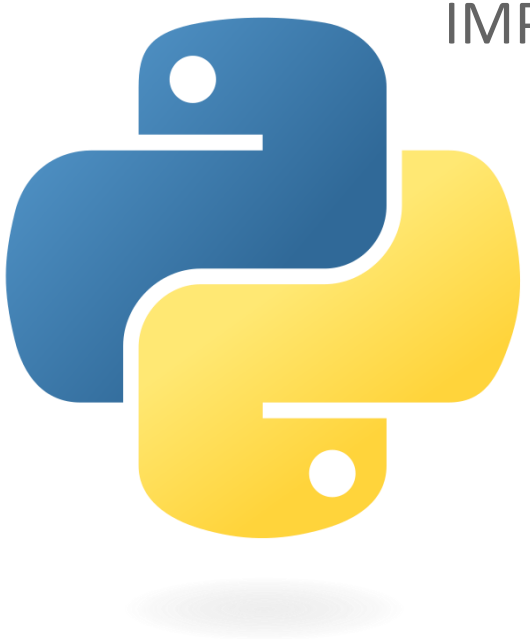


IMPARIAMO A PROGRAMMARE IN

python™

LEZIONE 4

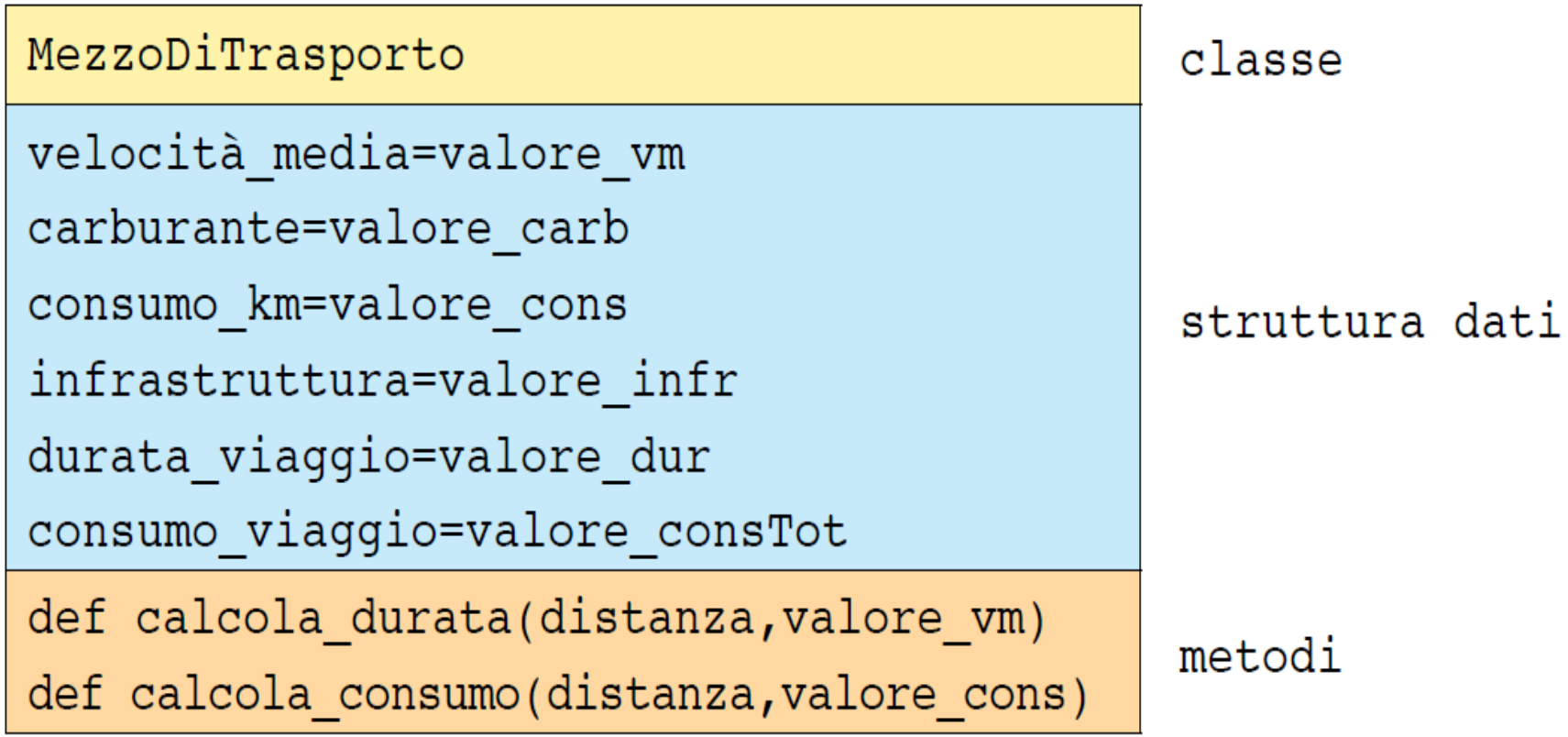
Capitolo 4 – Introduzione alla programmazione a oggetti

