Informatique II

Langages - & - Syntaxe Python

EXERCICES

Blocs d'Iteration
for et while

Exercices : opérations sur les blocs d'itérations

a - Palindrome

Ecrire une fonction **isPalindrome**(chaine) qui prend en argument une chaine de caractères, et qui renvoie un booléen pour déterminer si cette chaine est un palindrome ou pas (exemple KAYAK).

- Doit-on privilégier une boucle for ou une boucle while ?
- On fera une première fonction à l'aide d'une boucle while
- On proposera une seconde fonction avec une boucle for
  [Puis optimiser votre boucle for avec l'instruction **break**]
- Le fait d'écrire une fonction permet une écriture encore plus concise.
  Refaire ces 2 fonctions en usant habilement de la commande **return**.

- Ecrire une fonction **inverseChaine**(chaine) qui renvoie la chaine passée en argument mais écrite à l'envers.
- Proposer alors une fonction **isPalindromeInv**(chaine) très simple qui détermine à nouveau si la chaine est un palindrome.
- Parmi les méthodes abordées laquelle vous parait la plus rapide ? Comment chiffrer les temps de calcul ?

b - **La suite Syracuse [une question à 1000€]**

On a Sn la suite de Syracuse, définie comme suit :

Soit $S_0 = N$ avec N un entier naturel non nul

$$Si \quad S_n \quad pair \quad \Rightarrow \quad S_{n+1} = \frac{S_n}{2}$$

$$Si \quad S_n \quad impair \quad \Rightarrow \quad S_{n+1} = 3S_n + 1$$

On veut montrer que Sn converge vers 1 pour tout N !!!

- Réaliser un programme qui demande un entier N (avec la commande input) et qui affiche la succession des valeurs Sn puis le nombre d'étapes pour arriver à 1

- Tester votre code sur 10 valeurs de N d'ordres de grandeur très différents. Conclusion ?

- On voudrait PROUVER l'algorithme c-à-d démontrer qu'il donnera toujours 1 en un nombre fini d'étape. Quel type d'argument faut-il apporter.
  Je vous donne 1000€ si vous me donner la démonstration complète !!!

c - Séquence ADN

- Écrire un programme qui demande à l'utilisateur de saisir une séquence ADN sous la forme d'une chaine de caractère et qui détermine la séquence du brin complémentaire comme ci-dessous :
[Rq : on pourra introduire une liste de correspondance pour déterminer la complémentarité]
- Vérifier votre code en calculant le complémentaire du complémentaire.

```
$ python3 ADN_comp.py

entrer une sequence ADN (A,C,G et T uniquement) :
ACGTATATCGATAGCTAATCGATAGTACGATAGTCTAGATAGCTTTAGTATCGTATCCGATCGTTAGCTAGCT
sequence OK

Sequence :
ACGTATATCGATAGCTAATCGATAGTACGATAGTCTAGATAGCTTTAGTATCGTATCCGATCGTTAGCTAGCT
Complementaire :
TGCATATAGCTATCGATTAGCTATCATGCTATCAGATCTATCGAAATCATAGCATAGGCTAGCAATCGATCGA
```

- Écrire une fonction **rechercheGène**(chaine, mot) qui renvoie la liste de tous les indices où commencent le mot en question dans la chaine si il y a en. Si il n'y en a pas elle renvoie une liste vide.
On peut procéder par une double boucle ou être plus astucieux.
- Rechercher le gène "ATATA" dans "ATATATACGTAGTCATATATATACCGATATA"
- Soit N et M les longueurs de la chaine et du mot : évaluer le coût en temps de l'algorithme.

d - Chiffres romains

Écrire un programme qui demande une année en chiffre romains et qui renvoie l'année en chiffres romains convertie en notation décimale.

```
Entrer une année en chiffres romains
MCMLXXXIV
1984
```

On introduira une liste de liste pour les correspondances :

```
convert = [ [   'M',   'D',  'C',  'L',  'X',  'V',  'I' ],
            [1000,  500,  100,  50,   10,    5,   1  ] ]
```

- Écrire une fonction **valDec**(carac) qui prend en argument un caractère romain majuscule et renvoie la valeur associée.

