TP_Listes_852_2016-2017

December 3, 2016

0.1 Présentation des listes avec pythontutor.com

http://www.pythontutor.com/visualize.html#code=c1+%3D+%5B851,+852,+853%5D%0Ac2+%3D+c1%0Ac3+%
12%0A%0Adef+modifier_entier(n%29%3A%0A++++n+%3D+-12%0A++++%0Amodifier_tab(c1%29%0A%0Am
frontend.js&cumulative=false&heapPrimitives=false&textReferences=false&py=3&rawInputLstJSON=%5B%5D

1 Définition et création de listes

1.0.1 Exercice 1 Des listes jouets

```
In [6]: t1=[10,30,42,2,17,5,30,-20]
        t2=[i**2 for i in range(-3,6)]
        t3=[i**3 for i in range(1000) if (i%5) in [0,2,4]]
        t4=[841.0]
        for i in range(20):
            t4.append(t4[i]/3+28)
        #t5 identique à t2
        t5 = []
        for i in range(-3,6):
            t5.append(i**2)
        #t6 identique à t3
        t6 = []
        for i in range(1000):
            if (i%5) in [0,2,4]:
                t6.append(i**3)

In [3]: t7 = ['ab', 'bc', 'a', 'bb']
        t8 = [t7[i] for i in range(len(t7)) if 'a' in t7[i]]

In [4]: t7, t8

Out[4]: (['ab', 'bc', 'a', 'bb'], ['ab', 'a'])

In [2]: t5, t2

Out[2]: ([9, 4, 1, 0, 1, 4, 9, 16, 25], [9, 4, 1, 0, 1, 4, 9, 16, 25])

In [4]: len(t6)
```

```
Out[4]: 600

In [5]: t6 == t3

Out[5]: True
```

2 Manipulation de listes

2.1 Accès aux éléments

2.1.1 Exercice 2 Longueur d'une liste

```
In [7]: len(t1)  #donne la longueur de t1

Out[7]: 8

In [93]: [len(L) for L in [t1,t2,t3,t4]]

Out[93]: [8, 9, 600, 21]
```

2.1.2 Exercice 3 Procédure échangeant deux valeurs dans une liste/tableau

```
In [7]: L = [i for i in range(851, 854)]

In [8]: L

Out[8]: [851, 852, 853]

In [9]: L[0], L[1] = L[1], L[0]

In [10]: L

Out[10]: [852, 851, 853]

In [94]: def echange(t, i, j):
             """echange les valeurs des termes t[i] et t[j] du tableau t"""
             t[i], t[j] = t[j], t[i]
             # inutile de retourner t puisque les listes sont passées par référence

In [95]: t = [k for k in range(10)]
         t

Out[95]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [96]: echange(t, 1, 9)

In [97]: t

Out[97]: [0, 9, 2, 3, 4, 5, 6, 7, 8, 1]
```

2.2 Liste de listes, matrices

```
In [98]: M = [ [1, 2, 3], [4,5, 6] ]
         M

Out[98]: [[1, 2, 3], [4, 5, 6]]

In [99]: len(M)

Out[99]: 2

In [100]: M[1]

Out[100]: [4, 5, 6]

In [101]: M[0][2]

Out[101]: 3

In [102]: for i in range(len(M)):
              p = len(M[i])
              print('|',end='')
              for j in range(p - 1):
                  print(M[i][j], end=' , ')
              print(M[i][p - 1], end='|\n')

|1 , 2 , 3|
|4 , 5 , 6|
```

2.2.1 Exercice 4

```
In [9]: def dimensions(M):
            return (len(M), len(M[0]))

In [10]: d = dimensions([[0,1,2], [3,4,5]])

In [11]: d

Out[11]: (2, 3)

In [12]: d[0]

Out[12]: 2

In [13]: d[1]

Out[13]: 3

In [14]: type(d)

Out[14]: tuple
```

```
In [15]: d[0] = 4

         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-15-92a878cd05cb> in <module>()
     ----> 1 d[0] = 4


         TypeError: 'tuple' object does not support item assignment
```

2.3 Slicing

```
In [2]: L = [i for i in range(10)]

In [12]: L

Out[12]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [13]: L[2] #élément en position 3

Out[13]: 2

In [14]: #dernier élément
         L[len(L) - 1]

Out[14]: 9

In [3]: L

Out[3]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

On peut indexer les éléments à partir de la fin Le dernier a pour index -1 Le premier cad L[0] a pour index -len(L)

```
In [16]: L[-1]

Out[16]: 9

In [17]: L[-len(L)]

Out[17]: 0
```

Découpage de tranches ou slicing

```
In [18]: L

Out[18]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [19]: # la tranche des éléments de L entre l'index 2
         #et l'index 5 exclu, c'est une liste
         L[2:5]

Out[19]: [2, 3, 4]

In [21]: #trois derniers éléments
         L[-3:]

Out[21]: [7, 8, 9]

