IFT-1004 - Introduction à la programmation

Module 6 : Le paradigme orienté objet

Honoré Hounwanou, ing.

Département d'informatique et de génie logiciel
Université Laval

Table des matières

La modélisation orientée objet

Objectif

Apprendre à analyser un problème par une approche *ascendante* (*« bottom up »*) permettant d'identifier les objets naturellement présents dans l'environnement (modélisation des données).

Objectif

Apprendre à analyser un problème par une approche *ascendante* (*« bottom up »*) permettant d'identifier les objets naturellement présents dans l'environnement (modélisation des données).

Apprendre à identifier les interactions requises entre ces objets d'un point de vue d'une décomposition fonctionnelle du programme à produire (modélisation des traitements).

La modélisation orientée objet (OO)

Objectif

Apprendre à analyser un problème par une approche *ascendante* (*« bottom up »*) permettant d'identifier les objets naturellement présents dans l'environnement (modélisation des données).

Apprendre à identifier les interactions requises entre ces objets d'un point de vue d'une décomposition fonctionnelle du programme à produire (modélisation des traitements).

Développer des librairies d'objets réutilisables.

Orienté objet ?

- Paradigme permettant de manipuler des concepts

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept
  - Exemple : un rectangle a deux caractéristiques : la longueur et la largeur, qui sont deux valeurs numériques

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept
  - Exemple : un rectangle a deux caractéristiques : la longueur et la largeur, qui sont deux valeurs numériques
  - On peut calculer l'aire et le périmètre d'un rectangle.

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept
  - Exemple : un rectangle a deux caractéristiques : la longueur et la largeur, qui sont deux valeurs numériques
  - On peut calculer l'aire et le périmètre d'un rectangle.
- On appelle ce concept « classe » d'objets

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept
  - Exemple : un rectangle a deux caractéristiques : la longueur et la largeur, qui sont deux valeurs numériques
  - On peut calculer l'aire et le périmètre d'un rectangle.
- On appelle ce concept « classe » d'objets
  - On appelle ces caractéristiques des **attributs** (ou **données membres**) de la classe

Orienté objet ?

- Paradigme permettant de manipuler des concepts
  - Exemples : Voiture, personne, forme géométrique...
- On **agrège** plusieurs types de base pour former un concept et on permet certaines fonctions particulières sur ce concept
  - Exemple : un rectangle a deux caractéristiques : la longueur et la largeur, qui sont deux valeurs numériques
  - On peut calculer l'aire et le périmètre d'un rectangle.
- On appelle ce concept « classe » d'objets
  - On appelle ces caractéristiques des **attributs** (ou **données membres**) de la classe
  - On appelle ces fonctions propres des **méthodes** de la classe

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : c_1 est une instance de cercle de rayon 1.5 cm

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : `c_1` est une instance de cercle de rayon 1.5 cm
  - Exemple : `c_2` est une **autre** instance de cercle de rayon 3 cm

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : `c_1` est une instance de cercle de rayon 1.5 cm
  - Exemple : `c_2` est une **autre** instance de cercle de rayon 3 cm
- Pour accéder à un attribut d'un objet, on écrit simplement : `objet.attribut`

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : `c_1` est une instance de cercle de rayon 1.5 cm
  - Exemple : `c_2` est une **autre** instance de cercle de rayon 3 cm
- Pour accéder à un attribut d'un objet, on écrit simplement : `objet.attribut`
- Un attribut est lui-même une variable associée à un objet

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : `c_1` est une instance de cercle de rayon 1.5 cm
  - Exemple : `c_2` est une **autre** instance de cercle de rayon 3 cm
- Pour accéder à un attribut d'un objet, on écrit simplement : `objet.attribut`
- Un attribut est lui-même une variable associée à un objet
  - Les classes forment donc une hiérarchie de concepts

Attributs d'une classe

- Un ***attribut*** d'une classe et un membre de la classe
  - Il est contenu dans celle-ci
- Un ***objet*** est une ***instance*** particulière d'une classe
  - On peut donc créer autant d'instances (objets) que nécessaire
  - Exemple : `c_1` est une instance de cercle de rayon 1.5 cm
  - Exemple : `c_2` est une **autre** instance de cercle de rayon 3 cm
- Pour accéder à un attribut d'un objet, on écrit simplement : `objet.attribut`
- Un attribut est lui-même une variable associée à un objet
  - Les classes forment donc une hiérarchie de concepts
  - Exemple : Une personne peut avoir un attribut « père » qui est lui-même une personne.