L'idée est de faire la somme des nombres correspondant aux lettres, mais parfois celles-ci sont à compter négativement. A vous de trouver comment identifier le signe et de calculer l'année en décimal.

- Écrire une fonction **roman2dec**( ) qui demande une chaine et réalise ce programme.

e - Tracé d'un polygone et calcul de Pi en précision machine

Soit a le coté d'un polygone et N son nombre de coté.

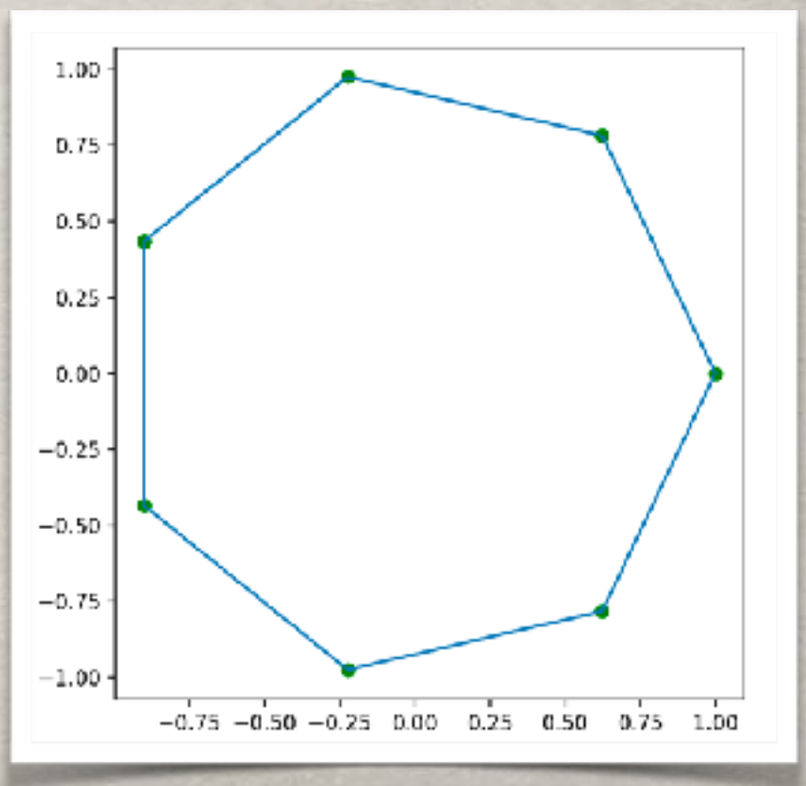
- Écrire une fonction **polygone**(a, N) qui renvoie la liste des points P avec P un tuple (x, y).
  On procèdera pour cela par des rotations en complexe avec le 1er point en (a, 0)
  [**from math import *** Utiliser **e**(alpha * 1J)** : rotation d'angle alpha et les **attributs .real .imag**]

- Écrire une fonction **coordXY(poly)** qui a partir du polygone défini ci-dessus renvoie le tuple (X, Y) des listes des abscisses X, et des ordonnées Y. Pour pouvoir faire le tracé.

- Récupérer les deux listes et faire le tracé d'un polygone avec plt.plot(X,Y); plt.show()
[from matplotlib import pyplot as plt]. Quel est le problème ? Comment corriger les deux listes ?

- Ecrire une fonction **surfacePolyN**(a, N) qui renvoie la surface du polygone.
  En déduire une valeur approchée de pi à l'aide de la surface d'un disque.

- Tester la convergence de la suite en regardant l'écart avec pi
  en mémoire dans python. Quelle valeur de N faut-il au minimum
  pour converger en précision machine (16 chiffres) ?



