Le graphisme en informatique

1 Principes généraux

Fonctionnement

La plupart des langages de programmation fournissent des **conteneurs** graphiques (des fenêtres) et des **primitives graphiques** (des fonctions) permettant d'y tracer des objets géométriques (points, lignes, rectangles, polygones, cercles, ellipses) ou d'y afficher du texte et des images.

Les couleurs

Les écrans reconstituent les couleurs par synthèse additive de trois couleurs primaires : rouge, vert et bleu. Ainsi, les couleurs sont généralement codées en informatique par 3 octets (soit 24 bits) : c'est le système

Au moins en théorie, un écran peut donc reproduire une gamme de couleurs différentes.

En HTML et dans d'autres situations, les couleurs sont codées par l'écriture hexadécimale des 3 octets.

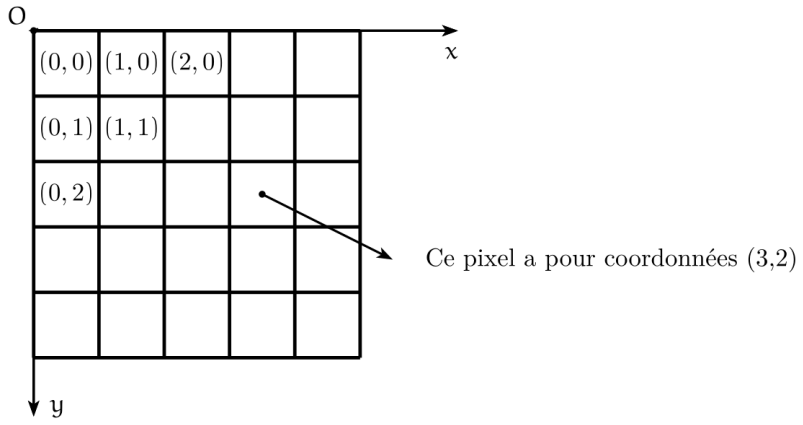
Dans le module `pygame`, les couleurs sont représentées par un triplet de trois entiers compris entre 0 et 255.

Couleur	Codage en Python	Codage HTML
Noir		
Blanc		
	$(0, 255, 0)$	
Bleu		
	$(255, 0, 255)$	
	$(128, 128, 128)$	

Note : il existe d'autres systèmes de représentation des couleurs (TSV, TSL, CMJN, ...).

Les coordonnées graphiques

Un écran est constitué d'une trame de pixels (de l'anglais *picture element*) : chacun de ces pixels peut s'éclairer avec une couleur quelconque. En informatique, les positions de chaque pixel sont déterminées par ses coordonnées dans un repère dont l'origine est située **en haut** à gauche de l'écran.



2 Le module `pygame`

Le module `pygame` est une bibliothèque de Python qui permet essentiellement :

- d'afficher une fenêtre ;
- d'y tracer ou d'y afficher ce que l'on veut ;
- d'écouter les événements utilisateurs (clavier, souris, fermeture de la fenêtre, *etc.*) ;
- de jouer du son.

La documentation complète du module `pygame` est en ligne : `https://www.pygame.org/docs/` (en anglais !).

3 Code minimal pour afficher une fenêtre

Considérons le script ci-dessous : les fonctions précédées du préfixe « `pygame.` » sont définies dans le module `pygame` (tout comme la fonction `sqrt` est définie dans le module `math`), tandis que la fonction `set_mode` est définie dans le sous-module `display` de `pygame`.

```
import pygame

pygame.init() # initialisation du module

size = (600, 400) # largeur, hauteur de la fenêtre
pygame.display.set_mode(size) # ouverture de la fenêtre

pygame.quit() # fermeture de la fenêtre
```

Ce script affiche une fenêtre, et celle-ci disparaît presque aussitôt !

Rappelons que Python est un langage impératif, qui exécute **séquentiellement** chaque instruction : il n'y a donc rien de surprenant à ce que la ligne 8 soit exécutée à la suite de la ligne 6.

Le script doit donc être modifié afin de maintenir la fenêtre ouverte, en utilisant une variable booléenne qui sert de verrou (verrou se dit *lock* en anglais, mais on aurait pu choisir tout autre nom de variable valide).

```
import pygame

pygame.init()

size = (600, 400)
pygame.display.set_mode(size)

lock = True # verrou
while lock: # boucle infinie permettant de maintenir la fenêtre ouverte
    lock = True

pygame.quit()
```

Problème évident : ce script ne se termine pas, même en cliquant sur la croix de fermeture de la fenêtre !

