# INF1 : Algorithmique et Programmation

## Cours 12 :
## Introduction à la programmation orientée objet et Ensembles

Domitile Lourdeaux

Université de technologie de Compiègne

Printemps 2024

# Motivations (1)

Les **classes** sont les principaux outils de la programmation orientée objet (POO)

**Ce type de programmation permet de** :

- **Structurer** les logiciels complexes en les organisant comme des ensembles d'objets qui interagissent entre eux et avec le monde extérieur

# Motivations (2)

**Bénéfices :**

1. **Construire** les différents objets **indépendamment** les uns des autres sans risques d'interférences (différents programmeurs)
   - **Grâce à** :
     - Encapsulation : fonctionnalités et variables enfermées dans l'objet
     - Interface de l'objet : accès à travers de procédures (e.g. mêmes noms)
   - **Permet d'éviter les variables globales**
     - Risques quand programmes volumineux : variables modifiées, redéfinies
2. **Construire de nouveaux objets** à partir d'objets préexistants
   - **Grâce à** :
     - Dérivation : construit une **classe enfant** à partir de la **classe parent**. (héritage de toutes les propriétés et les fonctionnalités)
     - Polymorphisme : attribut des comportements différents à des objets dérivants les uns des autres

Source : G. Swinnen. *Apprendre à programmer avec Python 3*. Ed. Eyrolles (disponible à la BUTC)

# Classes et objets

**Nous avons déjà rencontré les notions de classe et d'objet**

**En Python :**

- Les variables sont des objets
- Le type d'une variable correspond à une classe à laquelle sont associées des méthodes

**Exemple : les listes**

```
>>> liste = [8, 12, 26]
>>> type(liste)
<class 'list'>
>>> liste.append(53)
>>> print(liste)
[8, 12, 26, 53]
>>> print(liste.index(26))
2
```

**Nous allons voir comment définir de nouvelles classes et de nouveaux objets**

# Notion de classe et d'objet

**Une classe regroupe :**

- Les caractéristiques de l'entité qu'elle représente (**attributs**)
- Les **méthodes** effectuant des traitements sur cette entité

**Exemple :** la classe Cercle

- **Attributs** : centre, rayon
- **Méthodes** : périmetre, surface, ...

**Un objet est une instance d'une classe**

**Exemple :**

- l'**objet** cercle0 de centre $O$ et de rayon 5 est une **instance** de la **classe** Cercle

# Définition d'une classe

## Définition

**Une classe « encapsule » en général**

- Des attributs
- Des méthodes
- Un constructeur, qui est une méthode particulière, appelée à chaque fois qu'une instance est créée

**En Python**

- Une classe est définie à l'aide du mot-clé class
- Le nom du constructeur est obligatoirement `__init__`
- L'instance courante est désignée par self
- self doit être le premier paramètre des méthodes et donc aussi du constructeur

# Un exemple en Python

## La classe Point

Un point peut être représenté par ses coordonnées :

```
class Point:
    "Definition d'un point a partir de ses coordonnees"
    def __init__ (self, abscisse, ordonnee):
        "Constructeur"
        self.x = abscisse
        self.y = ordonnee

    def affiche(self):
        "Methode affiche"
        print(f'abscisse: {self.x}, ordonnee: {self.y}')

p = Point(3, 4) #Creation d'une instance de classe Point
p.affiche()  #Appel de la m thode affiche
abscisse: 3, ordonnee: 4
```

# La méthode constructeur

**Méthode exécutée automatiquement lors de la création d'une instance**

Des valeurs « par défaut » des paramètres peuvent être précisées

```
class Point:
    "Definition d'un point a partir de ses coordonnees"

    def __init__(self, abscisse = 0, ordonnee = 0):
        self.x = abscisse
        self.y = ordonnee

    def affiche(self):
        print(f'abscisse : {self.x}, ordonnee : {self.y}')
p = Point()
p.affiche()
abscisse : 0, ordonnee : 0
```

# Définition d'une méthode

**Définition analogue à celle d'une fonction mais :**

- Doit toujours être placée à l'intérieur de la définition d'une classe
- Contient au moins un paramètre placé en premier : **`self`**
- Le paramètre **`self`** est une référence à l'instance courante

```
class Point:
    "Definition d'un point a partir de ses coordonnees"

    def __init__(self, abscisse, ordonnee):
        self.x = abscisse
        self.y = ordonnee

    def affiche(self):
        print(f'abscisse : {self.x}, ordonnee : {self.y}')

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy

p = Point(3,4)
p.translate(2, 0)
p.affiche()
abscisse : 5, ordonnee : 4
```

# Accès aux attributs depuis l'extérieur d'une classe (1)

**Accès direct possible mais déconseillé**

```
print(p.x)
p.y = 10
```

**Il est fortement conseillé de définir si nécessaire des méthodes :**

- d'accès (accesseurs) : **getter**
- de modification (mutateurs) : **setter**

```
    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    def set_y(self, value):
        self.y = value

print(p.get_x())
p.set_y(10)
print(p.get_y())
```

# Accès aux attributs depuis l'extérieur d'une classe (2)

## Utilité

Les accesseurs et les mutateurs permettent de **sécuriser** (e.g. vérifications)

```
class Personne:
    def __init__(self, nom):
        "Constructeur"
        self.nom = nom

    def get_age(self):
        return self.age

    def set_age(self, value):
        if age <= 0 or age >= 150
            print('Age out of range')
        else:
            self.age = value
```

# Similitude et unicité

## Similitude

- Si deux objets sont créés à partir de la même classe avec les mêmes valeurs d'attributs, ils sont similaires, mais **pas identiques**
- Exemple :
  - p1 = Point(3, 4)
  - p2 = Point(3, 4)
- p1 et p2 référencent deux objets distincts

