> maListe = [1, 'a', True, [2, 3], 2.0]
>>> # Affichage de l'élément à l'indice 2
>>> maListe[2]
True
>>> # Insertion d'un élément à l'indice 2
>>> maListe.insert(2, 'John')
>>> maListe
[1, 'a', 'John', True, [2, 3], 2.0]
>>> # Affichage de l'élément à l'indice 2
>>> maListe[2]
'John'
```

```
>>> # Création d'une liste avec des éléments
>>> maListe = [1, 'a', True, [2, 3], 2.0]
>>> # Affichage de la liste
>>> maListe
[1, 'a', True, [2, 3], 2.0]
>>> # Ajout d'un élément à la fin de la liste avec append
>>> maListe.append("John")
>>> # Affichage de la liste
>>> maListe
[1, 'a', True, [2, 3], 2.0, 'John']
```

On peut remplacer un élément dans une liste en assignant une nouvelle valeur à l'indice de l'élément à changer.

```
>>> # Création d'une liste
>>> maListe = [1, 'a', True, [2, 3], 2.0]
>>> maListe
[1, 'a', True, [2, 3], 2.0]
>>> # Pour changer la valeur de l'élément 'a' à l'indice 1
>>> maListe[1] = 2
>>> maListe
[1, 2, True, [2, 3], 2.0]
```

Insertion d'éléments dans une liste

On peut agrandir une liste en lui associant les valeurs d'une autre liste. Cette technique s'appelle de la **concaténation.**

Pour faire de la concaténation de deux listes on peut utiliser la méthode **extend** de Python ou utiliser **+=.**

```
>>> # Création d'une liste 1
>>> liste1 =[1, 2]
>>> liste1
    [1, 2]
>>> # Création d'une liste 2
>>> liste2 = [2, 3]
>>> liste2
    [2, 3]
>>> # Ajout des éléments de la liste 2 à la fin de la liste 1
>>> liste1 += liste2 #=> équivaut à liste1 = liste1 + liste2
>>> liste1
    [1, 2, 2, 3]
```

Equivalent à

```
>>> # Création d'une liste 1
>>> liste1 =[1, 2]
>>> liste1
    [1, 2]
>>> # Création d'une liste 2
>>> liste2 = [2, 3]
>>> liste2
    [2, 3]
>>> # Ajout des éléments de la liste 2 à la fin de la liste 1
>>> liste1.extend(liste2)
>>> liste1
    [1, 2, 2, 3]
```

Lorsqu'on veut créer une nouvelle liste qui est la concaténation de deux listes il faut utiliser l'opérateur +.

```
>>> # Création d'une liste 1
>>> liste1 =[1, 2]
>>> liste1
    [1, 2]
>>> # Création d'une liste 2
>>> liste2 = [2, 3]
>>> liste2
    [2, 3]
>>> # Création d'une liste 3 qui est la liste 1 suivant des éléments de la liste 2
>>> liste3 = liste1 + liste2
>>> liste3
    [1, 2, 2, 3]
```

Suppression d'éléments dans une liste

Python nous offre deux méthodes pour supprimer un élément dans une liste soit avec le mot clé ***del*** ou ***remove.***

Le mot clé ***del*** permet de supprimer une variable ou une valeur d'une liste. La syntaxe pour utiliser ce mot clé est:

del variable_à_***supprimer*** **(pour supprimer une variable)**

del nom_liste[indice] **(pour supprimer un élément dans une liste)**

```
>>> # Création d'une variable
>>> msg = "Bonjour !"
>>> msg
'Bonjour !'
>>> # Suppression de la variable
>>> del msg
>>> msg
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    msg
NameError: name 'msg' is not defined
>>>
```