In [22]: #les trois premiers
         L[:3]

Out[22]: [0, 1, 2]

In [23]: #tous sauf les trois premiers et les trois derniers
         L[3:-3]

Out[23]: [3, 4, 5, 6]

In [25]: #les trois premiers suivis des trois derniers
         L[:3] + L[-3:]

Out[25]: [0, 1, 2, 7, 8, 9]
```

2.3.1 Exercice 5

```
In [105]: # dix derniers éléments de t3
          t3[-10:]

Out[105]: [952763904,
           955671625,
           961504803,
           967361669,
           970299000,
           976191488,
           982107784,
           985074875,
           991026973,
           997002999]

In [106]: # éléments de t3 sauf les 250 premiers et les 250 derniers
          t3[250:-250]

Out[106]: [72511713,
           73560059,
           74088000,
           75151448,
           76225024,
```

76765625,
77854483,
78953589,
79507000,
80621568,
81746504,
82312875,
83453453,
84604519,
85184000,
86350888,
87528384,
88121125,
89314623,
90518849,
91125000,
92345408,
93576664,
94196375,
95443993,
96702579,
97336000,
98611128,
99897344,
100544625,
101847563,
103161709,
103823000,
105154048,
106496424,
107171875,
108531333,
109902239,
110592000,
111980168,
113379904,
114084125,
115501303,
116930169,
117649000,
119095488,
120553784,
121287375,
122763473,
124251499,
125000000,
126506008,
128024064,

```
128787625,
130323843,
131872229,
132651000,
134217728,
135796744,
136590875,
138188413,
139798359,
140608000,
142236648,
143877824,
144703125,
146363183,
148035889,
148877000,
150568768,
152273304,
153130375,
154854153,
156590819,
157464000,
159220088,
160989184,
161878625,
163667323,
165469149,
166375000,
168196608,
170031464,
170953875,
172808693,
174676879,
175616000,
177504328,
179406144,
180362125,
182284263,
184220009,
185193000,
187149248,
189119224,
190109375,
192100033,
194104539,
195112000,
197137368]
```

```
In [107]: # cinq premiers éléments de t4 suivie des cinq derniers de t4
          t4[:5]+t4[-5:]

Out[107]: [841.0,
           308.3333333333333,
           130.77777777777777,
           71.59259259259258,
           51.86419753086419,
           42.000018561227925,
           42.000006187075975,
           42.00000206235866,
           42.00000068745289,
           42.00000022915096]
```

2.4 Modification, ajout et suppression d'éléments

2.4.1 Ajout d'un élément, extension, insertion

```
In [108]: li = [5, 6, 9, 12]
          id(li)

Out[108]: 3025647820

In [109]: li.append(-14)

In [110]: li

Out[110]: [5, 6, 9, 12, -14]

In [111]: id(li) #l'adresse mémoire de li n'est pas modifiée par append

Out[111]: 3025647820

In [112]: lj = [7, 8, 10]
          id(lj)

Out[112]: 3025646284

In [113]: lj = lj + [14]

In [114]: lj

Out[114]: [7, 8, 10, 14]

In [115]: id(lj) #une nouvelle adresse mémoire a été affectée à lj

Out[115]: 3025638028

In [116]: li.extend(lj)

In [117]: li
```

```
Out[117]: [5, 6, 9, 12, -14, 7, 8, 10, 14]

In [118]: id(li)  #l'adresse mémoire de li n'est pas modifiée par extend

Out[118]: 3025647820

In [119]: li[:4]

Out[119]: [5, 6, 9, 12]

In [120]: lj = li[:4] + lj

In [121]: lj

Out[121]: [5, 6, 9, 12, 7, 8, 10, 14]

In [122]: lj.insert(1, 13) #insertion de 13 en position 1

In [123]: lj

Out[123]: [5, 13, 6, 9, 12, 7, 8, 10, 14]

In [124]: lj[3:4] = [100]  #insertion de 100 à la place de 9 en position 3

In [125]: lj

Out[125]: [5, 13, 6, 100, 12, 7, 8, 10, 14]

In [126]: lj[3:3] = [99]  #insertion de 99 en position 3

In [127]: lj

Out[127]: [5, 13, 6, 99, 100, 12, 7, 8, 10, 14]
```

2.4.2 Suppression

```
In [128]: L = [42, "BCPST", 2 < 3, 3.14, [5, 4]]

In [129]: del L[2] #suppression de l'élément en position 2

In [130]: L

Out[130]: [42, 'BCPST', 3.14, [5, 4]]

In [131]: L[1:2] = []  #suppression de l'élément en position 1

In [132]: L

Out[132]: [42, 3.14, [5, 4]]

In [133]: val = L.pop(1)    #suppression de l'élément en position 1 et récupération

