```python
# ------------------------ VERSIONE 1 ------------------------

class persona: # definizione della classe persona
    # definizione degli ATTRIBUTI della classe
    nome = ''
    cognome = ''
    indirizzo = ''
    telefono = ''
    stato_civile = ''

    """
    Definizione dei METODI della classe. Il parametro self è un parametro speciale
    che deve sempre essere il primo parametro di un metodo della classe.
    Python assegna automaticamente al parametro self l'oggetto relativamente al quale
    il metodo è stato chiamato, quindi non deve essere specificato quando si chiama
    un metodo.
    """
    def cambia_indirizzo(self,s):
        self.indirizzo = s
    def cambia_telefono(self,s):
        self.telefono = s
    def cambia_stato_civile(self,s):
        self.stato_civile = s
    def stampa(self):
        print self.nome,self.cognome,self.indirizzo,self.stato_civile

"""
EREDITARIETA': la classe studente eredita dalla classe persona attributi e metodi
comuni e definisce metodi e attributi specifici
"""
class studente (persona):
    scuola = ''
    classe = 0

    def cambia_scuola(self,s):
        self.scuola = s
    def promosso(self):
```

```
        if self.classe == 5:
            print 'scuola finita'
        else:
            self.classe = self.classe + 1

p1 = persona()  """ creazione di un oggetto (o istanza) della classe persona
                    le parentesi sono obbligatorie"""

# assegnazione diretta di valori agli attributi
p1.nome = 'stefano'
p1.cognome = 'riccio'

# assegnazione tramite i metodi
p1.cambia_indirizzo('via html n. 10')
p1.cambia_stato_civile('coniugato')

p1.stampa() # risultato : stefano riccio via html n. 10

s1 = studente()
s1.nome = 'mario'
s1.cognome = 'rossi'
s1.cambia_indirizzo('via html n. 10')
s1.cambia_scuola('liceo')
s1.stampa() # risultato : mario rossi via html n. 10
# Il risultato del metodo "stampa" non permette di visualizzare
# la scuola e la classe di appartenenza.
# Vedere versione successiva

# ------------------------ VERSIONE 2 ------------------------

"""
L'operazione di istanziazione di una classe provoca la creazione di un oggetto.
Spesso e' necessario inizializzare i dati membri dell'oggetto nel momento della
istanziazione di esso. Nel nostro esempio e' utile inizializzare subito la persona
con il suo nome e cognome (infatti il nome e cognome sono assegnati alla nascita e
non cambiano piu').
Per fare questo la teoria della programmazione orientata agli oggetti
```

```
prevede l'uso di una funzione costruttore.
In python esiste qualcosa di simile: e' la funzione denominata "__init__()".
"""

class persona:
    indirizzo = ''
    telefono = ''
    stato_civile = ''

    """
    Segue il metodo COSTRUTTORE della classe che e' usato per inizializzare lo stato
    dell'oggetto che sta per essere creato.
    Nel caso del costruttore il valore di self e l'oggetto che si sta costruendo.
    Si e' anche eliminato l'assegnazione della stringa vuota alle due variabili
    (nome, cognome), perche' ora queste variabili vengono create all'interno
    dele costruttore __init__.
    """
    def __init__(self,n,c):
        self.nome = n
        self.cognome = c

    def cambia_indirizzo(self,s):
        self.indirizzo = s
    def cambia_telefono(self,s):
        self.telefono = s
    def cambia_stato_civile(self,s):
        self.stato_civile = s
    def stampa(self):
        # '\' permette di andare a capo in un'istruzione troppo lunga
        print self.nome,self.cognome,self.indirizzo, \
              self.stato_civile

class studente (persona):   # ecco l'ereditarieta' !
    scuola = ''
    classe = 1

    def cambia_scuola(self,s):
```

```
        self.scuola = s
    def cambia_classe(self,s):
        self.classe = s
    def promosso(self):
        if self.classe == 5:
            print 'scuola finita'
        else:
            self.classe = self.classe + 1
    def stampa(self):
        print self.nome,self.cognome,self.indirizzo,\
              self.stato_civile
        print 'scuola: '+self.scuola+' classe '+str(self.classe)

p1 = persona('stefano', 'riccio')   # le parentesi sono obbligatorie

p1.cambia_indirizzo('via html n. 10')   # richiamo un metodo
p1.cambia_stato_civile('coniugato')
p1.stampa() # risultato : stefano riccio via html n. 10

s1 = studente('mario','rossi')

s1.cambia_indirizzo('via html n. 10')
s1.cambia_scuola('liceo')
s1.cambia_classe(4)
s1.stampa() # risultato : mario rossi via html n. 10


# ------------------------ VERSIONE 3 ------------------------

"""
Esempio che segue realizza l'INCAPSULAMENTO: nella versione precedente le prime operazioni
sull'oggetto p1 sono state le inizializzazioni di indirizzo e stato_civile direttamente.
