Rudimenti di Python

senza rodimenti di Python

Contatti

_ Enrico Marino

_ Federico Spini

_ mail:
  _ `(marino|spini)@dia.uniroma3.it`

_ sito:
  _ `dia.uniroma3.it/~(marino|spini)/python`

Riferimenti

_ sito ufficiale: http://python.org/

_ tutorial ufficiale: http://docs.python.org/tutorial/

_ documentazione: http://docs.python.org/index.html

Classi

_ Le classi si definiscono tramite il comando `class`
  _ seguito dal nome della classe
  _ seguito dal nome della super classe tra parentesi tonde

```
class NomeClasse(object):
    pass
```

  _ `object` è la classe di più basso livello
    _ tutte le classi ereditano da `object`
  _ `pass` non fa nulla
    _ indica un blocco vuoto
  _ il tipo di una classe è `type`

```
type(NomeClasse)
>>> <type 'type'>
```

Classi
Instanziare una classe

_ Le classi si instanziano chiamando il costruttore

```
class Punto(object):
    pass

p = Punto()

type(p)
>>> <type 'instance'>
```

Classi
Attributi

- Le classi in Python sono dinamiche
  - possono cambiare la loro struttura durante l'esecuzione
  - permettono l'assegnazione dinamica degli attributi

```
p = Punto()
p.x = 0.5
p.y = 1.2
```

  - L'assegnazione dinamica degli attributi
    - permette di creare classi molto flessibili
    - richiede maggiore attenzione nella codifica per evitare bug
    - dovrebbe avvenire solo nel costruttore della classe

Classi
Metodi

- I metodi si definiscono con la stessa sintassi delle funzioni
  - hanno un parametro obbligatorio
    - chiamato per convenzione `self`
    - contenente l'istanza della classe che ha chiamato il metodo

```
class Punto(object):
    ...

    def trasla(self, x, y):
        self.x += x
        self.y += y
```

Classi
Metodi speciali

_ Costruttore `__init__`

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

p = Punto(10,20)
p.x
>>> 10
p.y
>>> 20
```

Classi
Metodi speciali

_ Rappresentazione `__repr__`

```
class Punto(object):
    ...
    def __repr__(self):
        return "Punto(", self.x, ",", self.y, ")"

p = Punto(10,20)
p
>>> Punto(10,20)
```

Classi
Overload degli operatori

_ Python permette l'overload degli operatori `+`, `-`, `*`, `/`
  _ tramite i metodi speciali __add__, __sub__, __mul__, __div__

```
class Punto(object):
    ...
    def __add__(self, p):
        return Punto(self.x + p.x, self.y + p.y)
    def __sub__(self, p):
        return Punto(self.x - p.x, self.y - p.y)
```

  _ il metodo `__add__` della classe Punto viene chiamato
    _ quando viene sommato un valore a una istanza Punto
  _ il metodo `__sub__` della classe Punto viene chiamato
    _ quando viene sottratto un valore a una istanza Punto

Classi
Overload degli operatori

_ I parametri per i metodi speciali per gli operatori sono due:
- _ il primo (self) è obbligatorio per tutti i metodi di una classe
  - _ conterrà l'istanza della classe a sinistra dell'operatore
- _ il secondo conterrà l'istanza della classe a destra dell'operatore

```
class Punto(object):
    ...
    def __add__(self, p):
        return Punto(self.x + p.x, self.y + p.y)

p1 = Punto(10,20)
p2 = Punto(1,2)
p1 + p2
>>>Punto(11,22)
```

Classi
Introspezione

_ Python permette l'introspezione
  _ ricavare informazioni sui tipi durante l'esecuzione del codice
    _ informazioni su attributi, metodi, parametri, variabili, …

_ `type` è la principale funzione di introspezione
  _ restituisce il tipo della variabile passata come parametro

