

[FIM] FONDAMENTI DI INFORMATICA per medicina e chirurgia high tech

L02: Classi e Oggetti

Dott. Giorgio De Magistris

demagistris@diag.uniroma1.it

CORSO DI LAUREA IN MEDICINA E CHIRURGIA HIGH TECH



SAPIENZA
UNIVERSITÀ DI ROMA

I3S

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA

DIAG

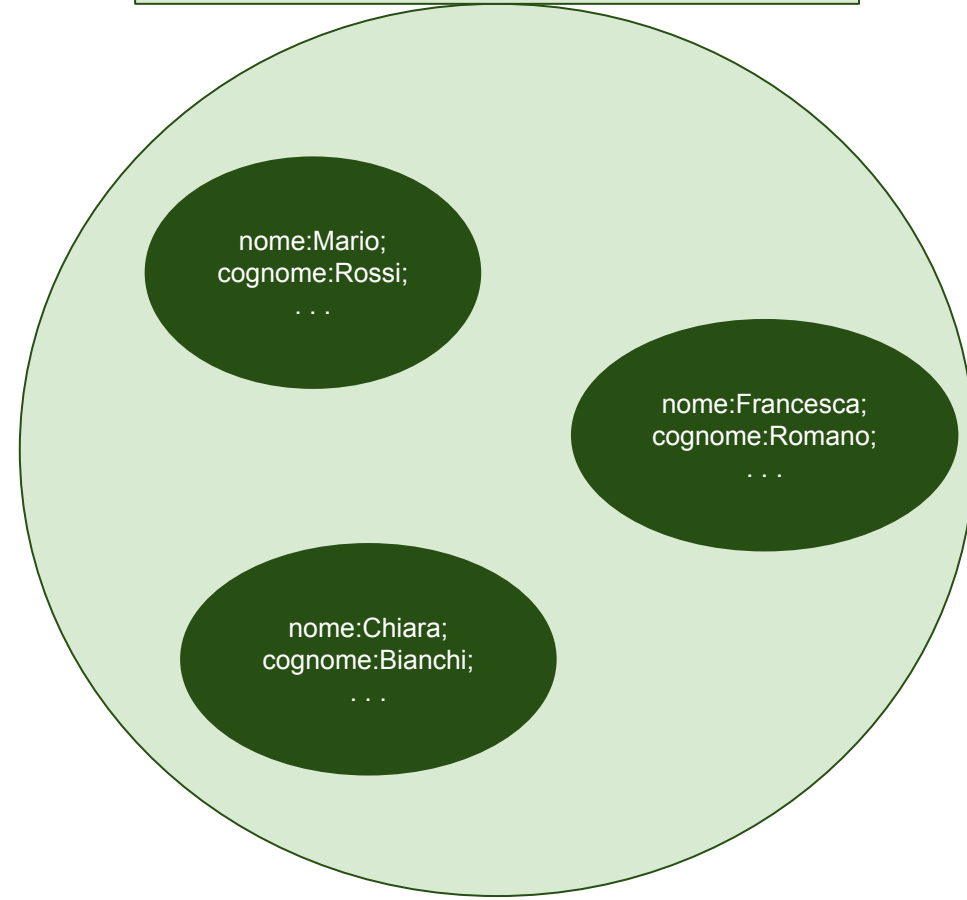
DIPARTIMENTO DI INGEGNERIA INFORMATICA, AUTOMATICA E GESTIONALE

TUTTI I DIRITTI RELATIVI AL PRESENTE MATERIALE DIDATTICO ED AL SUO CONTENUTO SONO RISERVATI A SAPIENZA E AI SUOI AUTORI (O DOCENTI CHE LO HANNO PRODOTTO). È CONSENTITO L'USO PERSONALE DELLO STESSO DA PARTE DELLO STUDENTE A FINI DI STUDIO. NE È VIETATA NEL MODO PIÙ ASSOLUTO LA DIFFUSIONE, DUPLICAZIONE, CESSIONE, TRASMISSIONE, DISTRIBUZIONE A TERZI O AL PUBBLICO PENA LE SANZIONI APPLICABILI PER LEGGE

Classi e Oggetti

- Le classi permettono di mettere insieme dati e funzionalità
- Creare una classe significa creare un nuovo **tipo** di oggetto
- Istanziare un oggetto significa creare un **oggetto** di quel tipo
- Si può pensare alla classe come ad un insieme e alle istanze della classe come gli elementi dell'insieme
- Definire una classe significa specificare delle proprietà comuni a tutti gli elementi dell'insieme

Classe Cittadino	
nome : str	data nascita : date
cognome : str	comune nascita : str
sex : str	
calcola_cod_fisc () -> str	



