



Pattern Recognition
and Applications Lab

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2024 - 2025

Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni



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)

Riepilogo Notazione Diagrammi di Classe

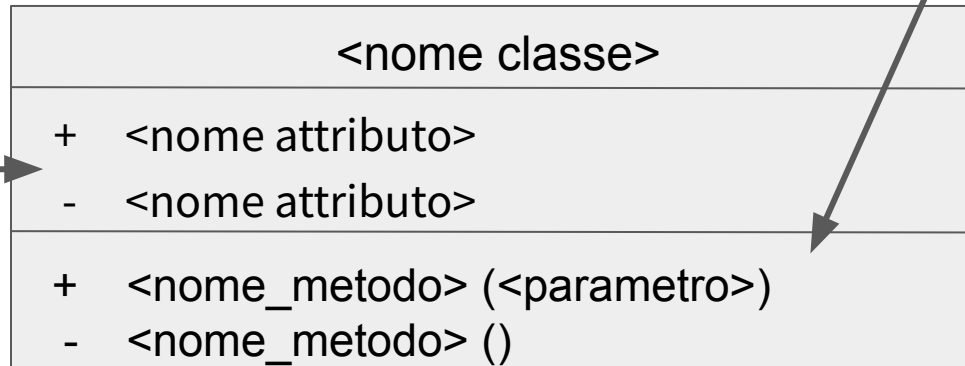
Relazioni:

- Composizione 
- Aggregazione 
- Ereditarietà 

Rappresentazione oggetti:

Visibilità

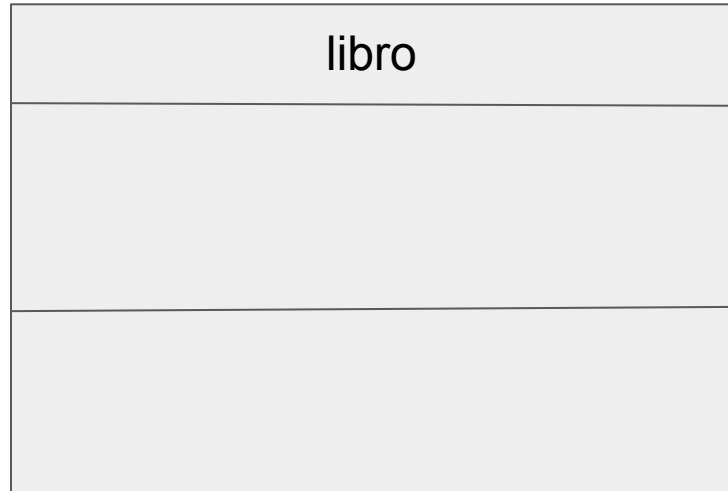
- + pubblico
- privato



Metodo che riceve
un parametro

Definizione di Classi

Supponiamo di voler definire una classe chiamata libro che non ha attributi e metodi.



Definizione di Classi

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, mentre per i nomi di attributi, metodi etc si utilizza la notazione con l'underscore. Il nome della classe sarà quindi CLibro.

In Python scriveremo:

```
class CLibro():  
    pass # l'istruzione pass è un'istruzione che non fa nulla
```

Questo codice definisce com'è fatta la classe libro.

Creazione di Istanze di Classe (Oggetti)

Una volta definita una classe possiamo creare delle istanze di quella classe (oggetti).

La sintassi per creare un'istanza è:

`<nome classe> (..)`

Vedremo nelle prossime slide cosa va eventualmente scritto al posto di ..

Creazione di Istanze di Classe (Oggetti)

Creiamo due istanze della classe libro e le memorizziamo in due variabili.

```
class CLibro():  
    pass
```

```
libro1 = CLibro() # la variabile libro1 è un riferimento ad un'istanza di CLibro
```

```
libro2 = CLibro() # la variabile libro2 è un riferimento ad un'altra istanza di CLibro
```

Definizione degli Attributi (di Istanza)

Di solito però le classi hanno diversi attributi.

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

La sintassi per definire un attributo è la seguente:

<referimento all'istanza>.<nome_attributo> = <valore>

Quando viene eseguita questa istruzione, il valore dell'attributo **<nome_attributo>** viene memorizzato nell'**istanza di classe.**

Definizione degli Attributi (di Istanza)

```
class CLibro():  
    pass
```

```
libro1 = CLibro()  
libro2 = CLibro()
```

```
libro1.titolo = "LPO"  
libro2.titolo = "Analisi1"
```

NB: sebbene Python ci permetta di definire attributi in questo modo, non andrebbero definiti esattamente così. Il modo corretto lo vedremo tra qualche slide...