Indice

4.1 Il paradigma della OOP

Indice

Una classe è costituita da una struttura dati e da metodi per la sua elaborazione.



4.1 Il paradigma della OOP

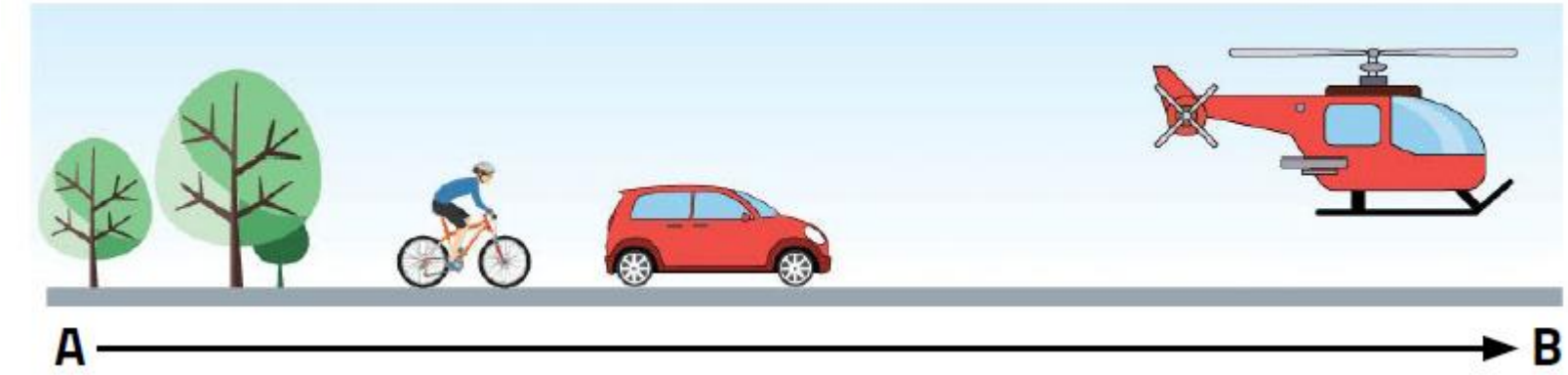
Indice

Un oggetto è un'istanza della classe.

auto:MezzoDiTrasporto()
velocità_media=80. carburante='benzina' consumo_km=0.1 infrastruttura=False durata_viaggio=30. consumo_viaggio=4.0

bici:MezzoDiTrasporto()
velocità_media=20. carburante='muscoli' consumo_km=0.0 infrastruttura=False durata_viaggio=120. consumo_viaggio=0.0

elicottero:MezzoDiTrasporto()
velocità_media=200. carburante='kerosene' consumo_km=1.0 infrastruttura=True durata_viaggio=12. consumo_viaggio=40.0



4.1 Il paradigma della OOP

La classe è un'astrazione che isola gli aspetti importanti degli oggetti da descrivere.

