

# Laboratorio 06

Programmazione - CdS Matematica

Michele Donini  
9 dicembre 2014

- Le classi rappresentano il meccanismo principale della programmazione orientata agli oggetti (OOP).
- Una classe è un'astrazione di un'entità del mondo reale:

```
class Persona():  
    '''classe persona'''
```

- Volendo creare un nuovo oggetto di tipo `Persona`, possiamo semplicemente scrivere:

```
alessia = Persona()
```

## Classi (2)

- Il costruttore di classe è un metodo molto particolare di una classe che serve all'atto della costruzione dell'istanza per inizializzare l'oggetto che stiamo creando.

```
class Persona():  
    def __init__(self,nome,cognome):  
        self.nome=nome  
        self.cognome=cognome  
  
    def stampa(self):  
        print "Nome: %s; Cognome: %s"%(self.nome,  
            self.cognome)
```

# Esercizio

- Simulare un sistema di prenotazione aule (molto semplificato).
- Il sistema sarà formato dalle seguenti classi: **Aula**, **Prenotazione**, **Dipartimento**, **Professore**.
- Campi dati consigliati:
  - Aula: nome (stringa), capienza (int).
  - Dipartimento: lista delle aule, lista delle prenotazioni, lista dei professori.
  - Professore: nome (stringa).
  - Prenotazione: aula, orario (intero compreso tra 7 e 19, il 7 si intende come prenotato nella fascia [7, 8]), professore.

Cosa sarà richiesto?

- Gestire le quattro classi in tutti i loro campi.
- Stampare: informazioni su un'aula, su un professore, su una prenotazione, sul dipartimento (inteso come vari metodi per vedere le aule del dipartimento, le prenotazioni, le aule ancora libere per una certa fascia oraria, le aule occupate nella giornata ed altri a vostra scelta).
- Metodi di gestione della prenotazione: un metodo che verifica se un'aula è libera in una certa fascia oraria, un metodo che permette l'aggiunta di prenotazioni verificando se l'aula è libera.
- Stampare una tabella doppia entrata (aula-orario) prenotazioni chiara e leggibile.

# Esercizio

Struttura esercizio:

`www.math.unipd.it/~mdonini/didattica/  
prenotazioni.py`

# Step 1

Creazione struttura di base delle classi Aula e Professore:  
costruttori e metodi di stampa e gestione delle informazioni di base.

# Step 1

Creazione struttura di base delle classi Aula e Professore:  
costruttori e metodi di stampa e gestione delle informazioni di base.  
Per esempio:

```
class Aula():  
    def __init__(self,nome,capienza):  
        """Crea un'aula"""  
        ...  
  
    def stampa(self):  
        print ...
```

# Verifica Step 1

```
a = Aula("Aula0",100)
a.stampa()
p = Professore("Fabio Aiolli")
p.stampa()
```

# Verifica Step 1

```
a = Aula("Aula0",100)
a.stampa()
p = Professore("Fabio Aiolli")
p.stampa()
```

```
Aula Aula0 con capienza 100
Professor Fabio Aiolli
```

## Step 2

Creazione struttura di base della classe Prenotazione.

## Step 2

Creazione struttura di base della classe Prenotazione.  
Possibile via da seguire:

```
class Prenotazione():  
    def __init__(self, aula, orario, prof):  
        """Crea una prenotazione (aula e prof sono di  
           tipo Aula e Professore. Mentre orario e' di  
           tipo int)"""  
        ...  
  
    def stampa(self):  
        ... # Hint: utilizzare i comandi definiti per le  
           altre due classi.
```

## Verifica Step 2

```
a = Aula("Aula0",100)
p = Professore("Fabio Aiolli")
prenoto = Prenotazione(a,15,p)
prenoto.stampa()
```

## Verifica Step 2

```
a = Aula("Aula0",100)
p = Professore("Fabio Aiolli")
prenoto = Prenotazione(a,15,p)
prenoto.stampa()
```

```
Prenotazione per le ore 15
Aula Aula0 con capienza 100
Professor Fabio Aiolli
```

## Step 3

Inizio creazione classe Dipartimento.

## Step 3

Inizio creazione classe Dipartimento.

Possibile via da seguire:

```
def __init__(self, nome):  
    """Crea un dipartimento"""  
    self.nome = nome  
    self.aule = []  
    self.professori = []  
    self.prenotazioni = []
```

## Step 3

Inizio creazione classe Dipartimento.

Possibile via da seguire:

```
def __init__(self, nome):  
    """Crea un dipartimento"""  
    self.nome = nome  
    self.aule = []  
    self.professori = []  
    self.prenotazioni = []
```

Creare e testare ora i metodi di aggiunta e stampa di aule e professori.

