



## UV2.1. Langage et algorithmique

### - CR1. Programmation Orientée Objet (POO)

---

A. Malek TOUMI

[toumiab@ensta-bretagne.fr](mailto:toumiab@ensta-bretagne.fr)

2017/2018

ENSTA Bretagne



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



# Définition

[toggle](#)[reset](#)

## Définition (Héritage)

Mécanisme spécifique aux langages orientés objet qui permet à une classe B (appelée sous-classe ou classe fille) d'hériter de toutes les propriétés d'une classe A (appelée super-classe ou classe mère).

Utilité :

- Factoriser du code
- Gérer des différents de manière similaire
- Augmenter la réutilisabilité



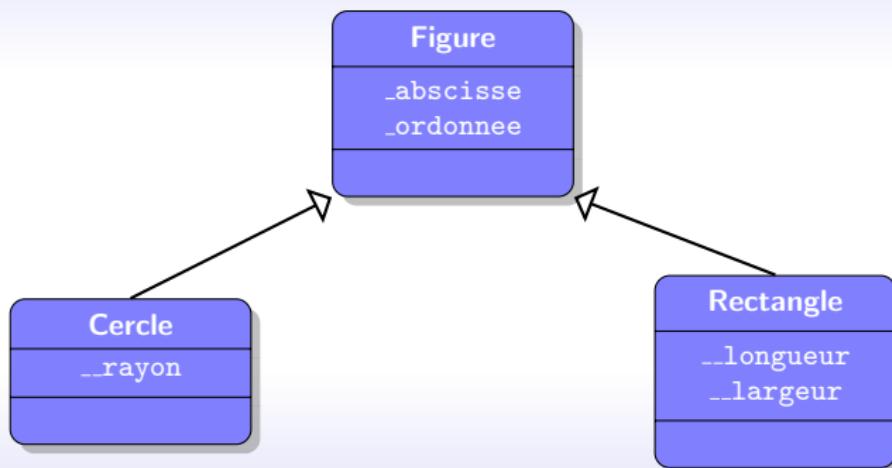
## Exemple

[toggle](#)[reset](#)

- Gestion de figures géométriques
  - Différents types de figures : cercles et rectangles
  - Propriétés communes (position du centre de la figure)
  - Propriétés spécifiques (rayon du cercle, longueur et largeur du rectangle)
- ⇒ trois classes



# Exemple

[toggle](#)[reset](#)



---

## Héritage multiple

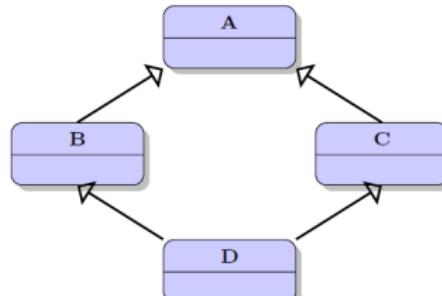
### Remarque (Héritage multiple)

Python, à l'encontre de certains langages, permet d'hériter de plusieurs classes.



## Héritage multiple

Exemple : D hérite de B et C, B et C héritent de A.





## Héritage multiple

### Remarque (Héritage multiple)

Python, à l'encontre de certains langages, permet d'hériter de plusieurs classes.

### Attention

L'héritage multiple entraîne des problèmes théoriques lorsqu'on hérite plusieurs fois du même parent.

⇒ linéarisation de l'ordre de l'héritage



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



# Héritage en Python

toggle

reset

- En Python : l'héritage est défini entre les parenthèses
- Le constructeur est la méthode nommée `__init__`
- Visibilité des variables d'instance : `(__)` ⇒ décorateurs
- Si aucun héritage n'est précisé : hérite de `object`

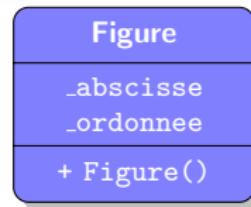
## Exemple (Figures)

```
class Figure (object):
    def __init__(self,x, y):
        self._abscisse = x
        self._ordonnee = y

f1 = Figure(10,2) # création de l'objet f1
f2 = Figure(0, 4) # création de l'objet f2
```



