Seconde, 12 novembre 2012

Présentation de Python et du module turtle

1 Environement Python et remarques générales

1.0.1 Lancer l'interpréteur Python :

Sous linux-ubuntu vous allez dans le menu Application/programmation et vous cliquez sur le programme IDLE
Sous windows : dans le menu Windows, cliquer sur le programme IDLE.

Dans la fenêtre « Python Shell » qui s'est ouverte choisir File/New Window. **C'est dans cette deuxième fenêtre que vous allez saisir votre programme. Pas dans la fenêtre « Python Shell ».**

Dans la nouvelle fenêtre qui s'est ouverte : File/Save As ... (indiquez le nom et l'emplacement du fichier qui contiendra votre programme ; donnez-lui le suffixe **.py**). Vous pouvez choisir votre clé USB (c'est le mieux) ou bien creer un répertoire à votre nom.

Saisissez votre programme. La coloration syntaxique permet de détecter les erreurs de syntaxe les plus superficielles. Pour sauvegarder : Ctrl-S. Pour lancer votre programme : F5. La fenêtre « Python Shell » devient la fenêtre active. C'est dans cette fenêtre qu'ont lieu les Entrées/Sorties de votre programme. C'est aussi dans cette fenêtre que s'affichent les messages d'erreur.

Pour les erreurs de syntaxe, revenir dans la fenêtre où vous avez saisi votre programme : le petit carré rouge qui y est apparu est en général juste après votre erreur de syntaxe.

Pour les autres messages d'erreur, lire le libellé de la première erreur affichée et relever le numéro de ligne indiqué. Dans la fenêtre où vous avez saisi votre programme : Alt-G puis saisissez le numéro de la ligne. Le curseur se positionne alors sur la ligne où s'est produit l'erreur.

Si un programme boucle indéfiniment, l'arrêter avec Ctrl-C.

1.0.2 Remarques générales d'informaticien

1. De préférence, les noms de fichiers ne doivent pas avoir d'espace ou de caractères spéciaux type ', !, : ...
2. Les raccourcis clavier suivants sont bien profitables :
   Ctrl S pour sauvegarder
   Ctrl C pour copier
   Ctrl V pour coller
   Ctrl X pour couper

1.1 Memento de turtle

Le langage Logo a été inventé pour initier les enfants à la programmation. Une tortue parcourt l'écran en suivant les ordres qui sont donnés par l'enfant. Elle tient un crayon au bout de sa queue, de sorte que sa trajectoire s'affiche à l'écran. Voici les principales instructions qui peuvent être données à la tortue :
Voici un aide mémoire des principales instruction de turtle.

`reset()` On efface tout et on recommence
`goto(x, y)` Aller à l'endroit de coordonnées x, y
`forward(distance)` Avancer d'une distance donnée
`backward(distance)` Reculer
`up()` Relever le crayon (pour pouvoir avancer sans dessiner)
`down()` Abaisser le crayon(pour recommencer à dessiner)
`color(couleur)` Couleur du tracé. <couleur> peut être une chaîne prédéfinie ('red', 'blue', 'green', etc.)
`bgcolor(couleur)` Couleur du fond d'écran. <couleur> peut être une chaîne prédéfinie
`left(angle)` Tourner à gauche d'un angle donné (exprimé en degrés)
`right(angle)` Tourner à droite
`width(épaisseur)` Choisir l'épaisseur du tracé
`fill(1)` Remplir un contour fermé à l'aide de la couleur sélectionnée
`write(texte)` <texte> doit être une chaîne de caractères délimitée avec des " ou des '

2 On commence pour de vrai

Python a été installé avec différents modules, spécialisés dans différentes tâches. Pour utiliser un module il faut commencer par l'importer. Ici notre programme va utiliser le module « turtle ». Il doit donc commencer par `from turtle import *` sur la première ligne. Pour voir la tortue à l'écran, il faut lui donner une forme. Cela se fait avec l'instruction `shape("turtle")`. On peut ensuite définir la taille de la fenêtre dans laquelle va se déplacer la tortue avec l'instruction `screensize(l,h)` où l et h sont les largeurs et hauteurs en pixels. Enfin c'est une bonne habitude de commencer par effacer l'écran avec l'instruction `clear()` et puis reinitialiser toutes les variables par `reset()`. Les cinq premières lignes du prochain fichier seront à répéter systématiquement.

