

Classes i objectes

Tecnologia de la Programació

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
<http://itic.cat>

26 de febrer de 2013



- 1 Programació orientada a objectes
- 2 Tipus definits pel programador
- 3 Objectes i atributs
- 4 Mètodes
- 5 Constructors
- 6 Modificadors
- 7 Doctests aplicats a classes
- 8 Per a la setmana vinent ...



Programació orientada a objectes

- OOP \equiv Paradigma de programació centrat en el concepte d'**objecte**.
- Un objecte representa un element concret del domini d'un programa i:
 - Posseeix un conjunt d'**atributs**.
 - Es manipula a través d'un conjunt de **mètodes**.
- Tot i ser conegut des dels 60's del s. XX, no és fins al voltant dels 80's que s'estén el seu ús.
- Molts dels llenguatges moderns adopten aquest paradigma. En particular: Modula3, Smalltalk, Java, C++ i Python.
- El paradigma és molt ric i transcendeix el llenguatge de programació. Abasta aspectes com ara la metodologia de treball, el test o el disseny.



Què és el tipus? I

- El **tipus** és la propietat d'un objecte que indica:
 - Quins valors (dades) pot emmagatzemar.
 - Quines operacions se li poden aplicar (a les dades que conté).
- Quan diem que un objecte és de tipus **str**, per exemple, significa que:
 - Pot emmagatzemar cadenes de caràcters.
 - Se li poden aplicar un conjunt determinat d'operacions entre les quals: **len**, **count**, **in**, etc.



Què és el tipus? II

Observeu algunes de les operacions del tipus **str**:

```
➤ s = "Piromania"
➤ print len(s)
9
➤ print 'r' in s
True
➤ print s[3:5]
om
➤ print s*2
PiromaniaPiromania
➤ type(s)
<type 'str'>
```



Exercici

- El projecte de programació d'INF:
 - 1 En quin domini es desenvolupa?
 - 2 Quines són les principals entitats amb que treballa?
 - 3 Quines són les seves operacions principals?
 - 4 Com es representen aquestes entitats?



Definició de nous tipus (classes) I

- Sovint és convenient disposar de nous tipus per **representar** elements del domini d'un programa més còmodament.
- Per exemple, podria ser interessant per a un cert programa representar el concepte «punt» a \mathbb{R}^2 .
- Podem definir nous tipus i «ampliar» el llenguatge amb la construcció **class**.

```
class Point(object):
    pass
```

- Compte amb **object**, que en molts llibre i apunts no s'usa!!



Definició de nous tipus (classes) II

- Una vegada definida la classe, podem crear objectes d'aquesta classe. Això ho anomenem **instanciar** la classe:

```
➤ p = Point()
➤ type(p)
<class '__main__.Point'>
```

- **p** és el nom d'un **objecte instància** de la **classe** Point. Sovint en diem només **instància**.
- A una instància se li poden afegir i esborrar **atributs**, que són dades que formen part de l'objecte:

```
➤ p.x = 10.0
➤ p.y = 5.0
```



Objectes i atributs

- Els objectes de classes definides pel programador són «ciutadans de ple dret». Per tant, tot el que coneixíem sobre objectes s'aplica:

```
>>> p = Point()
>>> p.x = 2.0
>>> p.y = 3.0
>>> q = p
>>> l = [1, 2, 'a', p]
>>> x = p.x + 2 * p.y
>>> print x
8.0

>>> q.x = 3.0
>>> print p.x
3.0
>>> print p
<__main__.Point object at 0xb751270c>
```

- L'assignació té la mateixa semàntica que en el cas de llistes. Recordi's la diferència entre assignar enters o llistes...
- L'operació de **print** és «estranya».



Foto!!

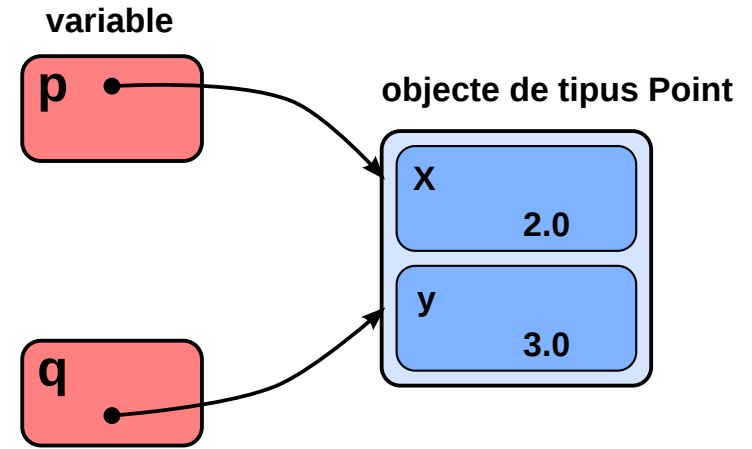


Figura: Una instància de classe Point i dues variables àlies una de l'altra



Mètodes I

- Una classe pot associar operacions als objectes que instància. Aquestes operacions s'anomenen **mètodes**.

```
class Point(object):
    def distance_to_origin(self):
        return math.sqrt(self.x*self.x + self.y*self.y)
```

- `self` representa l'objecte sobre el qual s'aplica el mètode. Ha de ser el primer paràmetre dels mètodes.
- Ara, els punts tenen operacions, com succeeix pels tipus predefinits:

```
>>> p = Point()
>>> p.x = 4.0 ; p.y = 4.0
>>> print p.distance_to_origin() # self és p
5.6568542494923806
```

- La crida d'un mètode es fa **qualificant** l'objecte al que s'aplica. Es comporta com una crida corrent.