# Héritage en Python

[toggle](#)[reset](#)

```
f = Figure(10,11) # Création de l'objet figure f
f1 = Figure (1,5) # Création de l'objet Figure f1
```



# Héritage en Python

toggle

reset

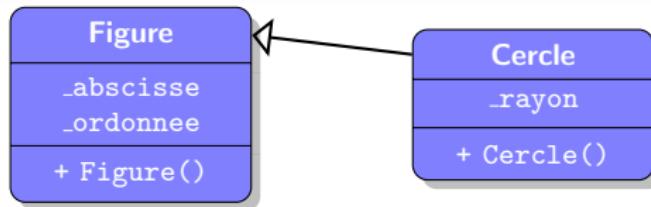
- En Python : l'héritage est défini entre les parenthèses
- Le constructeur est la méthode nommée `__init__`
- Visibilité des variables d'instance : `(__)`  $\Rightarrow$  décorateurs
- Si aucun héritage n'est précisé : hérite de `object`

## Exemple (Figures)

```
class Figure (object):
    def __init__(self,x, y):
        self._abscisse = x
        self._ordonnee = y
class Cercle (Figure):
    def __init__(self,x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r         # _abscisse et _ordonnee
```



# Héritage en Python

[toggle](#)[reset](#)

```
f= Figure(10,11) # Création de l'objet figure f
c= Cercle (1,5,3) # Création de l'objet Cercle c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```



# Appel du constructeur de la super-classe

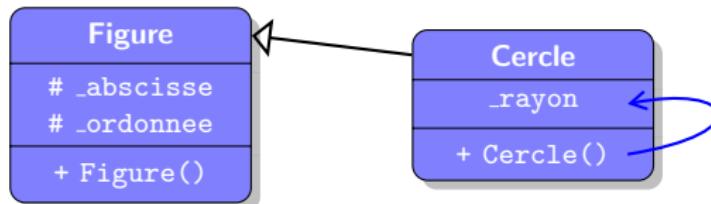
## Exemple (Figures)

```
class Figure(object):
    def __init__(self, a, b):
        self._abscisse = a
        self._ordonnee = b
class Cercle (Figure):
    def __init__(self, x=0, y=0, r=0):
        super().__init__(x,y) # définit et initialise
        self._rayon = r          # _abscisse et _ordonnee

c = Cercle(10, 3, 2) # création de l'objet c
```

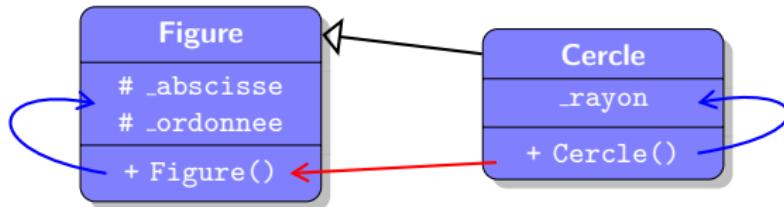


## Appel du constructeur de la super-classe





## Appel du constructeur de la super-classe





---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



## Définition

[toggle](#)[reset](#)

### Définition (Polymorphisme)

Mécanisme selon lequel un même message peut être envoyé vers des objets de types différents, chaque objet réagissant de façon originale.



## Définition

togglereset

### Définition (Polymorphisme)

Mécanisme selon lequel un même message peut être envoyé vers des objets de types différents, chaque objet réagissant de façon originale.

Objectif :

- Rendre une certaine complexité de programmation transparente
- Faciliter la conception de programmes



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



# Principe

[toggle](#)[reset](#)

- Plusieurs méthodes avec le même nom dans une hiérarchie de classes
- Méthode appelée : correspond à l'objet instancié
- Avantage : manipuler toutes les instances d'une hiérarchie de classes de manière uniforme



- Plusieurs méthodes avec le même nom dans une hiérarchie de classes
- Méthode appelée : correspond à l'objet instancié
- Avantage : manipuler toutes les instances d'une hiérarchie de classes de manière uniforme

### Exemple (Jeux d'échecs)

Méthode **deplacer()** pour les pièces : Roi, Reine, Fou, Cavalier, Tour et Pion (héritant chacun du type Pièce).



- Plusieurs méthodes avec le même nom dans une hiérarchie de classes
- Méthode appelée : correspond à l'objet instancié
- Avantage : manipuler toutes les instances d'une hiérarchie de classes de manière uniforme

### Exemple ()