```
>>> # Création d'une liste
>>> liste = [1, 'a', 2]
>>> liste
[1, 'a', 2]
>>> len(liste)
3
>>> # Suppression de l'élément à l'indice 1
>>> del liste[1]
>>> liste
[1, 2]
>>> len(liste)
2
```

Une erreur est générée lorsqu'on essaie de supprimer avec *del* une valeur dont l'indice n'existe pas dans la liste.

Suppression d'éléments dans une liste

La méthode ***remove*** contrairement à la méthode ***del***, permet de supprimer un élément dans une liste à partir de sa valeur non pas par son indice. La syntaxe pour utiliser ***remove*** est:

$$nom\_de\_la\_liste.remove(valeur\_a\_supprimer)$$

```
>>> # Création d'une liste
>>> liste = [1, 'a', 2, 'a', 3]
>>> liste
[1, 'a', 2, 'a', 3]
>>> len(liste)
5
>>> liste.remove('a')
>>> # Seul le premier 'a' a été supprimé
>>> liste
[1, 2, 'a', 3]
>>> len(liste)
4
>>> # Suppression du prochain 'a'
>>> liste.remove('a')
>>> liste
[1, 2, 3]
>>> # Suppression d'un élement qui n'existe pas dans la liste génère une erreur
>>> liste.remove('a')
Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    liste.remove('a')
ValueError: list.remove(x): x not in list
```

La méthode *remove* ne retire que la première occurrence de la valeur trouvée dans la liste.

La méthode *del* est une méthode globale de Python, elle permet de supprimer un élément dans une liste, une variable, un élément dans un dictionnaire….

La méthode *remove* est une méthode spécifique aux listes seuls les éléments de type *list* peuvent l'utiliser. Raison pour laquelle avant de l'utiliser, on la précède du nom de la liste.

Lecture de listes

Pour afficher les valeurs d'une liste, on peut utiliser les mots clés d'itération ***while*** ou ***for***.

La méthode consiste à accéder à chaque valeur de la liste à partir de leur indice qui sera généré par les mots clés de boucles.

```
>>> # Création d'une liste
>>> liste = [1, 3, "a", "b"]
>>> # Affichage des valeurs de la liste avec while
>>> indice = 0
>>> while indice < len(liste):
...     print(liste[indice])
...     indice += 1
...
...
1
3
a
b
```

Equivalent à

```
>>> # Création d'une liste
>>> liste = [1, 3, "a", "b"]
>>> # Affichage des valeurs de la liste avec for
>>> for indice in range(len(liste)):
...     print(liste[indice])
...
...
1
3
a
b
```

Avec la méthode *for*, l'indice est incrémenté à chaque parcours ce qui n'est pas le cas avec *while* il faut veiller à incrémenter l'indice sinon on rentre dans une boucle infinie.

Dans les exemples précédents, on affiche les éléments de la liste on aurait pu exécuter d'autres types d'opérations comme le filtrage ou la mise en majuscule des caractères…

Lecture de listes

Dans la majorité des cas, on utilisera la syntaxe d'itération avec ***for*** car le parcours est fait de manière implicite. La méthode avec *for* appliquée à une chaine de caractère va parcourir chaque caractère et si c'est une liste c'est le parcours de chaque élément de la liste qui sera fait.

La syntaxe est la suivante:

for element in nom_liste

```
>>> # Création d'une liste
>>> liste = [1, 3, "a", "b"]
>>> # Parcours avec for
>>> for elt in liste:
...     print(elt)
...
...

1
3
a
b
```

```
>>> # Création d'une chaine de caractères
>>> msg = "Hello"
>>> for carac in msg:
...     print(carac)
...
...

H
e
l
l
o
```

Les variables *elt* et *carac* vont prendre à chaque parcours une valeur de la liste.

Ce fonctionnement implicite de *for* sera aussi utilisé dans les autres types d'objets complexes comme les dictionnaires ou tuples.

La fonction *enumerate*

La boucle for permet de parcourir une liste en capturant les éléments dans une variable sans qu'on sache où ils sont dans la liste.

Python offre la fonction ***enumerate*** qui permet de parcourir une liste en récupérant en même temps la valeur de l'élément ainsi que son indice. ***enumerate*** sera combiné avec la boucle ***for.***

La syntaxe est la suivante:

$$\boldsymbol{for\ variableContenantLindice, variableContenantLaValeur\ in\ enumerate(nomDeLaListe)}$$