In [134]: val

Out[134]: 3.14

In [135]: L

Out[135]: [42, [5, 4]]
```

2.4.3 Les listes, des objets itérables

```
In [17]: L = [i for i in range(851, 854)]
```

Parcours sur les index

```
In [18]: for i in range(len(L)):
             print(i, L[i])
```

```
0 851
1 852
2 853
```

Parcours sur les valeurs

```
In [19]: for element in L:
             print(element)
```

```
851
852
853
```

Carrés de L

```
In [20]: [L[i] ** 2  for i in range(len(L))]
```

```
Out[20]: [724201, 725904, 727609]
```

```
In [21]: [ element ** 2 for element in L]
```

```
Out[21]: [724201, 725904, 727609]
```

2.4.4 Exercice 6

```
In [136]: t5 = [t2[2*i+1] for i in range(3)]

In [137]: # t5 contient les trois premiers éléments de t2 d'indice impair
          t2, t5

Out[137]: ([9, 4, 1, 0, 1, 4, 9, 16, 25], [4, 0, 4])

In [138]: t2[1:7:2] #trois premiers éléments de t2 d'indice impair par slicing

Out[138]: [4, 0, 4]

In [139]: t6 = [x**2 for x in t2]

In [140]: # t6 contient les carrés des éléments de t2
          t6
```

```
Out[140]: [81, 16, 1, 0, 1, 16, 81, 256, 625]

In [141]: t7 = [(x,y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

t7 avec Pythontutor

http://www.pythontutor.com/visualize.html#code=t7+%3D+%5B(x,y%29+for+x+in+%5B1,2,3%5D+for+y+
frontend.js&cumulative=false&heapPrimitives=false&textReferences=false&py=3&rawInputLstJSON=%5B%5D

t7 contient les couples (i,j) avec i in [1,2,3] et j in [3,1,4] et i!=j

2.4.5 Exercice 7 Listes en compréhension ou slicing ?

Liste des éléments d'indice pair de t1

```
In [142]: t1

Out[142]: [10, 30, 42, 2, 17, 5, 30, -20]

In [143]: # Par slicing
          t1[::2]

Out[143]: [10, 42, 17, 30]

In [144]: #Par liste en compréhension
          [t1[i]  for i in range(0, len(t1), 2)]

Out[144]: [10, 42, 17, 30]

In [ ]: [t1[i]  for i in range(0, len(t1)) if i%2 == 0]

In [ ]: L = []
        for i in range(0, len(t1)):
            if i%2 == 0:
                L.append(t1[i])
```

Liste des termes pairs de t1

```
In [145]: #par liste en compréhension
          [e for e in t1 if e%2 == 0]

Out[145]: [10, 30, 42, 2, 30, -20]

In [146]: #avec append
          pair = []
          for e in t1:
              if e % 2 == 0:
                  pair.append(e)
          print(pair)

[10, 30, 42, 2, 30, -20]
```

2.4.6 Exercice 8

```
In [ ]: def somme1(t):
            s = 0
            for i range(len(t)):
                s = s + t[i]
            return s
```

```
In [23]: def somme(tab):
             """somme des éléments d'un tableau, redéfinition de sum"""
             s = 0
             for terme in tab:
                 s += terme
             return s
```

```
In [148]: somme(t1)

Out[148]: 116
```

```
In [22]: def moyenne(tab):
             """moyenne des éléments d'un tableau sous forme de flottant"""
             return float(somme(tab))/len(tab)
```

```
In [150]: moyenne(t2)

Out[150]: 7.666666666666667
```

moyenne des carrés de t2

```
In [24]: moyenne([e**2 for e in t2])

Out[24]: 119.66666666666667
```

Racine carré

```
In [26]: from math import sqrt
         sqrt(2)

Out[26]: 1.4142135623730951

In [27]: 2 ** 0.5

Out[27]: 1.4142135623730951

In [28]: 2 ** (1/2)

Out[28]: 1.4142135623730951

In [29]: 2**1/2

Out[29]: 1.0
```

```
In [25]: def ecart_type1(tab):
             """écart-type des éléments d'un tableau avec la  formule de Konig"""
             tabcarre = [x**2 for x in tab]
             return (moyenne(tabcarre)-moyenne(tab)**2)**0.5

In [152]: ecart_type1(t2)

Out[152]: 7.803133273813083

In [153]: def ecart_type2(tab):
              """écart-type des éléments d'un tableau avec la définition et sans li
              m = moyenne(tab) # m est un flottant
              s = 0. #s sera un flottant
              # s recevra la somme des carrés des écarts à la moyenne
              for x in tab:
                  s = s +(x - m)**2
              return (s/len(tab))**0.5

In [154]: ecart_type2(t2)

Out[154]: 7.803133273813083
```

