Mathématiques supérieures le 9 Novembre 2023

TP n°1 - Mise en pratique : les modules `random` et `turtle`

TP Noté

Tous les documents sont autorisés.

IMPORTANT

À lire *attentivement* avant de commencer

Comment présenter le fichier à rendre ?

- Créer et enregistrer dans votre dossier personnel un nouveau fichier dans EduPython intitulé :

  NOM1-NOM2-NOM3.py

  où NOMi est à remplacer par les noms de chaque personne du groupe (en omettant les accents et espaces éventuels qu'il contient).

- Ce fichier sera présenté de la façon suivante :

```
##### NOM1 Prénom1 - NOM2 Prénom2 - NOM3 Prénom3

####Partie 1

##Q.1

"Votre résolution de la question Q1:"

##Q.2

#Q.2.a

"Votre résolution de la question Q2:a)"

etc...

####Partie 2

etc...
```

- Enregistrez ce fichier **régulièrement** - Ctrl+S !
- Une fois votre travail terminé, rendez-vous à l'adresse `https://classexo.fr/info` puis envoyez votre fichier.

1. Echauffement

Q1 : On dit qu'un mot est un *palindrome* s'il peut se lire de la même façon dans les deux sens de lecture. Par exemple, "radar" ou "ressasser" sont des palindromes.

Écrire une fonction `palindrome(chaine)` qui prend en argument une chaîne de caractères `chaine` et qui **renvoie** `True` si la chaine de caractères `chaine` est un palindrome et `False` sinon.

Q2 : On considère la suite récurrente :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \dfrac{u_n}{1+u_n^2} \end{cases}$$

a) Écrire une fonction `suite_u(n)` qui prend pour argument `n` un entier et qui **renvoie** le `n+1`-ième terme u_n de la suite récurrente précédente.

b) Écrire une fonction `indice_u(s)` qui prend pour argument `s` un nombre de type `float` de l'intervalle $]0,1[$ et qui **renvoie** le premier indice $n \in \mathbb{N}$ de la suite $(u_n)_{n\in\mathbb{N}}$ tel que $u_n \leq s$.
Par exemple, l'instruction `indice_u(0.02235)` *doit renvoyer* `999`

Dans la suite de ce TP, nous allons travailler les bases de Python vues dans le 1er chapitre du cours à travers les modules `random` et `turtle`. On commencera par importer ces deux modules :

```
from random import *
from turtle import *
```

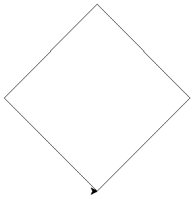
On rappelle les instructions suivantes concernant le module `turtle` :

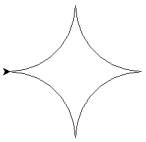
- l'instruction `reset()` permet d'ouvrir une nouvelle fenêtre de dessin lorsqu'il n'y en a pas avec la tortue "de base" ; et de réinitialiser tous les dessins de la tortue de base si la fenêtre est déjà ouverte.
- l'instruction `bye()` permet de fermer la fenêtre de dessin.

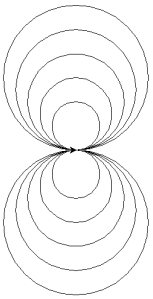
2. Exercices avec `random` et `turtle`

Q3 : Donner des suites d'instructions qui permettent de reproduire les dessins suivants avec le module `turtle` ; on commencera par ouvrir une fenêtre de dessin via la commande `reset()`.

