

# Programming Lab

Lezione 4

*Gli oggetti in Python*

Stefano Alberto Russo

# Programmazione ad oggetti

E' un paradigma di programmazione. Cambia molto.

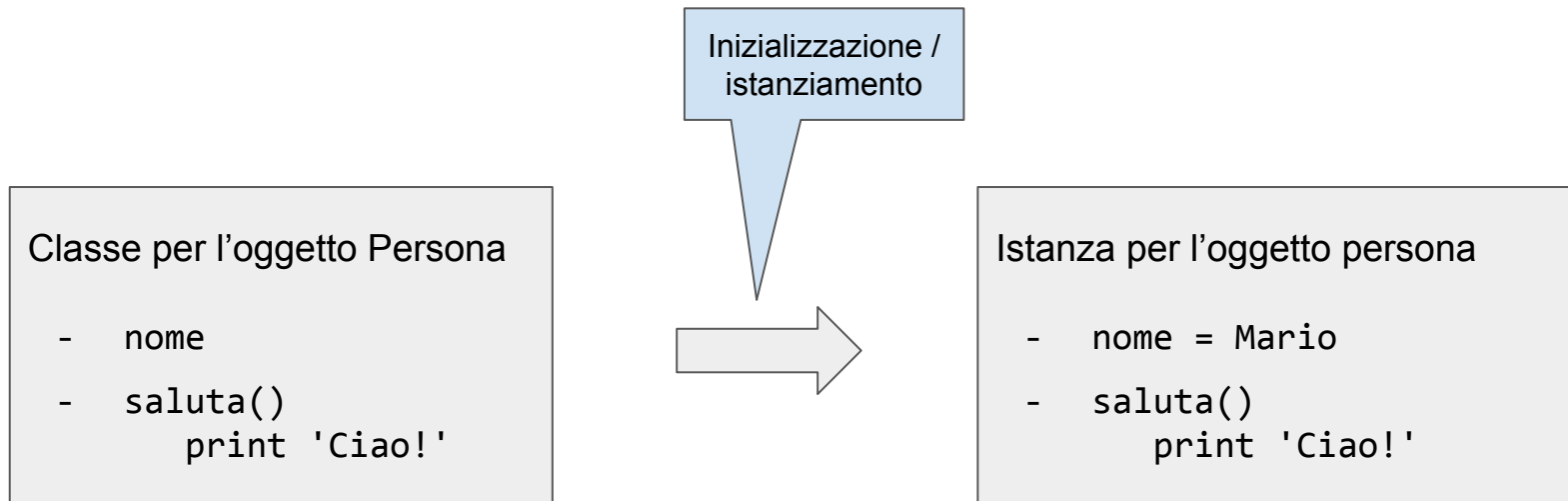
Gli **oggetti** sono definiti con le **classi**

- le funzioni negli oggetti/classi si chiamano **metodi**
- le variabili negli oggetti/classi si chiamano **attributi**

Una volta inizializzati diventano **istanze**

→ si parla infatti di **istanziare** un oggetto/classe

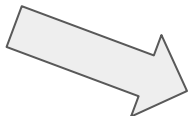
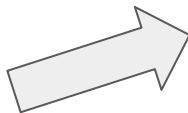
# Programmazione ad oggetti



# Programmazione ad oggetti

Classe per l'oggetto Persona

- nome
- saluta()  
    print 'Ciao!'