```
# Déclaration d'une liste de pièces
liste_pieces = [Pion(), Cavalier()]
# Appliquer la méthode déplacer à chaque pièce
for piece in liste_pieces :
    piece.deplacer()
```



# Principe

[toggle](#)[reset](#)

- Plusieurs méthodes avec le même nom dans une hiérarchie de classes
- Méthode appelée : correspond à l'objet instancié
- Avantage : manipuler toutes les instances d'une hiérarchie de classes de manière uniforme

## Exemple (Figures 2D, cercles, rectangles)

Méthode **aire()** : fournit la valeur l'aire pour Cercle et Rectangle.



## Exemple

[toggle](#)[reset](#)

### Exemple

```
import numpy as np
class Figure (object):
    ...
class Cercle (Figure):
    ...
    def aire():
        return np.pi * self._rayon**2
class Rectangle (Figure):
    ... # avec 2 var d'instance self._h, et self._l
    def aire():
        return self._h * self._l
```



## Utilisation des figures

toggle

reset

### Exemple

```
liste_figures = []
liste_figures.append(Cercle(10, 11, 2))
```



## Utilisation des figures

toggle

reset

### Exemple

```
liste_figures = []
liste_figures.append(Cercle(10, 11, 2))
liste_figures.append(Rectangle(10,5, 3, 4))
....
```



# Utilisation des figures

[toggle](#)[reset](#)

## Exemple

```
liste_figures = []
liste_figures.append(Cercle(10, 11, 2))
liste_figures.append(Rectangle(10,5, 3, 4))
....  
  
for i in range(len(liste_figures)):
    print(liste_figures[i].aire())
```



## Exemple de surcharge : méthode `__str__`

toggle

reset

- Méthode appelée pour convertir un objet en chaîne de caractères



## Exemple de surcharge : méthode `__str__`

toggle

reset

- Méthode appelée pour convertir un objet en chaîne de caractères
- ⇒ afficher directement les objets



## Exemple de surcharge : méthode `__str__`

togglereset

- Méthode appelée pour convertir un objet en chaîne de caractères  
⇒ afficher directement les objets
- Prédéfinie dans la classe `object`

Exemple vu (le type `Intervalle`) : l'affichage d'un objet `Intervalle`



## Exemple de surcharge : méthode `__str__`

togglereset

- Méthode appelée pour convertir un objet en chaîne de caractères  
⇒ afficher directement les objets
- Prédéfinie dans la classe `object`

Exemple vu (le type `Intervalle`) : l'affichage d'un objet `Intervalle`  
Exemple d'affichage d'une figure `f` :

```
f = Figure(3,5)  
print(f)
```



## Exemple de surcharge : méthode `__str__`

togglereset

- Méthode appelée pour convertir un objet en chaîne de caractères  
⇒ afficher directement les objets
- Prédéfinie dans la classe `object`

Exemple vu (le type `Intervalle`) : l'affichage d'un objet `Intervalle`  
Exemple d'affichage d'une figure `f` :

```
f = Figure(3,5)
print(f)
<__main__.Figure object at 0x00000000575DBA8>
```



## Exemple de surcharge : méthode `__str__`

togglereset

- Méthode appelée pour convertir un objet en chaîne de caractères  
⇒ afficher directement les objets
- Prédéfinie dans la classe `object`

Exemple vu (le type `Intervalle`) : l'affichage d'un objet `Intervalle`  
Exemple d'affichage d'une figure `f` :