Remarque : les longueurs ne sont pas importantes, les résultats de vos suites d'instructions doivent avoir la forme la plus proche possible du dessin demandé.

a) 

b) 

c) 

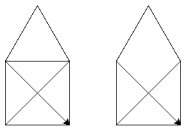
Q4 : Dans cette question, on utilisera le module `random`.

a) Écrire une fonction `de()` qui renvoie aléatoirement un entier uniformément entre 1 et 6.

b) Écrire une fonction `brelan()` qui **affiche** le résultat de 3 lancers de dés (réalisés avec la fonction `de()`) et qui renvoie `True` si les 3 lancers forment un brelan (= 3 résultats identiques) et `False` sinon.
Indication : dans le corps de la fonction, créer 3 variables `d1`, `d2` et `d3` qui contiendront les résultats de chaque lancer.

c) *Bonus :* Modifier la fonction précedente en commentant la partie d'affichage - `brelan()` n'affichera donc plus rien mais renverra toujours `True` ou `False` en fonction du résultat des lancers. Puis écrire une fonction qui permet de déterminer la proportion empirique du nombre de brelans réalisés dans un échantillon de `n` tentatives de brelans (où `n` est un entier qui sera argument de la fonction). Comparer le résultat de la fonction pour `n` grand avec l'espérance mathématique de la variable aléatoire de Bernouilli associée i.e. $X = 1$ si le brelan est réussi, 0 sinon.

Q5 : Ouvrir une fenêtre de dessin via la commande `reset()`.
Faire tracer à la tortue le dessin de gauche ci-dessous, sans utiliser l'instruction `penup()` (on pourra prendre 100 pixels pour la longueur des côtés du carré ; le toit est un triangle équilatéral) :

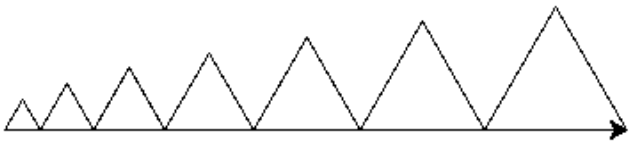


Bonus : est-il possible de faire réaliser le dessin de droite à la tortue avec les mêmes contraintes ?

Q6 : Ouvrir une fenêtre de dessin via la commande `reset()`.

a) Écrire une fonction `triangle(p)` qui prend en argument un entier `p` et qui trace un triangle équilatéral de côtés de longueur `p` pixels. À la fin du tracé, la tortue doit être dirigée horizontalement vers la droite.

b) Écrire une fonction `suite_triangle(n)` qui prend en argument un entier strictement positif `n` et qui trace une suite de `n` triangles équilatéraux côte-à-côte telle que les côtés du premier triangle sont de longueur 20 pixels et les côtés du $i+1$-ème triangle sont de longueur 10 pixels de plus que celle des côtés du i-ème triangle. Voici ce que doit tracer l'instruction `suite_triangle(7)` :



3. Jeu du "Pierre Feuille Ciseaux"

Le jeu du "Pierre Feuille Ciseaux" est très simple : deux joueurs présentent simultanément une de leurs mains devant eux en forme de pierre, feuille ou ciseaux. On désigne le vainqueur en comparant les mains selon la règle suivante : la pierre gagne sur les ciseaux, qui gagne sur la feuille, qui gagne sur la pierre. Si les joueurs jouent la même main, ils recommencent le jeu jusqu'à ce qu'un vainqueur soit désigné.

Q7 : Écrire une fonction `PFC()` qui **retourne** aléatoirement une chaîne de caractères uniformément parmi `'Pierre'`, `'Feuille'` et `'Ciseaux'`

Q8 : Écrire une fonction `comparaison(resultat1,resultat2)` qui :

- prend en argument deux chaînes de caractères `resultat1` et `resultat2` prenant pour valeurs potentielles `'Pierre'`, `'Feuille'` ou `'Ciseaux'`
- et **retourne** `1` si `resultat1` est gagnant sur `resultat2` ; `-1` s'il est perdant et `0` s'ils sont égaux.

