

## TP04 POURSUITE DE L'ÉCHAUFFEMENT

Continuons l'échauffement débuté lors des séances précédentes. Comme ce sera le cas durant toute l'année, votre dossier de travail est accessible<sup>1</sup> sur votre session dans Mes Documents/Devoirs/fleck/TP04/.

Partie I

### À préparer à la maison sans ordinateur

## 1 Valeur en sortie de boucle

Calculer (à la main) la valeur de  $S$  en sortie de boucle. Vérifier en tapant la boucle sur l'ordinateur.

```
1  ## Première boucle
2  S = 0
3  for i in range(11):
4      S = S + i
5  print(S)
6
7  ## Deuxième boucle (ressemblante à un détail près)
8  S = 0
9  for i in range(11):
10     S = S + 1
11  print(S)
12
13  ## Troisième boucle
14  S = 0
15  for i in range(13):
16     if i == 3:
17         S = S - 5
18     elif i%6 == 0:
19         S = S + 2
20     elif i%3 == 0:
21         S = S + 3
22  print(S)
```

## 2 Partie entière

Réfléchir à un algorithme simple permettant d'écrire la fonction `partie_entiere(x)` qui renvoie la partie entière du réel  $x$ . On rappelle que la partie entière d'un réel  $x$  (notée  $E(x)$ ) est l'entier le plus proche de  $x$  qui lui soit directement inférieur. Par exemple,

$$E(1,2345) = 1 \quad E(345,432) = 345 \quad \text{et} \quad E(-345,432) = -346$$

<sup>1</sup>En cas de problème informatique, il est toujours possible de récupérer ce dossier directement sur le site internet <http://pcsi.kleber.free.fr/IPT/>

### 3 Boucle while

On considère le code suivant

```
1 n = 1025478651
2 c = 0
3 while n != 0:
4     if (n%10)%2 == 0:
5         c += 1
6     n //= 10
7 print(c)
```

Que vaut  $c$  à la fin de la boucle ? Que réalise ce programme ? On pourra faire un tableau pour suivre l'évolution des variables dans la boucle.

Partie II

## Mise en jambe

### 1 Quelques listes à produire

1. Créer la liste  $L1$  constituée de 12 zéros.
2. Créer la liste  $L2 = [42, 41, 40, 39, \dots, 26, 25, 24]$  (il s'agit de remplir les  $\dots$  et de préférence pas à la main...)
3. Créer la liste  $L3 = [0, 0, 0, 2, 2, 2, 4, 4, 4, \dots, 10, 10, 10]$
4. Créer la liste  $L4 = [1, 2, 2, 3, 3, 3, \dots, 7, 7, 7, 7, 7, 7, 7, \dots, 10, 10]$ . Stocker son 40<sup>e</sup> élément dans la variable  $L4_{40}$ .
5. Créer la liste  $L5 = [1, 2, 1, 3, 2, 1, 4, 3, 2, 1, \dots, 9, 8, 7, 6, 5, 4, 3, 2, 1]$ . Stocker son 38<sup>e</sup> élément dans la variable  $L5_{38}$ .

### 2 Partie entière

Implémenter la fonction `partie_entiere(x)` pour laquelle vous avez préparé un algorithme à la section précédente.

### 3 ProjectEuler numéro 1

Si l'on liste tous les entiers naturels inférieurs à 10 qui soient soit multiple de 3, soit multiple de 5, on obtient 3, 5, 6 et 9 dont la somme fait  $3 + 5 + 6 + 9 = 23$ .

Écrire la fonction sans argument `probleme_1()` qui renvoie la somme de tous les entiers multiples de 3 ou de 5 qui soient strictement inférieurs à 1000.

### 4 ProjectEuler numéro 48

On peut calculer la somme  $1^1 + 2^2 + 3^3 + \dots + 10^{10} = 10\,405\,071\,317$ .

Écrire la fonction sans argument `probleme_48()` qui renvoie le nombre constitué des 10 derniers chiffres de la somme  $1^1 + 2^2 + 3^3 + \dots + 1000^{1000}$ .

Partie III

## Conjecture de Collatz

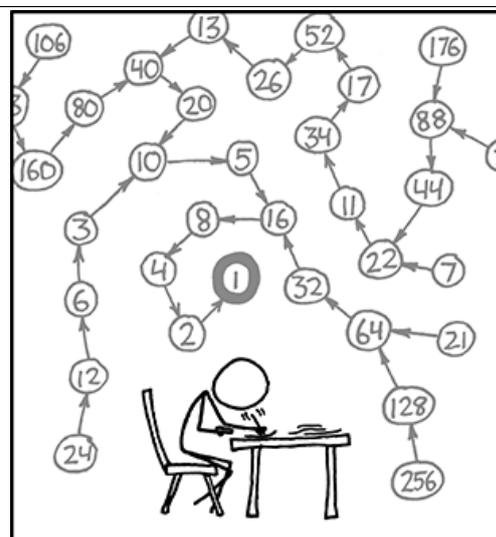
XkcD résume merveilleusement bien toute la beauté de la procédure de Collatz dans le dessin ci-contre. L'idée est de construire une suite récurrente en appliquant successivement une même fonction  $f$ , c'est-à-dire que l'on ait  $u_{n+1} = f(u_n)$ . Sauf que la fonction  $f$  est un peu spéciale: elle n'a pas le même comportement si l'entier donné en argument est pair ou impair. Plus particulièrement, elle s'écrit

$$f : n \mapsto \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 3n + 1 & \text{si } n \text{ est impair} \end{cases}$$

Par exemple, avec  $u_0 = 13$ , on obtient

13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, etc.