```
f = Figure(3,5)  
print(f)
```

```
def __str__(self):  
    return "Fig. Pos = ({0},{1})".format(self._abscisse,  
                                         self._ordonnee)
```



## Exemple de surcharge : méthode `__str__`

togglereset

- Méthode appelée pour convertir un objet en chaîne de caractères  
⇒ afficher directement les objets
- Prédéfinie dans la classe `object`

Exemple vu (le type `Intervalle`) : l'affichage d'un objet `Intervalle`  
Exemple d'affichage d'une figure `f` :

```
f = Figure(3,5)
print(f)
Fig. Pos = (3,5)
```

```
def __str__(self):
    return "Fig. Pos = ({0},{1})".format(self._abscisse,
                                           self._ordonnee)
```



# Quelques méthodes à surcharger

[toggle](#)[reset](#)

Méthode de la classe à surcharger	Utilisation
object. <code>__str__</code> (self)	<code>print(self)</code>
object. <code>__repr__</code> (self)	<code>self</code>
object. <code>__add__</code> (self, other)	<code>self + other</code>
object. <code>__sub__</code> (self, other)	<code>self - other</code>
object. <code>__mul__</code> (self, other)	<code>self * other</code>
object. <code>__and__</code> (self, other)	<code>self and other</code>
object. <code>__or__</code> (self, other)	<code>self or other</code>
object. <code>__len__</code> (self)	<code>len(self)</code>
object. <code>__getitem__</code> (self,i)	<code>x = self[i]</code>
object. <code>__setitem__</code> (self, i)	<code>self[i] = y</code>



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



## Principe

[toggle](#)[reset](#)

- Variables et méthodes propriétés de la classe, pas de l'instance
- ⇒ un seul exemplaire par classe
- Accessible depuis les classes et les instances
- Utilisation le décorateur `@classmethod` avec comme premier argument `cls` et non pas `self`



## Principe

togglereset

- Variables et méthodes propriétés de la classe, pas de l'instance
- ⇒ un seul exemplaire par classe
- Accessible depuis les classes et les instances
- Utilisation le décorateur `@classmethod` avec comme premier argument `cls` et non pas `self`

### Remarque

On peut utiliser la fonction built-in `classmethod` (`methodeClasse`) pour définir la méthode `methodeClass()` comme méthode de classe.



## Exemple

[toggle](#)[reset](#)

### Exemple

Comment compter les figures créées dans la classe **Figure** (et ses classes filles) ou attribuer un numéro unique à chaque objet ?



## Exemple

[toggle](#)[reset](#)

### Exemple

Comment compter les figures créées dans la classe **Figure** (et ses classes filles) ou attribuer un numéro unique à chaque objet ?

```
class Figure (object):
    nb_figure = 0 # variable de classe : compteur
    def __init__(self, x, y):
        self.abscisse = x
        self.ordonnee = y
```



## Exemple

[toggle](#)[reset](#)

### Exemple

Comment compter les figures créées dans la classe **Figure** (et ses classes filles) ou attribuer un numéro unique à chaque objet ?

```
class Figure (object):
    nb_figure = 0 # variable de classe : compteur
    def __init__(self, x, y):
        self._abscisse = x
        self._ordonnee = y
        Figure.nb_figures += 1
    @classmethod
    def get_nbfigures(cls):
        return Figure.nb_figures
```



## Exemple

[toggle](#)[reset](#)

### Exemple

Comment compter les figures créées dans la classe **Figure** (et ses classes filles) ou attribuer un numéro unique à chaque objet ?

```
class Figure (object):
    nb_figure = 0 # variable de classe : compteur
    def __init__(self, x, y):
        self._abscisse = x
        self._ordonnee = y
        Figure.nb_figures += 1
    @classmethod
    def get_nbfigures(cls):
        return Figure.nb_figures
```



## Exemple

[toggle](#)[reset](#)

### Exemple

Comment compter les figures créées dans la classe **Figure** (et ses classes filles) ou attribuer un numéro unique à chaque objet ?