```
type(1)       # <type 'int'>
type(1.)      # <type 'float'>
type("ciao")  # <type 'str'>
type(Punto)   # <type 'type'>
type(p)       # <type 'instance'>
```

Classi
Introspezione

- L'introspezione è molto utile nella definizione dei metodi
  - per definire il comportamento in base al tipo di parametro passato

```
def __add__(self, p):
    if type(p) is Punto:
        return Punto(self.x + p.x, self.y + p.y)
    if type(p) is int:
        return Punto(self.x + p, self.y + p)
    return NotImplemented
```

  - `is` permette di verificare il tipo restituito da `type`
  - `NotImplemented` indica che l'operazione non è supportata

Classi
Introspezione

_ Usando l'introspezione nel metodo `__add__` nell'esempio
  _ è possibile sommare un oggetto Punto con un altro oggetto Punto
  _ è possibile sommare un oggetto Punto con un intero
  _ non è possibile sommare un oggetto Punto con altri oggetti

```
Punto(1, 2) + Punto(10,10)
>>>Punto(11,12)

Punto(1,2) + 10
>>>Punto(11,12)

Punto(1,2) + 'ciao'
>>>TypeError: unsupported operand type(s) for +:
   'Punto' and 'str'
```

Classi
Introspezione

- Usando l'introspezione nel metodo `__add__` nell'esempio
  - non è possibile sommare un intero con un oggetto Punto

```
10 + Punto(1,2)
>>>TypeError: unsupported operand type(s) for +:
   'int' and 'Punto'
```

  - Quando Python incontra `10 + Punto(1,2)`
    - chiama il metodo `__add__` del primo operando (`int`)
  - Se il primo operando (`int`) non sa addizionare il secondo (`Punto`)
    - ritorna il tipo `NotImplemented`
  - Se il primo operando ritorna `NotImplemented`
    - Python verifica se il secondo operando implementa il metodo `__radd__`
  - Se il secondo operando implementa `__radd__`
    - chiama `__radd__` passando gli stessi parametri del metodo `__add__`

Classi
Introspezione

_ Usando il metodo `__radd__` nell'esempio
  _ è possibile sommare un intero con un oggetto Punto

```
def __add__(self, p):
    if type(p) is Punto:
        return Punto(self.x + p.x, self.y + p.y)
    if type(p) is int:
        return Punto(self.x + p, self.y + p)
    return NotImplemented

__radd__ = __add__

10 + Punto(1,2)
>>>Punto(11,12)
```

Classi
Ereditarietà

_ Python permette l'ereditarietà singola

```
class Punto(object):
    #...

class Punto3D(Punto):
    #...
```

_ Ogni classe deve discendere da `object`
  _ se non viene specificato Python crea una classe old-style
    _ usata nella vecchia gestione dei tipi di Python (in disuso)
    _ mentenuta per retro-compatibilità

Classi
Ereditarietà

_ Una classe può richiamare i metodi della classe da cui deriva

```
class Punto3D(Punto):
    def __init__(self, x, y, z):
        Punto.__init__(self, x, y)
        self.z = z

    def __add__(self, p):
        if type(p) is Punto3d:
            return Punto3D(self.x + p.x,
                self.y + p.y, self.z + p.z)
        if type(valore) is Punto:
            return Punto3D(self.x + p.x, self.y + p.y, self.z)
        if type(valore) is int:
            return Punto3D(self.x + p, self.y + p, self.z + p)
        return NotImplemented

    __radd__ = __add__
```

Classi
Documentazione

_ Si possono documentare i metodi
  _ specificando il testo di documentazione tra tre doppi apici (""")
  _ all'inizio del blocco

```
def trasla(self, dx, dy):
    """
    Trasla il punto di (dx,dy)
    """
    self.x += dx
    self.y += dy

>>> help(Punto.trasla)
Help on method trasla in module __main__:

trasla(self, dx, dy) unbound __main__.Punto method
    Trasla il punto di (dx,dy)
```