Introduzione alla programmazione in Python: Parte III

corso tenuto da:

Francesco Grigoli

organizzato da:

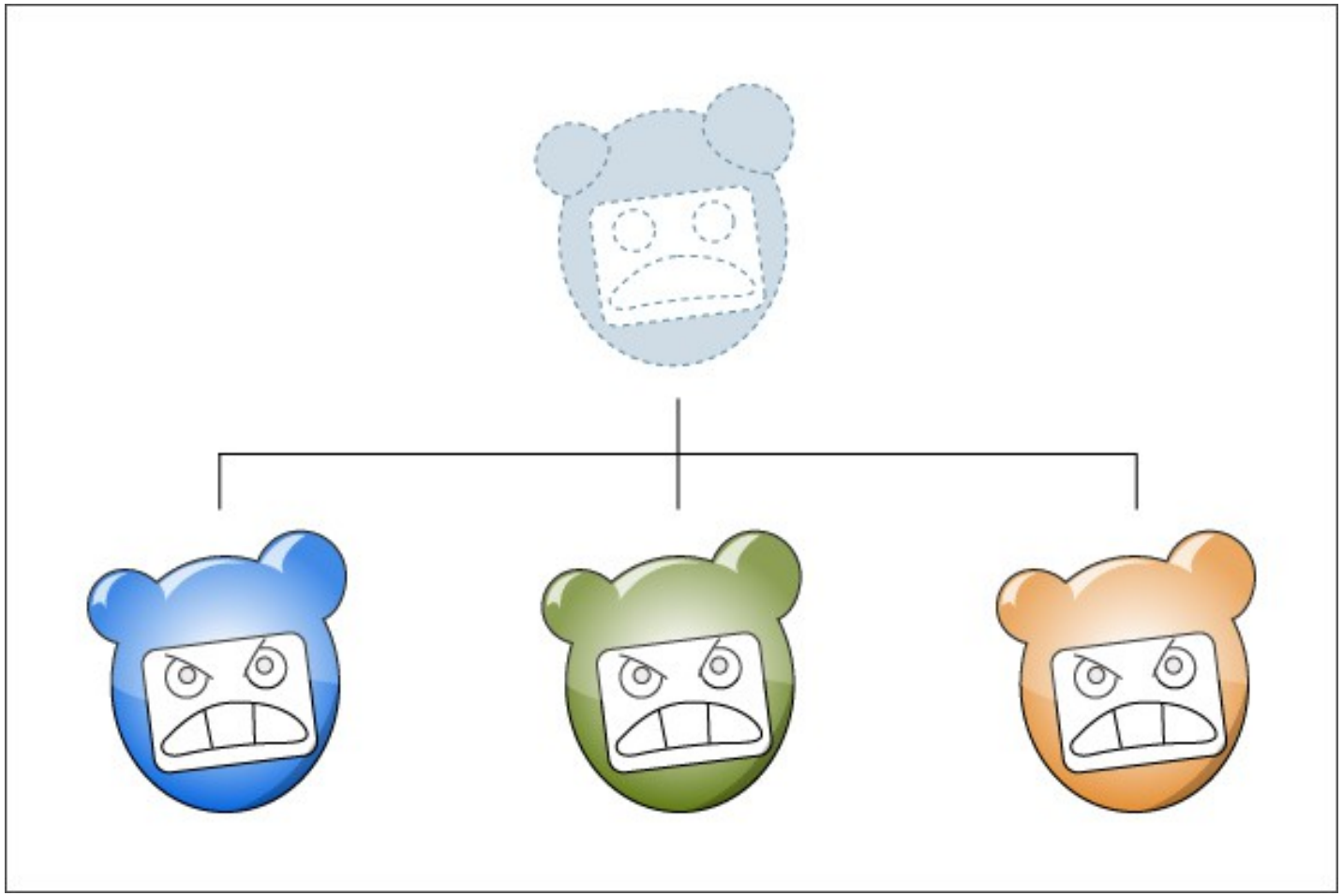
Associazione Next
Studio Mirabilia

con la collaborazione di:

ANFE, Sportello multifunzionale di Bagheria

Sommario

Introduzione all'OOP



La programmazione orientata agli oggetti

La programmazione orientata agli oggetti è un insieme di concetti che aiutano a strutturare un (gran) programma in un modo “naturale”.

