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
# 367241911503651005721610202208 25L
```

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

$res \leftarrow 1$
pour i *de* 1 *à* 843 **faire**
 $\llcorner$ $res \leftarrow res \times 3$
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**
 $\llcorner$ $res \leftarrow res \times i$
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 $|v_p - 2| < 10^{-6}$.

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.