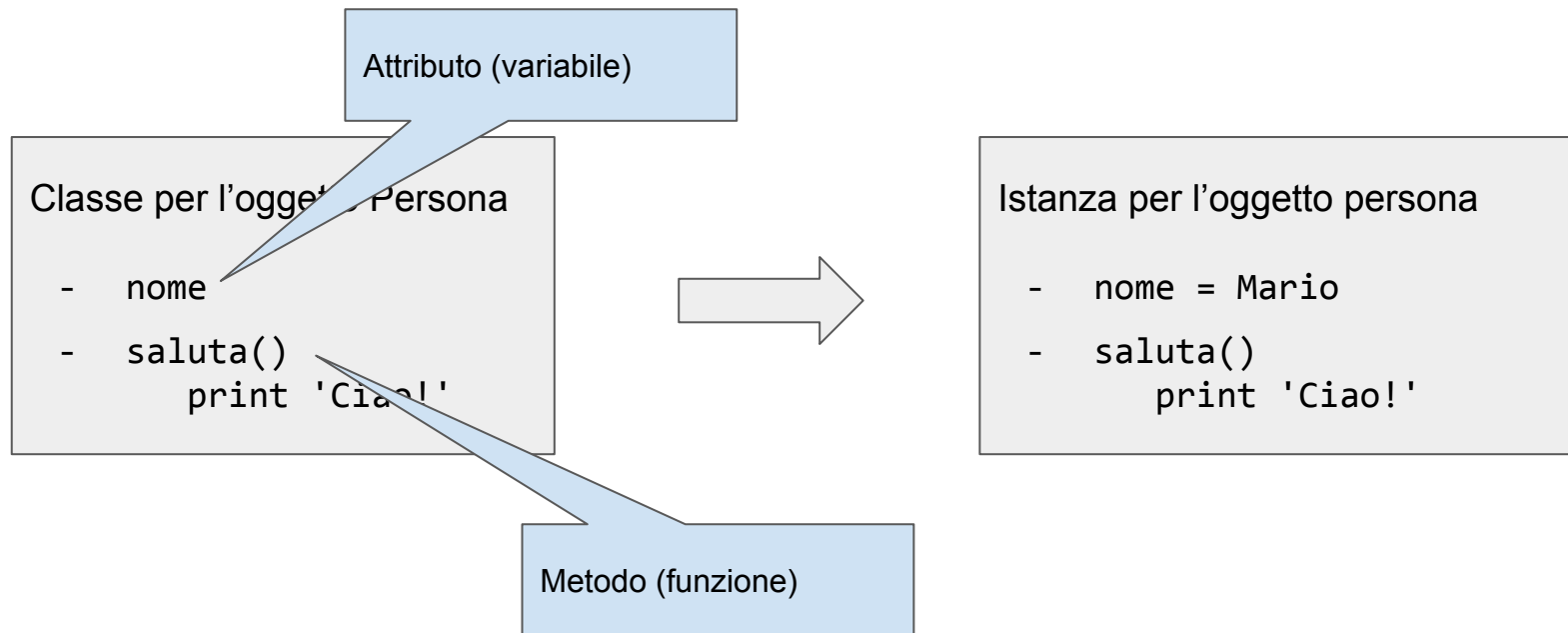
Istanza per l'oggetto persona

- nome = Mario
- saluta()  
    print 'Ciao!'