```
>>> # Création d'une liste avec des éléments de départ
>>> maListe = ['a', 1, 'toto', 4]
>>> for indice,valeur in enumerate(maListe):
...     print(f"Element à l'indice {indice}: {valeur}")
...
...
Element à l'indice 0: a
Element à l'indice 1: 1
Element à l'indice 2: toto
Element à l'indice 3: 4
```

Les indices commencent à partir de 0 car le premier élément est situé à l'indice 0.

La fonction *enumerate*

Parfois on verra l'utilisation de *enumerate* avec une seule valeur comme ci-dessous:

$$\boldsymbol{for\ valeurActuelle\ in\ enumerate(nomDeLaListe)}$$

Dans ce cas la variable va contenir deux éléments l'indice et la valeur qui sera dans un type spécial appelé ***tuple*** **(à voir chapitre 10).**

```
>>> # Création d'une liste avec des éléments de départ
>>> maListe = ['a', 1, 'toto', 4]
>>> # Enumerate avec une seule valeur
>>> for element in enumerate(maListe):
...     # element contient deux valeurs en indice 0 on a l'indice de l'élément et à l'indice 1 on la valeur
...     print("Elément actuelle de enumerate:", element)
...     # Récupération de l'indice et la valeur
...     indice = element[0]
...     valeur = element[1]
...     print(f"Elément à l'indice {indice}: {valeur}")
...
...
Elément actuelle de enumerate: (0, 'a')
Elément à l'indice 0: a
Elément actuelle de enumerate: (1, 1)
Elément à l'indice 1: 1
Elément actuelle de enumerate: (2, 'toto')
Elément à l'indice 2: toto
Elément actuelle de enumerate: (3, 4)
Elément à l'indice 3: 4
```

Les tuples sont semblables aux listes mais on ne peut modifier un tuple après sa création. Lorsqu'un tuple est créé on ne peut ni ajouter ou ni retirer un élément dans celui-ci.

De chaînes aux listes

On peut convertir une chaîne de caractères en liste, pour cela on utilise la méthode ***split*** **(éclater en français).**

La méthode ***split*** est une méthode des chaines de caractères et prend en paramètre un caractère définissant comment l'on va découper la chaîne initiale. ***split*** renvoie une liste.

```
>>> # Création d'une chaine
>>> chaine = "a,b,c,d,e"
>>> # Création d'une liste de caractères
>>> caracteres = chaine.split(",")
>>> # La variable 'caracteres' est une liste de caractères
... print(caracteres)
    ['a', 'b', 'c', 'd', 'e']
```

```
>>> # Création d'une chaine de caractères
>>> msg = "John Doé"
>>> # Création d'une liste de mots
>>> mots = msg.split(" ")
>>> # La variable 'mots' est une liste de mots
>>> print(mots)
    ['John', 'Doé']
```

Quand on utilise la méthode *split* sans spécifier un caractère de découpage, la chaîne sera découpée en se basant sur les espaces, les tabulations et les sauts de ligne.