Exercices de cours :

a - **Algorithme d'Euclide**
Ecrire l'algorithme étudié en classe [a venir] et le vérifier sur quelques exemples

b - **La tête au carré**
Ecrire un algorithme pour trouver tous les carrés qui s'écrivent sous la forme décimale
$aabb|_{10} = N^2$ [a et b compris entre 0 et 9]

c - **La suite de Finonacci Un**
On se propose d'étudier la reproduction des lapins ! On a au départ 1 couple de jeunes lapins : $U_0=1$. Un couple ne donne naissance à un autre couple qu'à partir du deuxième mois et donne un couple tous les mois : $U_0=1$ $U_1=1$ $U_2=1$

- Montrer que la suite peut s'écrire : $U_{n+1} = U_n + U_{n-1}$
- Calculer les 10 premiers termes de la suite de Fibonacci à la main.
- Ecrire un code qui calcule les N premiers termes de cette suite.

d - **Back to the roots !**
Soit Un la suite définie par $u_{n+1} = \frac{1}{2}\left(u_n + \frac{x}{u_n} \right)$ où x est un réel positif.

- Ecrire un code Python qui affiche les 20 valeurs successives de Un.
On choisira différentes valeurs de x et de U_0 pour tester la convergence.
- Vers quelle valeur converge la suite ?

E - **Le crible d'Erathostène**

On se donne la liste des entiers jusqu'à N :
Utiliser la méthode .**remove** pour enlever de la liste les nombres qui ne sont pas premiers et pour obtenir à la fin de l'algorithme, la liste des nombres premiers.

f - **Mad Max**

On donne un tableau de nombres entiers quelconques Tab. On veut trouver un algorithme qui va retourner la valeur maximale «Max» de ce tableau.

- Proposer un algorithme en réfléchissant sur la base du tableau suivant :

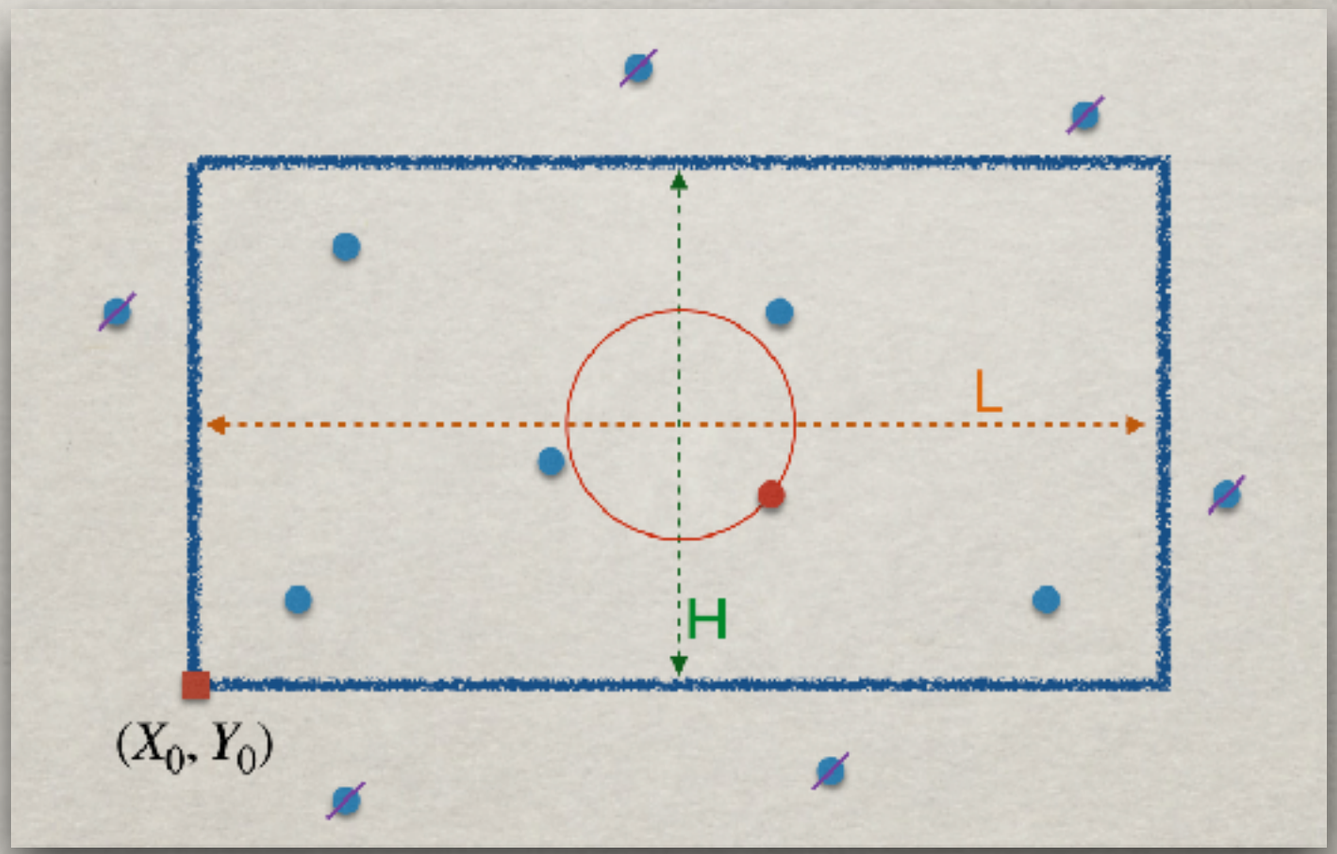
Tab = [15, 23, 57, 2, 85, 11, 15, 37, 8, 97, 64, 71, 3]