La solution consiste à écouter les événements utilisateurs et à passer la variable `lock` à `False` lorsque l'on clique sur cette croix (ou que l'on utilise le raccourci clavier Alt-F4, qui génère le même événement). De cette manière, le programme sort de la boucle `while`, exécute l'instruction `pygame.quit()` et se termine normalement.

On aboutit finalement au script ci-dessous, où les détails des lignes 10 à 12 seront expliqués ultérieurement.

```
import pygame

pygame.init()

size = (600, 400)
pygame.display.set_mode(size)

lock = True
while lock:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: # événement de fermeture de la fenêtre
            lock = False # on passe lock à False dans ce cas

pygame.quit()
```

4 Primitives graphiques

Nous disposons maintenant d'une fenêtre vide !

Pour y afficher des objets, les primitives graphiques du module `pygame` ont toutes besoin d'une référence vers un objet (de type `Surface`) qui représente la fenêtre. Comme cette référence est renvoyée par la fonction `set_mode`, on la stocke dans une variable (appelée `window` ci-dessous, mais on aurait pu choisir tout autre nom de variable valide).

```
import pygame

pygame.init()

size = (600, 400)
window = pygame.display.set_mode(size)
# window prend la valeur de l'objet renvoyé par la fonction set_mode.

lock = True
while lock:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            lock = False

pygame.quit()
```

Avant de présenter deux exemples de primitives graphiques, il faut enfin préciser que celles-ci ne dessinent pas directement sur la fenêtre mais dans une représentation en mémoire de celle-ci. Le module `pygame` fournit ainsi une fonction (dont l'appel s'écrit `pygame.display.update()`) qui permet justement de mettre à jour (ou « rafraîchir ») l'affichage de la fenêtre.

Remplir la fenêtre

La fonction `window.fill` permet de peindre la fenêtre de la couleur passée en paramètre.

Dessiner un rectangle

La fonction `pygame.draw.rect` permet de peindre un rectangle défini par un objet du type `pygame.Rect`.

Pour d'autres primitives graphiques, consulter la documentation de `pygame` et/ou demander à votre professeur.

En guise d'explications, on donne ci-dessous un exemple d'utilisation de ces primitives.

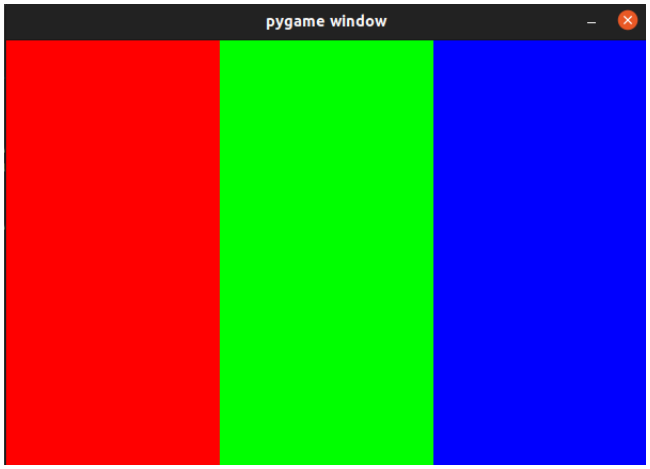
```
import pygame

pygame.init()

size = (600, 400)
window = pygame.display.set_mode(size)
# window prend la valeur de l'objet qui représente la fenêtre

gray = (128, 128, 128)
window.fill(gray) # peint la fenêtre en gris

blue = (0, 0, 255)
r = pygame.Rect(200, 100, 200, 100) # syntaxe : (left, top, width, height)
pygame.draw.rect(window, blue, r) # peint un rectangle bleu

yellow = (255, 255, 0)
r = pygame.Rect(300, 50, 50, 100)
pygame.draw.rect(window, yellow, r) # peint un rectangle jaune

pygame.display.update() # mise à jour de la fenêtre

lock = True
while lock:
    for event in pygame.event.get():
        if event.type == pygame.QUIT :
            lock = False

pygame.quit()
```

Précisions concernant la fonction `pygame.Rect(left, top, width, height)` :

- les paramètres `left` et `top` sont les coordonnées du coin supérieur gauche du rectangle ;
- les paramètres `width` et `height` sont la largeur et la hauteur du rectangle ;
- cette fonction renvoie un objet de type `Rect` passé en argument de la fonction `pygame.draw.rect`.

