Tableaux

Pour la représentation des entiers négatifs on a vu la méthode du complément (à 2)
Pour trouver la représentation de -4 **sur un octet** on procède ainsi :

1. On écrit 4 sur un octet -> 0000 0100
2. On prend le complémentaire de 4 -> 1111 1011
3. Puis on ajoute 1 à 1111 1011 et on obtient ainsi la représentation de -4 autrement dit 1111 1100

On aimerait écrire une fonction Python `complementaire(octet)`

1. prenant en entrée un octet, par exemple 0000 0100
2. et retournant le complémentaire de cet octet ici 1111 1011

Comment mémoriser un octet ? Comment mémoriser plusieurs valeurs nommées par un seul nom de variable ?

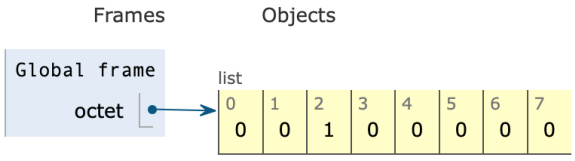
Un tableau permet de mémoriser plusieurs valeurs nommées **par un seul nom de variable**

Ainsi en Python à la variable `octet` on affecte une suite ordonnée de valeurs entre crochets et séparées par des virgules

Or l'écriture 0000 0100 signifie $0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2$ donc on va réécrire les bits de la gauche vers la droite dans l'ordre croissant des puissances de 2

```
octet = [0,0,1,0,0,0,0,0]
```

que l'on visualise ainsi



A la variable `octet` qui n'est qu'un nom est associé une **référence** (une adresse mémoire) permettant de retrouver toutes les valeurs du tableau

Cette référence est matérialisée par la flèche

Si on veut accéder à une valeur particulière

La première valeur du tableau est repérée par l'indice 0, si on veut accéder à cette valeur et par exemple afficher cette valeur

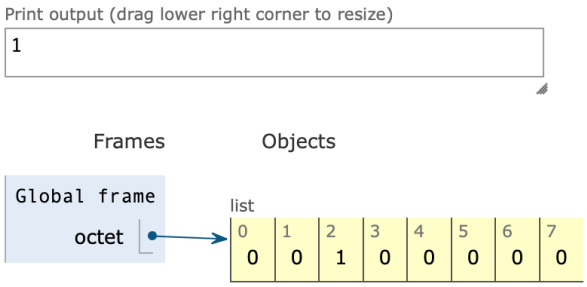
`print(octet[0])` et on obtiendra 0

La deuxième valeur du tableau est repérée par l'indice 1, si on veut accéder à cette valeur et par exemple afficher cette valeur

`print(octet[1])` et on obtiendra 0

La troisième valeur du tableau est repérée par l'indice 2, si on veut accéder à cette valeur et par exemple afficher cette valeur

`print(octet[2])` et on obtiendra 1, etc...



Attention à ne pas chercher une valeur au-delà de la taille du tableau sinon on aura une erreur

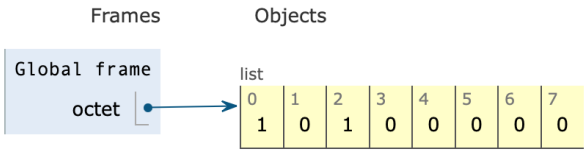
```
>>> octet = [0,1,0,1,0,0,0,0]

>>> octet[8]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: list index out of range
```

On peut modifier la valeur de chaque bit

Par exemple si on veut modifier le bit associé à 2^0 autrement dit le bit repéré par 0 on écrit

```
octet[0] = 1
```

on observe alors



Si on veut connaître la taille du tableau, autrement dit le nombre d'éléments qu'il contient, on dispose pour cela de la fonction `len()` pour **length**