Classi e Oggetti

- Per creare una classe bisogna usare la parola chiave **class** seguita dal nome della classe
- Per fare in modo che gli oggetti della classe vengano inizializzati con degli attributi bisogna definire la funzione **__init__** (costruttore), che prende come primo parametro l'oggetto stesso (**self**)
- Per inizializzare attributi dell'oggetto bisogna scrivere:
self.nome_attr = valore

```
class Cittadino:
    """
    Classe per rappresentare un cittadino
    italiano, la classe contiene tutte le
    informazioni necessarie per calcolare
    il codice fiscale
    """
    def __init__(self,
                  nome:str, cognome:str,
                  datan:date,
                  comune:str,
                  sesso:str) -> None:

        self.nome = nome
        self.cognome = cognome
        self.datan = datan
        self.comune = comune
        self.sesso = sesso

    def cf(self) -> str:
        return self.nome + \
               self.cognome + \
               str(self.datan) + \
               str(self.comune) + \
               self.sesso
```

Classi e Oggetti

- Per Istanziare un oggetto di una classe bisogna “chiamare” la classe (come fosse una funzione) con i parametri del costruttore
- Adesso la variabile `c1` contiene un riferimento ad un oggetto di tipo `Cittadino`
- Per accedere agli attributi dell'oggetto basta scrivere **`c1.nome_attributo`**

```
c1 = Cittadino("Mario", "Rossi", date(1990, 7, 2), "Roma", "M")  
  
print(c1.nome)  
  
print(c1.cognome)  
  
print(type(c1))
```

Classi e Oggetti

- Una volta creato un oggetto posso modificare gli attributi inizializzati nel costruttore

```
c1 = Cittadino("Mario", "Rossi", date(1990, 7, 2), "Roma", "M")  
  
c1.datan = date(1995, 8, 10)  
print(c1.datan)
```

Classi e Oggetti

- Le uniche operazioni che possiamo eseguire su una classe sono:
 - Riferimento ad attributo (attributo dati o funzione)
 - **Istanziamento**
- L'unica operazione che possiamo eseguire su un'istanza è:
 - **Riferimento ad attributo** (attributo dati o metodo)

```
class Cittadino:
    """ Classe per rappresentare un cittadino """
    numero_istanze = 0
    def __init__(self,
                  nome:str, cognome:str,
                  datan:date,
                  comune:str,
                  sesso:str) -> None:

        Cittadino.numero_istanze += 1
        self.nome = nome
        self.cognome = cognome
        self.datan = datan
        self.comune = comune
        self.sesso = sesso

    # Viene chiamata dall'interprete quando
    # l'oggetto viene eliminato dal garbage
    # collector
    def __del__(self):
        Cittadino.numero_istanze -= 1

c1 = Cittadino("Mario", "Rossi", date(1990, 7, 2), "Roma", "M")
print(Cittadino.numero_istanze)
c1 = None
print(Cittadino.numero_istanze)
```

Definizione Classe

stringa di documentazione, viene salvata nell'attributo `__doc__` della classe `Vec2D`

`__init__` è la funzione che viene invocata per istanziare un oggetto. Può essere modificata per aggiungere attributi all'oggetto (nel nostro esempio le variabili `x` e `y`)

Istanziamento degli oggetti `p1` e `p2`. Per istanziare un oggetto basta chiamare con le parentesi la classe, implicitamente viene invocata la funzione `__init__`

```
class Vec2D():
    """
    Classe per rappresentare vettori nel piano
    """

    def __init__(self,x:float,y:float) -> None:
        self.x = x
        self.y = y

    # this is a method
    def length(self)->float:
        return math.sqrt( self.x**2 + self.y**2 )

if __name__=="__main__":

    print(Vec2D.__doc__)

    p1 = Vec2D(1.,3.)
    p2 = Vec2D(3.,4.)

    print(p1.length())
    print(p2.length())
```

?