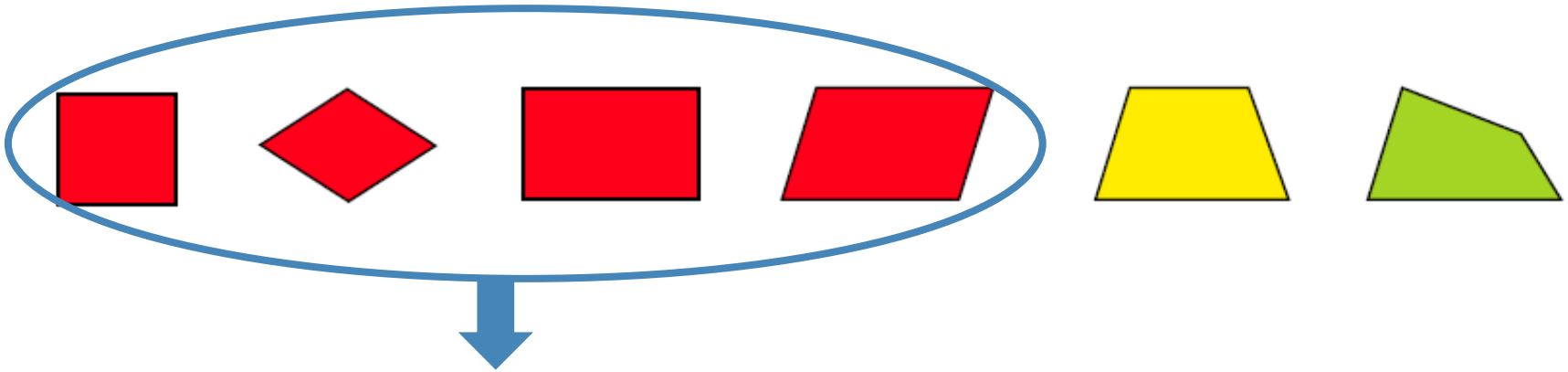


Figure che hanno i lati opposti uguali e paralleli

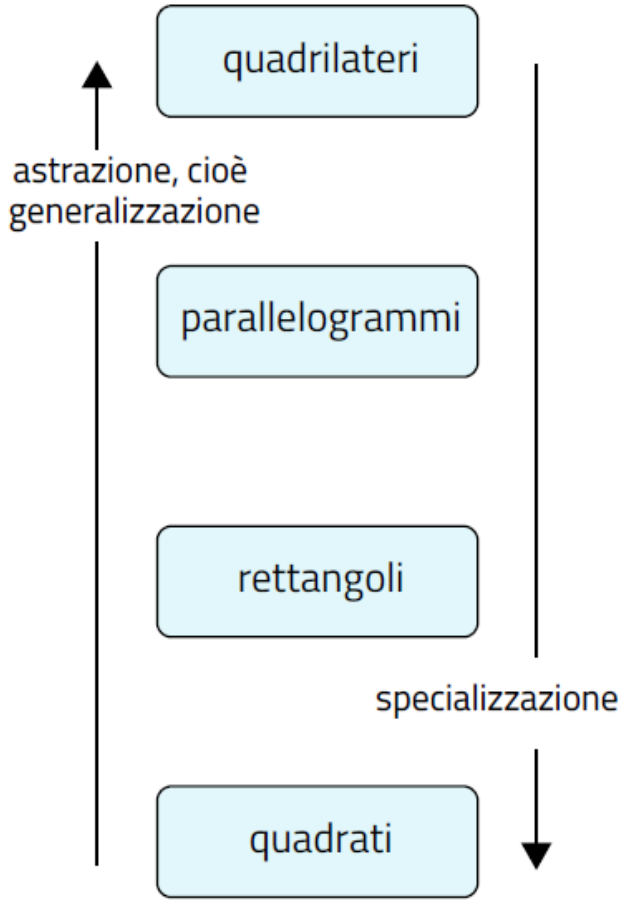
Si applica il principio della classificazione degli oggetti, ovvero la loro suddivisione in classi omogenee comprendenti oggetti dello stesso tipo.

4.1 Il paradigma della OOP

La specializzazione permette di definire delle sottoclassi.

