

Résumé du cours : Programmation orientée objet

HOUCMEDDINE FILALI

HOUSSEMEDDINE BOUZIDI

Un **objet** est une entité manipulée par un programme qui possède :

- une identité : `id(object)`
- un état : valeurs prises par ses attributs `obj.attribut` (`z.imag`, `z.real`, etc.)
- un comportement : spécifié par l'ensemble des méthodes applicables sur l'objet (`z.conjugate()`, `l.append(5)`, etc.)

En Python, un objet est créé à partir d'une classe (un moule à objets). Les classes décrivent les opérations (méthodes) applicable sur leurs instances. Par exemple, pour un objet de la classe `list`, il est possible d'appliquer les méthodes (`append`, `pop`, `insert`, `sort`, etc.). Les opérations natives (instanciation (`obj = NomClasse(params)`), opérations arithmétiques (+, -, *, **, etc.), indexage (`s[i]`, `s[i]=val`), invocation (`f(x)`), etc.) peuvent être supportés par une classe si celle dernière implémente les méthodes magiques (méthodes spéciales) reliées à ces opérations. Ces méthodes sont invoquées automatiquement par Python lorsque l'opération native associée est utilisée : (construction d'une nouvelle instance `__init__`, addition `__add__`, multiplication `__mul__`, indexage `__getitem__`, `__setitem__`, etc.).



Concepts de base

Description	Syntaxe	Exemple(s) & Remarques
Une classe représente un espace de nom « name space » contenant les noms accessibles à partir de la classe i.e (les attributs de classe et les méthodes).	<pre>class NomClasse : # corps de la classe attribut_classe = expression def methode1(self, params): # corps de la methode def methode2(self, params) # ...</pre>	<pre>class Point2D: def __init__(self, x, y): self.x = float(x) self.y = float(y) def __repr__(self): motif = "Point2D({},{})" return motif.format(self.x, self.y) def translator(self, dx, dy): #methode mutatrice self.x += dx self.y += dy</pre>
Constructeur (initialiseur) : le constructeur est une méthode magique (spéciale) appelée automatiquement par Python lors de la construction d'une nouvelle instance de la classe. Elle est dénotée par <code>__init__</code> . Les attributs d'instance sont typiquement créés dans cette méthode.	<pre>class NomClasse: def __init__(self, params): self.attribut_instance1 = expression1 ...</pre>	
Instanciation de classe : Créer un nouvel objet à partir de la classe. La méthode magique <code>__init__</code> de la classe instanciée est automatiquement appelée par Python.	<pre>obj = NomClasse(params)</pre>	<pre>z = complex(3,2) x = float(input()) p = Point2d(5,3) e = Etudiant("nawfel")</pre>

Concept d'attribut

Un attribut décrit une donnée caractéristique de l'objet. (l'ensemble des valeurs prises par les attributs d'un objet forment son état).

Description	Syntaxe	Exemple(s) & Remarques
Attributs d'instance : sont associés aux instances (objets) d'une classe. Définissent l'état d'une instance (chaque objet en possède ses propres valeurs) ex : <code>z.real, z.imag</code>	Définition d'un attribut d'instance : la création des attributs d'instance est typiquement effectuée au niveau de l'initialiseur (constructeur) : <pre>class NomClasse: def __init__(self, params): self.attr_instance = expr ...</pre> accès à un attribut d'instance : <code>instance.nom_attribut</code> créer/modifier un attribut d'instance : <code>instance.nom_attribut = expr</code>	<pre>z = complex(3,2) z.real #3.0 z.imag#2.0</pre>
Attributs de classe : — Créé une seule fois lors du chargement de la classe. — Partagés (en lecture et en écriture) entre toutes les instances de la classe.	Définition : <pre>class NomClasse: nom_attr_classe = expr</pre> Accès : <code>NomClasse.nom_attr_classe # (lecture)</code> <code>NomClasse.nom_attr_classe = expr # (écriture)</code>	<pre>class Joueur: compteur = 0 def __init__(self, nom): Joueur.compteur += 1 self.numéro = Joueur.compteur self.nom = nom ...</pre>



Concept de méthode

Une méthode est une fonction définie dans le corps d'une classe. Il existe deux plusieurs catégories de méthodes entre autres : les méthode ordinaires et les méthodes magiques.

Description	Syntaxe	Exemple(s) & Remarques
Méthode : une méthode décrit un service (opération) applicable sur une instance de la classe, elle peut éventuellement recevoir d'autres paramètres à part l'objet en question (dénoté par le nom <code>self</code>)	définition d'une méthode : <pre>class NomClasse : def nom_methd(self, params): corps de la méthode</pre> appel d'une méthode : <pre>instance.nom_methode(params) ou bien NomClasse.nom_methode(instance, params)</pre>	<pre>z = complex(3,2) z.conjugate() #3-2j complex.conjugate(z) l = list() l.append(5) list.append(1,5) l.insert(0,"ali") list.insert(1, 0, 'ali')</pre>



Méthodes magiques

une méthode magique est liée à une opération native de Python. Une méthode magique est automatiquement invoquée quand l'opération native associée est appliquée sur une instance d'une classe.