2.4.7 Exercice 9

```
In [26]: def maximum(tab):
             """retourne le maximum d'un tableau, redéfinition de max"""
             m = tab[0]
             for terme in tab[1:]: # ou for k in range(1, len(tab))
                 if terme > m:     # ou if tab[k] > m
                     m = terme    # ou m = tab[k]
             return m

         def maximum2(tab):
             """retourne le maximum d'un tableau, redéfinition de max"""
             m = tab[0]
             for i in range(1, len(tab)):
                 if tab[i] > m:
                     m = tab[i]
             return m

In [27]: def position_maximum(tab):
             """retourne la première position où le maximum est atteint"""
             pos,maxi = 0,tab[0]
             for i in range(1, len(tab)):
                 if tab[i] >= maxi: # si on met une inégalité large, la fonction re
                     pos,maxi = i,tab[i]
             return pos

In [157]: def liste_positions_maximum(tab):
              """retourne le maximum et la liste des positions où il est atteint"""
```

```
            tmaxi, maxi = [0],tab[0]
            for i in range(1,len(tab)):
                if tab[i] > maxi:
                    tmaxi,maxi = [i],tab[i]
                elif tab[i] == maxi:
                    tmaxi.append(i)
            return tmaxi,maxi
```

```
In [158]: maximum([3,4,4,2])
```

```
Out[158]: 4
```

```
In [159]: position_maximum([3,4,4,2])
```

```
Out[159]: 2
```

```
In [160]: liste_positions_maximum([3,4,4,2])
```

```
Out[160]: ([1, 2], 4)
```

2.4.8 Exercice 10

```
In [161]: def croissante(t):
              """Retourne un booleen indiquant si une liste est croissante"""
              for k in range(len(t) - 1):
                  if t[k] > t[k + 1]:
                      return False
              return True
```

```
In [162]: import operator  #module permettant de récupérer les opérateurs de base s

          def monotone(t):
              """Retourne un booleen indiquant si une liste est monotone"""
              assert len(t) >= 2, "La liste doit contenir au moins deux éléments"
              #on choisit la fonction de comparaison selon l'ordre des deux premier
              if t[0] < t[1]:
                  comparaison = operator.gt
              else:
                  comparaison = operator.lt
              for k in range(1, len(t) - 1):
                  if comparaison(t[k], t[k + 1]):
                      return False
              return True
```

```
In [163]: t = [k for k in range(10)]
          print(t)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [164]: croissante(t)

Out[164]: True

In [165]: t[::-1]   #inversion d'une liste par slicing

Out[165]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

In [166]: croissante(t[::-1] )

Out[166]: False

In [167]: monotone(t)

Out[167]: True

In [168]: monotone([0])

        ---------------------------------------------------------------------------
        AssertionError                            Traceback (most recent call last)

        <ipython-input-168-4a0d57b15642> in <module>()
    ----> 1 monotone([0])


        <ipython-input-162-36c599606821> in monotone(t)
          3 def monotone(t):
          4     """Retourne un booleen indiquant si une liste est monotone"""
    ----> 5     assert len(t) >= 2, "La liste doit contenir au moins deux éléments"
          6     #on choisit la fonction de comparaison selon l'ordre des deux premi
          7     if t[0] < t[1]:


        AssertionError: La liste doit contenir au moins deux éléments
```

2.4.9 Exercice 11 Une suite fortement récurrente, $u_0 = 1$ et $u_{n+1} = \sum_{k=0}^{n}(n+k)u_k$

Une première version du calcul des termes u_n où l'on mémorise les termes déjà calculés dans une liste.

```
In [169]: def suite_exo11V1(n):
              u = [1]
              for i in range(1, n + 1):
                  s = 0
                  j = i - 1
                  for k in range(0, i):
                      s = s + (j + k)*u[k]
                  u.append(s)
              return u[-1]
```

```
In [170]: [suite_exo11V1(n) for n in range(6)]

Out[170]: [1, 0, 1, 6, 44, 404]
```

Une seconde version où l'on mémorise simplement le terme précédent u_n, la somme $\sum_{k=0}^{n} u_k$ et la somme pondérée $\sum_{k=0}^{n} k u_k$.

```
In [ ]: def suite_exo11V2(n):
            somme = 1
            sommepond = 0
            u = 1
            for i in range(1, n + 1):
                u = (i - 1)*somme + sommepond
                somme = somme + u
                sommepond = sommepond + i*u
            return u

In [ ]: [suite_exo11V2(n) for n in range(6)]
```