L'OOP si basa su classi, attributi e metodi

- Una classe è un'entità che raggruppa le qualità (*attributi*) e i comportamenti (*metodi*) di un determinato oggetto.
- Nei linguaggi procedurali i dati sono spesso separati dalle funzioni, nella programmazione orientata agli oggetti dati e funzioni sono strettamente legati.

Classi e oggetti: Esempio

```
class lavoratore:
  'Classe comune per tutti i lavoratori'
  contatore_lavoratore = 0

  def __init__(self, nome, salario):
    self.nome = nome
    self.salario = salario
    lavoratore.contatore_lavoratore += 1

  def numero_impiegati(self):
   print "Impiegati Totali  %d" % . lavoratore.contatore_lavoratore

  def visualizza_lavoratore(self):
    print "Nome : ", self.nome,  ", Salario: ", self.salario
```

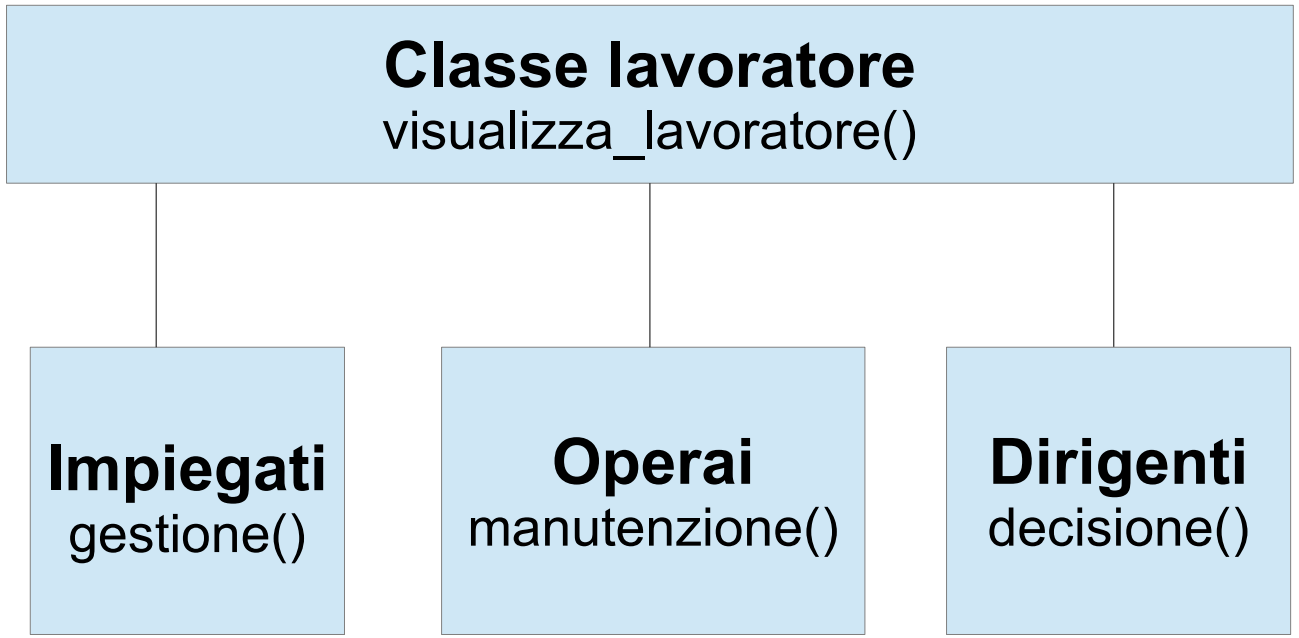
Classi e oggetti

- La parola chiave *class* introduce una classe.
- Per creare un esempio di oggetto, usiamo:

