TDO-seance2_exos

September 19, 2018

1 L3 E - S5 2018-19 - TDO - séance 2, solutions des exercices

1.1 Exercice 8 : années bissextiles

Ecrire une fonction booléenne `bissectile(n)` qui renvoie 1 si n est une année bissectile et 0 sinon. On rappelle que les années bissextiles reviennent tous les 4 ans, sauf lorsqu'elles sont un multiple de cent qui n'est pas un multiple de 400.

```
In [1]: def bissextile(n):
            if n%4 != 0:
                return False
            elif n %100 != 0:
                return True
            elif n %400 != 0:
                return False
            else:
                return True
```

```
In [2]: bissextile(1546),bissextile(1548), bissextile(1700),bissextile(1600)
```

```
Out[2]: (False, True, False, True)
```

On notera que l'usage des connecteurs logiques permet des formes beaucoup plus compactes :

```
In [3]: def bissextile2(n):
            return ( (n==4==0)  and ( n% 100 != 0)) or (n%400==0)
```

```
In [4]: bissextile2(1546),bissextile2(1548), bissextile2(1700),bissextile2(1600)
```

```
Out[4]: (False, False, False, True)
```

1.2 Exercice 9 : le crible d'Eratosthène

Ecrire une liste qui contient les nombres premiers inférieurs à 100, obtenus en appliquant la méthode du crible d'Eratosthène.

```
In [5]: K=[n for n in range(2,100)]
        L=[]
        i=1
        while len(K)>0:
                i=K[0]
                L.append(i)
                K.remove(i)
                for j in K:
                    if j%i==0:
                        K.remove(j)
        print (K)
        print (L)
```

```
[]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

1.3 Exercice 10 : somme de carrés

Ecrire une fonction qui, pour tout M réel postif, retourne le couple (n, m) où n est le plus petit entier tel que $m = 1 + 2^2 + 3^2 + ... + n^2 >= M$.

```
In [6]: def somme_des_carres_plus_grande_que(x):
            m,n=1,1
            while m<x:
                n=n+1
                m=m+n**2
            return(m,n)
```

```
In [7]: print(somme_des_carres_plus_grande_que(5))
        print(somme_des_carres_plus_grande_que(18))
        print(somme_des_carres_plus_grande_que(10000))
```

```
(5, 2)
(30, 4)
(10416, 31)
```

1.4 Exercice 11 : présent ou pas

Ecrire une fonction `present(x,L)` qui retourne `True` si x est dans L et `False` sinon (et qui vérifie que le second argument est bien une liste)

```
In [11]: def present(x,L):
             if type(L) != list:                ## si la condition est vérifiée, on fait...
                 return("Le second argument doit être une liste !!")
             else:
                 for a in L:
                     if x==a:
```

```
                    return True
            return False
```

```
In [12]: a=[23,'fr','an',6,4.5]+[x for x in range(50) if x % 2 == 0]
         print(a)
         b=(1,4,9)
         print(b)
```

```
[23, 'fr', 'an', 6, 4.5, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36,
(1, 4, 9)
```

```
In [13]: present(12,a)
```

```
Out[13]: True
```

```
In [14]: present(12,b)
```

```
Out[14]: 'Le second argument doit être une liste !!'
```

```
In [23]: def present2(x,L):
             if type(L) != list:                ## si la condition est vérifiée, on fait...
                 return("Le second argument doit être une liste !!")
             else:
                 if x in L:
                     return True
                 else:
                     return False
```

```
In [28]: print(present2(44,a))
         print(present2(43,a))
         print(present2(44,b))
```

```
True
False
Le second argument doit être une liste !!
```

1.5 Exercice 12 : présence d'une lettre dans un mot

Ecrire une fonction `presente(a,M)` qui retourne l'indice de la lettre a dans la chaîne M si a est dans M, qui renvoie -1 sinon et qui vérifie que le second argument est bien une chaîne de caractères.

```
In [7]: def presente(lettre,mot):
            indice=0
            if type(mot) != str:                ## si la condition est vérifiée, on fait...
                return("Le second argument doit être une chaîne de caractères !!")
            while indice < len(mot):
                if mot[indice]==lettre:
                    return indice
                indice += 1
            return -1
```

```
In [9]: presente('a','TRECFauia')

Out[9]: 5

In [14]: presente('a',456)

Out[14]: 'Le second argument doit être une chaîne de caractères !!'
```

1.6 Exercice 13 : écriture en base 16

Ecrire une fonction `H(n)` qui renvoie l'écriture en base 16 de l'entier n (les chiffres retenus pour l'écriture en base 16 sont $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$).

```
In [15]: def H(n):
             resultat=' '
             while n >0:
                 m=n%16
                 if m <10:
                     m=str(m)
                 elif m==10:
                     m='A'
                 elif m==11:
                     m='B'
                 elif m==12:
                     m='C'
                 elif m==13:
                     m='D'
                 elif m==14:
                     m='E'
                 elif m==15:
                     m='F'
                 resultat=m + resultat
                 n=n//16
             return resultat

In [16]: H(4),H(18),H(14),H(23*456)

Out[16]: ('4 ', '12 ', 'E ', '28F8 ')
```

1.7 Exercice 14 : Test de primalité

Ecrire une fonction `est_premier(N)` qui retourne `True` si l'entier $N >= 2$ est premier et `False` sinon en testant si N admet un diviseur parmi les entiers inférieurs ou égaux à sa racine carrée. On utilisera la fonction `sqrt` (qu'il faudra peut-être importer, par exemple en tapant `from math import *`)

```
In [17]: from math import *
         def est_premier(N):
             if  (type(N) != int or N <2):                ## si la condition est vérifiée, on fa
```

```
            return("on demande un entier >=2")
        resultat = True
        for i in range(2,int(sqrt(N))):
            if N%i == 0 :
                resultat = False
        return resultat
```

```
In [18]: est_premier(29)
```

```
Out[18]: True
```

```
In [19]: est_premier(28)
```

```
Out[19]: False
```

1.8 Exercice 15 : décomposition en produit de facteurs premiers

Ecrire une fonction `decomposition_en_facteurs_premiers(N)` qui retourne la décomposition en facteurs premiers de l'entier `N` >= 2 (nota bene : le résultat est attendu sous la forme d'une chaîne de caractères).

```
In [2]: def decomposition_en_facteurs_premiers(N):
            resultat = ''
            NN=N
            p=2
            while p<=NN :
                if (NN%p ==0) :
                    if resultat=='':
                        resultat = str(p)
                    else :
                        resultat = resultat + '*'+str(p)
                    NN=NN/p
                else  :
                    p=p+1
            if resultat=='':

                return str(N)
            else:
                return resultat
```

```
In [3]: decomposition_en_facteurs_premiers(306)
```

```
Out[3]: '2*3*3*17'
```

```
In [6]: decomposition_en_facteurs_premiers(336)
```

```
Out[6]: '2*2*2*2*3*7'
```

```
In [ ]:
```