

Algorithmique et Programmation avancée

3.5: Programmation orientée objet (POO)

Definition. La programmation orientée objet (POO) est un paradigme de programmation informatique qui consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs.

Objets en Python

Example 1. Exemple type en Python:

```
class New_Class(object):
    att_0="Salut" #Attributs propres a tout objet de cette classe
    def __init__(self,attribut_1,attribut_2):
        # Initialisation des attributs de l'objet
        self.att_1 = attribut_1
        self.att_2 = attribut_2

    def new_Method(self,opt_arg=None): # Une méthode de la classe
        conc=self.att_1+self.att_2
        l=len(conc)
        if opt_arg==None or opt_arg>l:
            opt_arg=l
        print(conc[:opt_arg])

if __name__=="__main__":
    objtest=New_Class("Bonjour ","Hello ")
    print(objtest.att_0)
    print(objtest.att_1)
    print(objtest.att_2)
    objtest.new_Method(10)
```

1: Prise en main

1. Créer une classe *Perso* avec les attributs *nom* et *prenom* et la méthode *presentation()*, qui affiche une chaîne de caractères contenant les nom et prenom de l'objet.
2. Ajouter un attribut *amis*, initialisé par défaut à une liste vide, ainsi qu'une méthode *ajoutAmi(self,other)* qui ajoute un objet de classe *Perso* à la liste d'amis de l'objet de façon réciproque. Tester en instanciant deux fois la classe *Perso*.

2: Une classe Point

1. Construire une classe *Point* qui prend deux attributs *x* et *y*. Afin d'améliorer l'affichage, on peut surcharger les méthodes spéciales issues de la classe mère *object* à savoir *__repr__* et *__str__* pour respectivement la lecture dans l'interpréteur et le print.

Exemple:

```
def __repr__(self):
    """surcharge du repr() """
    return "Point: abscisse:{}, ordonné:{}".format(self.x, self.y)
    #Coder la methode __str__ pour que le print affiche le point sous la forme (x,y).
```

2. Importer le module *matplotlib.pyplot* (`import matplotlib.pyplot as plt`) Compléter les méthodes *affichePoint(self,type = 'bo')* qui affiche un point sur le plot, *dist(self,other)* qui calcule la distance d'un point à l'autre, *afficheSeg(self,other)* qui trace le segment entre deux points et *nearest(self,liPoint)* qui trouve le plus proche voisin d'un point parmi une liste.

3. Ecrire un algorithme *closestPair* qui trouve la paire de points la plus proche parmi une liste et colore cette paire en rouge. (Plutôt que calculer toutes les paires, trier les points par rapport à leur abscisse et trouver une méthode plus efficace).

4. Construire une methode *angle(self, other1, other2)* qui donne l'angle entre trois points. Construire un algorithme *convexHull*, qui renvoie la liste des points de l'enveloppe convexe d'une liste de points et qui l'affiche.

3: Une classe Tree

Un arbre est un ensemble organisé de noeuds dans lequel chaque noeud a un père et un seul, sauf un noeud que l'on appelle la racine.

1. Construire une classe Node ayant pour attribut une valeur et une liste de fils (initialisée à []). Proposer une méthode qui ajoute un fils.
2. Proposer une méthode simple d'afficher un arbre et la coder.