2.5 Pour compléter

2.5.1 Exercice 12

```
In [ ]: def compartab(t1, t2):
            """Compare si deux tableaux sont égaux"""
            taille1 = len(t1)
            if taille1 != len(t2):
                return False
            for k in range(taille1):
                if t1[k] != t2[k]:
                    return False
            return True

In [ ]: def compartab2(t1, t2):
            """Compare si deux tableaux sont égaux avec l'itérateur zip"""
            if len(t1) != len(t2):
                return False
            for e1, e2 in zip(t1, t2):
                if e1 != e2:
                    return False
            return True

In [171]: from random import randint
          alea1 = [randint(0, 20) for _ in range(10)]
          alea2= [randint(0, 20) for _ in range(10)]

In [172]: alea1

Out[172]: [16, 13, 15, 16, 17, 16, 3, 17, 3, 18]
```

```
In [173]: alea2

Out[173]: [3, 8, 19, 17, 11, 1, 13, 11, 18, 16]

In [174]: compartab2(alea1, alea2)

Out[174]: False

In [175]: compartab(alea1, alea2)

Out[175]: False

In [176]: compartab([1,0, 1], [1,0,1]), compartab2([1,0, 1], [1,0,1])

Out[176]: (True, True)

In [177]: def inversion(t):
              L = []
              for e in t:
                  L.insert(0, e)
              return L

In [178]: def inversion2(t):
              L = []
              for k in range(len(t) - 1, -1, -1):
                  L.append(t[k])
              return L

In [179]: def inversion3(tab):
              """idem mais avec une liste en compréhension"""
              return [tab[i] for i in range(-1,-len(tab)-1,-1)]

In [180]: inversion(alea1)

Out[180]: [18, 3, 17, 3, 16, 17, 16, 15, 13, 16]

In [181]: inversion2(alea1)

Out[181]: [18, 3, 17, 3, 16, 17, 16, 15, 13, 16]

In [182]: #vérification avec un slicing
          alea1[::-1]

Out[182]: [18, 3, 17, 3, 16, 17, 16, 15, 13, 16]

In [183]: def remplace(t, x, y):
              """Remplace dans t toutes les occurences de x par y"""
              for k in range(len(t)):
                  if t[k] == x:
                      t[k] = y
```

```
In [184]: alea1

Out[184]: [16, 13, 15, 16, 17, 16, 3, 17, 3, 18]

In [185]: remplace(alea1, 4, 7)

In [186]: alea1

Out[186]: [16, 13, 15, 16, 17, 16, 3, 17, 3, 18]

In [187]: def remplace2(t, x, y):
              """Remplace dans t toutes les occurences de x par y
              Retourne une nouvelle liste"""

              def auxiliaire(e):
                  """Fonction auxiliaire"""
                  if e != x:
                      return e
                  return y

              return list(map(auxiliaire, t))

In [188]: def remplace3(tab,x,y):
              """Remplace dans t toutes les occurences de x par y
              Retourne une nouvelle liste"""
              return [element if element != x else y for element in tab]

In [189]: alea1

Out[189]: [16, 13, 15, 16, 17, 16, 3, 17, 3, 18]

In [190]: remplace2(alea1, 20, 0)

Out[190]: [16, 13, 15, 16, 17, 16, 3, 17, 3, 18]
```

3 Référencement et copie de listes

3.1 Référencement, aliasing

3.1.1 Exercice 13 Lien vers PythonTutor

http://pythontutor.com/visualize.html#code=k%20%3D%20%5B10,%2015,%2012%5D%0At%20%3D%20k%0A
12%5D%0Ali2%20%3D%20li1%5B%3A%5D%0Ali1%5B1%5D%5B1%5D%20%3D%2010&cumulative=false&curI
frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

En Python, on peut considérer que les variables de types simples (`int`, `bool`, `float`) sont bien l'association d'un nom et d'une valeur.

Les variables de type `list` sont différentes. Dans un autre langage comme le `C`, on dirait qu'il s'agit de **pointeurs** c'est-à-dire de variables dont les valeurs sont des adresses de zone mémoires où sont stockées les valeurs proprement dites. On parle aussi de **référence** pour la valeur d'une liste et ce type de mécanisme est appelé **indirection**.

Par exemple lorsqu'on écrit `L = [1, 3.14, [2, 3]]`, la valeur de L n'est pas `[1, 3.14, [2, 3]]` mais l'adresse de la zone mémoire où est stockée `[1, 3.14, [2, 3]]`

Ainsi lorsqu'on assigne la valeur de `L` à une autre liste `T`, on donne à `T` l'adresse mémoire associée à `L`, celle qui pointe vers la zone mémoire où est stockée `[1, 3.14, [2, 3]]`. Cette dernière peut être vue comme une série d'octets contigus où sont stockés l'entier `4`, le flottant `3.14` et la valeur de la liste `[2, 3]`. Mais la valeur de la liste `[2, 3]` est elle même une adresse mémoire, celle de la zone mémoire contigue où sont stockés les entiers `2` et `3`.