Q9 : Écrire une fonction `JeuPFC(joueur1,joueur2)` qui prend en argument deux chaînes de caractères `joueur1` et `jouer2` prenant pour valeurs les noms des deux joueurs au "Pierre Feuille Ciseaux" et réalise le jeu suivant :

- fait "jouer" chacun des deux joueurs grâce à la fonction `PFC` et **affiche** le résultat de chaque joueur
- compare leurs résultats grâce à la fonction `comparaison`
- recommence les deux étapes précedentes tant qu'il y a match nul
- puis **affiche** le nom du vainqueur

Remarque : on pourra également réaliser un compteur de tour pour l'afficher à chaque étape. Voici un exemple de ce qu'on pourra obtenir :

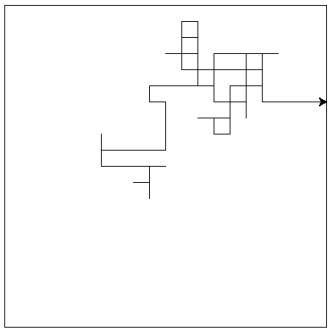
```
>>> JeuPFC('Sainte Croix','Saint Euverte')
Tour de jeu 1  :  Sainte Croix fait Pierre et Saint Euverte fait Feuille
Fin du match : Saint Euverte a gagné haut la main !

>>> JeuPFC('Sainte Croix','Saint Euverte')
Tour de jeu 1  :  Sainte Croix fait Pierre et Saint Euverte fait Ciseaux
Fin du match : Sainte Croix a gagné haut la main !

>>> JeuPFC('Sainte Croix','Saint Euverte')
Tour de jeu 1  :  Sainte Croix fait Ciseaux et Saint Euverte fait Ciseaux
Tour de jeu 2  :  Sainte Croix fait Ciseaux et Saint Euverte fait Ciseaux
Tour de jeu 3  :  Sainte Croix fait Pierre et Saint Euverte fait Ciseaux
Fin du match : Sainte Croix a gagné haut la main !
```

4. Une marche aléatoire de la tortue (NON NOTÉ)

On souhaite réaliser une marche aléatoire de la tortue sur un quadrillage ”imaginaire” de la fenêtre de dessin du module `turtle` dans un carré de côté de longueur 400 pixels. Voici une illustration de ce qui est attendu :



Q10 : Quels sont les effets de la fonction suivante ?

```
def initialisation():
    reset()
    tcadre = Turtle()
    tcadre.penup()
    tcadre.forward(200)
    tcadre.left(90)
    tcadre.pendown()
    tcadre.forward(200)
    for k in range(4):
        tcadre.left(90)
        tcadre.forward(400)
    tcadre.hideturtle()
```

Q11 : Écrire une fonction `deplacement(r)` qui prend en argument un entier `r` compris entre 0 et 3 et qui, en supposant que la tortue de base est présente dans la fenêtre de dessin et est dirigée horizontalement vers le droite :

- si `r` vaut 0, déplace la tortue de 20 pixels vers la **droite** puis met la tortue dirigée horizontalement vers le droite ;
- si `r` vaut 1, déplace la tortue de 20 pixels vers la **gauche** puis met la tortue dirigée horizontalement vers le droite ;
- si `r` vaut 2, déplace la tortue de 20 pixels vers le **haut** puis met la tortue dirigée horizontalement vers le droite ;
- si `r` vaut 3, déplace la tortue de 20 pixels vers le **bas** puis met la tortue dirigée horizontalement vers le droite ;

Q12 : Écrire une fonction `marche()` qui réalise la marche aléatoire de la tortue présentée précédemment avec comme condition d'arrêt, la condition suivante : si la tortue ”touche” un bord du carré dessiné par la fonction initialisation, on arrête la marche. On fera renvoyer à la fonction `marche()` le nombre de pas de la tortue lors de sa marche.
On appellera la fonction `initialisation()` au tout début de la fonction `marche` afin de tracer le

cadre de la marche aléatoire.

Quelques indications :

— Créer deux variables `h` (pour "horizontal") et `v` (pour "vertical") qui, au fur et à mesure de la marche, enregistrent respectivement la position horizontale et verticale de la tortue (leur but est de nous permettre de savoir quand la tortue touche un bord du carré !)

— On pourra créer une variable `r=randint(0,3)` et l'instruction `deplacement(r)` pour effectuer chaque pas aléatoire.