

Programmation en langage Python.

Structures de contrôle conditionnelles.

Inspiré de travaux réalisés en C à l'UVSQ – site de Versailles

MISE EN SITUATION.

Le but de cet ensemble d'exercices est de maîtriser la programmation des structures de contrôle conditionnelles.

C'est-à-dire que les instructions comprises dans ces structures seront exécutées sous condition :

Les structures de contrôle conditionnelles.

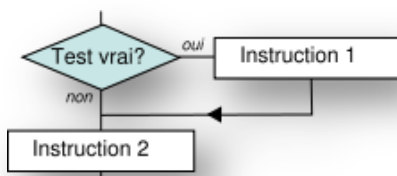
Voici les structures qui seront étudiées lors de cette séquence :

- ✓ Le `if`,
- ✓ le `if ... else`,
- ✓ le `if ... elif`,
- ✓ le `if ... elif ... else`.

Le test : `if`

Cette structure de contrôle permet d'exécuter une instruction ou une suite d'instructions seulement si une condition est vraie.

Syntaxe :



```
if a < b and a != 0:
    # bloc : Instruction 1
    m = a
    print("a =", m, "il est donc <inférieur à b et différent de zéro")
# bloc : Instruction 2
print("voici l'instruction suivante")
```

On évalue la condition (Test vrai ?) :

- ✓ si elle est vraie, on exécute le bloc `Instruction 1` et on passe au bloc suivant `Instruction 2`,
- ✓ si elle est fausse, on passe directement au bloc `Instruction 2`.



L'indentation marque le début de la structure de contrôle.

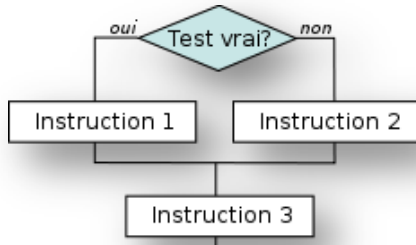
L'arrêt de **l'indentation**, signifie que la structure de contrôle est terminée.

Selon la **PEP8**, l'Indentation doit être faite avec la touche `tabulation` ou avec 4 espaces.

Le test: `if ... else`

Cette structure de contrôle permet d'exécuter soit une série d'instruction, soit une autre, en fonction du résultat d'une condition.

Syntaxe :



```

if a < b and a != 0:
    # bloc : Instruction 1
    m = a
    print("a =", m, "il est donc <inférieur à b et différent de zéro")
else:
    # bloc : Instruction 2
    print("a est supérieur à b ou il est nul")

# bloc : Instruction 3
print("voici l'instruction suivante")
    
```

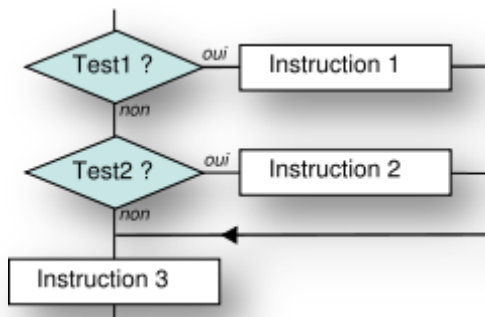
On évalue la condition (Test vrai ?),

- ✓ si elle est vraie, on exécute le bloc Instruction 1 et on passe directement au bloc Instruction 3,
- ✓ si elle est fausse, on exécute le bloc Instruction 2 et on passe au bloc suivant Instruction 3.

Le test: `if ... elif`

Cette structure de contrôle permet de choisir une série d'instruction en fonction du résultat de plusieurs conditions.

Syntaxe :



```

if a < b:
    # bloc : Instruction 1
    print("a est inférieur à b")
elif a > b:
    # bloc : Instruction 2
    print("a est supérieur à b")

# bloc : Instruction 3
print("instruction suivante")
    
```

Le test: `if ... elif ... else`

```

if a < b:
    # bloc : Instruction 1
    print("a est inférieur à b")

elif a > b:
    # bloc : Instruction 2
    print("a est supérieur à b")

elif a == b:
    # bloc : Instruction 3
    print("ils sont égaux")

else:
    # bloc : Instruction 4
    print("c'est bizarre ça ")

# bloc : Instruction 5
print("instruction suivante")
    
```

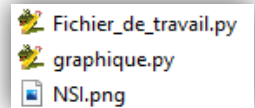
Attention : dans les structures de contrôle conditionnelles, lorsqu'une condition a été validée, les autres ne sont même pas évaluées.

TRAVAIL DEMANDE

Pour les 7 premiers exercices, nous utiliseront la bibliothèque SDL (avec Pygame). Afin de gagner en efficacité, il est vivement conseillé de reprendre votre fichier du TP précédent et de le renommer.

Rappel, le fichier python sur lequel vous travaillez fait appel à la bibliothèque `graphique.py` ainsi qu'à l'image `NSI.png`.

En conséquence ces trois fichiers doivent être enregistrés dans le même dossier.



Rappel sur les TUPLES :

Il est possible de définir les tuples (ici coordonnées des points) dans des constantes. Exemple :

```
p1 = (40, 10)
p2 = (340, 310)

pygame.draw.line(fenetre, black, p1, p2, 3)
```

Pour récupérer les données d'un tuple (coordonnées du point cliqué) indépendamment et pouvoir les modifier, il y a deux possibilités :

```
p1 = wait_clic()
p1_x = p1[0]
p1_y = p1[1]
```

```
p1_x, p1_y = wait_clic()
```

Dans les deux cas précédents, on enregistre le premier élément du tuple dans la variable `p1_x` et le deuxième dans la variable `p1_y`.