Potenzialmente, potremmo fare così...

In questo modo abbiamo aggiunto ad entrambe le istanze un attributo titolo e gli abbiamo assegnato un valore.

Definizione degli Attributi (di Istanza)

```
class CLibro():
```

```
    pass
```

```
libro1 = CLibro()
```

```
libro2 = CLibro()
```

```
libro1.titolo = "LPO"
```

```
libro2.titolo = "Analisi1"
```

NB: le istanze hanno lo stesso attributo ma valori differenti per quell attributo.

Potenzialmente, potremmo fare così...

In questo modo abbiamo aggiunto ad entrambe le istanze un attributo titolo e gli abbiamo assegnato un valore.

Utilizzo degli Attributi (di Istanza)

Per ottenere il valore di un attributo la sintassi è:

<referimento all'istanza>. <nome_attributo>

Utilizzo degli Attributi (di Istanza)

```
class CLibro():  
    pass
```

```
libro1 = CLibro()
```

```
libro1.titolo = "LPO"  
print(libro1.titolo)
```

Stamperà: LPO

Definizione di Metodi

Supponiamo di voler far sì che questa classe abbia anche un metodo che permette di stampare a schermo il titolo del libro.

| libro |
|-------------------|
| + titolo |
| + stampa_titolo() |

Definizione di Metodi

Nella definizione della classe dovremmo definire un metodo. Un metodo è sostanzialmente una funzione.

Questo metodo dovrà essere a conoscenza degli **attributi di una specifica istanza** (in quanto il titolo è diverso per ogni istanza).

Questo tipo di metodi viene chiamato **metodi di istanza**.

Definizione di Metodi

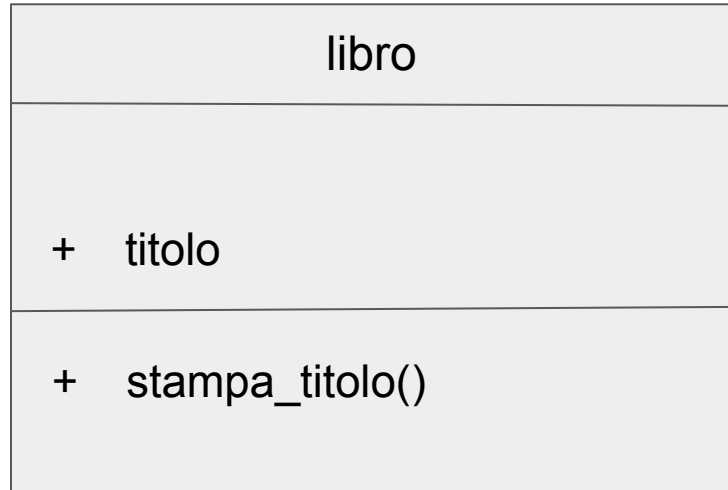
```
class CLibro():  
    def stampa_titolo(self):  
        print (self.titolo)
```

Questo metodo ha un **parametro** che per convenzione viene chiamato “**self**” e **riceve in automatico un riferimento all’istanza della classe**.

Grazie a questo parametro nella definizione della classe abbiamo un riferimento alla classe che possiamo utilizzare, ad esempio, per ottenere il valore di suoi attributi.

Definizione di Metodi

NB: nel diagramma di classe non specifichiamo il parametro self, perchè è un parametro che abbiamo perchè utilizziamo Python, mentre il diagramma di classe deve essere generico.



Definizione di Metodi

I metodi di istanza vanno invocati con la seguente sintassi:

<referimento all'istanza>.<nome del metodo> (<arg1> .. <arg n>)

Dove <arg1>.. <arg1 n> sono gli eventuali argomenti del metodo che devono essere passati al metodo.

NB: non dobbiamo passare al metodo l'istanza della classe. Questa gli viene passata in automatico dall'interprete.

Quindi se abbiamo un metodo con il solo parametro self, non dovremo passargli nessun argomento.