2.1 Succession d'instructions

Pour donner plusieurs instructions successives à la tortue, il suffit d'écrire ces instructions à la suite les unes des autres, en allant à la ligne à chaque fois, avec la même indentation.

1

1. Ecrire le programme suivant nommé `carre.py`

```
from turtle import *
shape("turtle")
screensize(1000,1000)
reset()
clear()
forward(200)
right(90)
forward(200)
```

2. Compléter le programme pour qu'il dessine un carré.
3. Modifier le programme pour que le fond de la fenêtre soit noir et le tracé en blanc (utilser le memento du paragraphe précédent et les commandes `bgcolor color` à placer après `reset()` et avant les commandes de déplacement de la tortue.

2.2 Variables

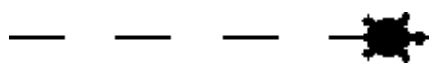
Bon et si maintenant on veut faire un carré deux fois plus grand ou deux fois plus petit. Il va falloir changer le 200 dans le programme précédent par 400 (et cela 4 fois). Il est donc intréssant d'introduire une variable que l'on nommera taille et qu'on aura à changer qu'une seule fois sur une seule ligne du programme. En Python comme dans beaucoup de programme les variables peuvent prendre les noms que l'on veut mais sans accent et sans caractères spéciaux du type &, !, :, ... Le programme aura alors cette allure :

```
...
taille = 400
...
forward(taille)
right(90)
...
```

2 Modifier le programme carre.py en conséquence et vérifier qu'il marche.

2.3 Boucles « pour »

3 Avec les instructions `up()` et `down()` du mémento faire un programme pointilles.py qui réalise la figure suivante. Vous utiliserez une variable `taille` pour la taille des pointillés (taille=20 sur le dessin).

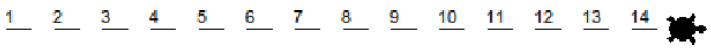


Vous avez du remarquer que votre programme effectue 4 fois la même chose et si on veut faire une ligne plus longue c'est encore pire... Pour éviter cela on utilise donc une boucle pour.

```
from turtle import *
shape("turtle")
screensize(1000,1000)
reset()
clear()
taille=15
for i in range(1,15) :
 write(i)
 forward(taille)
 up()
 forward(taille)
 down()
```

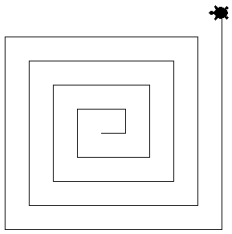
Les lignes indentées (décalées vers la gauche) sont répétées 14 fois : une première fois avec la variable i valant 1, une deuxième fois avec i valant 2, une troisième fois avec i valant 3 et une quatrième fois avec i valant 4 etc ... Pour obtenir ce résultat on fait précéder ces cinq lignes par « for i in range(1,15) » qui crée une variable i et lui donne successivement toutes les valeurs entières de 1 inclus à 15 exclus. La commande `write(i)` permet d'écrire la valeur de `i` à l'endroit ou se trouve la tortue.

4 Tapez ce programme `pointilles2.py` et testez-le. Vous devez obtenir la figure suivante.



Avez-vous compris ce que fait cette boucle ? au début `i`=1 puis on trace un trait et un blanc puis on recommence avec `i`=2 ...

5 Ecrire maintenant un programme nommé `SpiraleCarree.py` qui réalise la figure suivante :
Pour cela vous prendrez une longueur de base par exemple (taille=30). On commence par les deux premiers traits du centre qui on même longueur (taille) puis les deux suivants qui ont pour longueur 2*taille et ainsi de suite. Il s'agit donc de faire une boucle et d'utiliser la variable `i` dans la boucle. Vous devriez avoir une instruction du type `forward(i*taille)` dans votre programme ...



