I11 - Programmation I : Python
TP 6

Semestre 1 2017

Module PocketGL

Dans les exercices suivants, on fera appel au module `pocketgl`. Ce dernier propose quelques fonctions qui simplifient le travail du programmeur qui souhaite avoir une sortie graphique 2D. Il comporte notamment les fonctions suivantes :

- `init_window(pnom, pla, pha)` : à partir du moment où l'on souhaite afficher une fenêtre graphique, cette fonction doit être la *première* appelée des fonctions du module `pocketgl`,
- `main_loop()` : cette fonction doit être la *dernière* appelée des fonctions du module `pocketgl` (c'est la boucle d'attente active propre aux GUI),
- `current_color(*args)` : définit la couleur courante, accepte soit un triplet RVB d'entiers compris entre 0 et 255, soit un nom (en anglais) de couleur,
- des fonctions de dessin : `point(px, py)`, `box(px1, py1, px2, py2)`, `circle(px, py, pr, pep=1)`, `line(px1, py1, px2,py2, pep=1)`...
- `refresh()` : permet de forcer l'affichage du dessin (si des instructions de dessin sont dans une boucle, l'affichage ne se fait par défaut qu'après la boucle).

Exercice 1. Premiers pas

1. Recopier le code suivant et exécuter le:

```
from pocketgl import *

init_window('premiers pas', 400, 400)
point(400,400)
point(402,400)
current_color("blue")
box(0,0,50,100)
current_color("red")
circle(2500,250,40,3)
current_color(0,255,0)
line(50,50,200,100,2)
main_loop()
```

2. Écrire un programme qui affiche une fenêtre de 500x500 pixels et la remplit de barres horizontales de 10 pixels de large en dégradé de gris.

Exercice 2. Test d'équirépartition

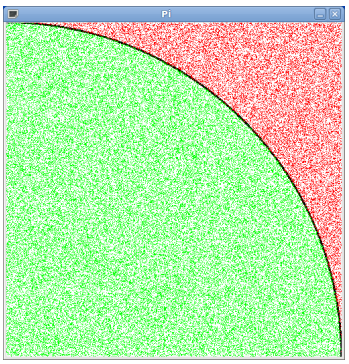
On souhaite vérifier le fonctionnement de l'un des générateurs de nombres aléatoires proposé par Python dans son module `random`. En effet, l'ordinateur étant par essence un automate déterministe, il n'est pas facile de créer un programme qui soit non prévisible i.e qui simule l'effet du hasard.

1. Ecrire un programme destiné à vérifier le fonctionnement du générateur de nombres aléatoires `randrange` de Python en simulant 10000 lancers d'un dé à 6 faces, en mémorisant les résultats dans une liste de 6 éléments, puis en affichant le nombre de sorties de chaque face.

2. Appliquer la même procédure en simulant 10000 lancers de deux dés à 6 faces puis présenter le résultat sous la forme d'un histogramme grâce au module `pocketgl`. Quelle forme générale doit avoir cet histogramme si le tirage des nombres est bien équiréparti ?

Exercice 3. Évaluation du nombre π par la méthode Monte Carlo

Soit un quart de cercle de rayon $R = 1$ inscrit dans un carré de coté 1.



L'aire du carré vaut $(R)^2$ soit 1. L'aire du quart de cercle vaut $\pi R^2/4$ soit $\pi/4$.
En choisissant N points aléatoires (à l'aide d'une distribution uniforme) à l'intérieur du carré, la probabilité que ces points se trouvent aussi dans le cercle est

$$p = \frac{\text{aire du cercle}}{\text{aire du carré}} = \frac{\pi}{4}$$

Soit n, le nombre points effectivement dans le cercle, il vient alors

$$p = \frac{n}{N} = \frac{\pi}{4},$$

d'où

$$\pi = 4 \times \frac{n}{N}.$$

1. Déterminer une approximation de π par cette méthode. Pour cela, pour N itérations, choisir aléatoirement les coordonnées d'un point entre 0 et 1 (fonction `random()` du module `random`), calculer la distance entre le centre du cercle et ce point et déterminer si cette distance est inférieure au rayon du cercle. Le cas échéant, le compteur n sera incrémenté.

   Que vaut l'approximation de π pour 1000 itérations ? 10000 ? 100000 ? Remarque : il faut compter environ 1 mn d'attente pour 1 million de points.

2. En utilisant le module `pocketgl`, illustrer cette méthode en colorant en vert les points dans le cercle et en rouge les points en dehors du cercle. Afficher des résultats intermédiaires de l'approximation de π durant l'animation.

Exercice 4. ($\star$)Tracer de fonction

1. Définir une fenêtre de 700x500 pixels à l'intérieur de laquelle se trouve un repère orthonormé. Le point d'intersection (de coordonnées (350,250) dans l'image) représentera le point de coordonné (0,0) du repère.

2. On décide qu'une unité correspond dans l'image à 100 pixels. Ainsi le point de coordonnées (1,2) dans le repère à pour coordonnées dans l'image (450,450). Tracer en gris les droites d'équation $x = 1$ et $y = 1$.

3. Tracer en gris tout un quadrillage dont les intersections sont les points à coordonnées entières dans le repère.

4. Tracer en rouge la courbe d'équation $y = cos(x)$ sur l'intervalle $[-3, 3]$.

5. Pour chaque valeur entière x de l'intervalle $[-3, 3[$, tracer en bleu un rectangle entre les points $(x, 0)$ et $(x + 1, \cos(x + 1))$.

6. Afficher dans l'interprète la somme des surfaces des rectangles en comptant les rectangles sous l'axe des abscisses négativement.

7. Écrire une fonction `integrale_cos( pas )` qui trace un rectangle entre les points $(x, 0)$ et $(x + 1, \cos(x + 1))$ pour chaque x de l'intervalle $[-3, 3[$ séparés par un pas de `<pas>` et affiche la somme des surfaces de ces rectangles.

8. ($\star\star$) Modifier le script pour qu'il affiche la plus grande valeur de `<pas>` sous forme de puissance de $1/2$ à partir de laquelle on obtient une approximation à 10^{-4} de la valeur théorique de la surface sous la courbe. On utilisera les fonctions `clear_screen()`, `refresh()` et la fonction `sleep` du module `time` afin de voir l'évolution de l'approximation graphiquement.