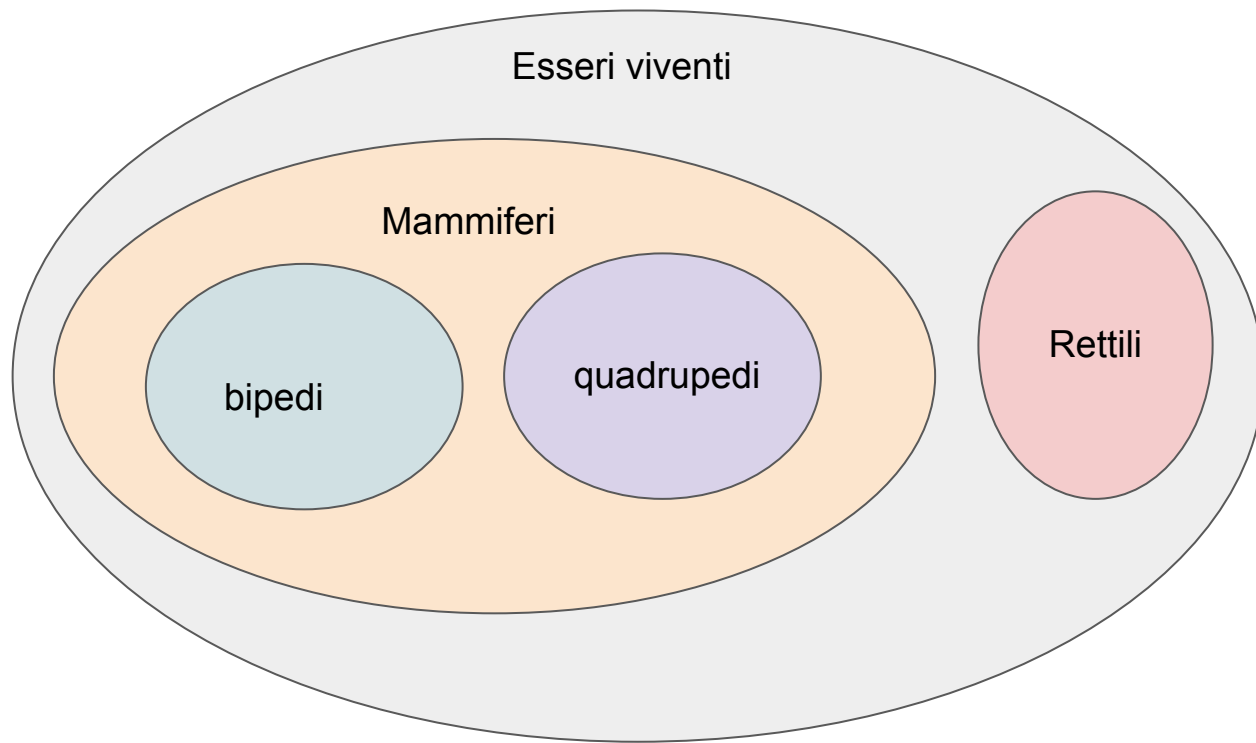
Istanza per l'oggetto persona

- nome = Lucia
- saluta()  
    print 'Ciao!'

