Boucles et tests

D'après un TP de Stéphane Gonnord

Buts du TP

- Continuer à dompter l'environnement.
- Écrire encore et encore des boucles simples et des tests.

EXERCICE 1 *Créer (au bon endroit) un dossier associé à ce TP. Dans ce dossier, placer une copie du fichier* `cadeau-tp-boucles.py` *fourni dans le dossier partagé de la classe (ou sur le web).*

Lancer Pyzo, sauvegarder immédiatement le fichier du jour au bon endroit. Écrire une commande absurde, de type `print(5*3)` *dans l'éditeur ; sauvegarder et exécuter.*

1 Quelques boucles

EXERCICE 2 *Calculer* $\sum_{k=831}^{944} k^{10}$*, si possible sans regarder le corrigé du tp précédent... mais en le consultant tout de même si la difficulté vous semble insurmontable !*

Le résultat sera copié collé de l'interpréteur vers le fichier `.py`, et commenté. À ce moment du TP, votre feuille de travail dans l'éditeur doit contenir quelque chose comme ci-contre :
On continue par des boucles basiques pour calculer a^b et $n!$

```
# -*- coding: utf-8 -*-

# Exo 1 : Fait !

# Exo 2 :
somme = 0
...
# >>> somme
# 36724191150365100572161020220825L
```

EXERCICE 3 *Calculer* 3^{843} *en appliquant l'algorithme basique suivant :*

$res \leftarrow 1$
pour i *de* 1 *à* 843 **faire**
  $res \leftarrow res \times 3$
Résultat : res
Comparer avec le résultat de `3**843`

EXERCICE 4 *Calculer* 100! *en appliquant l'algorithme suivant :*

$res \leftarrow 1$
pour i *de* 2 *à* 100 **faire**
  $res \leftarrow res \times i$
Résultat : res

Comparer avec le résultat fourni par la fonction `factorial` *de la bibliothèque* `math` *:*

```
>>> import math
>>> math.factorial(...)
```
ou
```
>>> from math import factorial
>>> factorial(...)
```
ou
```
>>> from math import *
>>> factorial(...)
```

EXERCICE 5 1. *On considère la suite* (u_n) *définie par* $\begin{cases} u_0 = 3 \\ u_{n+1} = 3u_n + n \end{cases}$.

Ecrire un script Python qui prend en entrée un entier n et qui retourne le terme de rang n de la suite (u_n)*. Attention ! Il faudra adapter la formule, cf. correction.*

2. *On admet que la suite* (v_n) *définie par* $\begin{cases} v_0 = 1 \\ v_{n+1} = 1 + \frac{2}{v_n} \end{cases}$ *est définie pour tout* $n \in \mathbb{N}$ *et converge vers 2.*

Ecrire un script qui détermine le plus petit entier p tel que $\left|v_p - 2\right| < 10^{-6}$.

EXERCICE 6 *Dans l'exercice suivant, on va calculer la somme des chiffres d'un gros entier. Si* $\varphi(n)$ *désigne la somme des chiffres de n (dans son écriture décimale...), on a par exemple* $\varphi(843) = 15$*. Pour calculer* $\varphi(1234567654398)$*, on peut prendre une variable* `somme` *dans laquelle on va sommer les décimales, en les faisant parallèlement disparaître du nombre initial. Par exemple,* $n = 1234567654398$ *et* $somme = 0$ *au départ. Après une étape,* $n = 123456765439$ *et* $somme = 8$ *; après deux étapes,* $n = 12345676543$ *et* $somme = 17$*... et après 13 étapes,* $n = 0$ *et* $s = 63$ *: la somme vaut* 63*. L'idée est, à chaque étape, de faire passer la dernière décimale de n dans la somme, puis de la faire disparaître de n.*

Project Euler, problème numéro 20
Calculer la somme des décimales de 100! *de la façon suivante :*

$somme \leftarrow 0$
$n \leftarrow 100!$
tant que $n > 0$ **faire**
  $somme \leftarrow somme + (n\%10)$
  $n \leftarrow n//10$
Résultat : $somme$

EXERCICE 7 *Pour chaque script déterminer le nombre d'étoiles affichées :*

```
#Script 1
i, j = 100, 100
while i> 0:
    i = i-1
    print('*')
    while j>0:
        j = j-1
        print('*')
```

```
#Script 2
i = 100
while i> 0:
    i = i-1
    print('*')
    j = 100
    while j>0:
        j = j-1
        print('*')
```

```
#Script 3
i = 100
while i> 0:
    i = i-1
    j = 100
    while j>0:
        j = j-1
        print('*')
```

```
#Script 4
i, j = 100, 50
while i>j:
    i = i-1
    print('*')
    while j>0:
        j = j-1
        print('*')
    j = 50
```

EXERCICE 8 Suite de Fibonacci.
La suite de Fibonacci est définie par $f_0 = 0$, $f_1 = 1$ *et pour tout* $n \in \mathbb{N}$, $f_{n+2} = f_n + f_{n+1}$.