Specializzando una classe si ottengono delle sottoclassi, che definiscono classi all'interno della classe più generale.

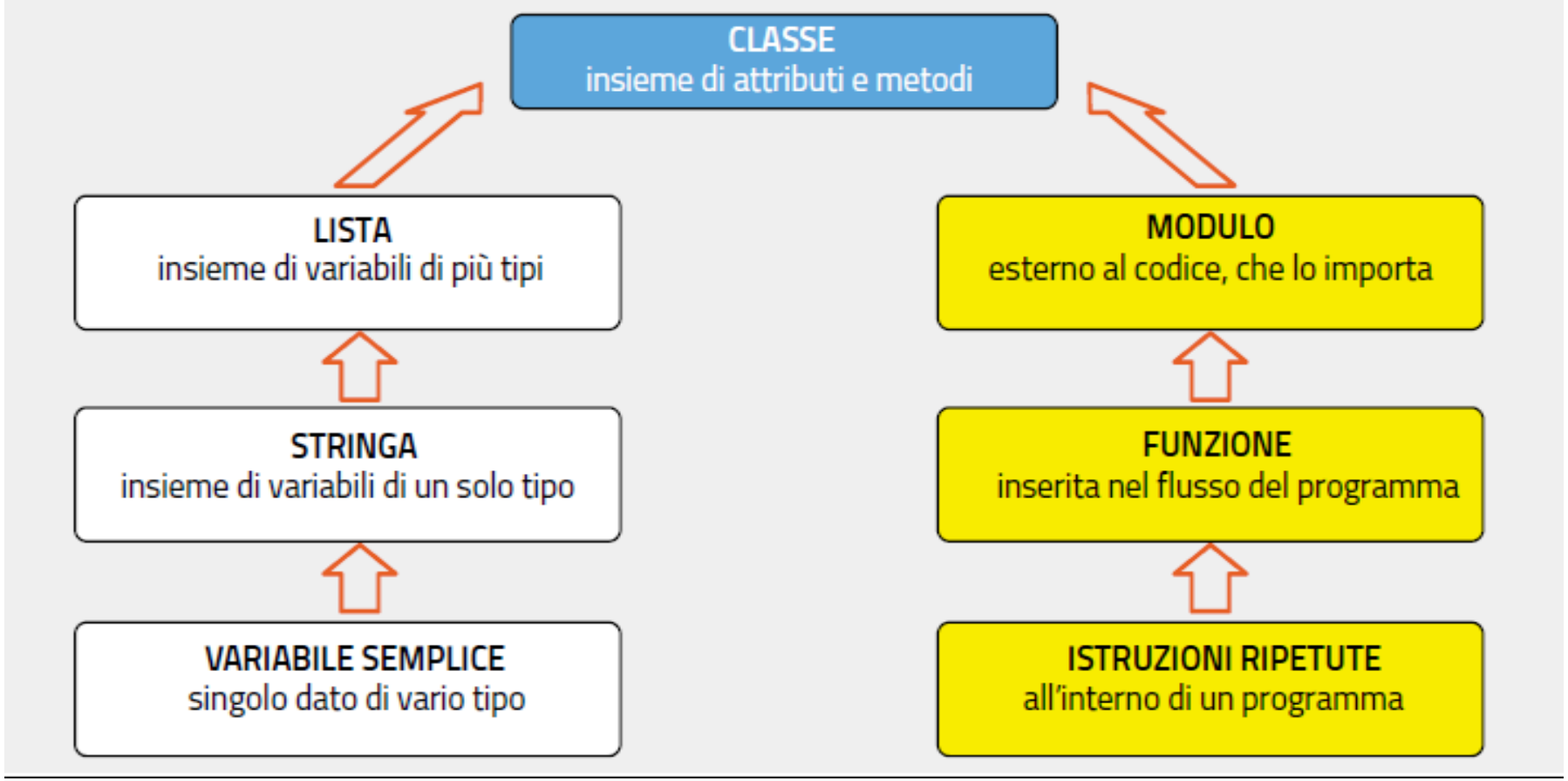
Il processo inverso a quello di specializzazione viene detto processo di generalizzazione



4.1 Il paradigma della OOP

Indice

La classe è un insieme di membri: attributi e metodi, che rappresentano le proprietà di un sistema e le operazioni possibili su di esso.