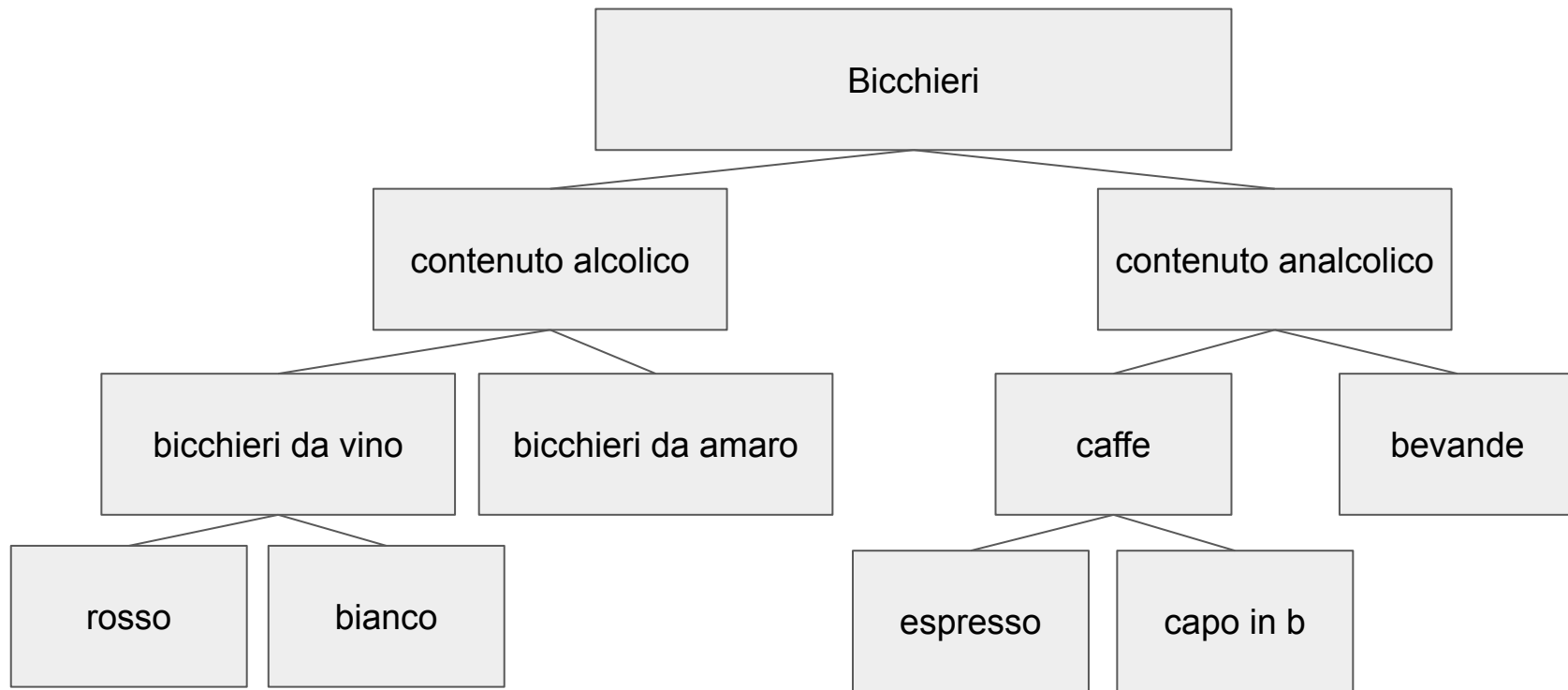
# Programmazione ad oggetti



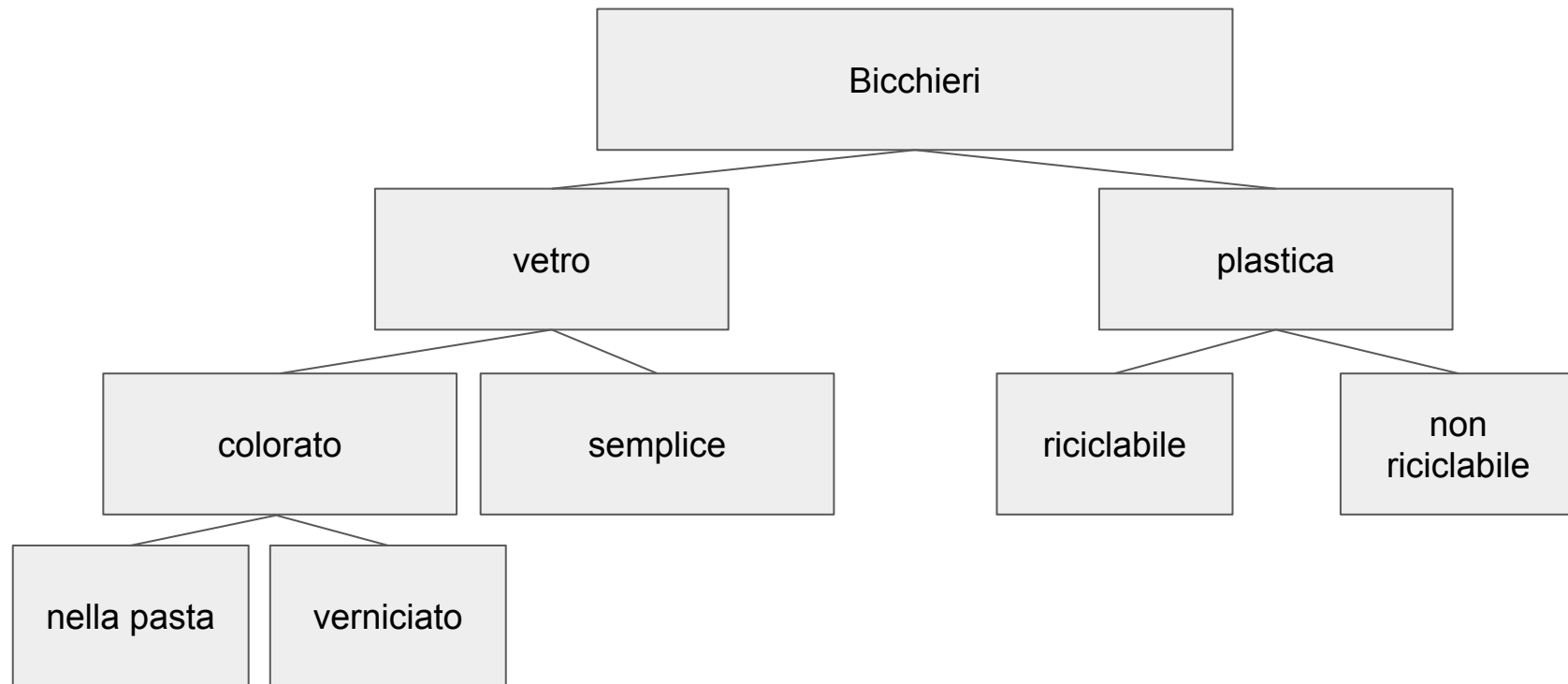
# Programmazione ad oggetti



# Programmazione ad oggetti



# Programmazione ad oggetti





# Programmazione ad oggetti

Gli oggetti si usano principalmente perchè:

- Permettono di rappresentare bene delle gerarchie (e sfruttare le caratteristiche in comune)
- Una volta istanziati, permettono di mantenere facilmente lo *stato* (senza diventare matti con strutture dati di appoggio)

# Convenzioni

In Python c'è una convenzione di stile ben precisa:

- caratteri **minuscoli** e **underscore** per le **variabili** e le **istanze** degli oggetti
- notazione **CamelCase** per il nome delle **classi**

Inoltre, doppi underscore prima e dopo il nome di un metodo indicano un metodo ad uso esclusivamente interno (esempio: `__str__`, oppure `__doc__`)

Gli apici valgono sia singoli che doppi, ma conviene usarli singoli per il codice, doppi nelle stringhe

```
mystring = 'Il mio nome è "Mario" e sono una persona'
```

# In Python tutto è un oggetto

```
>>> my_string_2 = 'corso di laboratorio di programmazione'
>>> dir(my_string_2)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>> my_string_2.title()
'Corso Di Laboratorio Di Programmazione'