- *Calculer* f_n *à la main, pour* $n \leqslant 10$.
- *Écrire un algorithme permettant de calculer* f_{100}.
- *Programmer cet algorithme en Python.*
- *Que vaut finalement* f_{100} *? Et* f_{1000} *?*

Pour ceux qui sèchent, un algorithme est proposé en dernière partie de TP.

EXERCICE 9 Algorithme d'Euclide

1. *L'algorithme des différences permet de déterminer le PGCD de deux entiers.*

   Le tableau ci-dessous donne un exemple d'exécution pour déterminer le PGCD noté $a \wedge b$ *des entiers* $a = 75$ *et* $b = 30$.

a	b	*différence*
75	30	45
45	30	15
30	15	15
15	15	0

   Écrire un programme en Python implémentant ce premier algorithme de calcul du PGCD de deux entiers.

2. *Un autre algorithme connu pour déterminer le PGCD utilise la propriété arithmétique suivante de la division euclidienne :*

   si $a = qb + r$ *avec* $0 \leqslant r < b$ *alors* $a \wedge b = b \wedge r$.

   Écrire un programme en Python implémentant ce second algorithme de calcul du PGCD de deux entiers.

EXERCICE 10 Project Euler problem 9

A Pythagorean triplet is a set of three natural numbers, $a < b < c$, for which, $a^2 + b^2 = c^2$
For example, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.
There exists exactly one Pythagorean triplet for which $a + b + c = 1000$.
Find the product abc.

2 Autour des nombres premiers

EXERCICE 11 *Importer la fonction* `est_premier` *du fichier* `cadeau_tp_boucles.py` *et exécuter :*

```
from cadeau_tp_boucles import est_premier

for n in range(20):
    if est_premier(n):
        print(n)
```

EXERCICE 12 Un peu de complexité
Lire le code de la fonction `est_premier` *: combien réalise-t-elle d'« opérations élémentaires » lorsqu'elle est exécutée avec en entrée un entier pair ? Et un entier premier ?*

EXERCICE 13 Complexité à la louche (difficile)
Sachant que «à la louche, la proportion d'entiers $\leqslant N$ qui sont premiers est de l'ordre de $\frac{1}{\ln N}$ », évaluer le nombre d'opérations élémentaires nécessaires pour tester la primalité des entiers $\leqslant N$.
Pour $N = 10^6$, le calcul va-t-il prendre un temps de l'ordre du pouillème de seconde, de la minute, ou de la journée ?

EXERCICE 14 *Combien il y a-t-il d'entiers plus petits que 100 qui sont premiers ? Même chose pour les entiers majorés par 10^4 puis 10^6.*

EXERCICE 15 *Combien existe-t-il de $n \leqslant 10^6$ tels que n et $n+2$ sont premiers ?*

$cpt \leftarrow 0$
pour *n allant de* 1 *à* ... **faire**
  si *n est premier* **alors**
    $cpt \leftarrow cpt + 1$
Résultat : cpt

EXERCICE 16 *Trouver le plus petit entier n supérieur à 10^{10} tel que n et $n+2$ sont premiers.*

EXERCICE 17 *Compter précisément le nombre de divisions euclidiennes effectuées pour tester la primalité des entiers majorés par 10^6.*

3 Observons une suite d'entiers

On s'intéresse ici à la suite définie par son premier terme $u_0 = 42$ puis la relation de récurrence $u_{n+1} = 15091 u_n \mod 64007$ pour tout $n \in \mathbb{N}$.