## Unicité

- Mais si p2 est défini à partir de p1 :
  - p1 = Point(3, 4)
  - p2 = p1
- p1 et p2 référencent le même objet

# Objets composés d'objets (1)

**Un objet peut être composé d'autres objets**

- Exemple : la classe Cercle contient un objet Point (son centre) créé indépendamment

```
from Point3 import Point
from math import pi

class Cercle:
    "Definition d'un cercle et de son centre"

    def __init__(self, centre, rayon):
        self.centre = centre
        self.rayon = rayon

    def perimetre(self):
        return pi * self.rayon ** 2
```

```
>>> p = Point(0, 0)
>>> c = Cercle(p, 5)
>>> print(round(c.perimetre(), 2))
78.54
```

# Objets composés d'objets (2)

## Création d'un objet lors de l'initialisation d'un autre objet

```
from Point3 import Point
from math import pi

class Cercle:
    "Definition d'un cercle et de son centre"

    def __init__(self, x_centre, y_centre, rayon):
        self.centre = Point(x_centre, y_centre)
        self.rayon = rayon
        self.perimetre=self.perim()

    def perim(self):
        return pi * self.rayon ** 2
```

```
>>> c = Cercle(0, 0, 5)
>>> print(c.centre.x)
0
>>> c.centre.affiche()
abscisse : 0, ordonnée : 0
>>> print(round(c.perimetre(), 2))
78.54
```

# Héritage (1)

```python
class CompteBancaire(object):
    def __init__(self, nom, solde):
        self.nom = nom
        self.solde = solde

    def affiche(self):
        print(f'Le solde du compte de {self.nom} est : {self.solde} euros')

class CompteEpargne(CompteBancaire):
    def __init__(self, nom, solde, taux):
        CompteBancaire.__init__(self, nom, solde)
        self.taux = taux

    def affiche(self):
        CompteBancaire.affiche(self)
        print(f"Le taux d'int r t est de {self.taux}%")

    def interets_annuels(self):
        return self.solde * self.taux / 100
```

# Héritage (2)

## Remarques

- La classe CompteEpargne est dérivée de CompteBancaire
- La méthode affiche de CompteEpargne surcharge celle de CompteBancaire
- Elle appelle la méthode parente (de même que le constructeur)

# Rappel : types composites en python

**Séquences : chaines, listes, tuples**

- Suites ordonnées d'éléments
- Accessibles par un index (nombre entier)

**Dictionnaires**

- Eléments non ordonnés
- Accessibles à l'aide d'une « clé »

**Ensembles**

- Eléments non ordonnés, uniques

# Définition

## Définition

Un ensemble (set en Python) est une collection d'éléments

- Non ordonnée
- Non indexée
- Sans duplication d'élément

Il est donc impossible de :

- Récupérer un élément par sa position
- Modifier un de ses éléments par l'indexation

# Création d'un ensemble

## Ensemble vide

- s = set()

## Création directe

- voyelles = {'a', 'e', 'i', 'o', 'u', 'y'}
- Duplication impossible :

```
>>> s = {4, 5, 5, 6}
>>> s
{4, 5, 6}
```

## A partir d'une chaine de caractères

- voyelles = set("aeiouy")

## A partir d'une liste

- voyelles = set(['a', 'e', 'i', 'o', 'u', 'y'])
- impairs = set([1, 3, 5, 7, 9])

# Ajout / Suppression

**Ajout d'un élément**

- impairs = set([1, 3, 5, 7]
- impairs.add(9)

**Ajout de plusieurs éléments**

- impairs.update([11, 13, 15, 16])

**Suppression d'un élément**

- impairs.remove(16)

**Copie**

- impairs2 = impairs.copy()

# Appartenance

**Opérateur : `in`**

**Exemples :**

```
if note in accord:
    print("...")

if note in {do, mi, sol}:
    print("...")
```

# Inclusion

## Méthodes : `issubset`, `issuperset`

## Exemples :

```
notes = {'do', 're', 'mi', 'fa', 'sol'}
dofa =  {'do','fa'}

if dofa.issubset(notes):
    print(dofa, ' est inclus dans ', notes)
else:
    print(dofa, " n'est pas inclus dans ", notes)
```

## Exécution :

{'fa', 'do'} est inclus dans {'fa', 'mi', 'ré', 'sol', 'do'}

# Union

## Méthodes : `union`

## Exemples :

```
a = {2, 5, 9}
b = {3, 5, 7}
c = a.union(b)
print(c)
```

## Exécution :

{2, 3, 5, 7, 9}

# Intersection

## Méthodes : `intersection`

## Exemples :

```
a = {2, 5, 9}
b = {3, 5, 7}
c = a.intersection(b)
print(c)
```

## Exécution :

{5}

# Différences ensemblistes

**Opérateur** : − ou **`difference()`**

**Exemples :**

```
a = {2, 5, 9}
b = {3, 5, 7}
c = a.difference(b)
d = b - a
print(c, d)
```

**Exécution :**

```
{9, 2} {3,7}
```

# Itérations

**Itération possible mais ordre quelconque !**

**Exemples :**

```
notes = {'do', 're', 'mi', 'fa', 'sol'}
for notes in notes
    print(note, end=",")
```

**Exécution :**

sol,do,fa,ré,mi,

# Tirage du loto (1)

```
import random

t = []
for i in range(6):
    nouveau = False
    while not nouveau:
        n = random.randint(1, 49)
        nouveau = True
        for c in t:
            if n == c:
                nouveau = False
    t.append(n)

print(t)
```

# Tirage du loto (2)

## Version plus simple avec un ensemble

```
import random

tirage = set()
while len(tirage) != 6:
    n = random.randint(1, 49)
    tirage.add(n)
print(tirage)
```

# Questions...