Definizione di Metodi

```
class CLibro():  
    def stampa_titolo(self):  
        print (self.titolo)
```

```
libro = CLibro() # crea un'istanza della classe libro  
libro.titolo = "LPO" # definisce l'attributo titolo e gli assegna un valore  
libro.stampa_titolo() # invoca il metodo stampa_titolo
```

Stamperà: LPO

Definizione di Metodi

```
class CLibro():  
    def stampa_titolo(self):  
        print (self.titolo)
```

NB: notate che **quando viene invocato il metodo non dobbiamo passare come argomento l'istanza**. Non abbiamo:
`libro.stampa_titolo(libro)`.
Viene passato in automatico da python.

```
libro = CLibro() # crea un'istanza della classe libro  
libro.titolo = "LPO" # definisce l'attributo titolo e gli assegna un valore  
libro.stampa_titolo() # invoca il metodo stampa_titolo
```

Stamperà: LPO

Definizione di Metodi

```
class CLibro():  
    def stampa_titolo(self):  
        print (self.titolo)
```

Problemi nella definizione di questa classe:

- 1) Per ogni istanza dovremmo scrivere il codice per definire gli attributi nel codice che utilizza la classe.
- 2) Sarebbe complicato per chi usa la classe capire quali attributi deve definire.

```
l = CLibro() # crea un'istanza della classe libro  
l.titolo = "LPO" # definisce l'attributo titolo e gli assegna un valore  
l.stampa_titolo() # invoca il metodo stampa_titolo
```

Stamperà: LPO

Definizione di Metodi

```
class CLibro():  
    def stampa_titolo(self):  
        print (self.titolo)
```

Problemi nella definizione di questa classe:

- 1) Per ogni istanza dovremmo scrivere il codice per definire gli attributi nel codice che utilizza la classe.
- 2) Sarebbe complicato per chi usa la classe capire quali attributi deve definire.

Soluzione: Gli attributi vengono definiti utilizzando un metodo apposito, chiamato metodo “inizializzatore”.

```
l = CLibro() # crea un'istanza della classe libro  
l.titolo = "LPO" # definisce l'attributo titolo e gli assegna un valore  
l.stampa_titolo() # invoca il metodo stampa_titolo  
Stamperà: LPO
```

Definizione di Metodi

```
class CLibro():  
    def __init__(self, titolo):  
        self.titolo = titolo
```

In questo modo è chiaro a chi osserva la definizione della classe, quali sono i suoi attributi!

```
def stampa_titolo(self):  
    print (self.titolo)
```

Il metodo inizializzatore (`__init__`) è un metodo speciale di Python che ha il compito di inizializzare l'istanza.

In questo metodo possiamo definire gli attributi.

Definizione di Metodi

Come passiamo i parametri a questo metodo?

Il metodo iniziatore viene invocato in automatico quando un'istanza della classe viene creata.

Come avevamo detto, la sintassi per creare un'istanza è:

<nome classe> (..)

Dove al posto dei puntini vanno indicati gli argomenti che devono essere passati al metodo iniziatore.

Definizione di Metodi

```
class CLibro():  
    def __init__(self, titolo):  
        self.titolo = titolo  
  
    def stampa_titolo(self):  
        print (self.titolo)
```

```
titolo = "LPO"
```

```
l = CLibro(titolo) # viene creata un'istanza dell'oggetto e invocato il metodo  
inizializzatore, al quale viene passato come argomento il contenuto della  
variabile titolo
```

```
l.stampa_titolo()
```


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..

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 - 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_pubblicazione = int(input("inserisci l'anno di pubblicazione del libro "))

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

Ottenere il Valore di un Attributo

Se volessimo stampare il valore dell'attributo titolo fuori dalla definizione della classe.

```
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.

```
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.

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 il metodo *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.

Questo metodo avrà quindi un parametro: *anno_corrente*.

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.

Visibilità di Metodi e Attributi

Il metodo *calcola_eta_libro* in realtà è un metodo *privato*.

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

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

Visibilità di Funzioni e Attributi

In Python, per definire un metodo o un attributo come privato si fa precedere il nome del metodo/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 metodi come privati?

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

- Definire un metodo come **privato** gli segnala che molto probabilmente quel metodo **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 il metodo `calcola_eta_libro`.

Questo metodo 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 il metodo *calcola_sconto*.

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

Questo metodo richiamerà al suo interno il metodo *calcola_eta_libro*, quindi anch'esso dovrà avere due parametri (senza contare `self`), 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>(<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() |

(Il metodo `stampa_voto_finale` calcola il voto finale come la media aritmetica dei voti conseguiti nei parziali.)

Esercizio: Creazione della Classe Esame

1) Definiamo il metodo `__init__` .

```
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
```

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(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.