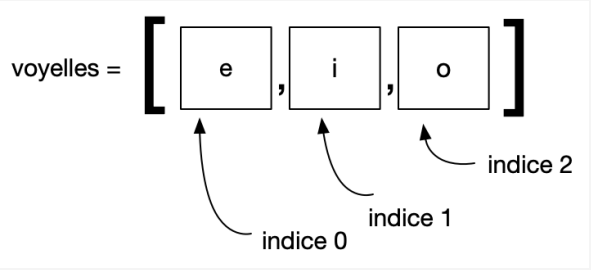
Fiche Python Listes

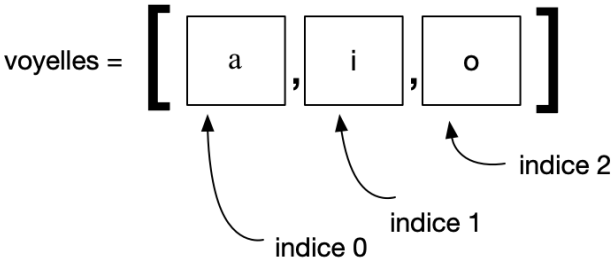
Une **liste** est une collection ordonnée d'objets. Une **liste** est entourée de **crochets** `[ ]`
On accède à un élément d'une liste grace à sa position, appelée *indice*. Le premier élément a pour indice zero.

```
voyelles = ['e','i','o']
type(voyelles)
# affiche list
voyelles[0]
# affiche e
voyelles[1]
# affiche i
voyelle[2]
# affiche o
voyelles[-1]
# affiche o
voyelles
# affiche ['e','i','o']
```



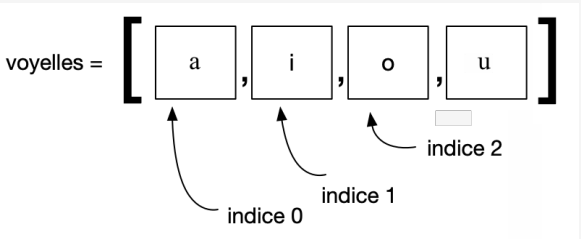
Modifier un élément de liste: On accède et on modifie un élément grâce à son indice.

```
voyelles = ['e','i','o']
voyelles[0] = "a"
voyelles
# affiche ["a","i","o"]
```



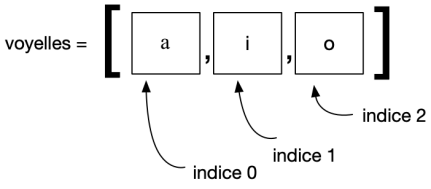
Ajouter un élément à la liste: Méthode **`append`**

```
voyelles = ["a","i","o"]
len(voyelles)
# affiche 3
voyelles.append('u')
voyelles
# affiche ["a","i","o","u"]
len(voyelles)
# affiche 4
```



Retirer le dernier élément de liste: Méthode **pop**

```
voyelles = ["a","i","o","u"]
voyelles.pop()
# affiche "u"
voyelles
# affiche ["a","i","o"]
```



Exercice:

```
semaine = ["lundi","mardi","mercredi"]
semaine.pop()
semaine.append("jeudi")
semaine[0] = "dimanche"
print(semaine)
# affiche … … ..
```

Fiche Python 2 Parcours de listes

Parcourir une liste sert à appliquer un traitement sur chacun de ses éléments (affichage, modification, calcul sur ses valeurs, …).
Il existe 2 moyens de parcours d'une liste avec une boucle bornée

```
L = [25, 54, 77, 100]
```

Parcours par élément: L'itérable prend la valeur des éléments de liste

```
for x in L:
    print(x)

# affiche 25
# affiche 54
# affiche 77
# affiche 100
```

Parcours par indice: L'itérable prend la valeur des **indices** des éléments de liste

```
for i in range(len(L)):
    print(i)

# affiche 0
# affiche 1
# affiche 2
# affiche 3
```

```
for i in range(len(L)):
    print(L[i])

# affiche 25
# affiche 54
# affiche 77
# affiche 100
```

Parcours et méthodes de listes: appliquer un traitement à chaque valeur de X et l'ajouter à Y

```
X = [25, 54, 77, 100]
Y = []
for val in X:
    Y.append(val * 2)
print(Y)
# affiche [50, 108, 154, 200]
```

Exercice: additionner les valeurs de liste

```
L = [25, 25, 25, 100]
s = 0
for x in L:
    s = s + x
print(s)
# affiche … … …
```