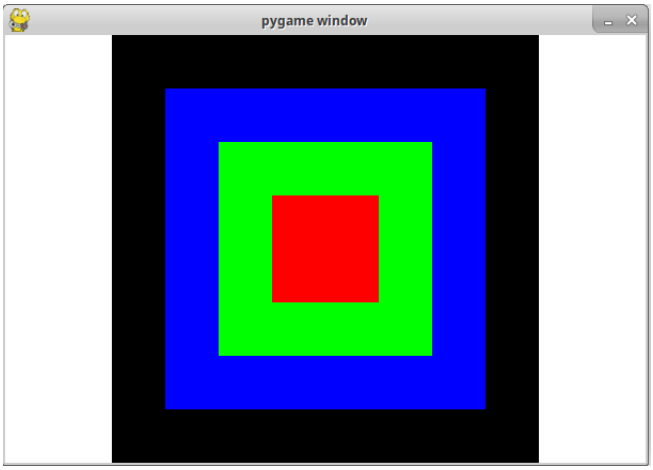
Exercice 1

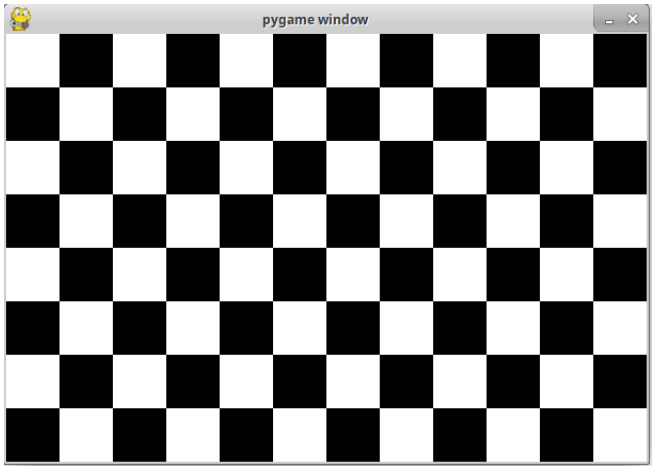
1. Récupérer le fichier `graphics.py` sur notre site : https://holzermarie.github.io/data/premiere/src/graphics.py.
2. Adapter le code existant pour afficher la fenêtre ci-dessous.

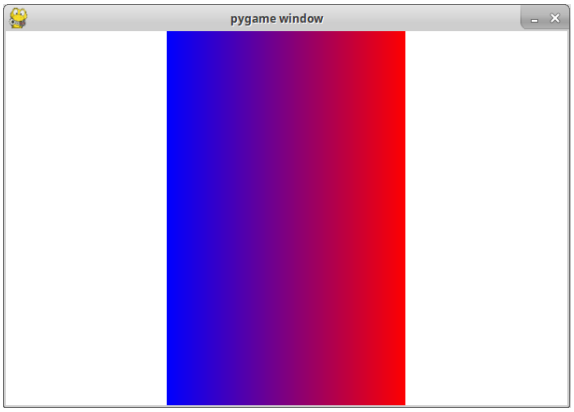


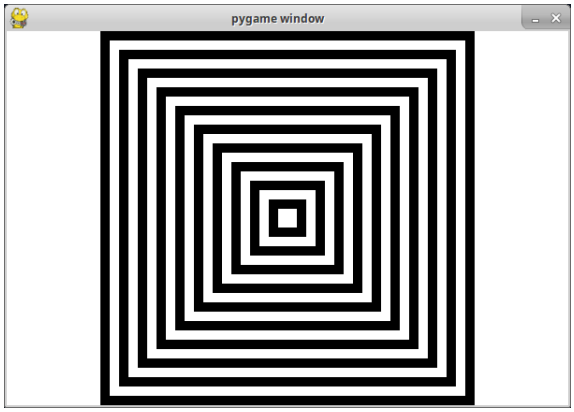
Exercice 2

Concevoir des scripts (un script = un fichier) qui réalisent les affichages ci-dessous.









Indications :

- pour le dégradé : on utilise des rectangles de largeur 1 ;
- pour la dernière figure : elle est constituée de 10 carrés noirs et de 10 carrés blancs dessinés alternativement, du plus grand au plus petit.