

Séance 01 :

Objets mutables et non mutables

1. Cours

C.F. cours

2. Exemple 1

On donne le code suivant.

```
1 A = [k**2 for k in range(5)]
2 B = [k**2 for k in range(10, 15)]
3 C = A.extend(B) # Concatene B a A
4 del A, B
```

1. *Que contient la liste `C` et pourquoi ?*

La liste `C` est vide, ce n'est même pas une liste, car la méthode `extend` pour une `list` ne retourne rien.

2. *Comment modifier ce code pour qu'il remplisse son rôle ?*

Il faut modifier la ligne 4. par `C = A + B`.

3. Exemple 2

On donne les commandes suivantes :

```
1 a = ["t", "o", "t", "o"]
2 a.remove("o")
3 c = a.remove("o")
4 b = "toto"
5 b.strip("o") # supprime "o" s'il se trouve en debut ou fin de chaine
6 d = b.strip("o")
```

1. *Deviner le contenu des variables `a`, `b`, `c` et `d` sans exécuter le code.*

`a = "tt"`, `b = "tot"`, `c = null` et `d = null`.

4. Exercice 1 : Différence entre deux listes

1. *Ecrire une fonction `diff(a, b)` qui calcule la différence entre deux listes `a` et `b`.*

```
1 def diff(a, b):
2     result = a.copy()
3     for el in b:
4         if (el in result):
5             result.remove(el)
6     return (a, b, result)
7
8
9 # ----- Tests -----
10 if __name__ == "__main__":
11     a = [2, 7, 8, 15, 2, 3, 4, 5, 2, 3]
12     b = [2, 13, 8, 3, 2, 7, 8, 3]
13     d = diff(a, b)
14     print(
15         'liste a :\n\t{}\nliste b :\n\t{}\ndiff(a, b) :\n\t{}'
16         .format(a, b, d[2])
17     )
```

5. Exercice 2 : Nombres premiers

1. En utilisant `remove` donner la liste des 168 nombres premiers de 1 à 1 000 en utilisant le crible d'Ératosthène. Faire moins de 1 500 tours de boucle. De même trouver les 1 229 nombres premiers de 1 à 10 000 en faisant moins de 20 000 tours de boucle.

```

1 def prime(n):
2     """Returns the list of the n first n prime numbers,
3     and and the number of loop turns"""
4
5     count = 0 # loop counter
6
7     nb = [k for k in range(2, n+1)]
8     for i in range(2, n+1):
9         for el in nb:
10            if el % i == 0 and el != i:
11                count += 1 # On compte combien de tours de boucle on fait
12                nb.remove(el)
13     return (count, nb)

```

Résultats :

```

1  ===== Test pour n = 1000 =====
2  Le programme fait 831 tours (max : 1500 tours).
3  Temps d execution : 0.008s
4  Nombre de resultats : 168
5  Resultats : [2, 3, 5, 7, '...', 991, 997]
6
7  ===== Test pour n = 10000 =====
8  Le programme fait 8770 tours (max : 20000 tours).
9  Temps d execution : 0.595s
10 Nombre de resultats : 1229
11 Resultats : [2, 3, 5, 7, '...', 9967, 9973]

```

6. Exercice 3 : Suite de Syracuse

- (a) Déclarer la fonction `S` en utilisant reste et quotient dans les nombres entiers.

```

1 def S(n):
2     if n % 2 == 0:
3         return n/2
4     elif n != 1:
5         return 3*n+1
6     else:
7         return 1

```

- (b) Afficher la liste des valeurs $u_p = S^p(c)$ pour p allant de 0 à $+\infty$ et en s'arrêtant dès que $u_p = 1$.

```

1 def qb(x):
2     up = x
3     while up != 1:
4         print(up)
5         up = S(up)

```

- (c) Tester pour x dans la liste `[2, 7, 19, 23, 29]`.

```

1 def qc():
2     X = [2, 7, 19, 23, 29]
3     for x in X:
4         qb(x)

```

- (d) Trouver les entiers x compris entre 3 et 300 pour lequel le vol (c'est-à-dire la longueur de la suite pour obtenir 1) est le plus long.

```

1 def qb(x):
2     up = x
3     results = [up]
4     while up > 1:
5         up = S(up)
6         results.append(up)
7     return results
8
9
10 def qd():
11     X = [k for k in range(3, 301)]
12     volMax = 0
13     result = []
14     for x in X:
15         a = qb(x)
16         vol = len(a)
17         if vol > volMax:
18             result = []
19             volMax = vol
20             result.append(x)
21         elif vol == volMax:
22             result.append(x)
23     return result, volMax

```

On trouve que le vol le plus long est $v = 128$, et est atteint pour $x \in \{231, 235\}$.

- (e) *Calculer l'intersection entre deux suites de syracuse en renvoyant la longueur de l'intersection et les derniers éléments où les deux suites sont différentes. En cas d'inclusion, on renvoie le complexe 1j pour indiquer qu'une suite est incluse dans l'autre.*

```

1 def contains(small, big):
2     for i in range(len(big)-len(small)+1):
3         for j in range(len(small)):
4             if big[i+j] != small[j]:
5                 break
6         else:
7             return i, i+len(small)
8     return False
9
10
11 def interSyracuse(i1, i2):
12     i1, i2 = qb(i1), qb(i2)
13     grande, petite = [], []
14     # On selectionne la plus petite liste
15     if max(len(i1), len(i2)) == len(i1):
16         grande, petite = i1, i2
17     else:
18         grande, petite = i2, i1
19     for k in range(len(petite), 1, -1):
20         subPetite = petite[-k:]
21         if contains(subPetite, grande):
22             if subPetite == petite:
23                 return (1j)
24             else:
25                 maxi = subPetite[0]
26                 o = petite.index(maxi)
27                 p = grande.index(maxi)
28                 return (len(subPetite), petite[:o][-1], grande[:p][-1])

```

On trouve :

$interSyracuse(7, 320) = (9, 80, 13)$
 $interSyracuse(7, 230) = 1j$