```
>>> # Création d'une chaine avec des espaces, tabulations et saut de lignes
>>> chaine = "Hello World\nToto\tRoro"
>>> print(chaine)
    Hello World
    Toto    Roro
>>> # Création d'une liste de mots
>>> mots = chaine.split()
>>> print(mots)
    ['Hello', 'World', 'Toto', 'Roro']
```

Des listes aux chaînes

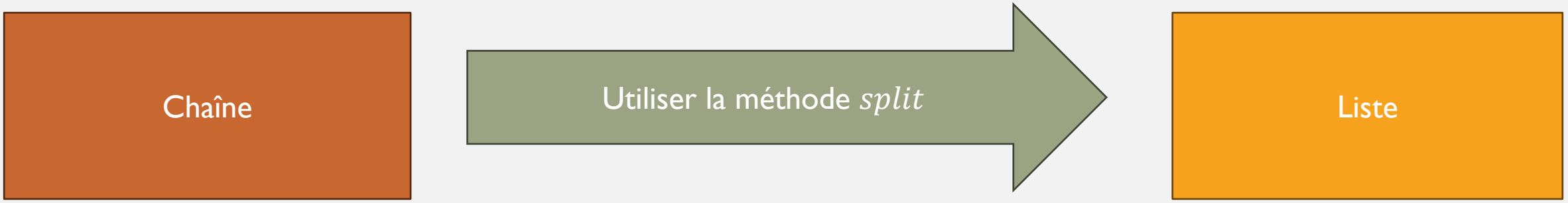
On peut aussi transformer une liste en chaine de caractères en utilisant la méthode ***join*** des chaines de caractères avec la syntaxe suivante:

$$'caractèreDeJointure'.join(nomDeLaListe)$$

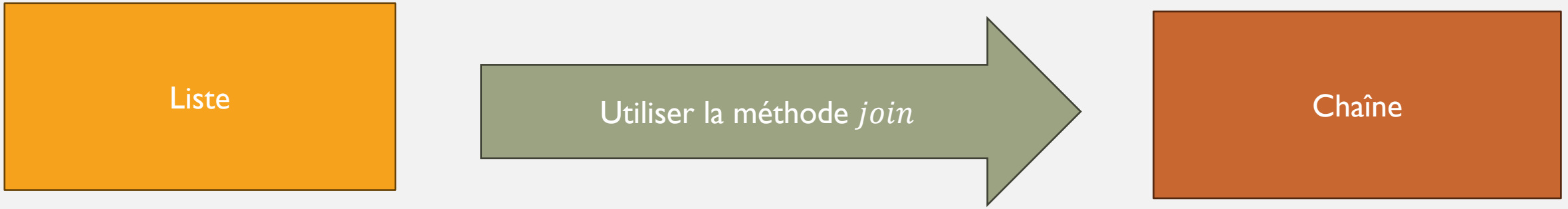
Cette méthode renverra une chaine de caractères en associant tous les éléments de la liste avec le caractère de jointure.

```
>>> # Création d'une liste de mots
>>> maListe = ["Toto", "Roro"]
>>> print(maListe)
['Toto', 'Roro']
>>> # Création d'une chaine avec un espace entre chaque mot
>>> chaine1 = " ".join(maListe)
>>> print(chaine1)
Toto Roro
>>> # Création d'une chaine avec | entre chaque mot
>>> chaine2 = "|".join(maListe)
>>> print(chaine2)
Toto|Roro
```

Split vs Join



$$\boldsymbol{nomDeLaListe.split('caractèreDeSeparation')}$$



$$\boldsymbol{'caractèreDeJointure'.join(nomDeLaListe)}$$

Parcours de liste efficace

Les programmeurs Python utilise une méthode efficace pour parcourir une liste en une ligne de code en la modifiant ou en la filtrant. On parle de **list comprehensions** (compréhension de liste en français).

```
>>> # Création de liste de nombres
>>> nombres = [1, 2, 3, 4]
>>> print(nombres)
    [1, 2, 3, 4]
>>> # Création d'une liste qui va contenir le double de chaque nombre
>>> double = [2 * n for n in nombres]
>>> # La variable 'double' est bien une liste
>>> print(type(double))
    <class 'list'>
>>> print(double)
    [2, 4, 6, 8]
>>> # Création d'une liste qui va contenir le carré de chaque nombre
>>> carre = [elt * elt for elt in nombres]
>>> # La variable 'carre' est bien une liste
>>> print(type(carre))
    <class 'list'>
>>> print(carre)
    [1, 4, 9, 16]
```

Avec cette méthode, le résultat $2 * n$ est spécifié avant le parcours de la liste $for\ n\ in\ nombres$.

Quand Python interprète le code il va parcourir la liste d'origine et utiliser chaque élément de la liste pour appliquer l'opération et renvoyer ensuite le résultat obtenu sous la forme d'une liste qui est de la même longueur que celle d'origine si aucune condition de filtrage.

Parcours avec filtrage avec condition

On peut utiliser la compréhension de liste pour créer une liste avec filtrage par exemple en prenant des éléments de la liste de départ en ignorant certains éléments pour former la nouvelle liste.