```

# In Python tutto è un oggetto

examples.py

```
my_string = 'a,b,c'  
print(my_string)  
print(my_string.split(','))  
print(my_string)
```

```
> python examples.py  
a,b,c  
['a', 'b', 'c']  
a,b,c
```

examples.py

```
my_list = [1,2,3,4]  
print(my_list)  
print(my_list.reverse())  
print(my_list)
```

```
> python examples.py  
[1, 2, 3, 4]  
None  
[4, 3, 2, 1]
```

# Oggetti con operazioni in-place e non

examples.py

```
my_string = 'a,b,c'  
print(my_string)  
print(my_string.split(','))  
print(my_string)
```

Questa è un'operazione (funzione, metodo) che quando viene eseguita torna il risultato

```
> python examples.py  
a,b,c  
['a', 'b', 'c']  
a,b,c
```

examples.py

```
my_list = [1,2,3,4]  
print(my_list)  
print(my_list.reverse())  
print(my_list)
```

Questa è un'operazione (funzione, metodo) che quando viene eseguita modifica l'oggetto, non torna niente!

```
> python examples.py  
[1, 2, 3, 4]  
None  
[4, 3, 2, 1]
```

# Definire oggetti

objects.py

```
class Person():  
    pass  
  
person = Person()  
print(person)
```

```
> python objects.py  
<__main__.Person object at 0x7ff378a93fa0>  
> |
```

# Definire oggetti

objects.py

```
class Person():  
    pass
```

```
person = Person()  
print(person)
```