EXERCICE 18 *Que vaut u_1 ? Et u_{10} ? Et u_{10^6} ?*

EXERCICE 19 *Compter le nombre de $n \leqslant 10^7$ vérifiant les conditions suivantes :*

1. *u_n est pair ;*
2. *u_n est premier ;*
3. *$u_n \mod 3 = 1$;*
4. *$u_n \mod 3 = 1$ et u_n est premier ;*
5. *u_n est pair et u_n est premier ;*
6. *n est pair et u_n est premier.*

4 Autour de la multiplication et l'exponentiation modulaire

EXERCICE 20 *Sans calculatrice : quelle est la dernière décimale de 17×923 ?*
Quelle est la dernière décimale de $123345678987654 \times 836548971236$?

Donc finalement : *pour connaître $ab \mod 10$, on fait le produit de $a \mod 10$ par $b \mod 10$, et on regarde ce produit modulo* 10.

On montrerait sans problème que ce résultat reste valable modulo n'importe quel entier. De même, pour calculer $a^b \mod c$, on peut faire b multiplications par a et réduire modulo c à chaque étape.

$res \leftarrow 1$
pour *i de* 1 *à b* **faire**
  $res \leftarrow res \times a \mod c$
Résultat : res

EXERCICE 21 *Expliquer l'intérêt de cette façon de procéder par rapport à la version «on calcule a^b, puis on réduit le résultat modulo c». Calculer ainsi $123456^{654321} \mod 1234567$.*

EXERCICE 22 Project Euler : problem 48
The series, $1^1 + 2^2 + 3^3 + \cdots + 10^{10} = 10405071317$. Find the last ten digits of the series, $1^1 + 2^2 + 3^3 + \cdots + 1000^{1000}$.

5 Pour ceux qui s'ennuient

EXERCICE 23 *Pour une tombola, on a vendu tous les billets numérotés $1, 2, 3, \ldots, n$ où n est un entier supérieur ou égal à 2016.*
On détermine les numéros des billets gagnants de la façon suivante : on écrit de gauche à droite la liste des entiers de 1 à n sur un tableau puis on passe en revue cette liste dans l'ordre croissant en effaçant les entiers qui sont les triples des nombres non effacés. On obtient donc la liste dont les premiers nombres sont : 1, 2, 4, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18,.... *On décide que les numéros effacés sont les gagnants. Les autres sont perdants.*

1. *Justifier que le numéro 100 est perdant. En déduire que 300 est gagnant.Le numéro 2016 est-il perdant ou gagnant ?*
   Démontrer que si le numéro a est perdant alors le numéro $9a$ l'est également.Le numéro 729 est-il gagnant ou perdant ?
   Parmi les numéros qui sont des puissances de 3, lesquels sont perdants ?
2. *Écrire en Python une fonction* `gagnant(m)` *qui retourne* `True` *si l'entier $m \geqslant 1$ est gagnant et* `False` *sinon.*
3. *Écrire en Python une fonction* `nombre_gagnant(n)` *qui retourne le nombre de gagnants parmi les entiers m tels que $1 \leqslant m \leqslant n$.*
   Si le temps d'exécution dépasse dix secondes pour $m = 10^9$, il faut revoir l'algorithme utilisé ...

```
>>> nombre_gagnant(2016), nombre_gagnant(10**9)
504, 249999999
```

EXERCICE 24 Project Euler : problem 39
If p is the perimeter of a right angle triangle with integral length sides, {a,b,c}, there are exactly three solutions for p = 120.
{20,48,52}, {24,45,51}, {30,40,50} For which value of p $\leqslant$ 1000, is the number of solutions maximised ?

6 Besoin d'indications ?

- Exercice 8. On calcule les valeurs du couple (f_k, f_{k+1}) pour k allant de 0 à 99. L'idée est que si $(f_k, f_{k+1}) = (a, b)$, alors au rang suivant : $(f_{k+1}, f_{k+2}) = (b, a+b)$, ce qui donne un algorithme assez simple :

  $(a, b) \leftarrow (0, 1)$
  pour *k de* 1 *à* 99 **faire**
    # À l'entrée $(a, b) = (f_{k-1}, f_k)$
    $(a, b) \leftarrow (b, a+b)$ # Et à la sortie $(a, b) = (f_k, f_{k+1})$
  Résultat : b

  On trouvera $f_{100} = 354224848179261915075$ et $f_{1000} = 434665...849228875$.
- Exercice 18. On a $u_1 = 57750$, $u_{10} = 52866$ et $u_{10^6} = 14919$.
- Exercice 19. On calcule les termes de proche en proche, en mettant à jour 6 compteurs :