```
>>> # Création de liste des nombres de 0 à 10
>>> nombres = [n for n in range(11)]
>>> print("Les nombres de 0 à 10 sont:")
Les nombres de 0 à 10 sont:
>>> print(nombres)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> # Création d'une liste des nombres pairs de 0 à 10 basée sur la liste précédente
>>> pairs = [n for n in nombres if n % 2 == 0]
>>> print("La liste des nombres pairs entre 0 et 10 sont:")
La liste des nombres pairs entre 0 et 10 sont:
>>> print(pairs)
[0, 2, 4, 6, 8, 10]
```

L'instruction de filtrage de la liste d'origine doit toujours être spécifiée à la fin. Dans ce programme, la liste résultat ***pairs*** a une taille deux fois plus petite que celle de départ ***nombres***.

Méthodes usuelles de liste

Python offre plusieurs méthodes pour manipuler les listes qui sont répertoriées sur la documentation 5. Structures de données — Documentation Python 3.12.4.

Pour vérifier qu'un élément est dans une liste on utilisera:

$$element\ in\ nom\_liste$$

```
>>> liste = ['a', 1, 2, 'a']
>>> 'a' in liste
True
>>> 'b' in liste
False
>>> 1 in liste
True
```

Pour ajouter un élément à la fin de la liste:

$$nomDeLaListe.append(element\_a\_ajouter)$$

```
>>> liste = ['a', 1, 2, 'a']
>>> print(liste)
['a', 1, 2, 'a']
>>> 'b' in liste
False
>>> liste.append('b')
>>> print(liste)
['a', 1, 2, 'a', 'b']
>>> 'b' in liste
True
```

Méthodes usuelles de liste

Pour inverser l'ordre des éléments de la liste, on utilisera:

$$nomDeLaListe.reverse()$$

```
>>> liste = [1, 2, 3]
>>> print(liste)
[1, 2, 3]
>>> liste.reverse()
>>> print(liste)
[3, 2, 1]
```

Pour compter le nombre de fois où un élément est présent dans une liste on utilisera:

$$nomDeLaListe.count(element)$$

```
>>> liste = ['a', 1, 2, 'a']
>>> print(liste)
['a', 1, 2, 'a']
>>> liste.count('a')
2
>>> liste.count('b')
0
>>> liste.count(1)
1
```

Encore plus de méthodes sur la documentation 5. Structures de données — Documentation Python 3.12.4.

La fonction id avec les listes

Si la mémoire de l'ordinateur était représentée comme une étagère avec plusieurs tiroirs numérotés, la valeur d'une variable serait stockée dans un des tiroirs.

Chaque tiroir serait appelé case mémoire et le numéro de tiroir le numéro de la case mémoire.

Python nous offre la fonction *id* qui permet d'avoir le numéro de la case mémoire où est stockée la valeur d'une variable. La syntaxe est la suivante:

$$\boldsymbol{id(nomDeLaVariable)}$$

```
>>> # Déclaration d'une variable
>>> a = 1
>>> id(a)
140703568552744
>>> b = "John"
>>> id(b)
2661724191984
```

Dans l'exemple précédent, on voit que les variables a et b ont un numéro de case mémoire différent ce qui signifie que les valeurs des variables sont stockées à deux endroits (tiroirs) différents.

La fonction *id* ne renvoie pas tout le temps, le même numéro de case mémoire. La valeur obtenue varie d'un ordinateur à l'autre.

La fonction id avec les listes

Quand dans un programme deux variables de types primitifs comme $int, float, bool\ et\ str$ ont la même valeur, alors les deux variables partage la même case mémoire tant qu'elle représente la même valeur.