1 Tableaux et références

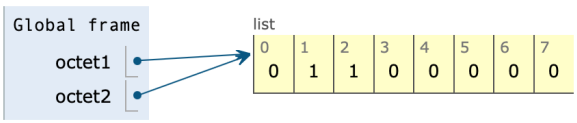
Attention un tableau peut être référencé par plusieurs variables ce qui peut emmener à des situations que l'on n'avait pas forcément prévu

Regardons cela sur un exemple

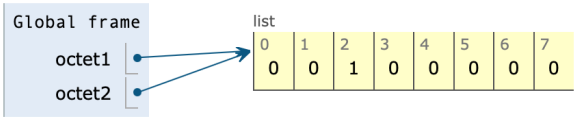
```
octet1 = [0,1,1,0,0,0,0,0]
octet2 = octet1
octet2[1] = 0
```

La variable `octet1` référence les valeurs [0,1,1,0,0,0,0,0]

L'affectation `octet2 = octet1` fait que les deux variables référencent les même valeurs



Par conséquent l'instruction `octet2[1] = 0` change la valeur à l'indice 1 dans le tableau `octet2` **mais aussi dans le tableau** `octet2`



2 Parcours d'un tableau

Pour définir la fonction `complementaire(octet)` il faut une commande permettant d'accéder à toutes les valeurs du tableau,(on dit aussi de parcourir le tableau)

Il existe deux façons de parcourir le tableau au moyen d'une boucle for

1. La première consiste à parcourir tous les indices du tableau de 0 jusqu'à $n-1$ où n est la longueur du tableau

   ```
   for i in range(len(octet)):
       octet[i] = 1 - octet[i]
   ```

   En procédant ainsi on peut modifier la valeur de chaque bit et obtenir le complémentaire

2. La deuxième nous permet **seulement de récupérer la valeur de chaque bit mais ne nous permet pas de modifier la valeur de chaque bit** et par conséquent ne marche pas ici

   Cependant dans des problèmes où on n'aura besoin que de récupérer les valeurs sans modifier les valeurs il est préférable d'utiliser cette méthode pour des raisons de lisibilité du code

   ```
   for bit in octet:
       bit = 1 - bit
   ```

   On peut donc définir maintenant la fonction

   ```
   def complementaire(octet):
       """
       octet est un tableau de taille 8
       """
       for i in range(len(octet)):
           octet[i] = 1 - octet[i]
   ```

   Nous avons vu un résultat important :

   Une fonction peut modifier le contenu d'un tableau passé en argument de cette fonction

   Cependant ceci pose le problème de la perte de la donnée de départ qui a été modifiée

   On aimerait à la fois conserver la donnée de départ et obtenir le complémentaire de l'octet de départ

Pour cela on va créer un nouveau tableau de même taille **en compréhension**, faire les affectations dans ce nouveau tableau et retourner la référence de ce nouveau tableau

```
def complementaire2(octet):
    """
    octet est un tableau de taille 8
    """
    #on initialise un tableau de taille 8 uniquement avec des 0
    comp = [0]*8
    for i in range(len(octet)):
        comp[i] = 1 - octet[i]
    return comp
```

3 Recherche dans un tableau

Un tableau peut servir à mémoriser des documents par exemple

```
documents = ['boucle_for.pdf', 'boucle_while.pdf',..., 'tableaux.pdf']
```

On aimerait savoir si le document `'binaire.pdf'` se trouve dans le tableau `documents`
Une première façon de faire est d'utiliser la commande `in` de Python

```
def appartient(nom,documents):
    return nom in documents
```

Voici un exemple du fonctionnement de `in` en console

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

>>> 3 in tab
True

>>> 4 in tab
False
```

Une autre façon est de parcourir le tableau avec une boucle `while` en faisant attention à protéger le booléen `nom != documents[i]` par le booléen `i < len(documents)` pour éviter l'erreur "list index out of range"