  $(c_1, c_2, ..., c_6) \leftarrow (0, 0, ..., 0)$
  $x \leftarrow 42$ # x va représenter le terme courant u_n
  pour *n de* 0 *à* 10^6 **faire**
    # On traite ici u_n
    si $x \mod 2 = 0$ **alors**
      $c_1 \leftarrow c_1 + 1$
    si ... **alors**
      ...
    ...
    si ... **alors**
      $c_6 \leftarrow c_6 + 1$
    $x \leftarrow 15091x \mod 64007$ # Calcul du terme suivant
  Résultat : $(c_1, ..., c_6)$
- Exercice 22. Il s'agit de calculer cette somme (mais aussi chaque terme) modulo 10^{10}.
- Exercice 23. Pour chaque puissance impaire de 3 inférieure ou égale à m, on dénombre les entiers $3^{2k+1} \times a$ avec a non divisible par 3.
- Exercice 24 Mémoizer dans un tableau/liste les décompositions déjà trouvées.

Corrigé du TP 03 Boucle

Chenevois-Jouhet-Junier-Rebout

1 Quelques boucles

1.1 Import de bibliothèques/modules

```
import math
```

1.2 Exercice 2

```
s = 0
for k in range(831, 945):
    s += k**10 #ou s = s + k**10

"""
>>> s
3672419115036510057216102022082 5
"""
```

1.3 Exercice 3

```
res = 1
for i in range(1, 844):
    res = res * 3

"""
>>> res-3**843
0
"""
```

1.4 Exercice 4

```
facto100 = 1
for i in range(2, 101):
    facto100 = facto100 * i
"""
>>> facto100
933262154439....1686400000000000000000000000
>>> import math
>>> facto100 - math.factorial(100)
```



```
0
"""
```

1.4.1 Exercice 5

Calcul du terme de rang n de $u(0) = 3$ et $u(n+1) = 3 \times u(n) + n$. On trouve $u(0) = 3$, $u(1) = 9$, $u(2) = 28$ ...

```
n = int(input('Entrez un entier n : '))
u = 3
for k in range(1, n+1):
    u = 3*u + k - 1
print('u({:d})={:d}'.format(n, u))
```

Soit la suite (v_n) définie par $v(0) = 1$ et pour tout entier $n \geqslant 0$, $v(n+1) = 1 + 2/v(n)$. Algorithme de seuil pour déterminer le plus petit indice p tel que $|v(p) - 2| < 10^{-6}$.

On trouve $v(21) = 2,00000072$

```
n = 0
v = 1
while abs(v - 2) >= 1e-6:
    n += 1
    v = 1 + 2.0/v
print('v({:d})={:1.8f}'.format(n,v))
```

1.5 Exercice 6 Projet Euler Problème 20

```
somme_dec = 0
n = math.factorial(100)
while n > 0:
    somme_dec += n%10
    n = n//10
print(somme_dec)

"""
>>> somme_dec
648
"""
```

1.6 Exercice 7

- Script 1 : la boucle externe est exécutée 100 fois, à chaque itération elle affiche une `*` et la boucle interne est exécutée une seule fois où elle affiche 100 '*' soit un total de 200 `*`
- Script 2 : la boucle externe est exécutée 100 fois, à chaque itération elle affiche une `*` et la boucle interne est exécutée 100 fois et à chaque itération de la boucle externe elle affiche 100 `*` soit un total de $100 + 100 \times 100 = 101 \times 100$ `*`
- Script 3: la boucle externe est exécutée 100 fois, à chaque itération elle n'affiche aucune `*` et la boucle interne est exécutée 100 fois à chaque itération de la boucle externe elle affiche 100 `*` soit un total de 100×100 `*`
- Script 4: la boucle externe est exécutée 50 fois, à chaque itération elle affiche une `*` et la boucle interne est exécutée 50 fois à chaque itération de la boucle externe elle affiche 50 `*` soit un total de $50 \times (1 + 50)$ `*`

1.7 Exercice 8

```
n = int(input('Entre le rang n : '))
f0, f1 = 0, 1
for k in range(1, n+1):
    f0, f1 = f1, f0+f1
print('fibo({:d})={:d}'.format(k,f0))

"""
>>> (executing cell "EXO 8 #########" (line 92 of "tp-python-sup-2015-03.py"))
Entre le rang n : 0
fibo(0)=0

>>> (executing cell "EXO 8 #########" (line 92 of "tp-python-sup-2015-03.py"))
Entre le rang n : 100
fibo(100)=354224848179261915075

>>> (executing cell "EXO 8 #########" (line 92 of "tp-python-sup-2015-03.py"))
Entre le rang n : 1000
fibo(1000)=43466557686937456435...5166849228875
"""
```

