POO & Python

Programmation Orientée Objet
&
Langage Python

UVSQ université PARIS-SACLAY
ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

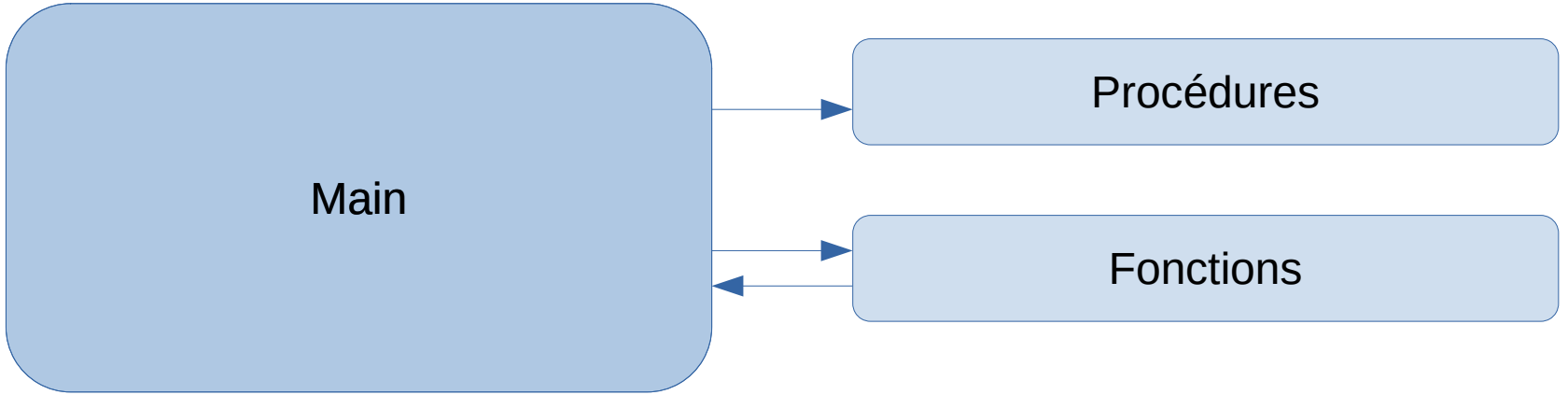
Olivier Snoeck

Olivier.snoeck@uvsq.fr

Les programmations

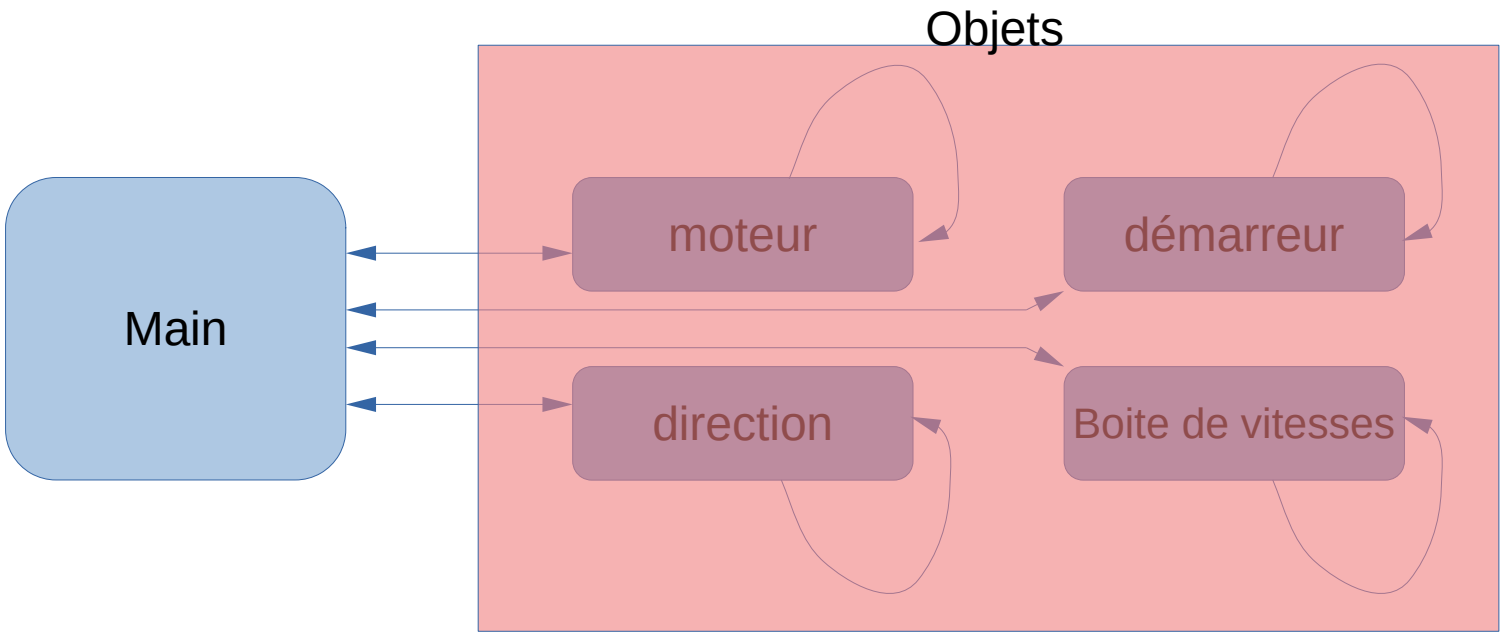
- La programmation procédurale
- La programmation orientée objet