“pass” è l’istruzione  
nulla, serve per avere  
un blocco vuoto

Questa operazione *istanzia* una classe di oggetto.  
Ovvero, da una generica definizione di un oggetto ne  
creo uno specifico.

```
> python objects.py  
<__main__.Person object at 0x7ff378a93fa0>  
> |
```

# Definire oggetti

objects.py

```
class Person():  
    def __init__(self, name, surname):  
        # Set name and surname  
        self.name = name  
        self.surname = surname  
  
person = Person('Mario', 'Rossi')  
print(person)  
print(person.name)  
print(person.surname)
```

```
> python objects.py  
<__main__.Person object at 0x7f8a75ac0fa0>  
Mario  
Rossi  
> |
```



# Definire oggetti

la funzione “init” è quella che è responsabile di inizializzare l'oggetto. Se non c'è viene usata quella di default che non fa nulla.

objects.py

```
class Person():  
  
    def __init__(self, name, surname):  
  
        # Set name and surname  
        self.name = name  
        self.surname = surname  
  
person = Person('Mario', 'Rossi')  
print(person)  
print(person.name)  
print(person.surname)
```

“self” vuol dire “me stesso”, “me istanza di classe”. E' obbligatorio come parametro in tutti i metodi degli oggetti (salvo casi particolari)

```
> python objects.py  
<__main__.Person object at 0x7f8a75ac0fa0>  
Mario  
Rossi  
> 
```

# Definire oggetti

objects.py

```
class Person():  
  
    def __init__(self, name, surname):  
  
        # Set name and surname  
        self.name = name  
        self.surname = surname  
  
    def __str__(self):  
        return 'Person "{} {}".format(self.name, self.surname)  
  
person = Person('Mario', 'Rossi')  
print(person)
```

```
> python objects.py  
Person "Mario Rossi"
```

```
> 
```

# Definire oggetti

objects.py

```
class Person():  
  
    def __init__(self, name, surname):  
  
        # Set name and surname  
        self.name = name  
        self.surname = surname  
  
    def __str__(self):  
        return 'Person "{} {}".format(self.name, self.surname)  
  
person = Person('Mario', 'Rossi')  
print(person)
```

Funzione ad uso interno che vado a sovrascrivere. E' responsabile della rappresentazione in formato stringa dell'oggetto.

```
> python objects.py  
Person "Mario Rossi"  
> |
```

## objects.py

```
# Import the random module
import random

class Person():

    def __init__(self, name, surname):

        # Set name and surname
        self.name = name
        self.surname = surname

    def __str__(self):
        return 'Person "{} {}".format(self.name, self.surname)

    def say_hi(self):

        # Generate a random number between 0, 1 and 2.
        random_number = random.randint(0,2)

        # Choose a random greeting
        if random_number == 0:
            print('Hello, I am {} {}'.format(self.name, self.surname))
        elif random_number == 1:
            print('Hi, I am {}'.format(self.name))
        elif random_number == 2:
            print('Yo bro! {} here!'.format(self.name))

person = Person('Mario', 'Rossi')
person.say_hi()
```

```
> python objects.py
Hello, I am Mario Rossi.
```

```
> python objects.py
Hi, I am Mario!
```

```
> python objects.py
Yo bro! Mario here!
```

## objects.py

```
# Import the random module
import random

class Person():

    def __init__(self, name, surname):

        # Set name and surname
        self.name = name
        self.surname = surname

    def __repr__(self):
        return 'Person "{} {}"'.format(self.name, self.surname)

    def say_hi(self):

        # Generate a random number between 0, 1 and 2.
        random_number = random.randint(0,2)

        # Choose a random greeting
        if random_number == 0:
            print('Hello, I am {} {}'.format(self.name, self.surname))
        elif random_number == 1:
            print('Hi, I am {}'.format(self.name))
        elif random_number == 2:
            print('Yo bro! {} here!'.format(self.name))

person = Person('Mario', 'Rossi')
person.say_hi()
```