1.8 Exercice 9 : algorithme d'Euclide

- Algorithme des différences

```
def euclide_difference(a, b):
    while a != b:
        if b > a: #on doit avoir a >= b pour que a - b >= 0
            a, b = b, a
        a, b = b, a - b
    return a

"""
In [10]: [euclide_difference(30, k) for k in range(1, 20)]
Out[10]: [1, 2, 3, 2, 5, 6, 1, 2, 3, 10, 1, 6, 1, 2, 15, 2, 1, 6, 1]
"""
```

- Algorithme classique

```
def euclide_classique(a, b):
    while b != 0:
        a, b = b, a%b
    return a

"""
In [13]: [euclide_classique(30, k) for k in range(1, 20)]
Out[13]: [1, 2, 3, 2, 5, 6, 1, 2, 3, 10, 1, 6, 1, 2, 15, 2, 1, 6, 1]
"""
```


1.9 Exercice 10 : Projet Euler problème 9

```
a, trouve = 1, False

#a<b<c et a+b+c=1000 donc a<1000//3 et 1000 - (a+b) > b

while not trouve and a < 1000//3:
    b = a+1
    while not trouve and b < (1000 - a)//2:
        c = 1000 - a - b
        if c**2 == a**2 + b**2:
            trouve = True
            produit = a*b*c
        b += 1
    a += 1
print('{0:} + {1:}+ {2:} = 1000 et {0:}**2 + {1:}**2 = {2:}**2 \
et produit = {3:}'.format(a-1 , b-1 , c, produit))

#200 + 375+ 425 = 1000 et 200**2 + 375**2 = 425**2 et produit = 31875000
```

2 Autour des nombres premiers

2.1 Exercice 11

Dans une fonction si chaque clause se termine par un `return` on n'a pas besoin d'une structure `if elif else`. Si une clause est réalisée, le `return` fait sortir de la fonction et ce qui suit n'est pas exécuté.

```
from math import sqrt

def est_premier(n):
    '''test de primalité'''
    if n <= 1:
        return False
    for d in range(2, int(sqrt(n)) + 1):
        if n%d == 0:
            return False
    return True

for n in range(20):
    if est_premier(n):
        print(n, end=',')
```

2.2 Exercice 12

- Si n est pair, un ou deux tests sont effectués et une division euclidienne.
- Si n est premier, environ $\lfloor\sqrt{n}\rfloor$ tests et divisions euclidiennes sont effectués.

2.3 Exercice 13

- Pour les entiers premiers, le coût de traitement sera de l'ordre de : $N/(\ln(N)) \times \sqrt{N}$, soit $10^9/(6\ln(10))$ pour $N = 10^6$: c'est acceptable. Un microprocesseur à 2,5 GHz peut réaliser un milliard de cycles par seconde donc si un cycle peut réaliser 4 unités de calculs en virgule flottantes, il peut réaliser 10 milliards de ces opérations par secondes soit 10 **gigaflops**. La puissance de calcul peut être estimée grossièrement par la formule *puissance = fréquence* × *nombre d'opérations simultanées* × *nombre de coeurs* Voir https://interstices.info/idee-recue-comparer-la-puissance-de-deux-ordinateurs-cest-facile/ pour plus de détails.

- C'est plus difficile à évaluer pour les composés : certains auront un coût proche de $\sqrt{N}$, mais ils seront peu nombreux. La plupart auront un premier diviseur faible, donc on peut espérer un coût qui soit plus proche de N ou $N\ln(N)$ que de $N^{3/2}$. Finalement le terme dominant dans la complexité est représenté par les entiers premiers.