Méthodes d'une classe

- Les ***méthodes*** d'une classe définissent les fonctions permettant la modification de ses attributs

Méthodes d'une classe

- Les ***méthodes*** d'une classe définissent les fonctions permettant la modification de ses attributs
- Une méthode est généralement appelée pour le cas particulier d'une instance de la classe, en vue de la modifier : `instance.methode(args, ...)`

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre
- Elle permet de définir les opérateurs applicables sur ses données

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre
- Elle permet de définir les opérateurs applicables sur ses données
- En théorie, une classe protège donc ses attributs en ne permettant leur modification qu'au travers de ses méthodes

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre
- Elle permet de définir les opérateurs applicables sur ses données
- En théorie, une classe protège donc ses attributs en ne permettant leur modification qu'au travers de ses méthodes
- Par défaut en Python, un attribut est directement manipulable et fait partie de l'interface publique de la classe

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre
- Elle permet de définir les opérateurs applicables sur ses données
- En théorie, une classe protège donc ses attributs en ne permettant leur modification qu'au travers de ses méthodes
- Par défaut en Python, un attribut est directement manipulable et fait partie de l'interface publique de la classe
  - Avantages : simplifie l'utilisation de l'objet et permet d'éviter d'écrire des méthodes « `getter` » et « `setter` » redondantes

Encapsulation

- Une classe réunit donc des données et des fonctions applicables sur ces données
  - On ne devrait manipuler ses attributs qu'à travers ses méthodes. On appelle ceci son ***interface publique***.
- Elle permet de définir un nouveau type de donnée plus complexe selon le concept que l'on souhaite manipuler ou la tâche que l'on veut résoudre
- Elle permet de définir les opérateurs applicables sur ses données
- En théorie, une classe protège donc ses attributs en ne permettant leur modification qu'au travers de ses méthodes
- Par défaut en Python, un attribut est directement manipulable et fait partie de l'interface publique de la classe
  - Avantages : simplifie l'utilisation de l'objet et permet d'éviter d'écrire des méthodes « `getter` » et « `setter` » redondantes
  - Désavantages : l'encapsulation peut facilement être contournée par un utilisateur de la classe

Héritage

Il est possible de construire une classe à partir d'une autre classe.

- On dit que cette classe ***hérite*** de tous les attributs et toutes les méthodes de son ancêtre

Héritage

Il est possible de construire une classe à partir d'une autre classe.

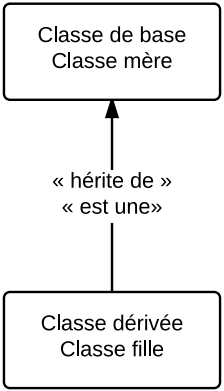
- On dit que cette classe ***hérite*** de tous les attributs et toutes les méthodes de son ancêtre
- On dit aussi que la classe est ***dérivée*** de son ancêtre, que celui-ci est la ***classe de base***