```
def appartient(nom,documents):
    i = 0
    while i < len(documents) and nom != documents[i] :
        i += 1
    return i < len(documents)
```

Enfin pour éviter l'erreur "list index out of range" on parcourt le tableau avec une boucle for de laquelle on peut sortir par `return` à condition que l'on ait trouvé la valeur cherchée

```
def appartient(nom,documents):
    for doc in documents:
        if nom == doc:
            return True
    return False
```

Exercices

Ex 1

Voici un programme en python

```
liste1 = [1,1]
liste2 = [0,0]
liste1[0] = 2
liste1 = liste2
print(liste1[0])
```

Qu'observe-t-on à l'écran ? (Justifier par un dessin)

A :) 2 B) : 1 C) : 0

Ex 2

Voici un programme en python

```
liste1 = [0,0,0]
liste2 = [1,1,1]
liste3 = [liste1,liste2]
liste2 = liste1
print(liste3)
```

Qu'observe-t-on à l'écran ? (Justifier par un dessin)

A :) [liste1,liste2] B) : [[0,0,0],[1,1,1]] C) : [[0,0,0],[0,0,0]]

Ex 3

Définir une fonction `moyenne(tableau)` qui retourne la moyenne des notes contenues dans `tableau`

Ex 4

1. Définir une fonction python `maximum(liste)` qui retourne la plus grande valeur trouvée dans la liste de nombres `liste`
   Par exemple `maximum([-2,3,4,3,1.5])` retourne 3
2. Définir une fonction python `maximumIndice(liste)` qui retourne l'indice de la première occurence de la plus grande valeur trouvée dans la liste de nombres `liste`
   Par exemple `maximumIndice([-2,3,4,3,1.5])` retourne l'indice 1 en effet la plus grande valeur 3 se situe aux indices 1 et 3 et on retourne 1 car c'est le premier indice

Ex 5

1. Définir une fonction `echange(tableau,i,j)` qui échange ldans le tableau les éléments d'indice i et j

2. Se servir de la fonction précédente pour définir une fonction `inverser(tableau)` qui **modifie** le tableau `tableau` en inversant l'ordre des éléments du tableau
   Ainsi à la console on obtient

```
tableau = [1,2,3]
>>> tableau
[1,2,3]
>>> inverser(tableau)
>>> tableau
[3,2,1]
```

3. Faire la même chose sans modifier le tableau de départ

Ex 6

Pour mesurer les différences entre deux tableaux de même taille contenant des entiers compter le nombre de différences

Définir une fonction `distance(tableau1,tableau2)` qui réalise ce travail

Ex 7 : opérateur &

Le & (lire et) correspond à la fonction logique `et` mais appliqué aux représentations des nombres en binaire, bit par bit.
Ainsi par exemple 110 & 100 = 100
Tester cette fonction à la console puis définir une fonction `et(liste1,liste2)` où liste1 et liste2 sont deux listes de même longueur contenant que des 0 et des 1 et retournant la liste résultat de liste 1 & liste2

Ex 8

On lance un dé à six faces **tant que les chiffres de 1 à 6 ne sont pas tous sortis**

On aimerait connaître le nombre de lancers moyen pour que les six chiffres apparaissent

1. Engendrer en compréhension une liste `chiffresSortis` de 7 booléens False.
   Lorsque le 2 sort on fait chiffresSortis[2] = True
2. Définir une fonction Python `pasTousSortis(chiffresSortis)` qui retourne Vraie si tous les chiffres ne sont pas sortis et Faux sinon
3. Définir une fonction `nbLancers()` qui retourne le nombre de lancers de dé effectués pour faire apparaître tous les chiffres
4. Définir une fonction `nbLancersMoyen(nbRepetitions)` qui retourne le nombre de lancers moyen de dé effectués pour faire apparaître tous les chiffres

Ex 9

Prolonger l'exercice précédent

Problème : Y-a-t-il une relation entre le nombre de faces d'un dé et le nombre moyen de lancers nécessaire pour faire apparaître tous les nombres sur les faces ?