2.4 Exercice 14

```
for nmax in [10**2, 10**4, 10**6]:
    cpt = 0
    for n in range(1,nmax+1):
        if est_premier(n):
            cpt += 1
    print(' {:d} entiers premiers <= {:d}'.format(cpt, nmax))
```

- 25 entiers premiers $\leqslant 100$
- 1229 entiers premiers $\leqslant 10000$
- 78498 entiers premiers $\leqslant 1000000$

2.5 Exercice 15

```
cpt = 0
for n in range(2, 10**6-1):
    if est_premier(n) and est_premier(n+2):
        cpt += 1
print(cpt)
```

8169 couples de premiers jumeaux inférieurs à 10**6

2.6 Exercice 16

```
n = 10**10+1
while not est_premier(n) or not est_premier(n+2):
    n += 1
print("Le plus petit couple d'entiers premiers jumeaux \
supérieurs à 10**10 est ",(n,n+2))
```

Le plus petit couple d'entiers premiers jumeaux supérieurs à 10^{10} est $(10000000277, 10000000279)$.

2.7 Exercice 17

```
def est_premier2(n):
    '''test de primalité'''
    global div  #compteur de divisions, variable globale pour l'exo 17
    if n <= 1:
        return False
    if n <= 3:
```



```
        return True
    for d in range(2, int(n**(1/2))+1):
        div += 1
        if n%d == 0:
            return False
    return True

cpt = 0 #compteur de couples de premiers jumeaux
div = 0 #compteur de divisions
for n in range(2,10**6):
    if est_premier2(n):
        cpt += 1
print('Il y a {} entiers  premiers  inférieurs à 10**6'.format(cpt))
print(div,'divisions ont été effectuées.')
```

Il y a 78498 entiers premiers inférieurs à 10^6 et 67740403 divisions ont été effectuées.

3 Observons une suite d'entiers

3.1 Exercice 18

```
def u(n):
    u = 42
    for k in range(1, n+1):
        u = (15091*u) % 64007
    return u


for n in [1, 10, 10**6]:
    print('u(%s)=%s'%(n,u(n)))
```

u(1)=57759 , u(10)=15421 et u(1000000)=14918.

3.2 Exercice 19

Pour éviter de répéter les memes calculs mieux vaut dresser une table de tous les premiers inférieurs à 64007 plutot que faire 10^7 tests de primalité. Quand on doit utiliser les mêmes données un grand nombre de fois dans une boucle, mieux vaut calculer ces données une seule fois en dehors de la boucle.

```
premier = [est_premier(k) for k in range(64007)]

"""
>>> print(premier[:8])
[False, False, True, True, False, True, False, True]
"""

c = [0]*6  #compteurs des six conditions
u = 42
for n in range(1, 10**7+1):
    u = (15091*u)%64007
    if u%2 == 0:
        c[0] += 1
```

```
    if premier[u]:
        c[1] += 1
    if u%3 == 1:
        c[2] += 1
    if u%3 == 1 and premier[u]:
        c[3] += 1
    if u%2 == 0 and premier[u]:
        c[4] += 1
    if n%2 == 0 and premier[u]:
        c[5] += 1
print(c)

"""
[4999968, 1001923, 3333434, 499050, 156, 493240]
"""
```

On n'a pas utilisé de `if elif else` car les clauses ne sont pas exclusives.

3.3 Exercice 20

```
"""
>>> ((17%10)*(923%10))%10
1
>>> ((123345678987654%10)*(836548971236%10))%10
4
"""
```

3.4 Exercice 21

```
res = 1
a = 123456
c = 1234567
for i in range(1, 654322):
    res = (res*a)%c
print(res)
"""
1075259
"""
```

```
print("{:d} est plus rapide à calculer que {:d}".format(res,(123456**654321)%1234567)))
```

1075259 est plus rapide à calculer que 1075259.

3.5 Exercice 22 : Projet Euler problème 48

```
series, m  = 0, 10**10
for i in range(1,1001):
    res = 1
    for k in range(1, i+1):
        res = (res*i)%m
    series = (series+res)%m
print(series)
```


Réponse : 9110846700

4 Pour ceux qui s'ennuient

4.1 Exercice 23 Tombola

```
def gagnant(m):
    while m%9 == 0:
        m = m // 9
    return m%3 == 0

"""
>>> list(map(gagnant, [100,300,729,2016]))
[False, True, False, False]
"""

def nombre_gagnant(n):
    '''Retour le nombre d'entiers gagnants parmi
    les entiers entre 1 et n'''
    # on détermine d'abord le plus grand entier k tel que 3**k <= m
    p = 1
    e = 0
    while p <= n:
        p *= 3
        e += 1
    e = e - 1  #3**e <= m et 3**(e+1)>m
    gagnant = 0  #compteur de gagnant
    for k in range(1, e + 1):
        #pour les 3**k avec k impair
        #on rajoute le nombre de multiples de 3**k <= m
        # de la forme a*3**k avec a pas divisible par 3
        if k%2 == 1:
            q = n//3**k
            gagnant += q - q//3
    return gagnant

def nombre_gagnant2(m):
    '''Plus court mais plus long car complexité cachée'''
    return len([m  for n in range(1 , m + 1) if gagnant(m)])
```

```
from timeit import timeit
from math import log
```

La fonction `timeit` du module `timeit` affiche le temps cumulé d'exécution d'une commande pour un nombre fixé d'exécutions.

