INFORMATIQUE
BCPST2

Sujet de travaux dirigés N° 1

Itérations simples

I. ♡ *Sommes et produits.*

1. Écrire et tester une fonction `def Fact(N)` qui donne pour résultat $N! = 1 \times 2 \times \ldots \times N$.
2. Écrire et tester une fonction `def SumPow(N,p)` qui donne pour résultat $\sum_{k=1}^{N} k^p$.
3. Écrire et tester une fonction `def Fib(N)` qui donne pour résultat le N-ième nombre de Fibonacci F_N. On rappelle que ceux-ci sont définis par $F_0 = 0$, $F_1 = 1$ et, pour $n \geqslant 2$, $F_n = F_{n-1} + F_{n-2}$.
4. Pour tout entier $n > 0$ on définit a_n comme le plus petit entier $a_n = q \geqslant n$ tel que $\sum_{k=n}^{q} \frac{1}{k} > 1$. Écrire et tester une fonction `def A(n)` qui donne pour résultat a_n.

II. ♡ *Positions dans une liste.* Dans cette partie on s'intéresse à une liste non vide supposée de contenir que des nombres.

1. Écrire et tester une fonction `def Pos(x,L)` qui donne pour résultat la (première) position de x dans L ou `None` si $x \notin L$.
2. Écrire et tester une fonction `def PosMax(L)` qui donne pour résultat la (première) position du maximum des valeurs de L.

III. ⋆ *Diviseurs.*

1. Écrire et tester une fonction `def ListeDiv(n)` dont le résultat est la liste de tous les diviseurs entiers de n, depuis 1 jusqu'à n lui-même. On utilisera éventuellement une écriture en compréhension.
2. En déduire une fonction `def ListePremiers(n)` dont le résultat est la liste de tous les nombres premiers compris dans $[\![2, n]\!]$. On utilisera éventuellement une écriture en compréhension.

On rappelle ici la syntaxe du calcul des quotient `q` et reste `r` de la division (euclidienne) de n par p, `q = n//p` et `r = n%p`.

IV. ⋆ *Nombres parfaits.* On appelle ainsi un entier naturel égal à la somme de ses diviseurs stricts (lui-même exclu). Ainsi 6 est un nombre parfait car ses diviseurs entiers stricts sont 1, 2 et 3 et il vérifie bien $6 = 1 + 2 + 3$.

1. Écrire une fonction `def TesteParf(n)` dont le résultat est `True` si n est un nombre parfait et `False` sinon. Tester avec 33 550 336 en particulier.
2. En déduire une fonction `def ListeParfaits(N)` dont le résultat est la liste des N premiers nombres parfaits. Ne pas dépasser $N = 4$ pour les tests !

V. ⋆ *Nombres d'Armstrong.* On appelle ainsi un entier naturel qui est égal à la somme des p-ième puissance des p chiffres qui le composent. Ainsi 153 en est un car c'est un nombre à 3 chiffres et $1^3 + 3^3 + 5^3 = 153$.

1. Écrire une fonction `def TesteArms(n)` dont le résultat est `True` si n est un nombre d'Armstrong et `False` sinon. Tester avec 8 208 et 93 084 en particulier.
2. Écrire une fonction `def ListeArms(N)` dont le résultat est la liste des nombres d'Armstrong à N chiffres. Tester avec $N \leqslant 5$.

VI. ♡ *Calcul et limite d'une suite définie par récurrence.* La suite étudiée est définie par $u_0 = 0$ et la relation $u_{n+1} = \dfrac{u_n + x}{u_n + 1}$, où $x \in \mathbb{R}^{+*}$. On peut montrer qu'elle a une limite si $n \to \infty$.

1. Écrire et tester la fonction `def U(x,n)` qui calcule u_n pour $n \geqslant 0$.
2. Écrire la fonction `def limU(x,epsilon)` qui évalue la limite de la suite en itérant jusqu'à trouver $|u_n - u_{n-1}| < \epsilon$. Comparer le résultat obtenu à la limite attendue.

VII. ⋆⋆ *Suite de Syracuse.* On appelle ainsi une suite d'entiers naturels définie par récurrence à partir d'un entier naturel quelconque $u_0 = k > 0$; si u_n est pair, u_{n+1} est sa moitié et, sinon, $u_{n+1} = 3u_n + 1$. Ainsi, à partir de $k = 5$ on peut construire la suite 5, $16 = 3 \times 5 + 1$, $8 = 16/2$, 4, 2, 1, 4, 2, ...

Après avoir atteint une première fois la valeur 1, le cycle *trivial* 4, 2, 1, ... se répète indéfiniment. On *admettra* la conjecture de Syracuse selon laquelle la suite de Syracuse de *n'importe quel entier* $k > 0$ strictement positif atteint la valeur 1 puis répète le cycle trivial.