Exercice: Sélection du plus grand

```
X = [25, 54, 77, 100]
Y = [23, 55, 79, 99]
Z = []
for i in range(len(X)):
    if X[i] > Y[i]:
        Z.append(X[i])
    else:
        Z.append(Y[i])
print(Z)
# affiche … …
```

Fiche Python 3 Listes et fonctions

Une fonction sert à isoler un morceau de code afin d'organiser le script, ou bien de réutiliser cette fonction à plusieurs endroits du programme. On déclare (on définit) une fonction par son nom, placé après le mot clé **`def`**

Une fonction peut avoir un ou des paramètres. Ceux-ci sont placés entre parenthèses lors de la déclaration.
Une fonction peut retourner une valeur.

Exemple:

```
# declaration de la fonction
def carre(x):
    return x * x
# appel de la fonction avec l'argument 5 placé sur le paramètre x
carre(5)
# retourne 25
```

Fonction somme de 2 paramètres: Voyons comment adapter le script qui calcule la somme de 2 valeurs

```
x = 125
y = 100
s = x + y
print(s)
# affiche 225
```

```
def sum(x , y):
    return x + y

sum(125, 100)
# retourne 225
```

Fonction somme sur une liste: Voyons comment adapter le script qui calcule la somme des éléments d'une liste

```
L = [25, 25, 25, 100]
s = 0
for x in L:
    s = s + x
print(s)
# affiche 175
```

```
def somme(T):
    s = 0
    for x in T:
        s = s + x
    return s

L = [25, 25, 25, 100]
somme(L)
# retourne 175
```

Exercice: Fonction moyenne sur une série de 3 valeurs

```
def moy(x, y, z):
    return .. .. ..
moyenne(12, 14, 16)
# retourne …
```

Exercice: **Fonction moyenne** sur les éléments de liste. Adapter le script de gauche

```
L = [25, 25, 25, 100]
s = 0
for x in L:
    s = s + x
print(s / len(L))
# affiche 43.75
```

```
def moyenne(T):
    s = 0
    …
    …
    …
    …
L = [25, 25, 25, 100]
somme(L)
# retourne …
```

Fiche Python 4 Dessiner sur une grille

Une grille est représentée en python par un tableau, c'est à dire une liste. Les données peuvent être les unes à la suite des autres (fichier image). Ou bien organisées dans une liste contenant des sous-listes (matrice).

Exemple:

```
# image binaire 4 * 4
img = [1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1]
# matrice
mat = [[1, 1, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0], [1, 1, 1, 1]]
```

Slice sur une liste: On souhaite extraire les 8 premières valeurs binaires de la liste `img`:
2 écritures équivalentes:

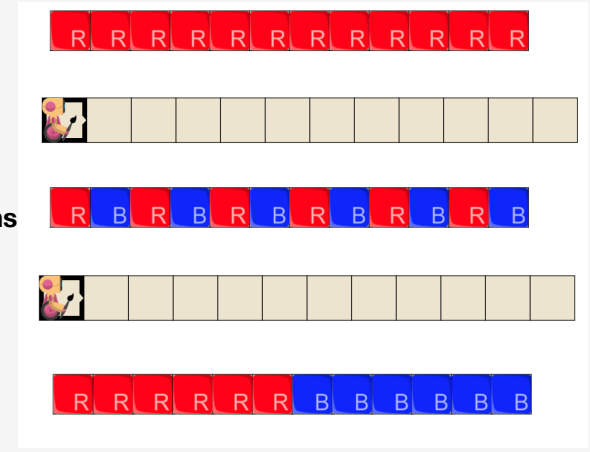
```
img[0:8]
img[:8]
# [1, 1, 0, 0, 1, 0, 0, 0]
```

Accéder à un élément de matrice:

```
print(mat[0])
# affiche [1, 1, 0, 0]

print(mat[0][0])
# affiche 1 (première ligne & première colonne)

print(mat[1][0])
# affiche 1 (deuxième ligne & première colonne)

print((mat[2][1])
# affiche 0 (troisième ligne & deuxième colonne)
```

Placer des valeurs identiques dans une liste:

```
L = []
for i in range(12):
    L.append(1)
L
# [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```



Utiliser une boucle avec une séquence d'instructions

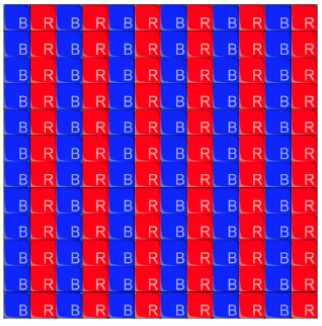
```
L = []
for i in range(…
    …
    …
L
# [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

