CONTROLE FLASH - PYTHON *(45 minutes)*

DATE : Novembre 2016 **GROUPE : 3**

NOM : **PRENOM :**

Q1. Complétez les lignes vides dans la session ci-dessous au toplevel.

```
>>> L = [[i,i+1] for i in range(3)]
>>> L

>>> L[0] = L[1]
>>> L

>>> L[0][0] = L[0][0] + 1
>>> L

>>> L[1] = 0
>>> L
```

Q2. Sans aucune **def**, rédigez un petit programme Python affichant le nombre d'entiers premiers inférieurs à 100000 et ne comportant pas le chiffre 2, ainsi que le plus grand d'entre eux. On suppose qu'il existe une fonction **est_premier(n)** retournant True ou False suivant que n est premier ou pas. *Pour ce problème (de niveau cc1) vous n'êtes pas obligé d'utiliser une liste.*

Q3a. Programmez une fonction **liste(f, p, a, b)** prenant deux fonctions f et p définies sur les entiers, ainsi que deux entiers $a \leq b$. Cette fonction retournera la liste des f(x) pour tous les entiers $x \in [a,b]$ vérifiant p (donc tels que p(x) vaut True).

```
def liste(f,p,a,b) :
```

Q3b. Comment utiliseriez-vous en une seule ligne au toplevel la fonction **liste** ci-dessus pour voir la liste des carrés des nombres premiers de [5,15] ? Utilisez la fonction **est_premier** de **Q**2.

```
>>> liste(
[25, 49, 121, 169]
```

Q4a. Programmez une fonction **promenade(n)** prenant un entier n > 0. Cette fonction va demander à la tortue d'avancer n fois de 50 pixels mais en tournant auparavant chaque fois d'un angle aléatoire. Après chaque avancée de 50 pixels, vous stockerez la position *(x,y)* de la tortue dans une liste de tuples et cette liste sera retournée comme *résultat* de la fonction **promenade**. Exemple :

```
promenade(4) --> [(15.45,-47.55), (62.73,-31.27), (19.00,-55.51), (-29.91,-45.12)]
```

```
def promenade(n) :          # on suppose turtle déjà importé
```

Q4b. Soit **L = promenade(20)**. Sans aucun ***def***, écrivez quelques lignes de code Python permettant d'afficher un point *(x,y)* **le plus bas** de la liste L (donc avec y minimum) atteint par la tortue. On trouverait (19.00,-55.51) dans l'exemple précédent.

Q5. On rappelle l'algorithme du calcul **dichotomique** de a^n par **récurrence**, pour a réel et n entier ≥ 0 : $a^0 = 1$ $a^{10} = (a^2)^5$ $a^{11} = a \times a^{10}$

Appliquez cet algorithme aux **polynômes creux** en programmant *en 3 lignes* une fonction **puis(p, n)** prenant un polynôme creux p et un entier n ≥ 0, et retournant le polynôme creux p^n.

```
puis([[1,1],[-1,0]],3) == [[1,3],[-3,2],[3,1],[-1,0]]        #  (X-1)³ = X³-3X²+3X-1
```

```
def puis(p,n) :             # par récurrence, et dichotomie
    if
```