Héritage

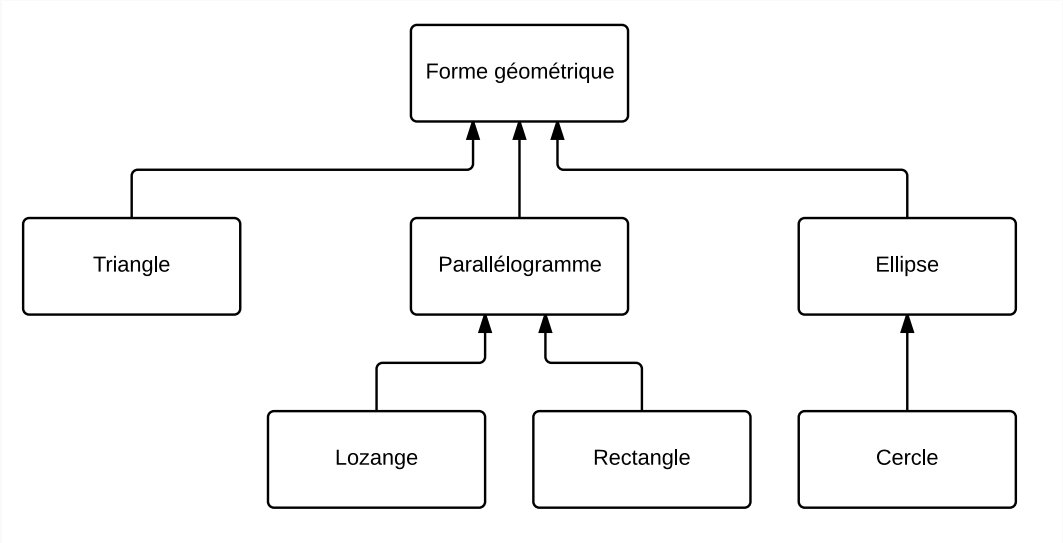
Il est possible de construire une classe à partir d'une autre classe.

- On dit que cette classe ***hérite*** de tous les attributs et toutes les méthodes de son ancêtre
- On dit aussi que la classe est ***dérivée*** de son ancêtre, que celui-ci est la ***classe de base***
- La classe dérivée correspond donc à un **cas particulier** de la classe de base, puisque celle-ci possède tous les attributs et les méthodes de son ancêtre et qu'elle peut en ajouter de nouveaux qui lui sont propres

Héritage


Classe de base
Classe mère
« hérite de »
« est une»
Classe dérivée
Classe fille

Héritage


Forme géométrique
Triangle
Parallélogramme
Ellipse
Lozange
Rectangle
Cercle

Polymorphisme

Le polymorphisme permet d'appeler une même méthode :

- sur tous les objets du type de la classe de base

Polymorphisme

Le polymorphisme permet d'appeler une même méthode :

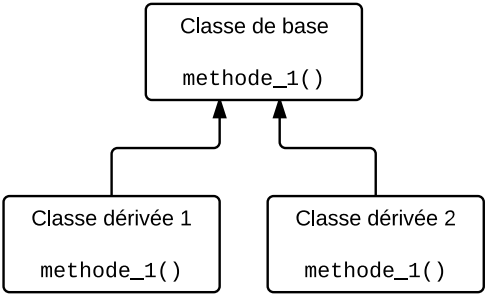
- sur tous les objets du type de la classe de base
- en prenant en considération les spécificités de chacune des classes dérivées

Polymorphisme

Le polymorphisme permet d'appeler une même méthode :

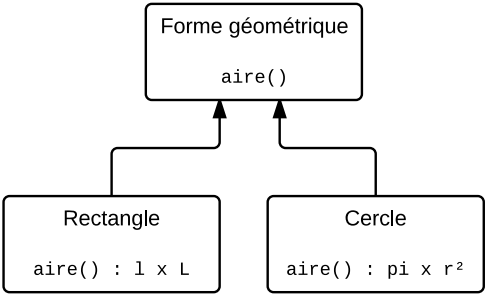
- sur tous les objets du type de la classe de base
- en prenant en considération les spécificités de chacune des classes dérivées
- exemple : on souhaite calculer l'aire de plusieurs formes géométriques

Polymorphisme



Les deux classes dérivées ont hérité de la `methode_1()` mais l'ont potentiellement adaptée à leur besoin.

Polymorphisme



Les deux classes dérivées ont hérité de la méthode `aire()`, mais l'ont adapté à leur spécificité.