Funzione (metodo) dell'oggetto. Anche chiamata interfaccia. Sono le funzioni che verranno testate per l'esame!

```
> python objects.py
Hello, I am Mario Rossi.
```

```
> python objects.py
Hi, I am Mario!
```

```
> python objects.py
Yo bro! Mario here!
```

# Estendere oggetti

objects.py

```
class Person():
    ...

class Student(Person):

    def __str__(self):
        return 'Student "{} {}".format(self.name, self.surname)

class Professor(Person):

    def __str__(self):
        return 'Prof. "{} {}".format(self.name, self.surname)

    def say_hi(self):
        print('Hello, I am professor {} {}.'.format(self.name, self.surname))
```

# Estendere oggetti

objects.py

```
class Person():  
    ...  
  
class Student(Person):  
  
    def __str__(self):  
        return 'Student "{} {}".format(self.name, self.surname)  
  
class Professor(Person):  
  
    def __str__(self):  
        return 'Prof. "{} {}".format(self.name, self.surname)  
  
    def say_hi(self):  
        print('Hello, I am professor. {} {}'.format(self.name, self.surname))
```

Estendo l'oggetto Persona declinandolo in Studente e Professore. Tutti i metodi che possedeva l'oggetto Persona sono automaticamente ereditati dagli oggetti Persona e Professore. Posso sovrascriverli o aggiungerne altri.

Sovrascrivo la rappresentazione in stringa dell'oggetto Persona per includere il titolo.

Sovrascrivo il metodo che saluta dell'oggetto Persona per avere un saluto di più consono ad un professore.

# Estendere oggetti

objects.py

```
class Person():
    ...

class Student(Person):

    def __str__(self):
        return 'Student "{} {}"'.format(self.name, self.surname)

class Professor(Person):

    def __str__(self):
        return 'Prof. "{} {}"'.format(self.name, self.surname)

    def say_hi(self):
        print('Hello, I am professor {} {}.'.format(self.name, self.surname))

    def original_say_hi(self):
        super().say_hi()
```



# Estendere oggetti

objects.py

```
class Person():
    ...

class Student(Person):

    def __str__(self):
        return 'Student "{} {}".format(self.name, self.surname)

class Professor(Person):

    def __str__(self):
        return 'Prof. "{} {}".format(self.name, self.surname)

    def say_hi(self):
        print('Hello, I am professor {} {}.'.format(self.name, self.surname))

    def original_say_hi(self):
        super().say_hi()
```

Con il “super” accedo alla funzione dell'oggetto padre, anche se l'ho sovrascritta

# Estendere oggetti

## Esempio

objects.py

```
class Person():
    ...

class Student(Person):

    def __str__(self):
        return 'Student "{} {}"'.format(self.name, self.surname)

class Professor(Person):

    def __str__(self):
        return 'Prof. "{} {}"'.format(self.name, self.surname)

    def say_hi(self):
        print('Hello, I am professor {} {}.'.format(self.name, self.surname))

    def original_say_hi(self):
        super().say_hi()
```

```
print('-----')

person = Person('Mario', 'Rossi')

print(person)
person.say_hi()

print('-----')

prof = Professor('Pippo', 'Baudo')

print(prof)
prof.say_hi()
prof.original_say_hi()

print('-----')
```

> python objects.py

```
-----
Person "Mario Rossi"
Yo bro! Mario here!
-----
Prof. "Pippo Baudo"
Hello, I am professor Pippo Baudo.
Yo bro! Pippo here!
-----
```

# Esercizio

Create un oggetto **CSVFile** che rappresenti un file CSV, e che:

- 1) venga inizializzato sul nome del file csv, e
- 2) abbia un attributo “name” che ne contenga il nome
- 3) abbia un metodo “get\_data()” che torni i dati dal file CSV come lista di liste, ad es: [ [ '01-01-2012', '266.0' ], [ '01-02-2012', '145.9' ], ... ]

*Provatelo sul file “shampoo\_sales.csv”.*

Poi, committate il file in cui l’avete scritto.