Avec les listes de listes, on peut donc avoir plusieurs niveaux d'indirections imbriquées les unes dans les autres. Il faut donc utiliser la fonction `deepcopy` du module `copy` pour réaliser une vraie copie (dite **copie profonde**), en cassant les références des listes (on dit aussi **déréférencer**) et en créant de nouvelles zones mémoires pour stocker les mêmes données que la liste de listes source.

```
In [1]: L = [[851, 852], [853, 854]]
        M = L

In [3]: L[0][1], M[0][1]

Out[3]: (852, 852)

In [4]: L[0][1] = 833

In [5]: L, M

Out[5]: ([[851, 833], [853, 854]], [[851, 833], [853, 854]])

In [6]: N = L[:]

In [9]: L[0][1] = 854

In [10]: L, N

Out[10]: ([[851, 854], [853, 854]], [[851, 854], [853, 854]])

In [12]: from copy import deepcopy
         P = deepcopy(M)

In [13]: P, M

Out[13]: ([[851, 854], [853, 854]], [[851, 854], [853, 854]])

In [14]: M[0][1] = 855

In [15]: M, P

Out[15]: ([[851, 855], [853, 854]], [[851, 854], [853, 854]])
```

3.2 Copie de listes

Lien vers l'exemple ci-dessous sur PythonTutor :
http://pythontutor.com/visualize.html#code=L+%3D+%5B0,0,0%5D*4%0AN+%3D+%5B%5B0,0,0%5D%5D*
frontend.js&cumulative=false&heapPrimitives=false&drawParentPointers=false&textReferences=false&showO

```
In [16]: L = [0,0,0]*4
         K = [[0,0,0]] + [[0,0,0]] + [[0,0,0]]  + [[0,0,0]]
         N = [[0,0,0]]*4
         N[1][1] = 2
         M = [[0,0,0] for i in range(4)]
         M[1][1] = 2
         P = [[0 for i in range(3)] for j in range(4)]
         P[1][1] = 2
         Q = [[0]*3 for j in range(4)]

         R = []
         for nligne in range(4):
             ligne = []
             for ncol in range(3):
                 ligne.append(0)
             R.append(ligne)

         S = []
         for nligne in range(4):
             S = S + [0]*3
```

Lien vers un autre exemple sur PythonTutor :
http://pythontutor.com/visualize.html#code=R+%3D+%5B0,0,0%5D%0AS+%3D+%5BR%5D+%2B+%5BR%
frontend.js&cumulative=false&heapPrimitives=false&drawParentPointers=false&textReferences=false&showO

3.2.1 Exercice 14

```
In [192]: def zeros(n,p):
              mat = []
              for nligne in range(n):
                  ligne = []
                  for ncol in range(p):
                      ligne.append(0)
                  mat.append(ligne)
              return mat

          def zeros2(n,p):
              """Retourne la matrice nulle de n lignes et p colonnes"""
              return [[0]*p for i in range(n)]

          def zeros3(n,p):
              """Mauvaise fonction car on réplique toujours
```

```
            la meme ligne"""
            return [[0]*p]*n

In [193]: a = zeros3(2,3)  #mauvaise  copie
          a

Out[193]: [[0, 0, 0], [0, 0, 0]]

In [194]: a[0][0] = 1

In [195]: a

Out[195]: [[1, 0, 0], [1, 0, 0]]

In [196]: b = zeros(2,3)  #bonne  copie
          b

Out[196]: [[0, 0, 0], [0, 0, 0]]

In [197]: b[0][0] = 1
          b

Out[197]: [[1, 0, 0], [0, 0, 0]]
```

3.2.2 Exercice 15 Copie de matrices (sans deepcopy)

Lien vers un exemple sur PythonTutor :
http://pythontutor.com/visualize.html#code=R+%3D+%5B0,0,0%5D%0AS+%3D+%5BR%5D+%2B+%5BR%
frontend.js&cumulative=false&heapPrimitives=false&drawParentPointers=false&textReferences=false&showOr

```
In [198]: def copie0(mat):
              """Vraie copie de mat sans partage de données"""
              n, p = dimensions(mat)
              mat2 = []
              for nligne in range(n):
                  ligne = []
                  for ncol in range(p):
                      ligne.append(0)
                  mat2.append(ligne)
              return mat2


          def copie01(mat):
              """Vraie copie de mat sans partage de données"""
              n, p = dimensions(mat)
              mat2 = zeros2(n, p )
              for i in range(n):
                  for j in range(p):
                      mat2[i][j] = mat[i][j]
              return mat2
```

```
def copie(m):
    """retourne une copie de la matrice m"""
    nlines,ncols = dimensions(m)
    n = zeros(nlines,ncols) # matrice copie
    for i in range(nlines):
        for j in range(ncols):
            n[i][j] = m[i][j]
    return n


def copie1(m):
    """copie de matrice avec une liste en compréhension"""
    return [[m[i][j] for j in range(len(m[0]))] for i in range(len(m))]


def copie2(m):
    """retourne une copie de la matrice m"""
    n = [] #matrice copie
    nlines,ncols = dimensions(m)
    for i in range(nlines):
        ligne = [] #vecteur ligne
        for j in range(ncols):
            ligne.append(m[i][j])
        n.append(ligne)
    return n
```

