TP 1 : Initiation à Python

Exercice 1. *Prise en main*

1. En utilisant vos identifiants ENT, logguez-vous sur le serveur **Windows (cours et TP)**. Certaines salles (p. ex. 007, 008 ou encore 214) sont équipées de postes PC : vous pouvez alors travailler directement sur le poste (et non sur le serveur).
2. Ouvrez l'explorateur de fichiers et trouvez le disque `M`.

   C'est un disque en réseau, le seul qui vous permettra de retrouver vos fichiers sur n'importe quel poste de travail (et aussi sur les machines virtuelles). Par ailleurs, les fichiers enregistrées sur un autre disque (C, D,...) risquent de disparaître !
3. Sur votre disque `M`, créez un dossier `PyMath`. Dans la suite, vous allez stocker tous vos fichiers dans ce dossier en les nommant, par exemple, `tp1exo2.py` etc. D'une manière générale, sauvegardez chaque exercice dans un autre fichier.

Exercice 2. *Boucle* `while`

Exécutez l'exemple de la boucle `while` donné dans l'aide mémoire. Puis déterminez le plus petit entier n tel que l'expression $\dfrac{n(n+1)}{2}$ dépasse 1 million (attention à la traduction de cette simple formule mathématique en Python ; réponse : 1414).

Exercice 3. *Structure modulaire et des fonctions mathématiques*

En Python toutes les fonctions et outils variés sont regroupés dans les *modules* (appelés aussi *bibliothèques* ou *packages*). Par exemple, les fonctions mathématiques usuelles se trouvent dans le module `numpy`. Pour en profiter, il faut d'abord le charger à l'aide de la commande

```
import numpy as np
```

On mettra cette commande une fois au début du programme (fichier). Ensuite, on peut utiliser une fonction mathématique en mettant le prefix `np`, par exemple : `np.sin(2)**2 + np.cos(2)**2`. Aussi, la constante π est définie dans `numpy` : `np.pi`.

1. Calculez $\sin(\pi^2)$ et le logarithme népérien (de base e) de 2050 (cf. l'aide-mémoire).
2. Calculez les logarithmes de base 10 des nombres suivantes : 90, 99, 101, 120, 999, 500, 1001 (et d'autres si vous voulez !) Quelle est la relation entre le résultat obtenu et le nombre de chiffres de l'entier initial ?
3. À partir de la dernière observation, codez en Python une formule mathématique permettant de déterminer le nombre de chiffres d'une valeur entière affectée à la variable `x`. On pourra utiliser la fonction `int` (cf. l'aide mémoire).

   Vérifiez sur quelques exemples faciles. Puis, à l'aide de votre formule, déterminez le nombre de chiffres de $2^{64} - 1$ (réponse : 20).

Exercice 4. *Fonctions*

1. Recopiez (dans un nouveau fichier) la fonction `carre_plus(n)` de l'aide mémoire.

   Plus bas dans le même fichier, faites appel à cette fonction avec $n = 5$ et affichez le résultat avec `print`.

   Calculez esuite `carre_plus(345) + carre_plus(567)` (réponse : 440518).

2. Écrire une fonction `racines(a,b,c)` qui, étant donnés 3 réels `a`, `b` et `c`, détermine et **retourne** le nombre des racines réelles de l'équation du second degré $ax^2 + bx + c = 0$ (on ne demande pas de la résoudre).

   Testez avec les équations suivantes :
   $x^2 + x - 2 = 0$, $\quad 4x^2 - 12x + 9 = 0$, $\quad 2x^2 + x + 1 = 0$.

Remarque : D'une manière générale, dans une fonction on n'utilisera jamais `print` pour afficher le résultat, mais `return` pour renvoyer le résultat pour la suite du programme.

Ainsi dans cet exercice, on va afficher le résultat renvoyé par la fonction `racines` à l'aide de `print` à l'extérieur de la fonction (sans décalage dans le fichier), par exemple : `print(racines(1, 2, 3))`.

Exercice 5. *Fonctions, boucles et tests*[1]

Si on écrit tous les entiers divisibles par 3 ou par 5 strictement inférieurs à 10, on obtient 3, 5, 6 et 9. Leur somme est 23.

Écrire une fonction `div35(n)` qui renvoie la somme des entiers divisibles par 3 ou par 5 strictement inférieurs à `n`. Tester : div35(1000) = 233168.

Que vaut `div35(2000) + div35(3000)` ?

Exercice 6. *La suite de Syracuse*

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2. Ainsi, en commençant avec n=10, on obtient la suite suivante :

$10 \to 5 \to 8 \to 4 \to 2 \to 1$.

Une fois tombée sur 1, la suite tourne en rond : $1 \to 2 \to 1 \to ...$

Selon la *conjecture de Syracuse*, quelle que soit la valeur initiale n choisie, la suite finira toujours par atteindre 1. Plusieurs projets visent à vérifier cette conjecture. On se propose ici d'étudier quelques questions basiques.

1. Écrire une fonction `syracuse(n)` qui, étant donné un entier `n`, retourne la longueur de la suite de Syracuse commençant par `n` (la suite se termine par la première occurrence de 1).

   Tester : `syracuse(10)` = 6, `syracuse(100)` = 19.
2. Calculer (et afficher) les longueurs de la suite de Syracuse pour la valeur initiale `n` variant de 10 à 25.
3. Trouver la longueur maximale de la suite de Syracuse pour la valeur initiale `n` variant de 10 à 10 000.

   Indication : Dans le code de la question précédente, on pourra introduire une variable `lmax`, initialisée à 0. A chaque tour de boucle, si on trouve une suite de longueur supérieure à `lmax`, on modifie cette dernière variable. Ainsi, à la fin, `lmax` sera la longueur maximale recherchée.
4. Trouver la valeur de `n` (entre 10 et 10000) qui produit la suite de la longueur maximale.

1. Cette exercice à été inspiré par le Project Euler, `http://projecteuler.net`