Placer des valeurs avec une rupture de séquence:

```
L = []
for i in range(…
    L.append(…
…
    …
# [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
```

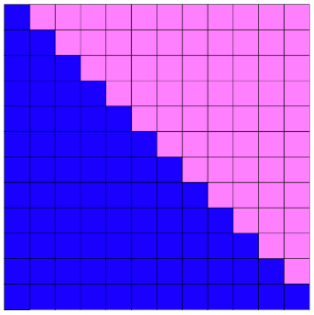
Utiliser des boucles imbriquées: matrice de lignes toutes identiques

```
mat = []
for i in range(12):
    ligne = []
    for j in range(..
        ligne.append(..
        ligne.append(..
    ..

mat
# [[0,1,0,1,0,1,0,1,0,1,0,1],[0,1,0,1,0,1,0,1,0,1,0,1], …
```



Utiliser un variant de boucle: matrice de lignes différentes

```
mat = []
for i in range(12):
    ligne = []
    for j in range(..
        ligne.append(..
    for k in range(..
        ligne.append(..

mat
# [[0,1,1,1,1,1,1,1,1,1,1,1],
   [0,0,1,1,1,1,1,1,1,1,1,1],
   [0,0,0,1,1,1,1,1,1,1,1,1]
```



Fiche Python 5 Algorithmes essentiels

Recherche séquentielle, recherche linéaire:

```
def recherche1(T, x)->int:
  """
  :param T: list of elements
  :param x: element
  :return: int, index of x in the list
           else -1
  """
  i = 0
  while i < len(T) and T[i] != x:
    i = i + 1
  if i == len(T):
    return -1
  else:
    return i
```

Recherche dichotomique:

```
def recherche_dicho(T,x):
    """
    :param T: sorted list of elements
    :param x: element
    :return: int, index of x in the list else -1
    """
    i_min = 0
    i_max = len(T) - 1
    while (i_max >= i_min):
        mid = (i_min + i_max)//2
        if T[mid] == x:
            return mid
        elif T[mid] < x:
            i_min = mid +1
        else:
            i_max = mid -1
    return -1
```

Inverser une liste:

```
def inverse(L1):
    """inverse les elements de la liste L1
    @return:
    L2: liste inversee
    """
    L2 = []
    l = len(L1)
    for rang in range(l):
        L2.append(L1[l-rang-1])
    return L2

>>> inverse(['N', 'O', 'E', 'L'])
['L', 'E', 'O', 'N']
```

Recherche du minimum:
il existe une fonction native en python qui retourne la valeur min dans une liste:

```
>>> min([0,-10, 5, 3])
-10
```

mais celle-ci ne retourne pas le RANG du minimum. Voici le programme de la fonction recherche_du_min qui retourne l'indice du min ainsi que la valeur min d'une liste:

```
def recherche_du_min(L):
    mini = L[0]
    ind_mini = 0
    for i in range(len(L)):
        if L[i] < mini:
            mini = L[i]
            ind_mini = i
    return ind_mini, mini

>>> recherche_du_min([0,-10, 5, 3])
1, -10
```

Recherche du maximum:
il existe une fonction native en python qui retourne la valeur max dans une liste:

```
>>> max([0,-10, 5, 3])
5
```

mais celle-ci ne retourne pas le RANG du maximum. Voici le programme de la fonction recherche_du_max qui retourne l'indice du max ainsi que la valeur max d'une liste:

```
def recherche_du_max(L):
    maxi = L[0]
    ind_maxi = 0
    for i in range(len(L)):
        if L[i] > mini:
            maxi = L[i]
            ind_maxi = i
    return ind_maxi, maxi

>>> recherche_du_max([0,-10, 5, 3])
2, 5
```

Tri par insertion:

```
def tri_insertion(tableau):
    for i in range(1,len(tableau)):
        en_cours = tableau[i]
        j = i
        #décalage des éléments du tableau }
        while j>0 and tableau[j-1]>en_cours:
            tableau[j]=tableau[j-1]
            j = j-1
        #on insère l'élément à sa place
        tableau[j]=en_cours
    return tableau

>>> tri_insertion([49, 4, 89, 13, 44, 60, 12, 2, 5, 62])
[2, 4, 5, 12, 13, 44, 49, 60, 62, 89]
```