4.1 Il paradigma della OOP

Indice

L'utilizzo della OOP rispetto alla programmazione procedurale porta delle differenze nella scrittura del software.

PROGRAMMAZIONE PROCEDURALE	PROGRAMMAZIONE A OGGETTI
• si devono gestire dati e funzioni separatamente	• dati e funzioni sono raggruppati negli oggetti
• si devono importare le funzioni per poterle usare	• non è necessario importare funzioni per operare sui dati
• si devono passare ogni volta i dati alle funzioni	• non è necessario passare i dati alle funzioni
• si deve verificare che dati e funzioni siano compatibili	• le funzioni sono sempre compatibili con i dati

4.2 La scrittura del codice

Indice

In Python le classi devono essere dichiarate all'inizio del programma oppure in modulo esterno che andrà importato.

```
class Persona:
    def __init__(self,nome,età):     # metodo inizializzatore
        self.nome=nome                # definizione degli...
        self.età=età                  # ...attributi della classe
    def maggiorenne(self,età)         # metodo specifico
```

Per primo si scrive il costruttore o inizializzatore, che imposta i valori degli attributi della classe e viene eseguito automaticamente all'atto della creazione di un oggetto.

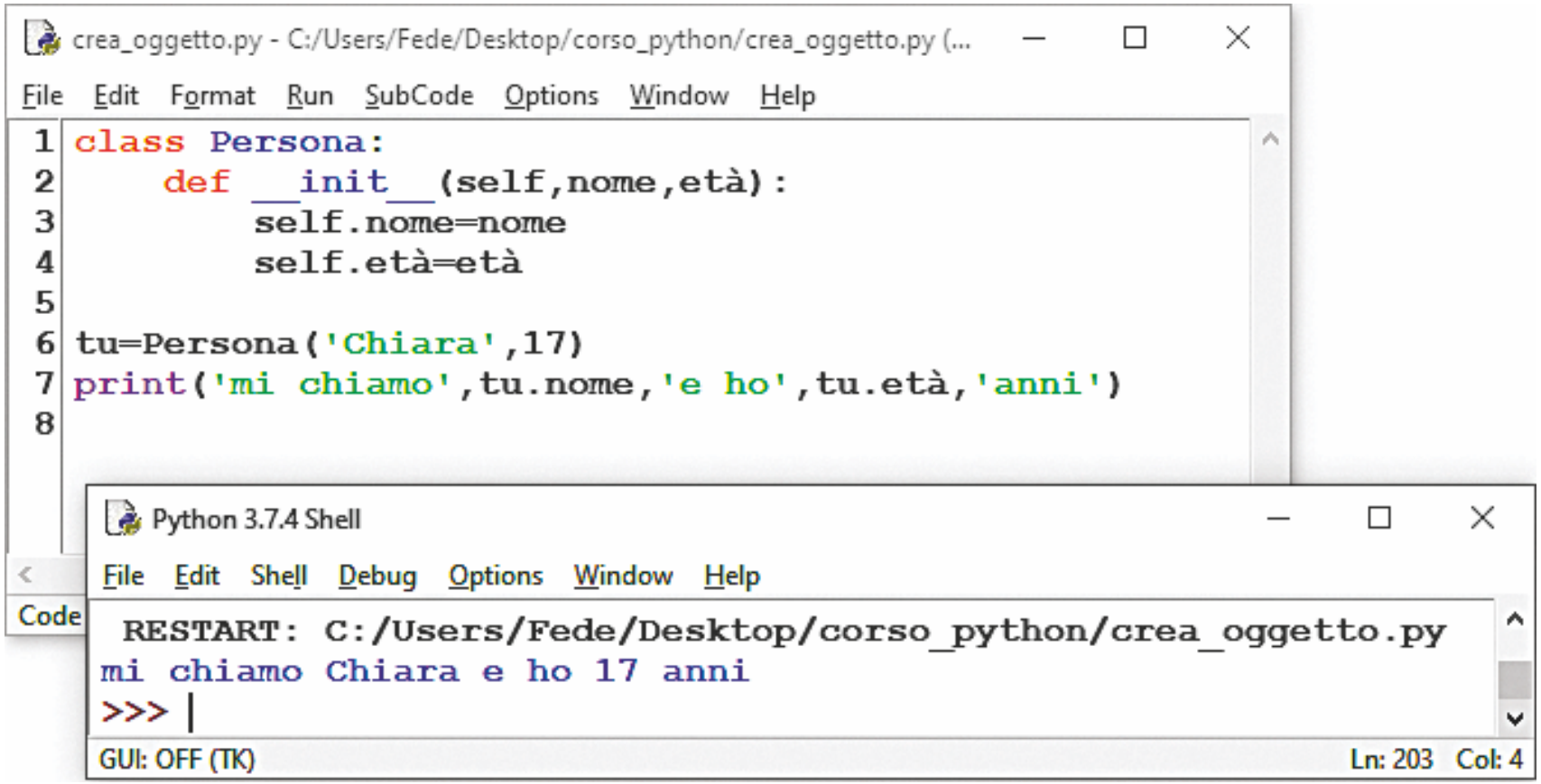
Il parametro **`self`** è obbligatorio e rappresenta l'oggetto considerato.