- aggiungi\_prof(self, prof)
- aggiungi\_aula(self, aula)
- stampa\_prof(self)
- stampa\_aule(self)

## Verifica Step 3

```
a = Aula("Aula0",100)
p = Professore("Fabio Aiolli")
dip = Dipartimento("Matematica")
dip.aggiungi_prof(p)
dip.aggiungi_prof(Professore("Andrea Burattin"))
dip.aggiungi_prof(Professore("Michele Donini"))
dip.stampa_prof()
dip.aggiungi_aula(a)
dip.aggiungi_aula(Aula("Aula1",150))
dip.aggiungi_aula(Aula("Aula2",150))
dip.aggiungi_aula(Aula("Aula3",30))
dip.aggiungi_aula(Aula("Aula4",500))
dip.stampa_aule()
```

# Verifica Step 3

Vogliamo come risultato:

I docenti del Dipartimento di Matematica sono:

Professor Fabio Aiolli

Professor Andrea Burattin

Professor Michele Donini

Le aule del Dipartimento di Matematica sono:

Aula Aula0 con capienza 100

Aula Aula1 con capienza 150

Aula Aula2 con capienza 150

Aula Aula3 con capienza 30

Aula Aula4 con capienza 500

## Step 4

Definiamo ora un metodo `def occupazione_aula(self,aula)` tale che la sua descrizione sia:

“Ritorna una lista di interi che rappresenta gli orari in cui l'aula e' occupata. Se l'aula non esiste ritorna `None`”

## Step 4

Definiamo ora un metodo `def occupazione_aula(self,aula)` tale che la sua descrizione sia:

“Ritorna una lista di interi che rappresenta gli orari in cui l'aula e' occupata. Se l'aula non esiste ritorna None”

```
def occupazione_aula(self,aula):  
    """Ritorna una lista di interi che rappresenta  
        gli orari in cui l'aula  
        e' occupata. Se l'aula non esiste ritorna None  
        """  
    if (aula not in self.aule):  
        return None  
    orari = []  
    for p in self.prenotazioni:  
        if p.aula == aula:  
            orari.append(p.orario)  
    return orari
```

## Verifica Step 4

```
a = Aula("Aula0",100)
p = Professore("Fabio Aiolli")
dip = Dipartimento("Matematica")
dip.aggiungi_prof(p)
dip.aggiungi_aula(a)
dip.prenotazioni.append(Prenotazione(a,15,p))
dip.prenotazioni.append(Prenotazione(a,9,p))
occupata = dip.occupazione_aula(a)
print occupata
```

## Verifica Step 4

```
a = Aula("Aula0",100)
p = Professore("Fabio Aiolli")
dip = Dipartimento("Matematica")
dip.aggiungi_prof(p)
dip.aggiungi_aula(a)
dip.prenotazioni.append(Prenotazione(a,15,p))
dip.prenotazioni.append(Prenotazione(a,9,p))
occupata = dip.occupazione_aula(a)
print occupata
```

[15, 9]

## Step 5

Ora siamo pronti a definire il metodo `aggiungi_prenotazione(self, prenotazione)`.

Questo metodo aggiunge la prenotazione controllando che:

- l'aula sia presente nel dipartimento
- il professore sia presente nel dipartimento
- l'orario sia in un intero tra le 7 e le 19
- non ci sia già una prenotazione coincidente (stessa aula e stesso orario)

Testare in modo opportuno il metodo.

## Step 6

Creare ora i metodi di stampa delle prenotazioni:

- `stampa__prenotazioni(self)`
- `stampa__prenotazioni__aula(self,aula)`
- `stampa__prenotazioni__orario(self,orario)`
- `aule__libere__orario(self,orario)`

Testare in modo opportuno questi 4 metodi.

## Step 7

Per concludere creiamo un metodo `stampa_tabella(self)` che stampi in modo chiaro una tabella a doppia entrata (orario-aula) che faccia capire in modo chiaro la situazione delle prenotazioni nel dipartimento.

Orario\Aule	Aula0	Aula1	Aula2	Aula3	Aula4
7	o	o	o	o	o
8	o	o	o	o	o
9	X	o	o	o	o
10	o	o	o	o	o
11	o	o	o	o	o
12	o	o	o	o	o
13	o	o	o	o	o
14	o	o	o	o	o
15	X	X	o	o	o
16	o	o	o	o	o
17	o	o	o	o	o
18	o	o	o	o	o
19	o	o	o	o	o