1. HISTOIRE DE CLIC.

Exercice_1.

Afficher une ligne verticale qui sépare l'écran en deux parties égales (`pygame.draw.line()`).

Attendre un clic de l'utilisateur et enregistrer les coordonnées (tuple) dans une variable (`p1` par exemple).

- ✓ Si le clic est à gauche de la ligne, afficher un cercle plein **bleu** (`pygame.draw.circle()`),
Aide : Pour cela, il faut vérifier si la coordonnée d'indice 0 est inférieure à la moitié de la fenêtre (`p1[0] < WIDTH/2`),
- ✓ s'il est à droite un cercle plein **rouge**.

Dans les deux cas le cercle sera centrée là où l'utilisateur a cliqué (en `p1`).

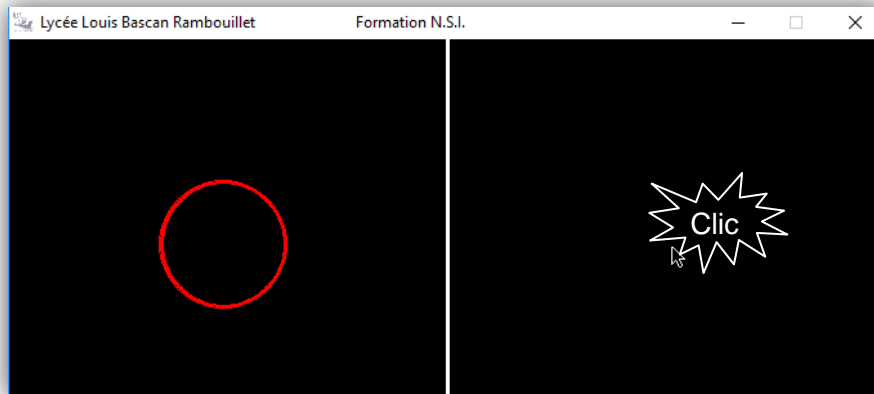
Exercice_2.

Modifier votre programme pour remplir le cahier des charges suivant :

Attendre un clic de l'utilisateur.

- ✓ Si le clic est à droite de la ligne, afficher un cercle vide **rouge** à la même ordonnée mais à gauche de l'écran,
- ✓ si le clic est à gauche, afficher un cercle vide **bleu** à la même ordonnée mais à droite de l'écran.

L'abscisse, pour tracer les cercles, devra être à l'opposé du clic.



Exercice_3.

Afficher une ligne verticale qui sépare l'écran en deux parties égales.

Attendre 3 clics.

- ✓ Si les trois clics sont du même côté, afficher un triangle reliant les 3 points,
- ✓ sinon ne rien afficher.

Vous pouvez utiliser la fonction [pygame.draw.polygon\(\)](#) pour dessiner le triangle.

```
pygame.draw.polygon()
    draw a polygon
    polygon(surface, color, points) -> Rect
    polygon(surface, color, points, width=0) -> Rect
    Draws a polygon on the given surface.
    Parameters:
    ◦ surface (Surface) -- surface to draw on
    ◦ color (Color or int or tuple(int, int, int, [int])) -- color to
      draw with, the alpha value is optional if using a tuple (RGB[A])
    ◦ points (tuple(coordinate) or list(coordinate)) -- a sequence
      of 3 or more (x, y) coordinates that make up the vertices of
      the polygon, each coordinate in the sequence must be a
      tuple/list/pygame.math.Vector2 of 2 ints/floats, e.g. [(x1, y1),
      (x2, y2), (x3, y3)]
    ◦ width (int) --
      (optional) used for line thickness or to indicate that the
      polygon is to be filled
      if width == 0, (default) fill the polygon
      if width > 0, used for line thickness
      if width < 0, nothing will be drawn
```

Exercice_4.

Afficher une ligne horizontale qui sépare l'écran en deux parties égales. Attendre deux clics.

- ✓ Si les clics sont chacun d'un côté de la ligne verticale, afficher une ligne **rouge** reliant les deux points où l'utilisateur a cliqué,
- ✓ si les deux clics sont du même côté de la ligne verticale, afficher une ligne **bleu** entre les deux points.

Exercice_5.

Afficher une ligne verticale et une horizontale qui sépare l'écran en quatre parties égales.

- ✓ Si le clic est en haut à gauche ou en bas à droite afficher un cercle **bleu**.
- ✓ Sinon afficher un cercle **rouge**.

Ecrire ce programme avec un seul test.

Exercice_6.

Afficher deux traits horizontaux qui séparent l'écran en trois zones égales. Attendre un clic.

- ✓ Si le clic est dans la zone du haut ou du bas, afficher un cercle **rose** ;
- ✓ Si le clic est dans la zone du milieu, afficher un cercle **marron**.

Exercice_7.

Attendre un clic de l'utilisateur.

Afficher un carré de 20 pixels de côté, centré sur le lieu du clic, dont la couleur dépend de la parité des coordonnées cliquées.

Utiliser le modulo (%) pour tester la parité.

	Abscisse paire	Abscisse impaire
Ordonnée paire	Rouge	Bleu
Ordonnée impaire	Jaune	Vert

Les coordonnées du point cliqué devront s'afficher dans la console pour vérification.

Faites valider votre programmation par le professeur.