```
>>> # Cas des entiers
>>> a = 1
>>> b = 1
>>> id(a)
140703568552744
>>> id(b)
140703568552744
>>> id(a) == id(b)
True
>>>
>>> # Si on change la valeur de a
>>> a = 2
>>> a
2
>>> b
1
>>> id(a)
140703568552776
>>> id(b)
140703568552744
>>> id(a) == id(b)
False
```

Partage la même case mémoire c'est pourquoi id est identique

Id de la variable *a* a changé car la valeur de la variable a n'est plus la même donc stocké dans une autre case mémoire.

Ce mécanisme de partage de la case mémoire, n'augmente pas le nombre de cases mémoire occupées. 100 variables ayant la même valeur et 50 variables ayant la même valeur sont liées à une seule case mémoire.

La fonction id avec les listes

Quand on utilise les types complexes comme les $list(liste), tuple\ et\ les\ dict(dictionnaires)$ cela n'est plus pareil. Toute déclaration d'une variable de type objet sera stockée dans une nouvelle case mémoire peu importe qu'une autre variable avec la même valeur existe ou non.

```
>>> # Déclaration d'une liste
>>> l1 = [1, 2]
>>> l2 = [1, 2]
>>> id(l1)
3106135014912
>>> id(l2)
3106128038592
>>> id(l1) == id(l2)
False
>>>
>>> # Déclaration de tuples
>>> t1 = 1, 2
>>> t2 = 1, 2
>>> id(t1) == id(t2)
False
>>>
>>> # Déclaration de dict
>>> d1 = {"nom": "toto"}
>>> d2 = {"nom": "toto"}
>>> id(d1) == id(d2)
False
```

Contrairement aux types primitifs ici on a une augmentation du nombre de cases mémoire occupées. 100 variables de types objet occuperont 100 cases mémoires différentes.

Lire l'article Quelle est le rôle de la fonction id en Python ? pour plus d'informations.

Exercices

Exercice 1

Ecrire un programme Python qui demande un nombre à virgule à l'utilisateur. L'utilisateur peut choisir de saisir un nombre en format français par exemple 3,12 ou format anglais 3.12.
Le programme doit renvoyer le nombre saisit par l'utilisateur avec 5 chiffres après la virgule.
Par exemple si l'utilisateur entre 3,99999999999 le programme doit renvoyer 3,99999.
Si l'utilisateur entre 3,99 le programme doit renvoyer 3,99000.

Exercice 2

Ecrire un script qui génère la liste des carrés et des cubes des nombres de 20 à 40.

Exercice 3

Ecrire un script qui demande à l'utilisateur de saisir un nom et le stocke dans une liste, le programme doit toujours demander un nom à l'utilisateur jusqu'à ce que l'utilisateur entre un caractère vide.
Ensuite le programme doit afficher la liste des noms entrés par l'utilisateur ainsi que le nombre de caractères que contient chaque nom. Par exemple:
John: 4 caractères
Doé: 3 caractères

Exercices

Exercice 4

Ecrire un programme qui va demander à l'utilisateur un nombre. Ensuite le programme lui demande s'il veut entrer un autre nombre ou non, continuer à demander un nombre tant que l'utilisateur dit Oui.
A la fin de la saisie, demander à l'utilisateur dans quel ordre il veut l'affichage (croissant ou décroissant).
Ensuite afficher la liste des nombres en ordre croissant ou décroissant, afficher le plus grand de ces nombres et le plus petit de ces nombres.

Exercice 5

Ecrire un programme qui demande à l'utilisateur de saisir une phrase quelconque. Ensuite le programme doit afficher le ou les mots le(s) plus long(s).

Exercice 6

Ecrire un programme qui demande à l'utilisateur de saisir un mot. Ensuite le programme doit afficher la liste des consonnes et des voyelles du mot entré par l'utilisateur.

Exercices

Mini-Projet 1

Ecrire un programme de Quiz de table de multiplication.
Le programme doit demander à l'utilisateur de choisir un mode entre:

- Facile (table de 0 à 10) 10 questions
- Moyen (table de 0 à 20) 20 questions
- Difficile (table de 0 à 50) 30 questions

Ensuite en fonction du mode le programme doit poser des questions sur le résultat de la table de multiplication entre deux nombres tirés au hasard.