Python assume che tutti i dati membri siano pubblici e quindi modificabili
dall'esterno dell'oggetto.
Per indicare a Python di tenere protetti i dati membri, e quindi realizzare
veramente l'incapsulamento, si deve anteporre al nome della variabile
i caratteri "__" (2 underscore).
```

```
Cosi' facendo nessuno puo' modificare il dato senza utilizzare l'apposita funzione.
"""

class persona:
    __indirizzo = ''
    __telefono = ''
    __stato_civile = ''

    def __init__(self,n,c):
        self.__nome = n
        self.__cognome = c

    def cambia_indirizzo(self,s):
        self.__indirizzo = s
    def cambia_telefono(self,s):
        self.__telefono = s
    def cambia_stato_civile(self,s):
        self.__stato_civile = s
    def stampa(self):
        # '\' permette di andare a capo in un'istruzione troppo lunga
        print self.__nome,self.__cognome,self.__indirizzo, \
              self.__stato_civile

class studente (persona):
    __scuola = ''
    __classe = 1

    def cambia_scuola(self,s):
        self.__scuola = s
    def cambia_classe(self,s):
        self.__classe = s
    def promosso(self):
        if self.__classe == 5:
            print 'scuola finita'
        else:
            self.__classe = self.__classe + 1
    def stampa(self):
```

```
        print self._persona__nome,self._persona__cognome,self._persona__indirizzo,\
              self._persona__stato_civile
        print 'scuola: '+self.__scuola+' classe '+str(self.__classe)


p1 = persona('stefano', 'riccio')   # le parentesi sono obbligatorie

p1.cambia_indirizzo('via html n. 10')   # richiamo un metodo
p1.cambia_stato_civile('coniugato')
p1.stampa() # risultato : stefano riccio via html n. 10

s1 = studente('mario','rossi')

s1.cambia_indirizzo('via html n. 10')
s1.cambia_scuola('liceo')
s1.cambia_classe(4)
s1.stampa() # risultato : mario rossi via html n. 10


# ------------------------ VERSIONE 4 ------------------------
"""
Esempio che realizza l'incapsulamento: in precedenza le prime operazioni
sull'oggetto p1 sono le inizializzioni di indirizzo e stato_civile direttamente.
Python assume che tutti i dati membri siano pubblici e quindi modificabili
dall'esterno dell'oggetto.
Per indicare a python di tenere protetti i dati membri, e quindi realizzare
veramente l'incapsulamento, si deve anteporre al nome della variabile
i caratteri "__" (2 underscore).
Cosi' facendo nessuno puo' modificare il dato senza utilizzare l'apposita funzione.
"""

class persona:
    __indirizzo = ''
    __telefono = ''
    __stato_civile = ''

    def __init__(self,n,c):
```

```
223         self.__nome = n
224         self.__cognome = c
225 
226     def cambia_indirizzo(self,s):
227         self.__indirizzo = s
228     def cambia_telefono(self,s):
229         self.__telefono = s
230     def cambia_stato_civile(self,s):
231         self.__stato_civile = s
232     def stampa(self):
233         # '\' permette di andare a capo in un'istruzione troppo lunga
234         print self.__nome,self.__cognome,self.__indirizzo, \
235               self.__stato_civile
236 
237 class studente (persona):
238     __scuola = ''
239     __classe = 1
240 
241     def cambia_scuola(self,s):
242         self.__scuola = s
243     def cambia_classe(self,s):
244         self.__classe = s
245     def promosso(self):
246         if self.__classe == 5:
247             print 'scuola finita'
248         else:
249             self.__classe = self.__classe + 1
250     def stampa(self):
251         print self._persona__nome,self._persona__cognome,self._persona__indirizzo,\
252               self._persona__stato_civile
253         print 'scuola: '+self.__scuola+' classe '+str(self.__classe)
254 
255 class segretario(persona):
256     __nome=''
257     def __init__(self,n):
258         self.__nome = n
259     """
```

```
    La classe segretario possiede un metodo per l'iscrizione (cioe' la creazione) di
    un nuovo studente. Questo metodo utilizza il costruttore della classe persona
    ereditato a sua volta dalla classe studente.
    """
    def iscrive_studente(self, sn, sc):
        st = studente(sn,sc)
        return st


p1 = persona('stefano', 'riccio')   # le parentesi sono obbligatorie

p1.cambia_indirizzo('via html n. 10')   # richiamo un metodo
p1.cambia_stato_civile('coniugato')
p1.stampa() # risultato : stefano riccio via html n. 10

s1 = studente('mario','rossi')

s1.cambia_indirizzo('via html n. 10')
s1.cambia_scuola('liceo')
s1.cambia_classe(4)
s1.stampa() # risultato : mario rossi via html n. 10

# Creiamo un'istanza di segretario e gli facciamo iscrivere un nuovo studente
seg1 = segretario('maria')
s2= seg1.iscrive_studente('gianni','pinotto')
s2.stampa()
```