La programmation procédurale est un paradigme de programmation qui gère des actions, de la logique, des événements.



Les programmations

- La programmation procédurale
- La programmation orientée objet

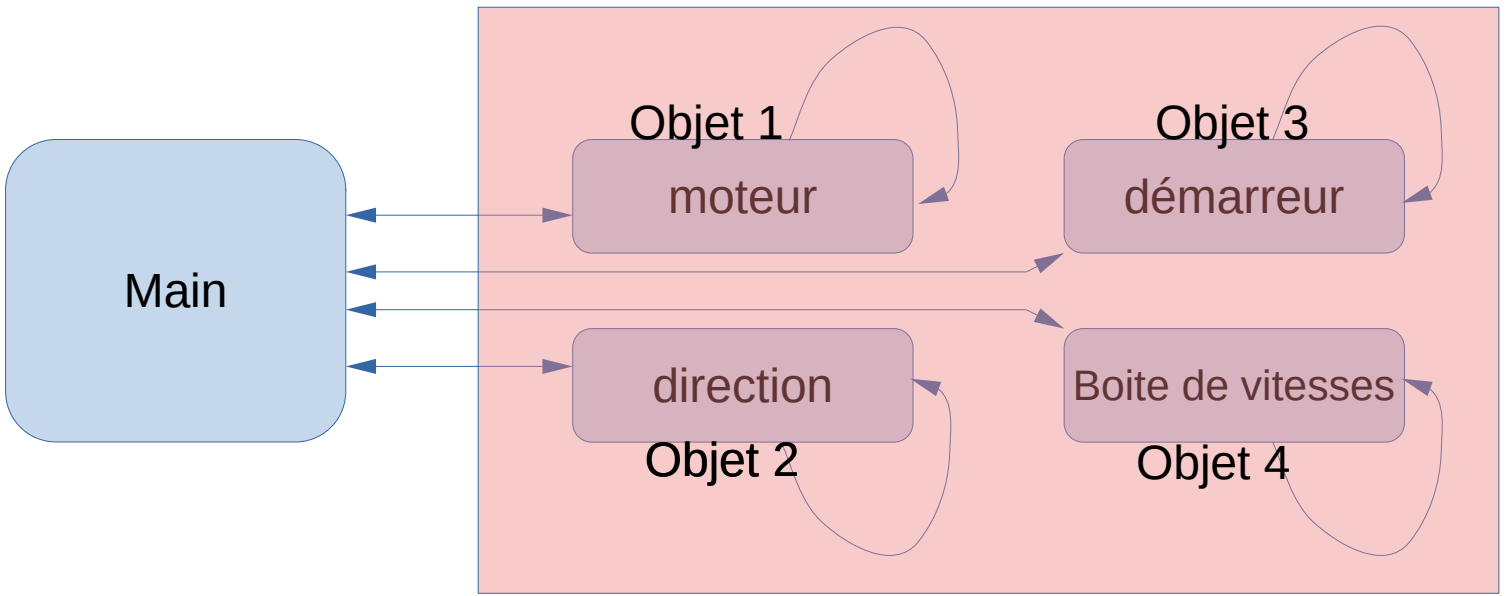


Les programmations

- La programmation procédurale

- La programmation orientée objet

La POO est un paradigme de programmation qui gère des briques logicielles



POO

La POO permet de :