Le nombre de questions dépendra du mode choisi:

- Facile 10 questions
- Moyen 20 questions
- Difficile 30 questions

A la fin de la série, le programme doit afficher la liste des erreurs commises par l'utilisateur ainsi que son score et son pourcentage de réussite (success*100/total).
Tenir compte des cas d'erreur.
Le test est réussi si le score en pourcentage est supérieur à 50%.

Exercices

Mini-Projet 2

Ecrire un programme qui va permettre d'enregistrer des utilisateurs avec leur nom et prénoms.
Le menu est composé de: ajouter une personne, supprimer une personne, recherche une personne, afficher la liste.
L'utilisateur doit faire un choix d'action et le programme doit permettre de gérer les personnes sachant qu'on ne peut avoir une personne de même nom et prénoms.
Par exemple John DOE ne peut être enregistré deux fois cependant Doé JOHN peut être enregistré même si John DOE y ait car les noms et les prénoms sont différents.
Lors de la recherche l'utilisateur fournit un nom et le programme doit afficher la liste des personnes qui ont soit le même nom ou soit le même prénom.
Pour supprimer une personne l'utilisateur doit fournir un nom et le programme lui affiche la liste des personnes correspondantes ensuite il doit entrer le numéro de la personne qu'il veut supprimer.
L'action affichage doit permettre d'afficher toutes les personnes enregistrées.
Une personne à un nom et plusieurs prénoms séparés par des espaces.
Lors de l'enregistrement respecter le format Prénoms NOM. Si l'utilisateur entre comme nom Doé et prénoms Toto John le stockage sera Toto John DOE.

Exercices

Mini-Projet 3

Le chiffre de César (ou chiffrement par décalage) est un algorithme de chiffrement très simple que Jules César utilisait pour chiffrer certains messages qu'il envoyait.

Le principe du chiffrement est le suivant:
Pour un message à chiffrer, on remplace chaque lettre par une lettre différente, située x lettres après dans l'alphabet où x est la valeur de la clé de chiffrement.
Si l'on considère que chaque lettre de l'alphabet est numérotée de 0 à 25 (A=0, B=1,…,Z=25), cela revient à additionner la valeur de la lettre du texte en clair avec la valeur de la clé pour trouver la valeur de la lettre qui va remplacer la lettre non chiffrée.
Par exemple pour chiffrer la chaine $ZETA$ avec une clé de chiffrement 1 on aura comme mot chiffré: $AFUB$ car la lettre 1 pas après Z est A et on recommence au début quand on arrive à la fin du tableau. On obtient le mot chiffré de $ZETA$ en remplaçant chaque lettre par la lettre qui la suit puisque le nombre de décalage est 1.

Dans le cas du déchiffrement on fait des sauts en arrière contrairement au chiffrement qui fait des sauts en avant. Une lettre chiffrée par une clé x est déchiffrée par la lettre se trouvant à x caractères avant la lettre dans l'alphabet. **Le chiffre de César est aujourd'hui abandonné car non sécurisé.**

Exercices

L'objectif est d'utiliser les connaissances apprises sur les listes pour écrire un programme Python qui permettra de déchiffrer/chiffrer un message fourni par un utilisateur avec une clé donnée.
Le programme doit pouvoir chiffrer et déchiffrer des messages contenant des lettres majuscules et minuscules ainsi que des nombres.

Lorsque le programme démarre, l'utilisateur doit choisir quel type d'opération il veut réaliser (chiffrement ou déchiffrement). Ensuite il doit saisir le texte qu'il veut chiffrer et la clé.
Le programme doit afficher le résultat de l'opération.

Ecrire le fichier Algorithme et ensuite le programme en Python.

Ci-dessous un exemple d'exécution (votre version peut être différente):

```
      CHIFFRE DE CESAR


Quelle opération voulez vous réaliser ?
1. Chiffrement
2. Déchiffrement
Entrer le numéro de l'opération:1

Entrer le message à chiffrer:Hello Joh● !