6 Revenons à notre programme qui trace un carré. Vous avez remarqué qu'il répète 4 fois les mêmes instructions. Simplifez-le en utilisant une boucle « pour ».
Dans ce dernier exercice, vous avez remarqué que la variable `i` (vous n'êtes pas obligé de l'appeler `i` d'ailleurs) n'est pas utilisée à l'intérieur de la boucle. Ainsi une boucle « pour » peut être utilisée pour répéter plusieurs fois exactement les mêmes instructions.

2.4 Procédures

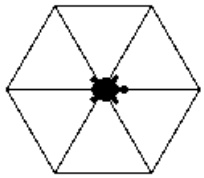
7 Vous n'avez sans doutes pas envie, à chaque fois que vous voulez tracer un carré, de réexpliquer à la tortue comment faire. Vous voudriez lui apprendre une bonne fois pour toutes comment faire, et qu'elle s'en souvienne ; de manière à pouvoir simplement lui dire « trace un carré ». C'est à cela que servent les procédures. Nous allons créer une procédure qui trace un carré. Voici la syntaxe à respecter.

```
def carre(taille) :
 # Ici taille est un parametre de la procedure
 # Il peut y en avoir plusieurs
 # Copier ici les instructions qui tracent
 # le carre en les indentant (decalage vers la droite)
# On revient a une indention nulle
carre(100) #trace un carre de cote 100
```

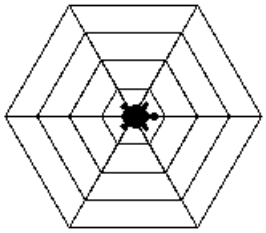
A cette occasion nous découvrons comment mettre des commentaires dans un programme, c'est-à-dire du texte libre qui ne sera pas pris en compte à l'exécution du programme : tout ce qui est entre # et la fin de la ligne est un commentaire.
La définition se termine lorsqu'on revient à une indentation nulle. Pour que la tortue trace effectivement un carré, il faut lui en donner l'ordre avec l'instruction carre(100) sans indentation.
Ecrire un programme `procedure.py` pour tester la procédure `carre`.

8 Compléter le programme `procedure.py` en y rajoutant une procédure à un paramétre `triangle(taille)` qui permet de tracer un triangle équilatéral et réaliser la figure suivante (avec une boucle "pour" et en utilisant les fonctions `up()` et `down()`).

9 Ecrire un programme `hexagonne.py` qui trace un hexagone en utilisant notre procédure « triangle » et une boucle « pour ».



10 Ecrire un programme qui trace une toile d'araignée en utilisant notre procédure « triangle » et deux boucles « pour » imbriquées l'une dans l'autre. Attention la variable dans chaque boucle ne peut pas avoir le même nom : `i` pour l'une et `j` pour l'autre (par exemple).



2.5 Instructions d'entrée et de sortie

2.5.1 Instructions de sortie

Pour afficher un résultat, un programme peut utiliser l'instruction print(expr1,expr2,...) qui évalue les expressions expr1, expr2, ... puis affiche le résultat dans la fenêtre "Python Shell". Certaines de ces expressions peuvent être des chaînes de caractères entre guillemets.

```
print("La somme de 2 et 3 vaut",2+3)
```

Remarque 1 *Pour afficher un résultat dans la fenêtre turtle nous avons déjà vu qu'on utilise la commande write.*

2.5.2 Instructions d'entrée

La fonction input permet à l'utilisateur d'entrer des données pour interagir avec le programme. Pour que l'ordinateur demande un entier, un réel respectivement une chaine de caractères à l'utilisateur on utilise l'une des 3 commandes suiavantes :

```
age=int(input("Entrez votre age : "))
nombre=float(input("donner un reel "))
nom=input("Quel est votre nom ?")
```

La fonction input a un argument, la chaîne de caractères ("Entrez votre âge : " par exemple). C'est ce message qui sera affché dans la fenêtre « Python Shell » pour demander à l'utilisateur de saisir son âge. Lorsque l'utilisateur aura saisi son âge (supposons qu'il ait 17 ans) et frappé sur la touche Entrée, la fonction input retournera la chaîne de caractères "17". La fonction input retourne toujours une chaîne de caractères. Si nous avons besoin de l'âge sous la forme d'un entier, il faut appliquer la fonction de conversion int comme ici. On applique float si on veut un réel.