- Développer un projet au sein d’une équipe,
- Pouvoir utiliser des outils collaboratifs,
- Faire évoluer le projet informatique,
- Sécuriser le projet informatique,
- Clarifier le rôle de chaque objet,
- Gérer des objets communs à plusieurs projets informatique,
- ….

POO

Les possibilités de Programmation Orientée Objet de Python reposent sur le concept de classe.

Une classe est la généralisation de la notion de type défini par l'utilisateur, dans lequel se trouvent associées à la fois des données (on parle de « membres donnée ») et des fonctions (on parle de « fonctions membre » ou de méthodes).

En P.O.O. pure, les données sont « encapsulées », ce qui signifie que leur accès ne peut se faire que par le biais des méthodes.

POO

Les classes

POO : Les classes

Un des aspects majeurs de la programmation par objets est la création de types abstraits, faciles à utiliser et faciles à modifier.
Pour améliorer la sécurité, on préférera séparer en 2 parties :

- La partie visible par l'utilisateur de l'objet
- La partie invisible par l'utilisateur de l'objet.

POO : Les classes

Un des aspects majeurs de la programmation par objets est la création de types abstraits, faciles à utiliser et faciles à modifier.
Pour améliorer la sécurité, on préférera séparer en 2 parties :

- La partie visible par l'utilisateur
- La partie invisible par l'utilisateur

Une classe est un type abstrait défini par l'utilisateur, qui contient des données et qui effectue des actions
Il y aura donc plusieurs parties :

- Les variables ou les données, appelées attributs,
- Les fonctions, appelées méthodes.

POO : Les classes

Une classe est un type abstrait défini par l'utilisateur, qui contient des données et qui effectue des actions
Il y aura donc plusieurs parties :

- Les variables ou les données, appelées attributs,
- Les fonctions, appelées méthodes.

POO : Les classes

Une classe est un type abstrait défini par l'utilisateur, qui contient des données et qui effectue des actions
Il y aura donc plusieurs parties :

- Les variables ou les données, appelées attributs,
- Les fonctions, appelées méthodes.

Exemple 1 :

```
class Heure:
    def __init__(self,heures,minutes, secondes):
        print("L'heure vient d'être créé")
        self.heures= heures
        self.minutes= minutes
        self.secondes= secondes
        print (heures, ":" ,minutes, ":" ,secondes )
    def affiche(self):
        # la méthode ne permet que d'afficher
        print (self.heures, ":" ,self.minutes, ":" ,self.secondes )
    def augmente(self):
        # la méthode permet d'augmenter les secondes...
        self.secondes=self.secondes+1
        if self.secondes >=60:
            self.minutes=+1
            self.secondes=0
            if self.minutes >=60:
                self.heures=+1
                self.minutes=0
    def raz(self):
        self.heures = self.minutes = self.secondes = 0
```

POO : Les classes

horloge = Heure(10,0,0)
#Ici, on instancie la classe en 1 objet. On y passe les paramètres du constructeur

POO : Les classes

```
horloge = Heure(10,0,0)
horloge.augmente()
print(horloge.secondes)
horloge.affiche()
```

```
L'heure vient d'être créé
10 : 0 : 0
1
10 : 0 : 1
```

POO : Les classes

```
horloge = Heure(10,0,0)
horloge.augmente()
print(horloge.secondes)
horloge.affiche()
horloge.raz()
```

```
L'heure vient d'être créé
10 : 0 : 0
1
10 : 0 : 1
```