In seguito si definiscono i metodi specifici della classe.

4.2 La scrittura del codice

Indice

Per creare un oggetto in Python si usa il nome della classe.



Non occorre indicare il parametro **`self`**.

4.2 La scrittura del codice

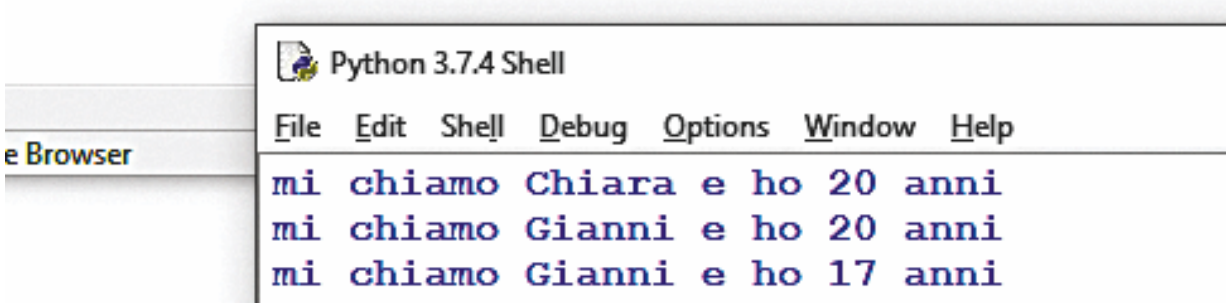
Indice

Il costruttore può assegnare valori di default nel caso non vengano forniti i parametri corrispondenti ad alcuni attributi.

```
class Persona:
    def __init__(self,n,e):
        self.nome=n
        self.età=20

tu=Persona('Chiara','')
print('mi chiamo',tu.nome,'e ho',tu.età,'anni')

io=Persona('Gianni',17)
print('mi chiamo',io.nome,'e ho',io.età,'anni')

io.età=17
print('mi chiamo',io.nome,'e ho',io.età,'anni')
```

non modifica il valore dell'attributo

per modificare il valore di default bisogna accedere all'attributo direttamente



4.2 La scrittura del codice

Indice

In Python gli attributi di una classe sono sempre visibili.
Per «nasconderli» occorre aggiungere al nome un doppio underscore __.

```
class Studente:
    def __init__(self,nome,età,classez):
        self.nome=nome
        self.età=età
        self.__classez=classez
    def stampa (self):
       print('nome:',self.nome,'età:',self.età,'classe:',self.__classez)
```

L'attributo **`__classez`** non sarà leggibile né modificabile.
Queste operazioni sono possibili solo attraverso un metodo specifico della classe.

