



Pattern Recognition
and Applications Lab

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2020 - 2021

Corso di Laurea in Ingegneria Elettrica, Elettronica e Informatica



University of Cagliari,
Italy

Department of Electrical and
Electronic Engineering



La Programmazione ad Oggetti in Python

In queste slide vedremo:

- Definizione di una classe
- Definizione di attributi (di istanza)
- Definizione di metodi (di istanza)

Definizione di Classi

Supponiamo il gestore di una libreria ci chieda di **creare una classe per poter memorizzare i dati di un libro.**




La classe dovrà permettere di:

- 1) memorizzare: titolo, autore, editore, prezzo, anno di pubblicazione.
- 2) calcolare il prezzo scontato del libro considerando che la libreria ha deciso di applicare, a tutti i libri pubblicati da più di 5 anni uno sconto del 5%.

Prima di tutto dobbiamo creare il diagramma di classe..

Riepilogo Notazione Diagrammi di Classe

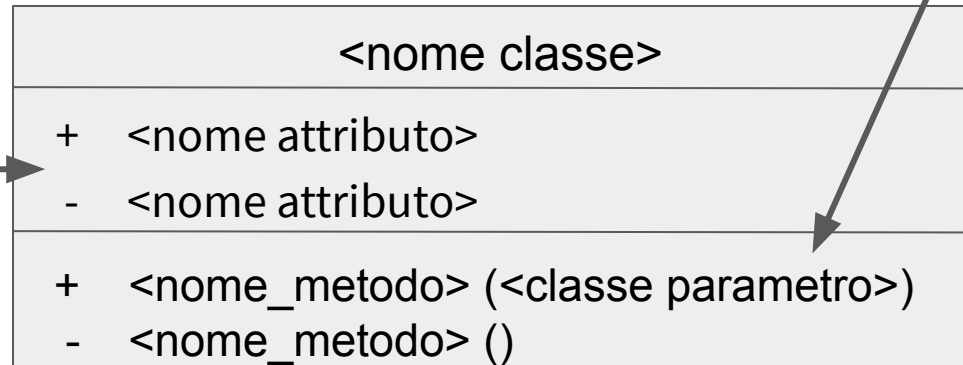
Relazioni:

- Composizione 
- Aggregazione 
- Ereditarietà 

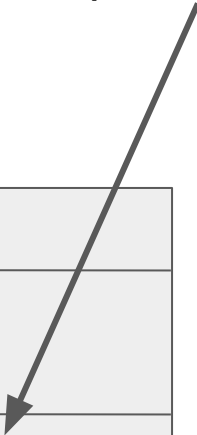
Rappresentazione oggetti:

Visibilità

- + pubblico
- privato



Metodo che riceve
un parametro



Definizione di Classi

Il diagramma di classe della classe libro.

libro
<ul style="list-style-type: none">+ titolo+ autore+ editore+ prezzo+ anno di pubblicazione
<ul style="list-style-type: none">+ calcola_prezzo_scontato()- calcola_eta_libro()- calcola sconto()

Definizione di Classi

Creiamo una classe *libro* che abbia gli attributi: *titolo, autore, editore, prezzo, anno_publicazione*. (Ignoriamo per ora il fatto che abbia dei metodi).

In Python, **per convenzione, i nomi delle classi** vengono scritti seguendo la notazione CamelCase e vengono preceduti da una C che indica che si tratta di una classe.

Il nome della nostra classe sarà quindi CLibro.

Definizione di Classi - Definizione degli Attributi

Gli oggetti libro appartenenti a questa classe avranno valori degli attributi differenti, dobbiamo quindi ricevere questi valori in input dall'utente.

Per ora, **facciamo sì che questi valori ci vengano forniti dall'utente quando un nuovo oggetto viene creato.**

In Python esiste una funzione apposita che viene richiamata quando un nuovo oggetto viene creato e ci permette di inizializzarlo, la funzione **`__init__`**. Questa funzione viene spesso chiamata **costruttore** dell'oggetto.

Spiegazione approssimativa della creazione di un oggetto. La descrizione dettagliata si trova nelle dispense “LPO_3_gli_oggetti_in_python_parte_4”.

Definizione di Classi - Definizione degli Attributi

Creiamo una classe *libro* che abbia gli attributi: *titolo*, *autore*, *editore*, *prezzo*, *anno_publicazione*.