On observe ci-dessous que le temps d'exécution de `nombre_gagnant(n)` semble proportionnel à $\log(n)/\log(3)$ et que celui de `nombre_gagnant2(n)` semble proportionnel à `n`.

On peut conjecturer une **complexité temporelle** logarithmique pour `nombre_gagnant` et linéaire pour `nombre_gagnant2`.

```
bench = [10**2, 10**3, 10**4, 10**5, 10**6, 10**7, 10**8, 10**9]
for n in bench:
    temps = timeit('nombre_gagnant({})'.format(n),
                   'from __main__  import nombre_gagnant',number = 10000)
    print(log(n,3)/temps)

"""
130.16506757074666
158.44562663545946
157.35055637870508
169.62033933138346
167.83313335771103
162.74572022382512
168.88623818133289
161.82620193271757
"""

bench = [10**2, 10**3, 10**4]
for n in bench:
    temps = timeit('nombre_gagnant2({})'.format(n),
                   'from __main__  import nombre_gagnant2',number = 10000)
    print(n/temps)

"""
259.88979357801304
254.6282087040769
"""
```

4.2 Exercice 24 : Projet Euler problème 39

On recherche le nombre maximal de configurations de triangles rectangles de périmètre donné parmi tous les périmètres inférieur ou égal à n.

- Algorithme 1 :

  3 boucles imbriquées : sur le perimetre puis sur les côtés a, b et c avec $a \leqslant b \leqslant c$ où $c = p - a - b$.

  La complexité est **cubique** en $O(n^3)$.

```
def euler39_cubique(n):
    pmax, nbsolmax = 0, 0
    for p in range(4, n + 1):
        nbsol = 0
        for a in range(1, p//3 + 1):
            for b in range(a, 2*p//3 + 1):
                if (p - a -b)**2 == a**2 + b**2:
                    nbsol += 1
        if nbsol > nbsolmax:
            nbsolmax, pmax = nbsol,p
    return pmax, nbsolmax

def euler39_cubique2(n):
    pmax, nbsolmax = 0, 0
    for p in range(3, 1001):
        nbsol = 0
        for c in range(p//3, p):
            b = int(c/2**0.5)
            while b < c and p > b + c:
                a = p - c - b
                if a**2 + b**2 == c**2:
                    nbsol += 1
                b += 1
        if nbsol > nbsolmax:
            pmax, nbsolmax = p, nbsol
    return pmax, nbsolmax
```

- Algorithme 2 :

  On crée un tableau des nombres de solutions pour les n périmètres. On le crible avec deux boucles imbriquées qui parcourent tous les couples avec $a \leqslant b \leqslant 2n/3$ tels que $\sqrt{a^2+b^2}$ est un entier inférieur ou égal à n.

  La complexité est **quadratique** en $O(n^2)$.

```
def euler39_quadratique(n):
    #cribleperi[p] devra contenir le nombre de solutions pour le périmètre p
    cribleperi = [0]*(n+1)
    for a in range(1, n//3 + 1):
        for b in range(a, 2*n//3 + 1):
            c = (a**2 + b**2)**(1/2)
            if c == int(c):
                p = a + b + int(c)
                if p <= n:
                    cribleperi[p] += 1
    return max(enumerate(cribleperi), key = lambda couple : couple[1])

"""
In [1]: euler39_cubique(1000)
Out[1]: (840, 8)

In [2]: euler39_quadratique(1000)
Out[2]: (840, 8)

In [3]: %timeit euler39_quadratique(1000)
10 loops, best of 3: 184 ms per loop

In [4]: %timeit euler39_cubique(1000)
1 loops, best of 3: 53.4 s per loop

In [5]: 52.4/184e-3
Out[5]: 284.7826086956522
"""
```

On a vérifié ci-dessus que le rapport de temps entre les deux algorithmes était de l'ordre de grandeur de n (ici 1000) à une constante près.