```
class Figure (object):
    nb_figure = 0 # variable de classe : compteur
    def __init__(self, x, y):
        self._abscisse = x
        self._ordonnee = y
        Figure.nb_figures += 1
    @classmethod
    def get_nbfigures(cls):
        return Figure.nb_figures
```



## Exemple

toggle

reset

```
x = Figure (2, 3)  
print(Figure.get_nbfigures())
```



## Exemple

toggle

reset

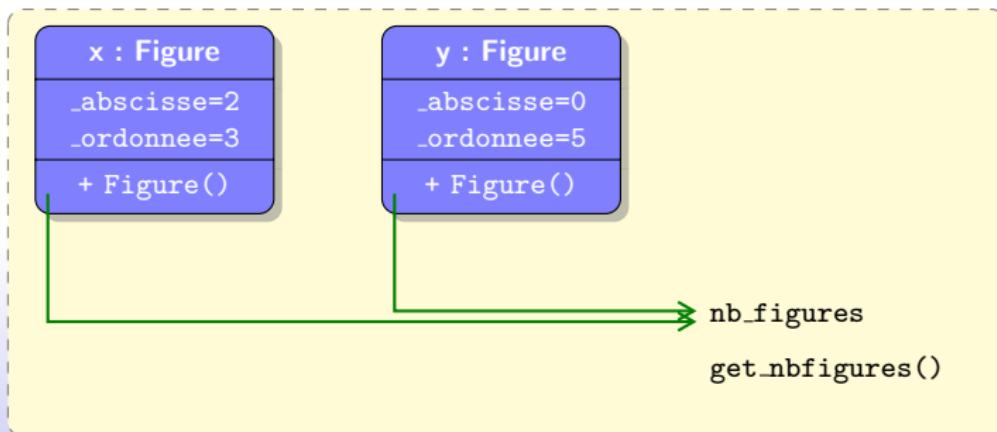
```
x = Figure (2, 3)
print(Figure.get_nbfigures()) # => 1
y = Figure (0, 5)
print(Figure.get_nbfigures())
```



## Exemple

[toggle](#)[reset](#)

```
x = Figure (2, 3)
print(Figure.get_nbfigures()) # => 1
y = Figure (0, 5)
print(Figure.get_nbfigures()) # => 2)
```





---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



# Méthodes abstraites

[toggle](#)[reset](#)

## Définition (Méthode abstraite)

Une méthode est abstraite si sa déclaration est précédée par le décorateur **@abstractmethod**. Elle ne contient alors aucun code



# Méthodes abstraites

[toggle](#)[reset](#)

## Définition (Méthode abstraite)

Une méthode est abstraite si sa déclaration est précédée par le décorateur **@abstractmethod**. Elle ne contient alors aucun code

## Exemple (Dans la classe Figure)

```
from abc import abstractmethod
class Figure(object):
    ...
    @abstractmethod
    def aire(self):
        """La méthode aire est déclarée abstraite.
        """
        pass
```



# Méthodes abstraites

[toggle](#)[reset](#)

## Définition (Méthode abstraite)

Une méthode est abstraite si sa déclaration est précédée par le décorateur **@abstractmethod**. Elle ne contient alors aucun code. Il n'est pas possible de l'appeler ssi :

- Sa classe est définie comme **classe abstraite** ⇒ définir dans les sous-classes.

## Exemple (Dans la classe Figure)