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_publicazione):
```

```
        ...
```

Questa funzione, viene chiamata automaticamente quando un oggetto viene creato. In questo codice, la funzione riceverà i valori degli attributi dell'oggetto.

Definizione di Classi - Definizione degli Attributi

La funzione `__init__` dovrà **definire gli attributi** e memorizzare i valori degli attributi dell'oggetto creato.

In Python, come le variabili, **gli attributi si definiscono assegnandogli un valore.**

La sintassi per definire un attributo è la seguente:

```
self.<nome_attributo> = <valore>
```

Quando viene eseguita questa istruzione l'attributo `<nome_attributo>` viene memorizzato nell'**istanza dell'oggetto** (alla quale ci si riferisce con ***“self”***).

Definizione di Classi - Definizione degli Attributi

Creiamo una classe *libro* che abbia gli attributi: *titolo*, *autore*, *editore*, *prezzo*, *anno_publicazione*.

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_publicazione):  
        self.titolo = titolo  
        self.autore = autore  
        self.editore = editore  
        self.prezzo = prezzo  
        self.anno_publicazione = anno_publicazione
```

Creazione di un'Istanza della Classe

Creiamo un'istanza della classe libro (oggetto appartenente alla classe libro).

```
titolo = input("inserisci il titolo del libro ")
autore = input("inserisci l'autore del libro ")
editore = input("inserisci l'editore del libro ")
prezzo = float(input("inserisci il prezzo del libro "))
anno_publicazione = int(input("inserisci l'anno di pubblicazione del libro "))
oggetto_libro = CLibro(titolo, autore, editore, prezzo, anno_publicazione)
```

Ottenere il Valore di un Attributo

Per ottenere il valore dell'attributo di un oggetto che abbiamo creato la sintassi è la seguente:

`<oggetto>.<nome_attributo>`

Esempio:

```
titolo = input("inserisci il titolo del libro ")
```

...

```
oggetto_libro = CLibro(titolo, autore, editore, prezzo, anno_publicazione)
```

```
print( oggetto_libro.titolo )
```

Stamperà il titolo valore dell'attributo "titolo" dell'oggetto.

Ottenere il Valore di un Attributo

Se volessimo fare la stessa cosa nel codice che definisce la classe la sintassi sarebbe:

```
self.<nome_attributo>
```

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_publicazione):  
        self.titolo = titolo  
        print(self.titolo)  
        self.autore = autore  
        self.editore = editore  
        self.prezzo = prezzo  
        self.anno_publicazione = anno_publicazione
```

Capire se un Oggetto Possiede un Attributo

In Python è possibile verificare se un oggetto possiede un attributo utilizzando la funzione **hasattr**.

... (chiediamo in input i valori degli attributi all'utente)

```
oggetto_libro = CLibro(titolo, autore, editore, prezzo, anno_pubblicazione)
```

```
print( hasattr(oggetto_libro, "titolo") )
```

Stamperà True perchè l'oggetto libro possiede l'attributo titolo.

Definizione di Classi - Definizione di un Metodo

Abbiamo detto che la classe libro deve anche permettere di calcolare il prezzo scontato del libro considerando che la libreria ha deciso di applicare, a tutti i libri pubblicati da più di 5 anni uno sconto del 5%.

Definiamo i metodi dell'oggetto.

Come abbiamo detto in precedenza **i metodi sono funzioni**.

Per questo oggetto abbiamo bisogno dei seguenti metodi:

- + calcola_prezzo_scontato
- calcola_eta_libro
- calcola_sconto

Definizione di Classi - Definizione di un Metodo

Cominciamo definendo la funzione *calcola_eta_libro*.

Poiché non abbiamo ancora visto come acquisire la data corrente utilizzando Python, facciamo sì che l'anno corrente venga inserito in input dall'utente.

Questa funzione avrà quindi un parametro: *anno_corrente*.

Definizione di Classi - Definizione di un Metodo

Questa funzione deve anche poter aver accesso all'attributo `anno_publicazione`, memorizzato nell'oggetto..

La funzione avrà quindi un altro parametro: l'oggetto.

L'oggetto viene sempre indicato come primo parametro della funzione e per convenzione il nome del parametro è ***self***.

Poiché ricevono l'istanza della classe come primo argomento questo tipo di metodi viene detto **metodi di istanza (instance method)**.

Vedremo che esistono altri tipi di metodo ma questi sono i più comuni.

Definizione di Classi - Definizione di un Metodo

```
class CLibro:  
    def __init__(self, titolo, autore, editore, prezzo, anno_publicazione):  
        ...  
  
