Introduction au Principe de la Modularité en Python

Introduction

La modularité est un principe fondamental en programmation. Elle permet de diviser un programme complexe en plusieurs parties plus simples appelées modules ou fonctions. Cela rend le code plus organisé, plus lisible et plus facile à maintenir.

Dans cette leçon, nous allons :

- Créer une maison avec turtle sans utiliser de fonctions.
- Identifier les répétitions et améliorer le code avec des fonctions.
- Construire une ville entière en utilisant une fonction générique dessiner_polygone.

1. Construire une maison sans modularité

Voici un code simple pour dessiner une maison en utilisant le module turtle :

```
from turtle import *
speed(10)
# Base de la maison
penup()
goto(-100, -100)
pendown()
fillcolor("blue")
begin_fill()
for _ in range(4):
    forward(200)
    left(90)
end_fill()
# Toit
penup()
goto(-120, 100)
pendown()
fillcolor("red")
begin_fill()
for _ in range(3):
    forward(240)
    left(120)
end_fill()
# Porte
penup()
goto(-30, -100)
pendown()
fillcolor("yellow")
begin_fill()
for _ in range(2):
    forward(60)
    left(90)
    forward(100)
    left(90)
end_fill()
# Fenêtre1 de la maison
penup()
goto(-80, 0)
pendown()
fillcolor("white")
begin_fill()
for _ in range(4):
    forward(50)
    left(90)
end_fill()
```

```
# Fenêtre2 de la maison
penup()
goto(30, 0)
pendown()
fillcolor("white")
begin_fill()
for _ in range(4):
    forward(50)
    left(90)
end_fill()
```

Problème : Ce code est long et répétitif. Si on veut dessiner plusieurs maisons ou modifier une forme, il faudra répéter plusieurs fois les mêmes instructions.

2. **Solution : Utiliser des fonctions pour simplifier le code et le rendre réutilisable**

Nous allons définir une fonction générique dessiner_polygone() pour simplifier le dessin.

```
from turtle import *
def dessiner_polygone(x, y, lo, la, couleur, nature):
    penup()
    goto(x, y)
    pendown()
    fillcolor(couleur)
    begin_fill()
    if nature == "triangle":
        for i in range(3):
            forward(lo)
            left(120)
    elif nature == "rectangle" or nature == "carree":
        for i in range(2):
            forward(lo)
            left(90)
            forward(la)
            left(90)
    end_fill()
dessiner_polygone(-100, -100, 200, 200, "blue", "carree")
dessiner_polygone(-120, 100, 240, 240, "red", "triangle")
dessiner_polygone(-80, 0, 50, 50, "white", "carree")
dessiner_polygone(30, 0, 50, 50, "white", "carree")
dessiner_polygone(-30, -100, 60, 100, "yellow", "carree")
```

3. Application : Créer une ville en exploitant la fonction polygone

Utiliser cette fonction pour construire une ville avec plusieurs maisons et immeubles.

Conclusion : En utilisant des fonctions :

- Le code est plus court et plus lisible.
- Le code est réutilisable. Il est facile d'ajouter de nouveaux bâtiments ou de modifier les formes.
- La répétition est évitée, ce qui limite les erreurs.

La structure suivante est appelée la structure conditionnelle. Elle permet d'exécuter un traitement si une condition est vérifiée.

```
if condition1 :
        Traitement1
elif condition2 :
        Traitement2
elif condition n-1 :
        Traitement n-1
else :
        Traitement n
```