

```
1 # ----- VERSIONE 1 -----
2
3 class persona: # definizione della classe persona
4     # definizione degli ATTRIBUTI della classe
5     nome = ''
6     cognome = ''
7     indirizzo = ''
8     telefono = ''
9     stato_civile = ''
10
11     """
12     Definizione dei METODI della classe. Il parametro self è un parametro speciale
13     che deve sempre essere il primo parametro di un metodo della classe.
14     Python assegna automaticamente al parametro self l'oggetto relativamente al quale
15     il metodo è stato chiamato, quindi non deve essere specificato quando si chiama
16     un metodo.
17     """
18     def cambia_indirizzo(self,s):
19         self.indirizzo = s
20     def cambia_telefono(self,s):
21         self.telefono = s
22     def cambia_stato_civile(self,s):
23         self.stato_civile = s
24     def stampa(self):
25         print self.nome,self.cognome,self.indirizzo,self.stato_civile
26
27     """
28     EREDITARIETA': la classe studente eredita dalla classe persona attributi e metodi
29     comuni e definisce metodi e attributi specifici
30     """
31 class studente (persona):
32     scuola = ''
33     classe = 0
34
35     def cambia_scuola(self,s):
36         self.scuola = s
37     def promosso(self):
```

```

38     if self.classe == 5:
39         print 'scuola finita'
40     else:
41         self.classe = self.classe + 1
42
43 p1 = persona() """ creazione di un oggetto (o istanza) della classe persona
44                 le parentesi sono obbligatorie"""
45
46 # assegnazione diretta di valori agli attributi
47 p1.nome = 'stefano'
48 p1.cognome = 'riccio'
49
50 # assegnazione tramite i metodi
51 p1.cambia_indirizzo('via html n. 10')
52 p1.cambia_stato_civile('coniugato')
53
54 p1.stampa() # risultato : stefano riccio via html n. 10
55
56 s1 = studente()
57 s1.nome = 'mario'
58 s1.cognome = 'rossi'
59 s1.cambia_indirizzo('via html n. 10')
60 s1.cambia_scuola('liceo')
61 s1.stampa() # risultato : mario rossi via html n. 10
62 # Il risultato del metodo "stampa" non permette di visualizzare
63 # la scuola e la classe di appartenenza.
64 # Vedere versione successiva
65
66 # ----- VERSIONE 2 -----
67
68 """
69 L'operazione di istanziamento di una classe provoca la creazione di un oggetto.
70 Spesso e' necessario inizializzare i dati membri dell'oggetto nel momento della
71 istanziamento di esso. Nel nostro esempio e' utile inizializzare subito la persona
72 con il suo nome e cognome (infatti il nome e cognome sono assegnati alla nascita e
73 non cambiano piu').
74 Per fare questo la teoria della programmazione orientata agli oggetti

```

```

75 prevede l'uso di una funzione costruttore.
76 In python esiste qualcosa di simile: e' la funzione denominata "__init__()".
77 """
78
79 class persona:
80     indirizzo = ''
81     telefono = ''
82     stato_civile = ''
83
84     """
85     Segue il metodo COSTRUTTORE della classe che e' usato per inizializzare lo stato
86     dell'oggetto che sta per essere creato.
87     Nel caso del costruttore il valore di self e l'oggetto che si sta costruendo.
88     Si e' anche eliminato l'assegnazione della stringa vuota alle due variabili
89     (nome, cognome), perche' ora queste variabili vengono create all'interno
90     dele costruttore __init__.
91     """
92     def __init__(self,n,c):
93         self.nome = n
94         self.cognome = c
95
96     def cambia_indirizzo(self,s):
97         self.indirizzo = s
98     def cambia_telefono(self,s):
99         self.telefono = s
100    def cambia_stato_civile(self,s):
101        self.stato_civile = s
102    def stampa(self):
103        # '\ ' permette di andare a capo in un'istruzione troppo lunga
104        print self.nome,self.cognome,self.indirizzo, \
105              self.stato_civile
106
107 class studente (persona):    # ecco l'ereditarieta' !
108     scuola = ''
109     classe = 1
110
111     def cambia_scuola(self,s):