2.6 Test de condition

On peut utiliser des tests logiques pour que le programme puisse prendre des décisions selon la valeur de certaines variables. La syntaxe est la suivante :

```
if <condition> :
  # Bloc d'instructions num 1
else :
  # Bloc d'instructions num 2
```

Les opérateurs de comparaison sont les suivants :

- $<$ pour strictement inférieur.
- $>$ pour strictement supérieur.
- $<=$ pour inférieur ou égal.
- $>=$ pour supérieur ou égal.
- $==$ pour égal.
- $!=$ pour different de .

On utilise aussi `or` et `and` pour combiner des conditions.

Exemple 1

```
a=float(input("donner a")
```

```
if a > 0:
   print("a est superieur a 0.")
else:
   print("a est inferieur ou egal a 0.")
```

On peut aussi utiliser la commande elif qui est la contraction de else if (lorsque on enchaine plusieurs tests).

Exemple 2

```
a=float(input("donner a")
if a > 0: # Positif
     print("a est positif.")
elif a < 0: # Negatif
     print("a est negatif.")
else: # Nul
        print("a est nul.")
```

11 Ecrire un programme test-triangle.py qui demande 3 réels $a; b; c$ qui détermine le plus grand d'entre eux et teste ensuite si le triangle est rectangle. Vous completerez le programme suivant :

```
a=float(input("donner a : "))
b=float(input("donner b : "))
c=float(input("donner c : "))

if a>b and a >c :
    print("a est le plus grand")
    max=a
    d1=b
    d2=c
elif  b>a and b >c :
    print("b est le plus grand")
    max=b
    ...
```

2.7 boucles « tant que »

La syntaxe génerale de cette boucle est la suivante :

```
while <condition> :
 <bloc de boucle>
```

Tant que la condition est respectée la boucle est effectuée.

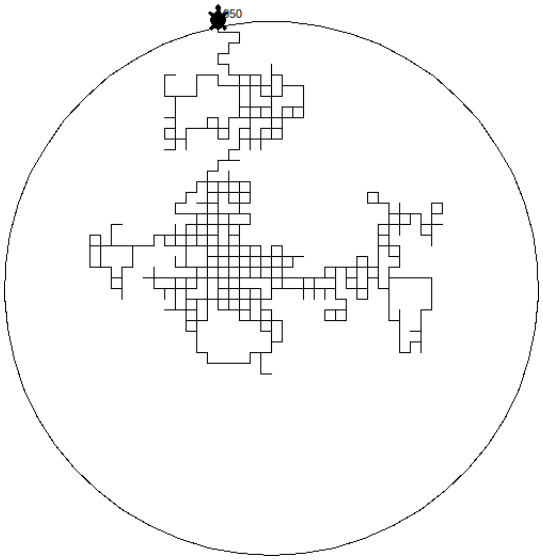
Exemple 3 *Le programme suivant demande un nombre $a > 10$. Tant que ce n'est pas le cas, il redemande :*

```
a=0
while a<10 :
 a=float(input("rentrer a"))
```

12 Ecrire un programme tortue-folle.py qui fonctionne de la manière suivante :

1. On fixe une variable rayon égal à 200 et une variable lm à 10.
2. On trace un cercle centré en (0,0) à l'aide des commandes `circle(rayon,360)` et`goto(x,y)`.
3. La tortue effectue des déplacements élémentaires de taille lm et à la fin de chacun d'entre eux elle peut continuer dans le même sens, tourner à gauche, à droite ou faire demi-tour (elle choisit au hasard). Ici on utilisera la commande i=randint(1,4) qui renvoie un entier aléatoire entre 1 et 4 en ajoutant au début du programme `from random import *` qui permet d'utiliser random.
4. la tortue se promène ainsi et le programme s'arrete lorsqu'elle sort du cercle. On utilise pour cela une boucle du type `while distance(0,0)<rayon :` où `distance(0,0)` renvoie la distance de la tortue à l'origine.

Vous devriez avoir une figure de ce genre :



Voilà un petit peu de recherche sur internet vous permettra de faire encore plein de choses avec cette petite tortue ... En vrac, voilà quelques figures.