POO : Les classes

```
horloge = Heure(10,0,0)
horloge.augmente()
print(horloge.secondes)
horloge.affiche()
horloge.raz()
horloge.affiche()
horloge.augmente()
horloge.affiche()
```

```
L'heure vient d'être créé
10 : 0 : 0
1
10 : 0 : 1
0 : 0 : 0
0 : 0 : 1
```

→ Modifiez l’affichage pour afficher la dizaine (exemple : 00 : 00 : 01)

Autre exemple

```
class Voiture:
    #les attributs (idem que les variables)
    marque="Lamborghini"
    couleur="rouge"

#affichage des attributs de la classe Voiture
print(Voiture.couleur)
print(Voiture.marque)
```

POO – Voiture 1.py

Autre exemple

```
class Voiture:
    #les attributs (idem que les variables)
    marque="Lamborghini"
    couleur="rouge"

#affichage des attributs de la classe Voiture
print(Voiture.couleur)
print(Voiture.marque)

#l'instanciation pour avoir un premier objet
voiture1 = Voiture()
print(voiture1.couleur)
print(voiture1.marque)

#l'instanciation pour avoir un second objet
voiture2 = Voiture()
print(voiture2.couleur)
print(voiture2.marque)
```

POO – Voiture 2.py

```python
class Voiture:
    #les attributs (idem que les variables)
    marque="Lamborghini"
    couleur="rouge"

#affichage des attributs de la classe Voiture
print(Voiture.couleur)
print(Voiture.marque)

#l'instanciation pour avoir un premier objet
voiture1 = Voiture()
voiture2 = Voiture()
print(voiture1.couleur)
print(voiture2.couleur)

#On change l'attribut de la classe
Voiture.couleur="jaune"
print(voiture1.couleur)
print(voiture2.couleur)
```

e

POO – Voiture 3.py

emple

```
class Voiture:
    #les attributs (idem que les variables)
    marque="Lamborghini"
    couleur="rouge"

#affichage des attributs de la classe Voiture
print(Voiture.couleur)
print(Voiture.marque)

#l'instanciation pour avoir un premier objet
voiture1 = Voiture()
voiture2 = Voiture()
print(voiture1.couleur)
print(voiture2.couleur)

#On change l'attribut de l'objet
voiture1.couleur="jaune"
voiture1.marque="Renault"
voiture2.couleur="vert"
voiture2.marque="Trabant"
print(voiture1.marque)
print(voiture1.couleur)
print(voiture2.marque)
print(voiture2.couleur)
```

POO – Voiture 4.py

```python
class Voiture:
    #les attributs (idem que les variables)
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

#l'instanciation pour avoir un premier objet et sa construction
voiture1 = Voiture("Porsche","jaune")
voiture2 = Voiture("Lamborghini","rouge")
print(voiture1.couleur)
print(voiture2.couleur)

#On change l'attribut de l'objet
voiture1.couleur="jaune"
voiture1.marque="Renault"
voiture2.couleur="vert"
voiture2.marque="Trabant"
print(voiture1.marque)
print(voiture1.couleur)
print(voiture2.marque)
print(voiture2.couleur)
```

POO – Voiture 5.py

```python
class Voiture:
    #les attributs (idem que les variables)
    voitures_crees = 0
    def __init__(self, marque, couleur):
        Voiture.voitures_crees+=1 #commun à toute la classe
        self.marque = marque
        self.couleur = couleur

#l'instanciation pour avoir un premier objet et sa construction
voiture1 = Voiture("Porsche","jaune")
voiture2 = Voiture("Lamborghini","rouge")
print(voiture1.couleur)
print(voiture2.couleur)
print(Voiture.voitures_crees)

#On change l'attribut de l'objet-> ça peut être dangereux
voiture1.couleur="vert"
voiture1.marque="Trabant"
print(voiture1.marque)
print(voiture1.couleur)
```