C'est ce qu'on appelle la suite de Collatz (ou suite de Syracuse) du nombre 13. Après avoir atteint le nombre 1, la suite de valeurs (1,4,2,1,4,2,1,4,2...) se répète indéfiniment en un cycle de longueur 3 appelé cycle trivial. La conjecture de Collatz est l'hypothèse selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1. Bien qu'elle ait été vérifiée pour les 5,7 premiers milliards de milliards d'entiers et en dépit de la simplicité de son énoncé, cette conjecture défie depuis de nombreuses années les mathématiciens. Paul Erdős a dit à son propos que « les mathématiques ne sont pas encore prêtes pour de tels problèmes »...



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

1. Implémenter la fonction  $f(n)$  ci-dessus et donner l'image des entiers de  $\llbracket 1 ; 30 \rrbracket$  par la fonction  $f(n)$ . On stockera ces images sous forme d'une liste dans la variable `trente_premiers`.
2. Soit  $u_0 = a \in \mathbb{N}^*$ . On appelle orbite de  $a$  la liste des termes de la suite  $u_{n+1} = f(u_n)$  jusqu'à ce que l'on tombe sur 1 (compris). Écrire le programme `orbite(a)` qui prend comme entrée l'entier  $a$  et qui renvoie son orbite sous forme d'une liste. Donner l'orbite de 27 et la stocker dans la variable `orbite_de_27`.

Plusieurs caractéristiques d'une orbites peuvent être explorées:

- son « temps de vol » correspond au nombre total d'entiers visités sur l'orbite.
- son « altitude » est donnée par le plus grand entier visité sur l'orbite.
- son « temps de vol en altitude » correspond au nombre d'étapes avant de passer strictement en dessous du nombre de départ.
- et enfin son « temps de vol avant la chute » correspond au nombre d'étapes minimum après lequel on ne repasse plus au-dessus de la valeur de départ.

Par exemple pour l'orbite du nombre 13, l'altitude vaut 40 (maximum de la suite), le temps de vol vaut 10, le temps de vol en altitude vaut 3 (on passe à  $10 < 13$  lors de la 3<sup>e</sup> étape) et le temps de vol avant la chute vaut 6 (on passe à  $8 < 13$  lors de la 6<sup>e</sup> itération de la fonction  $f$ ).

3. Écrire une fonction `temps_de_vol(a)` qui renvoie le temps de vol correspondant à l'orbite de  $a$ .
4. Écrire une fonction `altitude(a)` qui renvoie l'altitude de l'orbite de  $a$ . Pour s'entraîner, on implémentera la recherche du maximum « à la main ».
5. Écrire une fonction `temps_en_altitude(a)` qui renvoie le temps de vol en altitude correspondant à l'orbite de  $a$ . On prendra comme convention que ce temps vaut 1 si  $a$  vaut initialement 1.
6. Écrire une fonction `temps_avant_chute(a)` qui renvoie le temps de vol avant la chute pour l'orbite de  $a$ . À nouveau, on prend comme convention qu'il vaut 1 si  $a$  vaut initialement 1.

À présent que l'on dispose de ces outils, utilisons-les pour faire un peu de « data-mining » (variante autour du ProjectEuler numéro 14).

7. Pour des entiers de départ de valeur strictement inférieures à un million, déterminer à chaque fois
  - a) celui qui monte le plus haut en altitude et la valeur de cette altitude maximale (à stocker dans la liste `le_plus_haut` qui aura donc deux éléments `[a,altitude(a)]`).
  - b) celui dont le temps de vol est le plus long et la valeur de ce temps de vol (à stocker dans la liste `le_plus_long`).
  - c) celui dont le temps de vol en altitude est le plus long et la valeur de ce temps de vol (à stocker dans la liste `le_plus_long_en_altitude`).
  - d) celui dont le temps de vol avant la chute est le plus long et la valeur de ce temps de vol (à stocker dans la liste `le_plus_long_avant_la_chute`).
8. Ne trouvez-vous pas que la procédure suggérée fasse un peu « gachis » ? Implémentez une procédure unifiée qui puisse effectuer tous les calculs précédents en moins de temps (en sacrifiant peut-être un peu sur l'autel de la simplicité de conception). Estimez le gain de temps que l'on peut espérer et mesurez-le (via un `import time` et à l'aide de `time.clock()`, plus d'info avec `help(time.clock)`). N'oubliez pas de tester votre procédure en retrouvant les résultats précédent

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?  
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	2 WEEKS	1 DAY
	1 DAY					8 WEEKS	5 DAYS

Don't forget the time you spend finding the chart to look up what you save.  
And the time spent reading this reminder about the time spent.  
And the time trying to figure out if either of those actually make sense.  
Remember, every second counts toward your life total, including these right now.