Description	Syntaxe	Exemple(s) & Remarques																
Méthodes associées à la construction/destruction d'une instance	<table><tr><th>Opération</th><th>Méthode associée</th></tr><tr><td>inst = Classe(params)</td><td>... Classe.__init__(inst, params)</td></tr><tr><td>destruction d'un objet</td><td>obj.__del__()</td></tr></table>	Opération	Méthode associée	inst = Classe(params)	... Classe.__init__(inst, params)	destruction d'un objet	obj.__del__()											
Opération	Méthode associée																	
inst = Classe(params)	... Classe.__init__(inst, params)																	
destruction d'un objet	obj.__del__()																	
Méthodes associées à la représentation textuelle (informelle et formelle)	<table><tr><th>Opération</th><th>Méthode associée</th></tr><tr><td>str(obj)</td><td>obj.__str__()</td></tr><tr><td>repr(obj)</td><td>obj.__repr__()</td></tr></table>	Opération	Méthode associée	str(obj)	obj.__str__()	repr(obj)	obj.__repr__()	<pre>from numpy import array a = array([1,2,3]) a # repr(a) print(a) #print(str(a))</pre>										
Opération	Méthode associée																	
str(obj)	obj.__str__()																	
repr(obj)	obj.__repr__()																	
Méthodes associées aux opérations arithmétiques	<table><tr><th>Opérateur</th><th>Méthode associée</th></tr><tr><td>+</td><td>__add__</td></tr><tr><td>-</td><td>__sub__</td></tr><tr><td>*</td><td>__mul__</td></tr><tr><td>/</td><td>__truediv__</td></tr><tr><td>//</td><td>__floordiv__</td></tr><tr><td>%</td><td>__mod__</td></tr><tr><td>abs(obj)</td><td>__abs__</td></tr></table>	Opérateur	Méthode associée	+	__add__	-	__sub__	*	__mul__	/	__truediv__	//	__floordiv__	%	__mod__	abs(obj)	__abs__	<pre>l1 = [1,2]; l2 = [3] l1 + l2; l1.__add__(l2); list.__add__(l1, l2) s1 = {1,2,3}; s2 = {5} s1 - s2; s1.__sub__(s2); set.__sub__(s1, s2) z = 3+4j abs(z); z.__abs__(); complex.__abs__(z)</pre>
Opérateur	Méthode associée																	
+	__add__																	
-	__sub__																	
*	__mul__																	
/	__truediv__																	
//	__floordiv__																	
%	__mod__																	
abs(obj)	__abs__																	
Méthodes magiques associées aux opérations relationnelles	<table><tr><th>Opérateur</th><th>Méthode associée</th></tr><tr><td>==</td><td>__eq__</td></tr><tr><td>!=</td><td>__ne__</td></tr><tr><td><</td><td>__lt__</td></tr><tr><td>></td><td>__gt__</td></tr><tr><td><=</td><td>__le__</td></tr><tr><td>>=</td><td>__ge__</td></tr></table>	Opérateur	Méthode associée	==	__eq__	!=	__ne__	<	__lt__	>	__gt__	<=	__le__	>=	__ge__	<pre>l1 == l2; l1.__eq__(l2); list.__eq__(l1,l2) s1 <= s2; s1.__le__(s2); set.__le__(s1, s2)</pre>		
Opérateur	Méthode associée																	
==	__eq__																	
!=	__ne__																	
<	__lt__																	
>	__gt__																	
<=	__le__																	
>=	__ge__																	
Méthode associée aux objets appelables (callable)	instance(params) engendre l'invocation de type(instance).__call__(instance, params)	<pre>from numpy import poly1d p = poly1d([1,2,5]) # instantiation p(x) = x^2 + 2 x + 5 p(5); p.__call__(5); poly1d.__call__(p, 5) #40</pre>																
Méthodes associées aux opérations sur les itérables	<table><tr><th>Opération</th><th>Méthode associée</th></tr><tr><td>len(obj)</td><td>obj.__len__()</td></tr><tr><td>obj[index]</td><td>obj.__getitem__(index)</td></tr><tr><td>obj[index] = expr</td><td>obj.__setitem__(index, expr)</td></tr><tr><td>v in obj</td><td>obj.__contains__(v)</td></tr></table>	Opération	Méthode associée	len(obj)	obj.__len__()	obj[index]	obj.__getitem__(index)	obj[index] = expr	obj.__setitem__(index, expr)	v in obj	obj.__contains__(v)	<pre>l = [5, 8, -18] len(l); l.__len__(); list.__len__(l) l[0]; l.__getitem__(0); list.__getitem__(l,0) l[0] = 3j;l.__setitem__(0, 3j);list.__setitem__(l,0, 3j) 3j in l;l.__contains__(3j);list.__contains__(l,3j)</pre>						
Opération	Méthode associée																	
len(obj)	obj.__len__()																	
obj[index]	obj.__getitem__(index)																	
obj[index] = expr	obj.__setitem__(index, expr)																	
v in obj	obj.__contains__(v)																	

Héritage

L'héritage est un mécanisme de réutilisation de classes. La classe fille hérite toutes les méthodes définies dans sa classe mère. La classe fille peut éventuellement les redéfinir en ajoutant quelques instructions puis en déléguant l'appel vers la classe mère.

Description	Syntaxe	Exemple(s) & Remarques
l'héritage permet de créer une nouvelle classe à comportant les méthode d'une classe existante (classe mère). La classe fille (qui hérite) peut éventuellement définir ses propres méthodes et redéfinir les méthodes héritées de sa classe mère.	<pre>class ClasseFille(ClasseMère): # corps de la classe fille</pre>	<pre>class PointColoré(Point2D): def __init__(self, x, y, couleur): super().__init__(x,y) #ou bien Point2D.__init__(self, x, y) self.col = couleur def __repr__(self): m = "PointColoré({},{},{})" return m.format(self.x, self.y,\ self.col) def ChangerCouleur(self, nvcouleur): self.col = nvcouleur class dict_str_(dict): def __init__(self): dict.__init__(self) def __setitem__(self, key, value): assert isinstance(key, str) super().__setitem__(key, value) def update(self, other): for key in other: if not isinstance(key, str): raise \ Exception(" Erreur!!!") super().update(other)</pre>