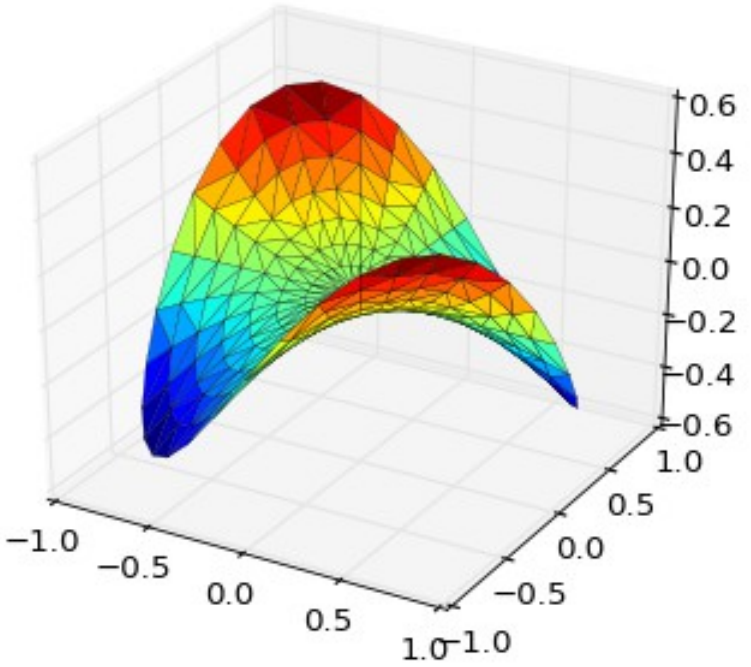
  Imp1 = lavoratore (’Francesco’, '1,000,000')
- Il metodo __init__() viene citato quando viene creato un esempio di oggetto. (inizializza i valori).
- I metodi necessitano di un riferimento (self) relativo all'istanza come primo argomento.
- I metodi funzionano nel seguente modo:

  Imp1.visualizza_lavoratore()

Sottoclassi ed Ereditarietà



Calcolo scientifico: (Numpy, Scipy e Matplotlib)



Creazione di array con NumPy

Conversione tra la lista di Python e gli array di NumPy:

```
>>> a = numpy.array([1,2,3], dtype=numpy.float64)
>>> a
array([ 1., 2., 3.])
>>> a.tolist()
[1.0, 2.0, 3.0]
```

Creazione di array di zeri e uno:

```
>>> numpy.zeros((2,3), dtype=numpy.int32)
array([[0, 0, 0],
       [0, 0, 0]])
>>> numpy.ones(2)
array([ 1., 1.])
>>> numpy.empty((2,2))
array([[ 76.14698799, 167.7812807 ],
       [ 167.78128071, 95.2773591 ]])
```

Creazioni di array NumPy

Creazione di una serie di numeri interi, data dal passo:

```
>>> numpy.arange(-4, 5, 2)
Array([-4, -2, 0, 2, 4])
```

Il numero degli elementi potrebbe non essere controllabile quando si usa questo per creare intervalli di numeri a virgola mobile (dovuta alla precisione di virgola mobile).

Create a range of floats, given number of elements:

```
>>> numpy.linspace(10, 20, 5)
array([ 10. , 12.5, 15. , 17.5, 20. ])
```

Attributi array NumPy visti da Python

- a.ndim: Numero di assi (dimensioni), rank
- a.shape: Tuple with extent in each dimension
- a.size: numero totale di elementi
- a.dtype: Data type
- a.itemsize: Size in bytes of each element
- a.data: Buffer containing actual data

Operatori matematici binari

Questi operatori lavorano elemento per elemento sugli array NumPy:

a + b add(a,b) a * b multiply(a,b)

a - b subtract(a,b) a / b divide(a,b)

a % b remainder(a,b) a ** b power(a,b)

```
>>> from numpy import * # <- dangerous
>>> a = arange(7,4,-1)
>>> b = arange(16,19)
>>> c = a + b #          <- same as: c = add(a,b)
>>> c
array([23, 23, 23])
>>> c / 4.6 #           <- scalar operands are ok
array([ 5., 5., 5.])
>>> add(a,b,a) #        <- in place add: a += b
Array([23, 23, 23])
```

Operatori di confronto e logici sono anche disponibili.

Funzioni matematiche

Funzioni matematiche NumPy lavorano elemento per elemento sugli array:

sin(x), sinh(x), arcsin(x), ..., arctan2(x,y)

exp(x), log(x), log10(x), sqrt(x)

absolute(x), conjugate(x)

ceil(x), floor(x), fabs(x)

hypot(x,y), fmod(x,y)

maximum(x,y), minimum(x,y)

```
from numpy import *
from pylab import *
x,y = meshgrid(arange(100)/5.,arange(100)/5.)
z = sin(x)*cos(y) *sin(0.3*x) *cos(0.3*y)
imshow(z)
show()
```

Indexing

- NumPy arrays use C storage order by default.
- Fastest index comes last.

```
>>> a = arange(16)
>>> a.resize(4,4)


>>> a[1,2]
6
```