    def calcola_eta_libro(self, anno_corrente):  
        ...
```

NB: la funzione `__init__` per convenzione viene definita per prima.

Definizione di Classi - Utilizzo di un Metodo

Supponiamo di voler utilizzare il metodo `calcola_eta_libro`.

... (prendiamo in input i valori degli attributi dell'oggetto)

```
oggetto_libro = CLibro(titolo, autore, editore, prezzo, anno_pubblicazione)
```

```
oggetto_libro.calcola_eta_libro(anno_corrente)
```

NB: notate che **quando viene richiamata la funzione non viene passato come argomento l'oggetto**. Non abbiamo:

```
oggetto_libro.calcola_eta_libro(oggetto_libro, anno_corrente)
```

L'oggetto viene sempre passato alla funzione come primo argomento in automatico da Python.

Visibilità di Funzioni e Attributi

La funzione *calcola_eta_libro* in realtà è una funzione *privata*.

Dalle specifiche in fatti, quello che l'oggetto deve permettere di fare è calcolare il prezzo scontato, che dipende dall'età del libro.

Questa funzione ci serve per dividere il problema (calcolare il prezzo scontato) in sottoproblemi. In particolare questa funzione risolve il sottoproblema di calcolare l'età del libro.

Visibilità di Funzioni e Attributi

In Python, per definire una funzione o un attributo come privato si fa precedere il nome della funzione/attributo da un underscore `_`.

Esempio:

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_publicazione):
```

```
        ...
```

```
    def _calcola_eta_libro(self, anno_corrente):
```

```
        ...
```

Visibilità di Funzioni e Attributi

NB: In Python l'utilizzatore di un oggetto è potenzialmente in grado di utilizzare tutto ciò che noi definiamo come “privato”.

Perchè quindi è importante definire attributi e funzioni come privati?

- Definire un **attributo** come **privato** segnala all'utente che **modificarlo potrebbe causare problemi**.

- Definire una **funzione** come **privata** gli segnala che molto probabilmente quella funzione **non è di interesse dell'utilizzatore dell'oggetto**
e.g., perchè serve solo internamente all'oggetto per svolgere dei calcoli.

Definizione di Classi - Definizione di un Metodo

Implementiamo la funzione `calcola_eta_libro`.

Questa funzione sottrae all'anno corrente l'anno di pubblicazione.

NB: l'anno di pubblicazione è un attributo e il suo valore è memorizzato nell'oggetto.

```
class CLibro:
```

```
...
```

```
def _calcola_eta_libro(self, anno_corrente):  
    eta_libro = anno_corrente - self.anno_pubblicazione  
    return eta_libro
```

Definizione di Classi - Definizione di un Metodo

Abbiamo bisogno dei seguenti metodi:

- + `calcola_prezzo_scontato`
- `calcola_eta_libro`
- `calcola_sconto`

Creiamo la funzione *calcola_sconto*.

Lo sconto sarà del 5% del prezzo per tutti i libri pubblicati da più di 5 anni.

Questa funzione richiamerà al suo interno la funzione *calcola_eta_libro*, quindi anch'essa dovrà avere due parametri, l'oggetto e l'anno corrente.

Definizione di Classi - Definizione di un Metodo

Per utilizzare un metodo definito all'interno della stessa classe la sintassi è la seguente:

```
self.<nome_metodo>(self, <parametro1>...<parametron> )
```

Anche quando utilizziamo un metodo della classe stessa il primo argomento che viene passato alla funzione da Python è l'oggetto stesso.

NB: essendo funzioni, le definizioni dei metodi potrebbero potenzialmente avere anche parametri di default.

Definizione di Classi - Definizione di un Metodo

```
class CLibro:
```

```
...
```

```
def _calcola_sconto(self, anno_corrente):  
    eta_libro = self._calcola_eta_libro(anno_corrente)  
    if eta_libro > 5:  
        sconto = self.prezzo * 5 / 100  
    else:  
        sconto = 0  
    return sconto
```

Definizione di Classi - Definizione di un Metodo

Definiamo ora l'ultimo metodo, il metodo pubblico: `calcola_prezzo_scontato`
Essendo un metodo pubblico il nome della corrispondente funzione non sarà preceduto dall'underscore.

```
class CLibro:
```

```
...
```

```
def calcola_prezzo_scontato(self, anno_corrente):  
    prezzo_scontato = self.prezzo - self._calcola_sconto(anno_corrente)  
    return prezzo_scontato
```