Composition

On peut combiner des objets ensemble pour construire de nouveaux objets. Une instance d'une classe peut donc être un attribut d'une autre classe. On parle alors de *composition*.

Composition

On peut combiner des objets ensemble pour construire de nouveaux objets. Une instance d'une classe peut donc être un attribut d'une autre classe. On parle alors de *composition*.

Par exemple, un dessin est composé de plusieurs formes géométriques.

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code
- Une classe est une entité cohérente et indépendante

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code
- Une classe est une entité cohérente et indépendante
- Si elle fait ce que l’on recherche, on la réutilise sans modification

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code
- Une classe est une entité cohérente et indépendante
- Si elle fait ce que l’on recherche, on la réutilise sans modification
- Si elle fait presque l’affaire mais n’est pas assez spécialisée, on la réutilise par héritage et on lui ajoute (change) ce qui manque (ne va pas)

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code
- Une classe est une entité cohérente et indépendante
- Si elle fait ce que l’on recherche, on la réutilise sans modification
- Si elle fait presque l’affaire mais n’est pas assez spécialisée, on la réutilise par héritage et on lui ajoute (change) ce qui manque (ne va pas)
- La fois suivante, on aura le choix entre la classe de base générale ou encore la classe dérivée plus spécialisée

L’orienté objet en bref

- La programmation orientée objet vise à favoriser la réutilisation du code
- Une classe est une entité cohérente et indépendante
- Si elle fait ce que l’on recherche, on la réutilise sans modification
- Si elle fait presque l’affaire mais n’est pas assez spécialisée, on la réutilise par héritage et on lui ajoute (change) ce qui manque (ne va pas)
- La fois suivante, on aura le choix entre la classe de base générale ou encore la classe dérivée plus spécialisée
- Le processus d’héritage peut être employé à volonté

Retour sur la modélisation réalisée au module 5

Retour sur la modélisation réalisée au module 5

Selon la modélisation des données effectuées au module 5, nous grouperons les informations qui caractérisent les objets à définir

Ce faisant, nous verrons si nous pouvons obtenir différents objets reliés par une relation de spécialisation, dont les objectifs sont :

- Le développement modulaire
- La réutilisation logicielle

Définir des classes d'objets (module 5)

Retour sur la modélisation des données effectuée au module 5.

Identifier l’apartenance des traitements (module 5)

Retour sur la modélisation des traitements effectuée au module 5, associer les traitements aux classes d’objets définies précédemment.

La programmation orientée objet

Objectif

Apprendre à définir en Python des classes, une hiérarchie de classes et les méthodes implémentant des traitements.

La définition de classes en Python

La syntaxe pour définir une classe en Python est :

```
class MaClasse(<classe de base>):
    var = valeur  # variable locale à la classe

    def __init__(self, y):  # fonction d'initialisation de la classe
        self.x = y  # déclaration d'un attribut

    def fonc_1(self, z, ...):  # fonction membre (méthode)
        self.x = z  # modification de l'attribut
        var = valeur  # déclaration d'une variable locale à la méthode
        ... # attention aux noms des variables!

    def fonc_2(self, ...):
        ...
```

Exemple

```python
class Cercle:
    pi = 3.14159  # variable locale à la classe

    def __init__(self, r):  # constructeur de la classe
        self.rayon = r  # déclaration d'un attribut

    def aire(self):  # méthode
        return self.rayon * self.rayon * self.pi

    def circonference(self):  # méthode
        return 2 * self.rayon * self.pi

    def afficher(self):  # méthode
        print("Rayon actuel:", self.rayon)
```

Les classes en Python

Un énoncé de type `class` est **exécutable** par son constructeur. Il produit un objet du type de la classe et affecte celui-ci à la variable.

```
>>> c_1 = Cercle(5)
>>> c_2 = Cercle(10)
>>> print(type(c_1))
<class '__main__.Cercle'>
```

Des variables locales à une classe peuvent être partagées par toutes les instances (objets).

```
>>> print(c_1.aire(), c_2.aire())
78.53975 314.159

>>> Cercle.pi = 2.0
>>> print(c_1.aire(), c_2.aire())
50.0 200.0
```

Une classe contient des fonctions membres dont le premier argument (`self`) réfère toujours à l'objet (l'instance) pour lequel la fonction a été appelée.