- On présentera d'abord l'algorithme sous la forme d'un diagramme à flèches.
- Ecrire le code Python et tester votre algorithme.
- Comment modifier l'algorithme pour trouver le minimum ?

- Peut-on prouver l'algorithme : proposer une preuve.
- Estimer la complexité de cet algorithme.

Problème : point le plus proche

[D'après CCP 2018]



Rq : ce petit algorithme intervient dans la reconstitution de la trajectoire d'une balle de tennis avec le système Hawk Eye afin de savoir si la balle sort ou non.

Soit une liste de points [P1, P2 , … , PN] où chaque point est sous la forme d'un tuple de ses coordonnées dans le plan (Px, Py)

Soient X0, Y0, L > 0 , H > 0 les bornes d'un rectangle dans le plan (cf figure).
window = (X0, Y0, L, H)

a - Ecrire une fonction **select**(listePoint, window) qui prend en argument la liste de points et la fenêtre et qui renvoie la liste des points qui sont au sein du rectangle bords inclus.

b - Ecrire une fonction **ballFinder**(listePoint, window) qui renvoie le point le plus proche du centre du rectangle si il est par ailleurs dans le rectangle et qui renvoie None si il n'y a pas de point au sein du rectangle

c - Vous proposerez une validation de vos fonctions à l'aide d'un exemple de points et d'une fenêtre de votre choix. Bonus : faire une représentation graphique et un générateur aléatoire de points.

Module Turtle : «Dessine moi une tortue»

```
>>> import turtle          # permet d'imporer le module turtle pour le dessin
                           # attention au mode graphique choisi

>>> turtle.down()          # pose le crayon
>>> turtle.up()            # leve  le crayon

>>> turtle.forward(100)    # avance le crayon de 100 pixel (=> trait si down)
>>> turtle.left(45)        # tourne à gauche de 45°
```

Rq : de même turtle.back => recule , turtle.right(90) tourne à droite

Exercices :

a - Tracer un carré de coté 123, un triangle rectangle [100, 200, ?]

b - Trouver un algorithme pour tracer un polygone de N-cotés de tailles 100