4 D'autres exercices

4.0.1 Exercice 16 Opérations sur les matrices

```
In [199]: def dimensions(M):
              return (len(M), len(M[0]))

          def somme_matrice(m,n):
              """retourne la matrice somme de deux matrices m et n"""
              (nlines, ncols) = dimensions(m)
              if dimensions(n) != (nlines, ncols):
                  return None
              s = zeros(nlines,ncols)
              for i in range(nlines):
                  for j in range(ncols):
                      s[i][j] = m[i][j]+n[i][j]
              return s

          def somme_matrice1(m,n):
```

```
    """retourne la matrice somme de deux matrices m et n"""
    assert dimensions(m)==dimensions(n), "Les matrices n'ont pas la même
    return [[m[i][j]+n[i][j] for j in range(len(m[0]))] for i in range(le


def somme_matrice2(m,n):
    """retourne la matrice somme de deux matrices m et n"""
    assert dimensions(m)==dimensions(n), "Les matrices n'ont pas la même
    s = []
    for i in range(len(m)):
        ligne = []
        for j in range(len(m[0])):
            ligne.append(m[i][j]+n[i][j])
        s.append(ligne)
    return s

def multscal_matrice(m,t):
    """multiplie tous les coefficients de la matrice m par le scalaire t'
    nlines,ncols=dimensions(m)
    s=zeros(nlines,ncols)
    for i in range(nlines):
        for j in range(ncols):
            s[i][j]=m[i][j]*t
    return s

def multscal_matrice1(m,t):
    """multiplie tous les coefficients de la matrice m par la scalaire t'
    return [[m[i][j]*t for j in range(len(m[0]))] for i in range(len(m))]

def transpose(m):
    """retourne la transposée d'une matrice"""
    nlines,ncols = dimensions(m)
    s = zeros(ncols,nlines)
    for i in range(ncols):
        for j in range(nlines):
            s[i][j] = m[j][i]
    return s

def transpose1(m):
    """retourne la transposée d'une matrice"""
    return [[m[i][j] for i in range(len(m))] for j in range(len(m[0]))]


def prod_matrice(m,n):
    """retourne la matrice poduit de m par n"""
    mlines,mcols = dimensions(m)
    nlines,ncols = dimensions(n)
    if mcols != nlines:
```

```
        return "Les dimensions ne sont pas compatibles"
    p = zeros(mlines,ncols)
    for i in range(mlines):
        for j in range(ncols):
            #p[i][j] = 0
            for k in range(mcols): # ou nlines c'est pareil
                #p[i][j] = p[i][j] + m[i][k]*n[k][j]
                p[i][j] += m[i][k]*n[k][j]
    return p


def prod_matrice1(m,n):
    """retourne la matrice poduit de m par n"""
    assert dimensions(m)[1]==dimensions(n)[0], "Les dimensions ne sont pa
    return [[sum([m[i][k]*n[k][j] for k in range(len(m[0]))]) for j in ra

def prod_matrice2(m,n):
    """retourne la matrice poduit de m par n"""
    assert dimensions(m)[1]==dimensions(n)[0], "Les dimensions ne sont pa
    p = []
    for i in range(len(m)):
        ligne = []
        lignem = m[i] #on stocke la ligne de m dans un pointeur
        for j in range(len(n[0])):
            coef = 0
            for k in range(len(m[0])):
                coef += lignem[k]*n[k][j]
            ligne.append(coef)
        p.append(ligne)
    return p

def diagonale(n,x):
    """Retourne une matrice diagonale avec que des x sur la diagonale"""
    return [[x if i == j else 0 for j in range(n)] for i in range(n)]


def puissance_mat(m,exposant):
    """Retourne la puissance de la matrice m d'exposant donné"""
    nlines,ncols = dimensions(m)
    assert nlines==ncols,"La matrice doit être carré"
    p = diagonale(nlines,1)
    for i in range(exposant):
        p = prod_matrice2(p,m)
    return p

## Exercice 14
def est_symetrique(m):
    """Retourne un booleen indiquant si la matrice m est symetrique"""
```

```
            n, p = dimensions(m)
            if n != p:
                return False
            for i in range(n):
                for j in range(i):
                    if m[i][j] != m[j][i]:
                        return False
            return True

        def est_symetrique2(m):
            n, p = dimensions(m)
            if n != p:
                return False
            return m == transposition(m)
```

4.1 Exercice 17

```
In [200]: def sommevect1(t1, t2):
              taille1 = len(t1)
              if  taille1 != len(t2):
                  return None
              t = []
              for k in range(taille1):
                  t.append(t1[k] + t2[k])
              return t

In [201]: def sommevect2(t1, t2):
              assert len(t1) == len(t2), "les vecteurs doivente etre de meme longue
              return [e1 + e2 for (e1, e2) in zip(t1, t2)]

In [202]: def sommevect3(t1, t2):
              assert len(t1) == len(t2), "les vecteurs doivente etre de meme longue
              return list(map(sum, zip(t1, t2)))

In [203]: sommevect1([1,2], [3,4]), sommevect2([1,2], [3,4]), sommevect3([1,2], [3,

Out[203]: ([4, 6], [4, 6], [4, 6])
```