Le constructeur

Le ***constructeur*** d'une classe est une fonction qui permet d'initialiser des objets de la classe.

- le constructeur porte toujours le nom `__init__`

Le constructeur

Le ***constructeur*** d'une classe est une fonction qui permet d'initialiser des objets de la classe.

- le constructeur porte toujours le nom `__init__`
- créer un objet de la classe provoquera automatiquement l'appel du constructeur, lorsque celui-ci existe

Le constructeur

Le ***constructeur*** d'une classe est une fonction qui permet d'initialiser des objets de la classe.

- le constructeur porte toujours le nom `__init__`
- créer un objet de la classe provoquera automatiquement l'appel du constructeur, lorsque celui-ci existe

Le constructeur

Le ***constructeur*** d'une classe est une fonction qui permet d'initialiser des objets de la classe.

- le constructeur porte toujours le nom `__init__`
- créer un objet de la classe provoquera automatiquement l'appel du constructeur, lorsque celui-ci existe

```
>>> c = Cercle('abc')
>>> c.afficher()
Rayon actuel: abc

>>> c.aire()
TypeError: 'str' object is not callable
```

Les variables de classe et les attributs

Une variable définie dans un énoncé `class` (en dehors de toute fonction) sera définie dans toutes les instances de la classe. On peut aussi définir des variables qui sont distinctes pour chaque instance de la classe.

Les variables de classe et les attributs

Une variable définie dans un énoncé `class` (en dehors de toute fonction) sera définie dans toutes les instances de la classe. On peut aussi définir des variables qui sont distinctes pour chaque instance de la classe.

```
class Toto:
    x = 42

>>> a = Toto()
>>> b = Toto()
>>> a.x, b.x
(42, 42)

>>> Toto.x = 43
>>> a.x, b.x
(43, 43)
```

Les variables de classe et les attributs

Attention : Python permet également de modifier une telle variable à partir d'une instance. Par contre, celle-ci se comporte ensuite comme un *attribut* et devient locale pour l'instance en question.

```
class Toto:
    x = 42

>>> a = Toto()
>>> b = Toto()
>>> a.x, b.x
(42, 42)

>>> a.x = 43
>>> a.x, b.x
(43, 42)
```

Il vaudrait mieux utiliser un vrai attribut dans ce cas.

```
class Toto:
    def __init__(self):
        self.x = 42

>>> a = Toto()
>>> b = Toto()
>>> a.x, b.x
(42, 42)

>>> a.x = 43
>>> a.x, b.x
(43, 42)
```

Définition et appel d'une méthode

Une méthode est appelée à l'aide d'une instance, suivi d'un point et du nom de la méthode. Remarquez que le paramètre `self` est automatiquement remplacé par l'instance avec laquelle a été appelée la méthode.

Définition et appel d'une méthode

Une méthode est appelée à l'aide d'une instance, suivi d'un point et du nom de la méthode. Remarquez que le paramètre `self` est automatiquement remplacé par l'instance avec laquelle a été appelée la méthode.

```
class Toto:
    def __init__(self, x):
        self.x = x

    def fonc(self):
        print("Cette instance de Toto a x égal à", self.x)
```

```
>>> t = Toto(42)
>>> t.fonc()
Cette instance de Toto a x égal à 42
```

Accéder à la classe de base

Un objet d'une classe dérivée peut accéder aux attributs et aux méthodes de la classe de base à l'aide de la fonction `super()`.

```
class Toto:
    def fonc(self):
        print("Appel de fonc() de la classe Toto")
class Tata(Toto):
    def fonc(self):
        print("Appel de fonc() redéfinie dans Tata")
        super().fonc()

>>> t_1, t_2 = Toto(), Tata()
>>> t_1.fonc()
Appel de fonc() de la classe Toto

>>> t_2.fonc()
Appel de fonc() redéfinie dans Tata
Appel de fonc() de la classe Toto
```