Definizione Classe

stringa di documentazione, viene salvata nell'attributo `__doc__` della classe `Vec2D`

`__init__` è la funzione che viene invocata per istanziare un oggetto. Può essere modificata per aggiungere attributi all'oggetto (nel nostro esempio le variabili `x` e `y`)

Istanziamento degli oggetti `p1` e `p2`. Per istanziare un oggetto basta chiamare con le parentesi la classe, implicitamente viene invocata la funzione `__init__`

```
class Vec2D():
    """
    Classe per rappresentare vettori nel piano
    """

    def __init__(self,x:float,y:float) -> None:
        self.x = x
        self.y = y

    # this is a method
    def length(self)->float:
        return math.sqrt( self.x**2 + self.y**2 )

if __name__=="__main__":

    print(Vec2D.__doc__)

    p1 = Vec2D(1.,3.)
    p2 = Vec2D(3.,4.)

    print(p1.length())    3.16...
    print(p2.length())    5.0
```


Funzioni vs Metodi

- Un metodo è una funzione che “appartiene” all’oggetto
- Nella definizione prende sempre come primo parametro il riferimento all’oggetto stesso (per convenzione chiamato **self**)
- Quando un metodo viene invocato da un oggetto con **oggetto.nome_metodo(params)** l’interprete python passa come primo argomento il riferimento all’oggetto stesso

```
class Vec2D():

    def __init__(self,x:float,y:float) -> None:
        self.x = x
        self.y = y

    # this is a method
    def length(self)->float:
        return math.sqrt( self.x**2 + self.y**2 )

if __name__=="__main__":

    print(type(Point2D.length))

    p1 = Vec2D(3,2)
    print(type(p1.length))

    print(p1.length())

    print(Vec2D.length(p1))
```



Funzioni vs Metodi

- Un metodo è una funzione che “appartiene” all’oggetto
- Nella definizione prende sempre come primo parametro il riferimento all’oggetto stesso (per convenzione chiamato **self**)
- Quando un metodo viene invocato da un oggetto con **oggetto.nome_metodo(params)** l’interprete python passa come primo argomento il riferimento all’oggetto stesso

```
class Vec2D():

    def __init__(self,x:float,y:float) -> None:
        self.x = x
        self.y = y

    # this is a method
    def length(self)->float:
        return math.sqrt( self.x**2 + self.y**2 )

if __name__=="__main__":

    print(type(Vec2D.modulus))    <class 'function'>

    p1 = Vec2D(3,2)
    print(type(p1.length))       <class 'method'>

    print(p1.length())           3.605...

    print(Vec2D.length(p1))      3.605...

    Se chiamo la funzione length
    dall’oggetto ( metodo ) il primo
    parametro ( self ) è implicito
```

Operatori

- Gli operatori in python non sono altro che metodi dell'oggetto
- I tre frammenti di codice sono equivalenti
 - nel primo uso l'operatore +
 - nel secondo chiamo la funzione `__add__` definita nella classe `int`
 - nel terzo chiamo il metodo `__add__` dell'oggetto `a`

```
a = 2
b = 3

c = a + b

print(c)
```

```
a = 2
b = 3

c = int.__add__(2,3)

print(c)
```

```
a = 2
b = 3

c = a.__add__(b)

print(c)
```

Override degli Operatori

- Gli operatori, possono essere sovrascritti, per cambiare la loro implementazione
- Nella mia classe Vec2D voglio che l'operatore + sommi rispettivamente le coordinate x e y

```
class Vec2D():

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

    def length(self) -> float:
        return math.sqrt( self.x**2 + self.y**2 )


    def __add__(self, v:Vec2D) -> Vec2D:
        return Vec2D(self.x + v.x, self.y + v.y)

    def __str__(self) -> str:
        return "Vec.x = {}, Vec.y = {}".format(self.x,self.y)

v1 = Vec2D(2,3)
v2 = Vec2D(3,5)

v3 = v1 + v2

print(v3)
```



Override degli Operatori

- Gli operatori, possono essere sovrascritti, per cambiare la loro implementazione
- Nella mia classe Vec2D voglio che l'operatore + sommi rispettivamente le coordinate x e y

```
class Vec2D():

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

    def length(self) -> float:
        return math.sqrt( self.x**2 + self.y**2 )

    def __add__(self, v:Vec2D) -> Vec2D:
        return Vec2D(self.x + v.x, self.y + v.y)

    def __str__(self) -> str:
        return "Vec.x = {}, Vec.y = {}".format(self.x,self.y)

v1 = Vec2D(2,3)
v2 = Vec2D(3,5)

v3 = v1 + v2

print(v3)
```

Vec.x = 5, Vec.y = 8

Slides distribuite con Licenza Creative Commons (CC BY-NC-ND 4.0) Attribuzione - Non commerciale - Non opere derivate 4.0 Internazionale

PUOI CONDIVIDERLE ALLE SEGUENTI CONDIZIONI

(riprodurre, distribuire, comunicare o esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato)

Attribuzione*

Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.

Non Commerciale

Non puoi utilizzare il materiale per scopi commerciali.

Non opere derivate

Se remixi, trasformi il materiale o ti basi su di esso, non puoi distribuire il materiale così modificato.

Divieto di restrizioni aggiuntive

Non puoi applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici a questa licenza