1. Écrire et tester la fonction `def Syracuse(k,n)` qui calcule le n-ième terme de la suite si $u_0 = k$.
2. Écrire et tester la fonction `def LongSyracuse(k)` qui calcule la longueur de la suite entre sa valeur initiale k et la première occurrence de la valeur 1.
3. Quelle est la plus longue suite de Syracuse parmi les 10^4 premières valeurs de k ?

VIII. ♡ *Chaînes de caractères : fonctions simples.*

1. Proposer et tester une fonction `def Pal(S)` dont le résultat est `True` si la chaîne `S` est un palindrome (indépendante du sens de lecture, comme par exemple 'kayak') et `False` sinon.
2. Proposer et tester une fonction `def MaxFreq(S)` dont le résultat est le symbole le plus fréquent de la chaîne `S` (espaces et ponctuations exceptés).

IX. ⋆⋆⋆ *Chaînes de caractères : fonctions plus complexes.*

1. Proposer et tester une fonction `Paires(S)` dont le résultat est le plus grand nombre de caractères consécutifs deux à deux égaux dans la chaîne `S`. Le résultat sera `None` si la chaîne est de longueur inférieure à 2 et 0 si elle ne comporte pas d'éléments consécutifs deux à deux égaux.
2. Une adresse électronique doit vérifier certains critères. Elle est composée de deux parties séparées par le caractère `@` (at) qui ne doit donc apparaître qu'une seule fois. Le reste, partie locale au début et adresse de serveur à la fin, peut contenir des lettres, des chiffres, les symboles particuliers pris dans la liste `!#$%&'*+-=?^_`{|}~/` et le point `.` ; toutefois celui-ci ne peut être ni répété, ni présent au début ni à la fin d'aucune des deux parties (locale et serveur). La partie serveur comporte obligatoirement au moins un point et la désinence finale sui le suit comporte au moins deux caractères : c'est le *top level domain* (TLD) à prendre dans une liste prédéfine (`com`, `org`, ...et le noms de domaine nationaux `fr`, `de`, etc.) Écrire et tester une fonction qui vérifie si une chaîne de caractères est une adresse mail valide ; elle prendra en paramètre une chaîne formée de tous les TDL admis, séparés par des points : `"com.org.fr.de"` par exemple.

X. ⋆⋆ *Fonction de Riemann.* Pour $s > 1$ on définit la fonction zêta de Riemann par la somme $\zeta(s) = \displaystyle\sum_{n=1}^{\infty} \frac{1}{n^s}$ et on peut montrer que, lorsque s est un entier pair, alors $\zeta(s) = \dfrac{\pi^s}{a_s}$ où a_s est un entier.

1. Écrire une fonction `def zeta(s, epsilon)` qui calcule une valeur approchée, avec une précision ϵ donnée sur le premier terme omis dans la somme, de $\zeta(s)$.

2. En déduire le calcul des 4 premiers entiers a_2, a_4, a_6 et a_8.

XI. ⋆⋆ *Quel jour sommes nous ?* En informatique il est d'usage de compter les dates depuis le *jour zéro*, le jeudi 1er janvier 1970. Ainsi une date quelconque peut-elle être considérée :

— comme le triplet `[j, m, a]` des numéros (entiers) du jour, du mois et de l'année (avec donc comme origine la valeur `zero = [1, 1, 1970]`) ;

— comme un décompte de jours, le nombre de jours écoulés depuis le jour zéro ;

— ou même (c'est la fonction `time()` de la bibliothèque `time`) comme le nombre de secondes écoulées depuis le jour zéro à zéro heure (il s'agit d'un nombre flottant qui peut donc être utilisé pour des chronométrages, en fraction de seconde).

1. Écrire et tester une fonction `def Bissextile(a)` logique qui indique les années à 366 jours (années multiples de 400 ou bien multiples de 4 mais pas de 100). En déduire les fonctions `def JoursAnnee(a)` et `def JoursMois(m, a)` donnant le nombre de jours dans une année donnée ou bien dans un mois donné.

2. Écrire et tester une fonction `def Decompte(date, base)` qui indique le nombre de jours qui sépare une date d'une date de base. Vérifier la conformité avec la définition indiquée ci-dessus de la fonction `time.time()`.

3. Écrire et tester une fonction `def Date(decompte, base)` qui calcule une date en fonction d'une date de base et d'un décompte de jours les séparant.

4. Écrire et tester une fonction `def Date(decompte, base)` qui calcule une date en fonction d'une date de base et d'un décompte de jours les séparant.