Portée des variables

```
x = 11  # attribut d'un module, variable globale

def f():
    print(x)  # affiche la variable globale

def g():
    x = 22    # variable locale, cache la variable
    print(x)  # globale du même nom

class C:
    x = 33  # variable associée à la classe C
    def h(self):
        x = 44  # variable locale
        self.x = 55  # attribut de l'instance
```

L’orienté objet en bref

- Encapsulation :

L'orienté objet en bref

- Encapsulation :
  - Une classe contient :

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances

L'orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d'autres instances
    - des méthodes pour modifier ces données

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d’autres objets, c’est à dire des instances de la classe

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d’autres objets, c’est à dire des instances de la classe
  - Une instance possède tous les attributs de sa classe et peut aussi en créer de nouveaux

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d’autres objets, c’est à dire des instances de la classe
  - Une instance possède tous les attributs de sa classe et peut aussi en créer de nouveaux
- Héritage

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d’autres objets, c’est à dire des instances de la classe
  - Une instance possède tous les attributs de sa classe et peut aussi en créer de nouveaux
- Héritage
  - Une classe peut hériter des attributs et méthodes d’une autre classe

L'orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d'autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d'autres objets, c'est à dire des instances de la classe
  - Une instance possède tous les attributs de sa classe et peut aussi en créer de nouveaux
- Héritage
  - Une classe peut hériter des attributs et méthodes d'une autre classe
- Polymorphisme

L’orienté objet en bref

- Encapsulation :
  - Une classe contient :
    - des données qui peuvent aussi être d’autres instances
    - des méthodes pour modifier ces données
  - Une classe est un objet capable de fabriquer d’autres objets, c’est à dire des instances de la classe
  - Une instance possède tous les attributs de sa classe et peut aussi en créer de nouveaux
- Héritage
  - Une classe peut hériter des attributs et méthodes d’une autre classe
- Polymorphisme
  - Un appel de méthode de plusieurs objets partageant une même classe de base sera fait avec les spécificités des objets (on appelle la version « spécialisée » de la méthode).

Exemple en Python

```python
class Point:
    """Point géométrique."""
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Exemple en Python

```
class Rectangle:
    """Un rectangle."""
    def __init__(self, origine, largeur, hauteur):
        self.origine = origine  # Ceci sera un point
        self.largeur = largeur
        self.hauteur = hauteur

    def trouve_centre(self):
        x_centre = self.origine.x + self.largeur / 2
        y_centre = self.origine.y + self.hauteur / 2
        return Point(x_centre, y_centre)

    def surface(self):
        return self.hauteur * self.largeur
```

Exemple en Python

```
class Carre(Rectangle):
    """Un carré = un rectangle particulier."""
    def __init__(self, origine, cote):
        super().__init__(origine, cote, cote)
        self.cote = cote

    def surface(self):
        return self.cote ** 2
```

Exemple en Python

Programme principal :

```
origine_rectangle = Point(40, 30)
origine_carre = Point(10, 25)
boite_rectangulaire = Rectangle(origine_rectangle, 100, 50)
boite_carree = Carre(origine_carre, 40)
centre_rectangle = boite_rectangulaire.trouve_centre()
centre_carre = boite_carree.trouve_centre()

print("Centre du rectangle :", centre_rectangle.x, centre_rectangle.y)
print("Centre du carré :", centre_carre.x, centre_carre.y)
print("Surface du carré : ", boite_carree.surface())
```

Résultat :

```
Centre du rectangle : 90.0 55.0
Centre du carré : 30.0 45.0
Surface du carré :  1600
```

Lectures et travaux dirigés

Lectures et travaux dirigés

- Chapitres 11 et 12 de G. Swinnen
- Travaux dirigés : Exercices sur la programmation orientée objet.