```

```

112     self.scuola = s
113 def cambia_classe(self,s):
114     self.classe = s
115 def promosso(self):
116     if self.classe == 5:
117         print 'scuola finita'
118     else:
119         self.classe = self.classe + 1
120 def stampa(self):
121     print self.nome,self.cognome,self.indirizzo,\
122           self.stato_civile
123     print 'scuola: '+self.scuola+' classe '+str(self.classe)
124
125 p1 = persona('stefano', 'riccio') # le parentesi sono obbligatorie
126
127 p1.cambia_indirizzo('via html n. 10') # richiamo un metodo
128 p1.cambia_stato_civile('coniugato')
129 p1.stampa() # risultato : stefano riccio via html n. 10
130
131 s1 = studente('mario','rossi')
132
133 s1.cambia_indirizzo('via html n. 10')
134 s1.cambia_scuola('liceo')
135 s1.cambia_classe(4)
136 s1.stampa() # risultato : mario rossi via html n. 10
137
138
139 # ----- VERSIONE 3 -----
140
141 """
142 Esempio che segue realizza l'INCAPSULAMENTO: nella versione precedente le prime operazioni
143 sull'oggetto p1 sono state le inizializzazioni di indirizzo e stato_civile direttamente.
144 Python assume che tutti i dati membri siano pubblici e quindi modificabili
145 dall'esterno dell'oggetto.
146 Per indicare a Python di tenere protetti i dati membri, e quindi realizzare
147 veramente l'incapsulamento, si deve anteporre al nome della variabile
148 i caratteri "__" (2 underscore).

```

```
149 Così' facendo nessuno puo' modificare il dato senza utilizzare l'apposita funzione.
150 ""
151
152 class persona:
153     __indirizzo = ''
154     __telefono = ''
155     __stato_civile = ''
156
157     def __init__(self,n,c):
158         self.__nome = n
159         self.__cognome = c
160
161     def cambia_indirizzo(self,s):
162         self.__indirizzo = s
163     def cambia_telefono(self,s):
164         self.__telefono = s
165     def cambia_stato_civile(self,s):
166         self.__stato_civile = s
167     def stampa(self):
168         # '\n' permette di andare a capo in un'istruzione troppo lunga
169         print self.__nome,self.__cognome,self.__indirizzo, \
170             self.__stato_civile
171
172 class studente (persona):
173     __scuola = ''
174     __classe = 1
175
176     def cambia_scuola(self,s):
177         self.__scuola = s
178     def cambia_classe(self,s):
179         self.__classe = s
180     def promosso(self):
181         if self.__classe == 5:
182             print 'scuola finita'
183         else:
184             self.__classe = self.__classe + 1
185     def stampa(self):
```

```

186     print self._persona__nome,self._persona__cognome,self._persona__indirizzo,\
187           self._persona__stato_civile
188     print 'scuola: '+self.__scuola+' classe '+str(self.__classe)
189
190
191 p1 = persona('stefano', 'riccio') # le parentesi sono obbligatorie
192
193 p1.cambia_indirizzo('via html n. 10') # richiamo un metodo
194 p1.cambia_stato_civile('coniugato')
195 p1.stampa() # risultato : stefano riccio via html n. 10
196
197 s1 = studente('mario','rossi')
198
199 s1.cambia_indirizzo('via html n. 10')
200 s1.cambia_scuola('liceo')
201 s1.cambia_classe(4)
202 s1.stampa() # risultato : mario rossi via html n. 10
203
204
205 # ----- VERSIONE 4 -----
206 """
207 Esempio che realizza l'incapsulamento: in precedenza le prime operazioni
208 sull'oggetto p1 sono le inizializzazioni di indirizzo e stato_civile direttamente.
209 Python assume che tutti i dati membri siano pubblici e quindi modificabili
210 dall'esterno dell'oggetto.
211 Per indicare a python di tenere protetti i dati membri, e quindi realizzare
212 veramente l'incapsulamento, si deve anteporre al nome della variabile
213 i caratteri "__" (2 underscore).
214 Così facendo nessuno può modificare il dato senza utilizzare l'apposita funzione.
215 """
216
217 class persona:
218     __indirizzo = ''
219     __telefono = ''
220     __stato_civile = ''
221
222     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     """

```

```
260     La classe segretario possiede un metodo per l'iscrizione (cioe' la creazione) di
261     un nuovo studente. Questo metodo utilizza il costruttore della classe persona
262     ereditato a sua volta dalla classe studente.
263     """
264     def iscrive_studente(self, sn, sc):
265         st = studente(sn,sc)
266         return st
267
268
269 p1 = persona('stefano', 'riccio') # le parentesi sono obbligatorie
270
271 p1.cambia_indirizzo('via html n. 10') # richiamo un metodo
272 p1.cambia_stato_civile('coniugato')
273 p1.stampa() # risultato : stefano riccio via html n. 10
274
275 s1 = studente('mario', 'rossi')
276
277 s1.cambia_indirizzo('via html n. 10')
278 s1.cambia_scuola('liceo')
279 s1.cambia_classe(4)
280 s1.stampa() # risultato : mario rossi via html n. 10
281
282 # Creiamo un'istanza di segretario e gli facciamo iscrivere un nuovo studente
283 seg1 = segretario('maria')
284 s2= seg1.iscrive_studente('gianni', 'pinotto')
285 s2.stampa()
```