Entrer la clé de chiffrement (entre 1 et 61):10

Le message chiffré est:

Rovvy Tyrx !
```

```
      CHIFFRE DE CESAR


Quelle opération voulez vous réaliser ?
1. Chiffrement
2. Déchiffrement
Entrer le numéro de l'opération:2

Entrer le message à déchiffrer:Rovvy Tyrx !

Entrer la clé de déchiffrement (entre 1 et 61):10

Le message déchiffré est:

Hello John !
```

Exercices

Mini-Projet 4

Le chiffre de Vigenère est un algorithme de chiffrement établi par le cryptographe français BLAISE DE Vigenère. C'est un crypto système poly-alphabétique (contrairement au chiffre de César qui est mono alphabétique), c'est-à-dire qu'il consiste à changer une lettre par une autre, mais cette dernière n'est pas toujours la même. Cela permet une plus grande sécurité. Cet algorithme utilise une clé dans notre cas sous la forme d'un mot ou d'une phrase que l'on choisira. Plus la clé sera longue, plus le cryptogramme sera sécurisé.

Pour utiliser le chiffre de Vigenère on dispose de deux méthodes de chiffrement/déchiffrement:

- Soit on utilise la table de Vigenère voir Chiffre de Vigenère.
- Soit on utilise la méthode mathématique.

Nous utiliserons dans cet exercice la méthode mathématique.

Avec la méthode mathématique, on part d'un tableau comme le chiffre de César composé des lettres de A à Z ayant respectivement les indices de 0 à 25.

Supposons:

- x : l'indice de la lettre de la clé
- y : l'indice de la lettre chiffrée
- z: l'indice de la lettre du texte en clair

Exercices

Mini-Projet 4

Si on veut chiffrer, on obtient l'indice de la lettre chiffrée par la formule:

$$y = (x + z)\ modulo\ 26$$

Par exemple pour chiffrer la lettre B (indice 1) du message avec la lettre de la clé H (indice 7) on aura:

$$y = (1 + 7)\ modulo\ 26 = 8\ modulo\ 26 = 8\ soit\ la\ lettre\ I$$

Le modulo est le reste de la division euclidienne en Python il est représenté par %.
Par exemple 2%1=0 et 5%2=1

Quand on veut déchiffrer on cherche l'indice z par les formules suivantes:

- Si $x > y$ alors $z = (26\ - x) + y$
- Sinon $z = y - x$

Par exemple pour déchiffrer la lettre I (indice 8) du message chiffré avec la lettre de la clé H (indice 7) on aura:

$$x = 7 < y = 8\ alors\ z = 8 - 7 = 1\ soit\ la\ lettre\ B$$

Les chiffres de César et Vigenère appartiennent à la famille du chiffrement symétrique car on utilise la même clé pour chiffrer et déchiffrer le message.

Exercices

L'objectif est d'utiliser les connaissances apprises sur les listes pour écrire un programme Python qui permettra de déchiffrer/chiffrer un message fourni par un utilisateur avec une clé donnée.
Le programme doit pouvoir chiffrer et déchiffrer des messages contenant des lettres majuscules et minuscules ainsi que des nombres.

Lorsque le programme démarre, l'utilisateur doit choisir quel type d'opération il veut réaliser (chiffrement ou déchiffrement). Ensuite il doit saisir le texte qu'il veut chiffrer et la clé.
Le programme doit afficher le résultat de l'opération.

Ecrire le fichier Algorithme et ensuite le programme en Python.

Ci-dessous un exemple d'exécution (votre version peut être différente):

```
     CHIFFRE DE VIGENERE


Quelle opération voulez vous réaliser ?
1. Chiffrement
2. Déchiffrement
Entrer le numéro de l'opération:1

Entrer le message à chiffrer:Bonjour

Veuillez entrer la clé:toto

Le message chiffré est:

UCGxHIK
```

```
     CHIFFRE DE VIGENERE


Quelle opération voulez vous réaliser ?
1. Chiffrement
2. Déchiffrement
Entrer le numéro de l'opération:2

Entrer le message à déchiffrer:UCGxHIK

Veuillez entrer la clé:toto

Le message déchiffré est:

Bonjour
```

FIN CHAPITRE 9