Mètodes II

- Els mètodes poden tenir paràmetres com passava en el cas de les funcions:

```
class Point(object):
    def distance_to_origin(self):
        return math.sqrt(self.x*self.x + self.y*self.y)

    def is_inside_circle(self, r):
        """ True iff point inside radius r circle """
        return self.distance_to_origin() < r
```

- Noteu com es crida un mètode des de l'implementació d'un altre.



Mètodes III

- Amb la nova implementació de Point es poden fer càlculs com aquests a l'interpret de Python:

```
➤ p = Point()
➤ p.x = 2.0; p.y = 2.0
➤ print p.distance_to_origin()
2.828427125
➤ print p.is_inside_circle(1)
False
➤ print p.is_inside_circle(3)
True
```



Constructors I

- Cada vegada que s'instancia un objecte d'una classe cal crear els atributs escaients. Altrament els mètodes, que assumeixen la seva existència, farien fallida.
- Aquest procés és pesat i procliu a produir errors.
- Existeix un mètode especial, amb un nom fix, que en cas d'existir s'executa automàticament en **temps d'instanciació** de l'objecte.
- Aquest mètode de nom `__init__()` s'anomena **inicialitzador**.



Constructors II

- A l'exemple del punt:

```
class Point(object):
    def __init__(self, x, y):
        self.x = x ; self.y = y

    def distance_to_origin(self):
        return math.sqrt(self.x*self.x + self.y*self.y)

    def is_inside_circle(self, r):
        return self.distance_to_origin() < r
```



Constructors III

- L'ús de la classe Point és ara molt més còmode i segur:

```
➤ p = Point(1.0, 1.0)
➤ print p.distance_to_origin()
1.4142
➤ print p.is_inside_circle(2.0)
True
➤ print Point(2.0,2.0).is_inside_circle(2.0)
False
```



Modificadors I

- Els mètodes poden modificar l'objecte:

```
class Point(object):
    ....
    def h_move(self, d):
        self.x += d
```

- Fixem-nos amb les conseqüències:

```
>>> p = Point(2.0, 2.0)
>>> p.h_move(-2.0)
>>> print p.distance_to_origin()
2.0
```

- h_move() és un **modificador**.
- Amb aquest canvi, la classe Point ha passat de ser **immutable** a ser **mutable**.



Finalment el Point

```
class Point(object):
    """ A point on an euclidean plane """

    def __init__(self, x, y):
        self.x = x ; self.y = y

    def distance_to_origin(self):
        return math.sqrt(self.x * self.x + self.y * self.y)

    def is_inside_circle(self, r):
        return self.distance_to_origin() < r

    def h_move(self, d):
        self.x += d
```



Exercici

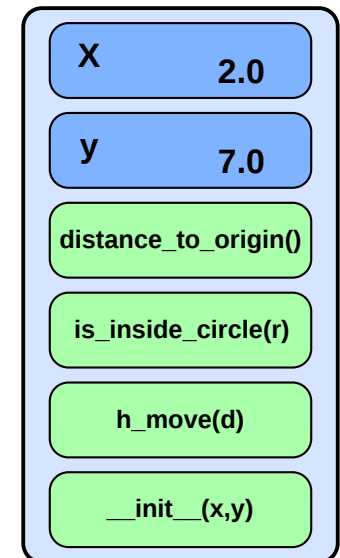
Amplieu la classe Point. Afegiu mètodes per a:

- 1 Desplaçar un punt verticalment.
- 2 Calcular l'origen.
- 3 Calcular la distància entre dos punts.
Després **refactoritzeu** el mètode distance_to_origin() per aprofitar aquest mètode més general.
- 4 Calcular el punt mig entre dos punts.
- 5 Donat un punt p , un punt q i un real $0 \leq r \leq 1$ desplaçar p cap a q en la proporció indicada per r .



Foto!!

- Instància de la classe Point.
- Totes les instàncies idèntica estructura.
- Instància conté atributs i mètodes.



Doctests sobre classes I

- Com en el cas de les funcions, les classes i els seus mètodes també poden testejar-se usant doctests.
- Hi ha dos espais preferents on afegir els doctests:
 - 1 En el docstring de la classe: quan els tests afecten més d'un mètode.
 - 2 En el docstring del mètode: quan els tests ho són específicament d'un mètode.



Doctests sobre classes II

```
class Point(object):
    """
    A point on an euclidean plane
    >>> p = Point(2,2); p.h_move(2); p.h_move(-2)
    >>> p.x
    2
    >>> p.y
    2
    >>> p = Point(0,0); p.h_move(2)
    >>> p.distance_to_origin()
    2.0
    """
    def is_inside_circle(self, r):
        """
        >>> Point(1,0).is_inside_circle(1)
        True
        >>> Point(0,2).is_inside_circle(1)
        False
        """
        return self.distance_to_origin() < r
```



Feina per aquesta setmana

- 1 Estudi de la teoria. A partir de la referència principal i les transparències. Inclou provar els conceptes en el computador i fer els exercicis de les transparències1.
- 2 Confecció d'un xuletari i un glossari del tema.
- 3 Solució dels problemes del tema.
- 4 Solució del problema especial, que té com objectiu aconseguir la solució més senzilla i entenedora possible.

