TP04 Poursuite de l'échauffement

Continuons l'échauffement débuté lors des séances précédentes. Pensez à utiliser GithubDesktop pour faire un « Pull » et récupérer le dossier du TP qui aura été déposé sur Gitlab par le Général Kléber.

Pensez aussi à écrire votre code *en premier lieu* sur feuille pour vous entraîner aux futures épreuves écrites d'informatique.

Partie I

Mise en jambe

I.1 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.

I.2 ProjectEuler numéro 48

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

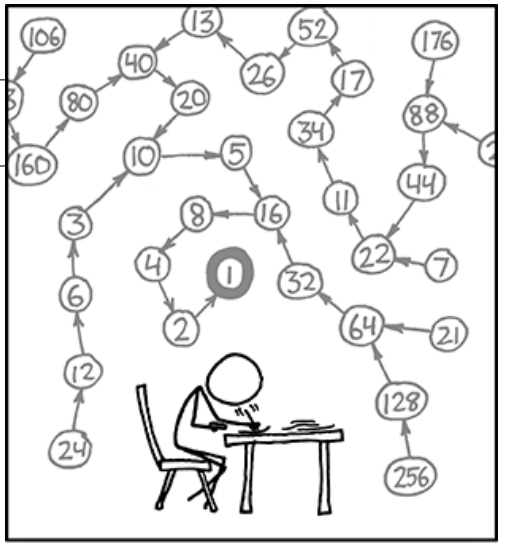
Écrire la fonction sans argument `probleme_48()` qui renvoie le nombre constitué des 10 derniers chiffres de la somme $1^1+2^2+3^3+\cdots+1000^{1000}$.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie II

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

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

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

$$13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, \text{ etc.}$$

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 Erdös a dit à son propos que « les mathématiques ne sont pas encore prêtes pour de tels problèmes »...

1. Implémenter la fonction `f(n)` ci-dessus.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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`.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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^e étape) et le temps de vol avant la chute vaut 6 (on passe à $8 < 13$ lors de la 6^e 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`.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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 ».

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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 inférieure à `n`, écrire les fonctions suivantes :
   (a) `le_plus_haut(n)` qui détermine celui qui monte le plus haut en altitude et la valeur de cette altitude maximale ;
   (b) `le_plus_long(n)` qui détermine celui dont le temps de vol est le plus long et la valeur de ce temps de vol ;
   (c) `le_plus_long_en_altitude(n)` qui déterminer celui dont le temps de vol en altitude est le plus long et la valeur de ce temps de vol ;
   (d) `le_plus_long_avant_la_chute(n)` celui dont le temps de vol avant la chute est le plus long et la valeur de ce temps de vol ;

   Bien sûr, vu que les calculs se ressemblent méchamment et plutôt que de faire des copier-coller de presque la même chose, vous voudrez peut-être écrire une fonction générique qui prenne à la fois `n` et la fonction (type `altitude` ou `temps_avant_chute`) dont on veut déterminer le maximum en argument pour en tirer les informations demandées.

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

8. Ne trouvez-vous pas que la procédure suggérée fasse un peu « gachis » ? En particulier, avec un peu de mémoïsation [1], vous pouvez faire cela bien plus rapidement.
   Ce qui prend du temps dans les procédure précédentes, c'est le calcul de l'orbite (en outre répété pour chacune des 4 propriétés que l'on veut vérifier). Il convient donc :
   — de ne calculer l'orbite qu'une seule fois ;
   — de s'arrêter dès qu'on descend en dessous du nombre de départ car on a déjà calculé le reste de l'orbite. Il convient donc de récupérer le résultat qui aura été stocké précédemment dans une liste ou un dictionnaire (dont la clé est le nombre de départ), au choix.

   Implémentez donc une procédure unifiée appelée `tout_en_un(n)` qui prend en argument le plus grand nombre jusqu'auquel on veut tester et renvoie une liste de doublets. Avec les notations de la partie précédente, ce serait

```
    return [le_plus_haut(n), le_plus_long(n), le_plus_long_en_altitude(n), le_plus_long_avant_la_chute(n)]
```

Stop Gitlab
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

1. Si, si, c'est la bonne orthographe, voyez `https://fr.wikipedia.org/wiki/Mémoïsation`