0	1	2	3
4	5	**6**	7
8	9	10	11
12	13	14	15

Slicing

- Gli operatori slicing riportano un **view** su un array.
- Questo **view** è esso stesso un array NumPy.
- Ma questo opera su i dati dell' array originale.

```
>>> a[1,:]
array([4, 5, 6, 7])

>>> b = a[1,:]
>>> b[:] = 0
>>> a
array([[ 0, 1, 2, 3], [ 0, 0, 0, 0], [ 8, 9, 10, 11], [12, 13, 14, 15]])
```

0	1	2	3
4	**5**	**6**	**7**
8	9	10	11
12	13	14	15

Slicing

- Lo slicing supporta range e strides.
- Strides posso essere negativi to get a reversed view.

```
>>> a[::2,1::2]
array([[ 1, 3], [ 9, 11]])
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

```
>>> a[1:3,1:3]
array([[ 5, 6], [ 9, 10]])
```

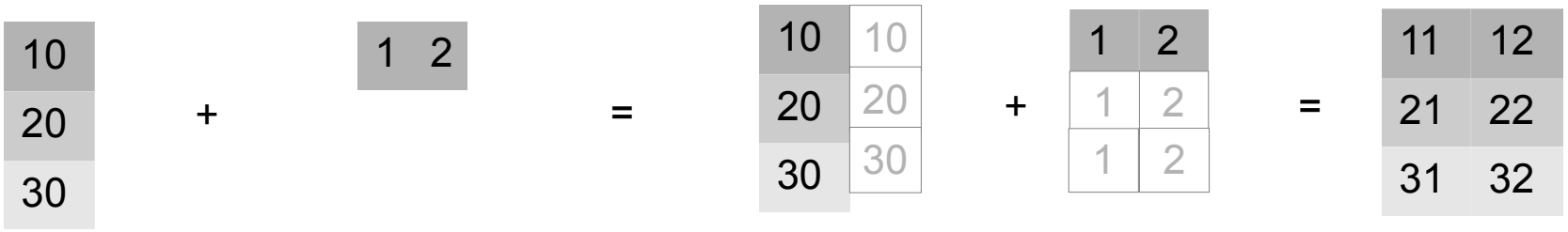
```
>>> a[1:3,2:0:-1]
array([[ 6, 5], [10, 9]])
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Trasmissione array

- Se possibile, gli array sono tese a dare possibile operazione desiderata.

```
>>> a = array([[1,2]])              # a row vector
>>> b = array([[10],[20],[30]])     # a column vector
>>> a+b
array([[11, 12], [21, 22], [31, 32]])
```



Trasmissione array e newaxis

- Newaxis è uno speciale indice che inserisce a dummy dimension.
- Ogni newaxis incrementa the arrays dimensionallity by 1.

```
>>> x = arange(100)[newaxis,:]/5. # a row vector
>>> y = arange(100)[:,newaxis]/5. # a column vector
>>> z = sin(x)*cos(y)

>>> from pylab import *
>>> imshow(z)
>>> show()
```

Metodi riduzione: reduce, accumulate

Gli operatori binari hanno metodi che lavorano sugli arrays.

- op.reduce(a,axis=0)
- op.accumulate(a,axis=0)

```
>>> a
array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
>>> add.reduce(a)                  # sum of each column
array([ 9, 12, 15])
>>> add.reduce(a, axis=1)          # sum of each row
array([ 3, 12, 21])
>>> add.accumulate(a, axis=1)
array([[ 0, 1, 3], [ 3, 7, 12], [ 6, 13, 21]])
```

Function method: outer

Il metodo outer fa un array 2D con il risultato op su tutte le combinazioni possibili degli elementi di a e di b.

- op.outer(a,b)

```
>>> add.outer(arange(3), arange(10))
array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
       [ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]])
```

Funzione array: where

y = where(condition, false, true)

```
>>> a = arange(10)

>>> a
Array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> where(a>=7, 99, a)
array([ 0, 1, 2, 3, 4, 5, 6, 99, 99, 99])
```

Ulteriori letture

- A. B. Downey, Think Python, O'Reilly
- M. Lutz, Learning Python, O'Reilly
- W. McKinney, Python for data analysis, O'Reilly
- www.numpy.org

Ringraziamenti

Desidero ringraziare Sebastian Heimann per avermi fornito le slides da cui trarre spunto e Lidia Di Blasi per averle tradotte. Ringrazio inoltre Giuseppe Gallo per aver organizzato il corso e il team dello sportello multifunzionale ANFE di Bagheria per aver messo a disposizione i loro locali.

…. Infine, grazie a tutti voi per aver partecipato...