POO – Voiture 6.py

```python
class Voiture:
    #les attributs (idem que les variables)
    voitures_crees = 0
    def __init__(self, marque, couleur):
        Voiture.voitures_crees+=1 #commun à toute la classe
        self.marque = marque
        self.couleur = couleur

    def afficher_marque(self):
        print(f"La voiture est une {self.marque}")

#l'instanciation pour avoir un premier objet et sa construction
voiture1 = Voiture("Porsche","jaune")
voiture2 = Voiture("Lamborghini","rouge")
voiture1.afficher_marque()
voiture2.afficher_marque()
print(Voiture.voitures_crees)

#On change l'attribut de l'objet-> ça peut être dangereux
voiture1.couleur="vert"
voiture1.marque="Trabant"
print(voiture1.marque)
print(voiture1.couleur)
```

POO – Voiture 7.py

POO avec un exemple complet

POO – Feux de signalisation.py

Exercice 2

Source

1/ On voudrait gérer les étudiants d'une institution à l'aide d'une classe Etudiant définie par :les attributs suivants :

- nom : nom d'un étudiant
- prénom: prénom d'un étudiant
- tabnotes : tableau contenant les notes d'un étudiant, sachant qu'un étudiant a au total 10 notes.

les méthodes suivantes :
- void saisie (), permettant la saisie d'un étudiant
- void affichage (), permettant l'affichage d'un étudiant
- float moyenne (), retourne comme résultat la moyenne des notes d'un étudiant.
- int admis (), retourne comme résultat la valeur 1, si un étudiant est admis et la valeur 0, sinon. Un étudiant est considéré comme étant

admis lorsque la moyenne de ses notes est supérieure ou égale à 10.
- int exae_quo (Etudiant E), retourne comme résultat la valeur 1, si deux étudiants ont la même moyenne et la valeur 0, sinon.

Ecrire la classe Etudiant dans le langage Python

Exercice 2

Source

2/ On voudrait maintenant représenter, à l'aide d'une nouvelle classe Etudiant_en_Maitrise, certains étudiants particuliers dans cette institution qui sont les étudiants en dernière année d'études. Ces étudiants possèdent en effet un attribut supplémentaire : note_memoire, qui représente la note de leur mémoire de fin d'études.

Les méthodes à associer à cette classe sont les suivantes :
- void saisie (), permettant la saisie d'un étudiant en maîtrise
- void affichage (), permettant l'affichage d'un étudiant en maîtrise
- float moyenne (), retourne comme résultat la moyenne des notes d'un étudiant en maîtrise
- int admis (), retourne comme résultat la valeur 1, si un étudiant est admis et la valeur 0, sinon. Un étudiant en maîtrise est considéré comme étant admis lorsque, d'une part, la moyenne de ses notes est supérieure ou égale à 10 et d'autre part la note obtenue pour son mémoire de fin d'études est supérieure ou égale à 10.

- int exae_quo (Etudiant_en_Maitrise E), retourne comme résultat la valeur 1, si deux étudiants ont d'une part la même moyenne et d'autre part, la même note de mémoire et retourne la valeur 0, sinon.

a) Quelles sont les méthodes qui sont à redéfinir dans la classe Etudiant_en_Maitrise ?
b) Ecrire la classe Etudiant_en_Maitrise dans le langage Python

TD

1) Écrire un programme dans lequel on demande à l'utilisateur de saisir un entier en gérant l'exception dans le cas où il ne saisit pas un entier correctement.
2) Écrire un programme dans lequel on demande à l'utilisateur de saisir un entier en gérant l'exception dans le cas où il ne saisit pas un entier correctement en lui demandant de refaire la saisie.
3) Écrire un programme dans lequel on demande à l'utilisateur de saisir son date de naissance en gérant l'exception dans le cas où il ne saisit pas une date valide en lui demandant de refaire la saisie.
4) Écrire un programme qui demande à l'utilisateur une date de départ et une date d'arrivée, générer une exception si la date d'arrivée et inférieure à la date de départ.
5) Créer une classe Élèves caractérisée par nom, âge et moyenne.

- L'âge doit être entre 18 et 26 sinon l'exception InvalidAgeException (elle affiche le message "L'âge doit être entre 18 et 26") est générée.
- La note doit être entre 0 et 20 sinon l'exception InvalidNoteException est générée (elle affiche le message "La note doit être entre 0 et 20").

→ Définir les constructeurs de la classe, les accesseurs et les méthodes ToString.
//Source : www.exelib.net : merci à eux