```
from abc import abstractmethod ABCMeta
class Figure(metaclass=ABCMeta):
    ...
    @abstractmethod
    def aire(self):
        """La méthode aire est déclarée abstraite.
        """
        pass
```



# Héritage d'une classe abstraite

togglereset

## Héritage

- Une sous-classe d'une classe abstraite peut être concrète si elle redéfinit toutes les méthodes abstraites. Sinon, elle est considérée abstraite
- Exemple : **aire** est redéfinie dans **Cercle** et **Rectangle**



# Héritage d'une classe abstraite

togglereset

## Héritage

- Une sous-classe d'une classe abstraite peut être concrète si elle redéfinit toutes les méthodes abstraites. Sinon, elle est considérée abstraite
- Exemple : **aire** est redéfinie dans **Cercle** et **Rectangle**

## Intérêt :

- Déclarer explicitement qu'une classe n'est pas instanciable
- Définir les prototypes des méthodes dans la super-classe



# Héritage d'une classe abstraite

togglereset

## Héritage

- Une sous-classe d'une classe abstraite peut être concrète si elle redéfinit toutes les méthodes abstraites. Sinon, elle est considérée abstraite
- Exemple : **aire** est redéfinie dans **Cercle** et **Rectangle**

## Intérêt :

- Déclarer explicitement qu'une classe n'est pas instanciable
- Définir les prototypes des méthodes dans la super-classe

Exemple : Liste de **Figure**



## Exemple de classe abstraite

[toggle](#)[reset](#)

```
from abc import abstractmethod, ABCMeta
class Figure(metaclass = ABCMeta):
    @abstractmethod
    def aire(self):
        pass
class Cercle (Figure) :
    ...
    def aire(self):
        return np.pi * self._rayon**2
    ...
```



## Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
class Figure(metaclass = ABCMeta):
    @abstractmethod
    def aire(self):
        pass
class Cercle (Figure) :
    ...
    def aire(self):
        return np.pi * self._rayon**2
    ...
class Rectangle (Figure) :
    ...
    def aire(self):
        return self._h *self._l
    ...
```



# Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
l_figures = []
# création d'un objet Cercle et ajout à la liste
l_figures.append(Cercle(10, 11, 1))
# création d'un objet Rectangle et ajout à la liste
l_figures.append(Rectangle(10,5, 3, 4))
# tentation d'instancier la classe abstraite Figure
```



# Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
l_figures = []
# création d'un objet Cercle et ajout à la liste
l_figures.append(Cercle(10, 11, 1))
# création d'un objet Rectangle et ajout à la liste
l_figures.append(Rectangle(10,5, 3, 4))
# tentation d'instancier la classe abstraite Figure
l_figures.append(figure(2,0))
```



# Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
l_figures = []
# création d'un objet Cercle et ajout à la liste
l_figures.append(Cercle(10, 11, 1))
# création d'un objet Rectangle et ajout à la liste
l_figures.append(Rectangle(10,5, 3, 4))
# tentation d'instancier la classe abstraite Figure
l_figures.append(Figure(2,0)) # => Error
```



# Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
l_figures = []
# création d'un objet Cercle et ajout à la liste
l_figures.append(Cercle(10, 11, 1))
# création d'un objet Rectangle et ajout à la liste
l_figures.append(Rectangle(10,5, 3, 4))
# tentation d'instancier la classe abstraite Figure
l_figures.append(Figure(2,0)) # => Error
# affichage de l'aire des objets
```



# Exemple de classe abstraite

togglereset

```
from abc import abstractmethod, ABCMeta
l_figures = []
# création d'un objet Cercle et ajout à la liste
l_figures.append(Cercle(10, 11, 1))
# création d'un objet Rectangle et ajout à la liste
l_figures.append(Rectangle(10,5, 3, 4))
# tentation d'instancier la classe abstraite Figure
l_figures.append(Figure(2,0)) # => Error
# affichage de l'aire des objets
for f in l_figures :
    print(f.aire())
```



---

## Sommaire

### 1 Héritage

Généralités

Héritage en Python

### 2 Polymorphisme

Généralités

Polymorphisme d'héritage (redéfinition, spécialisation)

### 3 Variables et méthodes de classe

### 4 Abstraction

Méthodes et classes abstraites

### 5 Horloge



## Horloge

---

toggle

reset