Utilizzo dell'Oggetto

Una volta definita la classe libro, possiamo creare un oggetto e utilizzare il metodo pubblico `calcola_prezzo_scontato`.

```
titolo = input("inserisci il titolo del libro ")
autore = input("inserisci l'autore del libro ")
editore = input("inserisci l'editore del libro ")
prezzo = float(input("inserisci il prezzo del libro "))
anno_publicazione = int(input("inserisci l'anno di pubblicazione del libro "))
oggetto_libro = CLibro(titolo, autore, editore, prezzo, anno_publicazione)
print(oggetto_libro.calcola_prezzo_scontato(2020))
```

Esercizio: Creazione della Classe Esame

Scrivere il codice Python della classe *Esame* il cui diagramma di classe è il seguente:

Esame
+ nome_esame + voto_primo_parziale + voto_secondo_parziale
- calcola_media() + stampa_voto_finale()

(La funzione `stampa_voto_finale` calcola il voto finale come la media dei voti conseguiti nei parziali.)

Esercizio: Creazione della Classe Esame

1) Definiamo la funzione `__init__` (il costruttore dell'oggetto).

```
class CEsame:
```

```
    def __init__(self, nome_esame, voto_primo_parziale, voto_secondo_parziale):
```

```
        self.nome_esame = nome_esame
```

```
        self.voto_primo_parziale = voto_primo_parziale
```

```
        self.voto_secondo_parziale = voto_secondo_parziale
```

Spiegazione approssimativa della creazione di un oggetto. La descrizione dettagliata si trova nelle dispense “LPO_3_gli_objetti_in_python_parte_4”.

Esercizio: Creazione della Classe Esame

2) Definiamo i metodi dell'oggetto.

```
def _calcola_media(self):  
    media = (self.voto_primo_parziale + self.voto_secondo_parziale)/2  
    return media  
  
def stampa_voto_finale(self):  
    voto_finale = self._calcola_media()  
    print("Il voto finale per l'esame ", self.nome_esame, " è ", voto_finale)
```

Esercizio: Creazione della Classe Esame

Proviamo a creare un'istanza di classe CEsame.

```
esame_lpo = CEsame("LPO", 28, 30)  
esame_lpo.stampa_voto_finale()
```

Stamperà:

Il voto finale per l'esame LPO è 29.0

Terminologia: Attributi vs Proprietà

Sebbene in molti linguaggi orientati agli oggetti il termine attributi e proprietà si utilizzino in modo intercambiabile, in Python questo non accade perchè, come vedremo, in Python il termine “proprietà” ha un significato specifico e differente.

Terminologia: Metodi vs Funzioni

Sebbene in molti linguaggi i termini funzione e metodo vengano usati in modo intercambiabile ([https://it.wikipedia.org/wiki/Funzione_\(informatica\)](https://it.wikipedia.org/wiki/Funzione_(informatica))) nei linguaggi orientati agli oggetti, generalmente si parla di **funzioni** quando ci si riferisce ad una **funzione non dichiarata nella definizione di una classe**, mentre si parla di **metodo** quando ci si riferisce a **funzioni definite dentro la definizione di una classe**.

Terminologia: Metodi vs Funzioni

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_pubblicazione):
```

```
        self.titolo = titolo
```

```
        self.autore = autore
```

```
        self.editore = editore
```

```
        self.prezzo = prezzo
```

```
        self.anno_pubblicazione = anno_pubblicazione
```

```
    def _calcola_eta_libro(self, anno_corrente):
```

```
        eta_libro = anno_corrente - self.anno_pubblicazione
```

```
        return eta_libro
```

Nei linguaggi orientati agli oggetti `_calcola_eta_libro` viene chiamato metodo.

Terminologia: Metodi vs Funzioni

```
def calcola_eta_libro(self, anno_corrente, anno_pubblicazione):  
    eta_libro = anno_corrente - anno_pubblicazione  
    return eta_libro
```

```
class CLibro:
```

```
    def __init__(self, titolo, autore, editore, prezzo, anno_pubblicazione):  
        self.titolo = titolo  
        self.autore = autore  
        self.editore = editore  
        self.prezzo = prezzo  
        self.anno_pubblicazione = anno_pubblicazione
```

Nei linguaggi orientati agli oggetti `calcola_eta_libro` viene chiamata funzione.

Metodi vs Funzioni

Nei linguaggi orientati agli oggetti si cerca di ridurre al minimo l'utilizzo di funzioni e di prediligere l'utilizzo di metodi.