4.3 Lavorare con moduli e liste

Indice

Le classi si possono memorizzare in moduli autonomi, che vengono poi importati nei programmi.

modulo_clas.py | rettangolo_mod.py

```
1  # modulo contenente la classe
2  class Rettangolo:
3      def __init__(self,b,h):
4          self.base=b
5          self.altezza=h
6      def area(self):
7          print("l'area del rettangolo misura",self.base*self.altezza)
8
```

modulo_clas.py | rettangolo_mod.py

```
1  # programma principale che usa il modulo
2  import modulo_clas
3  mio_rettangolo=modulo_clas.Rettangolo(5.3,8)
4  mio_rettangolo.area()
5
```

Istruzione di importazione

4.3 Lavorare con moduli e liste

Indice

Gli oggetti possono essere organizzati in liste.

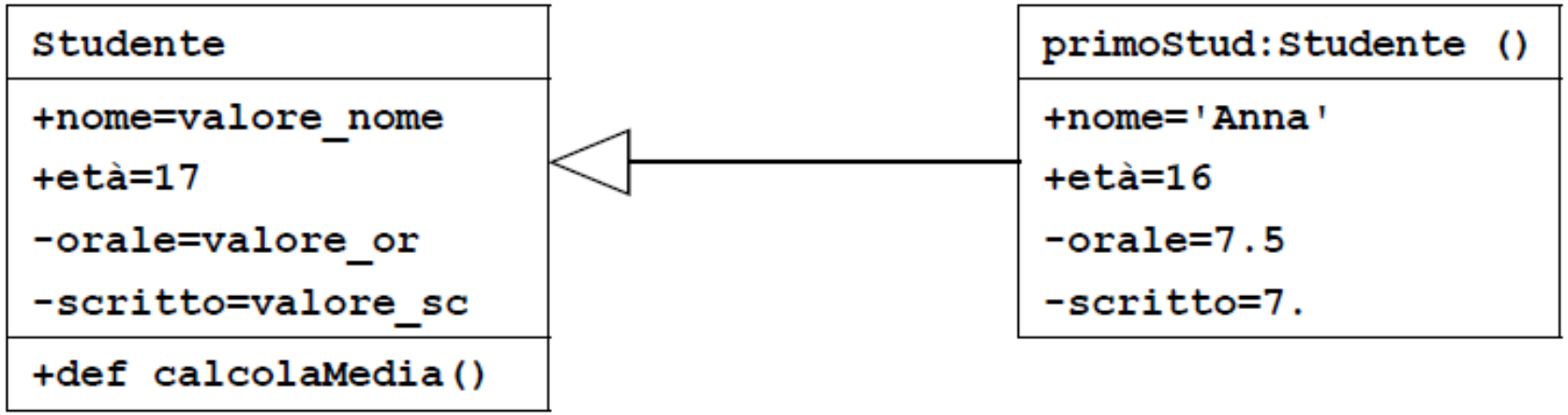
```
# creiamo una lista vuota
s_lista=[]
# chiediamo all'utente di inserire i dati
num=int(input('quanti studenti vuoi inserire? '))
for x in range(num):
    print('STUDENTE n.',x+1)
    dato1=input('inserisci il cognome: ')
    dato2=input('inserisci il nome: ')
    dato3=input('inserisci classe e sezione: ')
# creiamo un oggetto 's' con i dati inseriti
    s=Studente(dato1,dato2,dato3)
# aggiungiamo l'oggetto alla lista
    s_lista.append(s)
# a fine ciclo, stampiamo la lista degli oggetti
print('\nELENCO DEGLI STUDENTI:')
for el in s_lista:
    print(el.get_cognome(),el.get_nome(),el.get_classez())
```

Nell'esempio si considera definita la classe **`Studente`**.

4.4 Dal problema al progetto: l'UML

Indice

L'UML è il linguaggio più usato per la progettazione di sistemi software complessi.



La classe si rappresenta con un rettangolo tripartito, in cui sono scritti il nome della classe e gli attributi (con + e – a seconda della loro visibilità).

Le istanze di una classe vengono collegate alla classe con una freccia che punta verso la classe.