4.1.1 Exercice 18 Sommes cumulées

```
In [212]: def sommes_cumulees(t):
              """Complexité linéraire par rapport à la taille de la liste t"""
              taille = len(t)
              cumul = [t[0]] + [0]*(taille - 1)
              for k in range(1, taille):
                  cumul[k] = cumul[k - 1] + t[k]
              return cumul

In [209]: t1
```

```
Out[209]: [10, 30, 42, 2, 17, 5, 30, -20]

In [211]: sommes_cumulees(t1)

Out[211]: [10, 40, 82, 84, 101, 106, 136, 116]
```

4.1.2 Exercice 19

```
In [253]: def deux_plus_grosV1(t):
              """Retourne les deux plus grands elements du tableau/liste t.
              t est supposé avoir au moins 2 éléments"""
              taille = len(t)
              if taille < 2:
                  return None
              #on initialise gros avec les deux premiers éléments dans l'ordre déc
              if t[0] > t[1]:
                  gros = t[:2]
              else:
                  gros = t[1::-1]
              #on parcourt le tableau à partir de la position 2 (troisieme element)
              for k in range(2, len(t), 1):
                  #on insère l'élément courant à sa place dans la liste des deux p
                  courant = t[k]
                  j = 0
                  while j < 2 and gros[j] > courant:
                      j += 1
                  if j == 0:
                      gros = [courant,  gros[0]]
                  elif j == 1:
                      gros = [gros[0], courant]
              return gros

In [254]: deux_plus_grosV1([12, 4, 12, 1, 7, 3, 12,5])

Out[254]: [12, 12]

In [286]: def deux_plus_gros_rec(t):
              """Retourne les deux plus grands elements du tableau/liste t.
              t est supposé avoir au moins 2 éléments.  Fonction récursive"""
              taille = len(t)
              if taille < 2:
                  return None
              if taille == 2:
                   if t[0] > t[1]:
                      return t[:2]
                   elif t[0] < t[1]:
                      return  t[1::-1]
                   else:
                      return t[:1]
```

```
            else:
                gros = deux_plus_gros_rec(t[:-1])
                j = 0
                courant = t[-1]
                while j < len(gros) and gros[j] >= courant:
                    j += 1
                if j == 0:
                    return [courant,  gros[0]]
                elif j == len(gros) - 1 and courant != gros[0]:
                    return [gros[0], courant]
                else:
                    return gros
```

```
In [279]: def deux_plus_grosV2(t):
            """Retourne les deux plus grands elements du tableau/liste t.
            t est supposé avoir au moins 2 éléments.
            Dans cette version, on insère le sdecond plus gros uniquement s'il es
            taille = len(t)
            if taille < 2:
                return None
            #on initialise gros avec les deux premiers éléments dans l'ordre stri
            if t[0] > t[1]:
                gros = t[:2]
            elif t[0] < t[1]:
                gros = t[1::-1]
            else:
                gros = t[:1]
            #on parcourt le tableau à partir de la position 2 (troisieme element)
            for k in range(2, len(t), 1):
                #on insère l'élément courant à sa place dans la liste des deux pl
                courant = t[k]
                j = 0
                while j < len(gros) and gros[j] >= courant:
                    j += 1
                if j == 0:
                    gros = [courant,  gros[0]]
                elif j == len(gros) - 1 and courant != gros[0]:
                    gros = [gros[0], courant]
            return gros
```

```
In [285]: deux_plus_grosV2([12, 4, 12, 1, 7, 3, 12,5])

Out[285]: [12, 7]
```

Si tous les éléments sont identiques, la liste renvoyée est de taille 1.

```
In [276]: deux_plus_grosV2([12, 12, 12])

Out[276]: [12]
```

```
In [284]: deux_plus_gros_rec([12, 4, 12, 1, 7, 3, 12,5])

Out[284]: [12, 7]

In [283]: deux_plus_gros_rec([12, 12, 12])

Out[283]: [12]
```

4.1.3 Exercice 20

```
In [214]: def permutelignes(A,i,j):
              """permute les lignes i et j de la matrice m"""
              nlines,ncols = dimensions(A)
              assert repr(type(i))==repr(type(j))=="<class 'int'>" and 0<=i<nlines
              "i et j doivent être des entiers compris entre 0 et le nombre de lign
              A[i],A[j] = A[j],A[i]

In [206]: def rempl(A,x,y):
              """remplace toutes les occurences de x dans A par y"""
              nlines,ncols = dimensions(A)
              for i in range(nlines):
                  for j in range(ncols):
                      if A[i][j] == x:
                          A[i][j] = y
```

4.1.4 Exercice 21

```
In [217]: def pascal(n):
              """Retourne les n premières lignes du triangle de Pascal"""
              triangle = [[1]]
              for i in range(1, n):
                  ligne = [1]
                  for j in range(1, i):
                      ligne.append(triangle[i-1][j] + triangle[i-1][j-1] )
                  ligne.append(1)
                  triangle.append(ligne)
              return triangle

In [219]: pascal(5)

Out[219]: [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```