

Intelligenza Artificiale

Anno Accademico 2022 - 2023

***Classi e Oggetti
in Python***

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

SOMMARIO

- Programmazione Orientata agli Oggetti (OOP)
- Classi e Oggetti in Python
- Come istanziare un Oggetto
- Come istanziare i Metodi
- Class e Instance Attributes
- Esempi vari

OBJECT-ORIENTED PROGRAMMING

INTRODUZIONE

- La **Programmazione Orientata agli Oggetti** è un paradigma di programmazione che prevede di assolvere ai vari compiti mediante **oggetti** che cooperano tra loro.
- Ogni oggetto ha il proprio insieme di dati e un insieme di metodi che li elaborano.
- Nel linguaggio Python una **classe** descrive un insieme di oggetti che hanno lo stesso comportamento.
- Ciascuna classe definisce uno specifico insieme di **metodi**, da poter usare con i propri oggetti.
- L'insieme di tutti i metodi messi a disposizione da una classe, con la descrizione del loro comportamento, è l'**interfaccia pubblica** (**public interface**) della classe.

CLASSI E OGGETTI

COME DEFINIRE UNA CLASSE IN PYTHON

Tutte le definizioni di una classe hanno inizio con la parola chiave **class**, seguita dal nome della classe e da due punti.

Quella che segue è una definizione “minimale” di classe, il cui corpo è costituito solo dal placeholder **pass**, che indica dove andrà scritto il codice:

```
In [1]: class Dog:  
        pass
```

In Python è consuetudine scrivere i nomi delle classi secondo la cosiddetta notazione “**Camel-case**” (o “**CapitalizedWords**”).

Ad esempio, il nome di una classe relativa ad una specifica razza di cani come Jack Russell Terrier verrebbe scritta così:
JackRussellTerrier.

CLASSI E OGGETTI

ISTANZIARE UN OGGETTO

Data la precedente definizione di classe, possiamo istanziare un nuovo oggetto **Dog** scrivendo il nome della classe seguito dalle parentesi:

```
In [2]: Dog()
```

```
Out[2]: <__main__.Dog at 0x10429dcc0>
```

In questo modo abbiamo ottenuto un nuovo oggetto **Dog** a **0x10429dcc0**. Questa stringa rappresenta il **memory address** che indica dove è stato allocato in memoria l'oggetto **Dog**.

CLASSI E OGGETTI

ISTANZIARE UN OGGETTO

Proviamo ora ad istanziare un nuovo oggetto **Dog**:

```
In [3]: Dog()
```

```
Out[3]: <__main__.Dog at 0x10429d320>
```

Questa nuova istanza è allocata in un differente indirizzo di memoria. Essa è completamente distinta dal primo oggetto **Dog** che avevamo istanziato.

CLASSI E OGGETTI

ISTANZIARE UN OGGETTO

Per capire meglio quanto affermato, vediamo le seguenti istruzioni:

```
In [4]: a = Dog()  
        b = Dog()
```

```
In [5]: a == b
```

```
Out[5]: False
```

Abbiamo creato due oggetti **Dog**, assegnati rispettivamente alle variabili **a** e **b**.

Confrontando **a** e **b** mediante l'operatore "==" il risultato è **False**.

CLASSI E OGGETTI

CONSTRUCTOR METHOD

La definizione della classe **Dog** vista prima non è molto interessante per il momento.

In effetti sarebbe utile precisare alcune proprietà che tutti gli oggetti **Dog** dovrebbero avere (e.g., nome, età, razza, ecc.).

Le proprietà che tutti gli oggetti **Dog** devono avere possono essere definite in un metodo chiamato `.__init__()`.

Quando un nuovo oggetto **Dog** è creato, `.__init__()` imposta lo stato iniziale dell'oggetto assegnando i valori delle proprietà dell'oggetto. Ossia, tale metodo inizializza ogni nuova istanza della classe.

CLASSI E OGGETTI

CONSTRUCTOR METHOD

E' possibile passare al metodo tutti i parametri che vogliamo, ma il primo parametro dovrà essere sempre una variabile chiamata **self**.

Quando viene creata una nuova istanza di classe, l'istanza viene automaticamente passata al parametro **self** in modo che possano essere definiti nuovi **attributi** sull'oggetto.

CLASSI E OGGETTI

CONSTRUCTOR METHOD

Aggiorniamo la classe **Dog** con un metodo `.__init__()` che crea gli attributi `.name` e `.age`:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Nel corpo del metodo ci sono due istruzioni che usano la variabile **self**:

self.name = name crea un attributo chiamato **name** e gli assegna il valore del parametro **name**.

self.age = age crea un attributo chiamato **age** e gli assegna il valore del parametro **age**.

CLASSI E OGGETTI


CLASS E INSTANCE ATTRIBUTES

Gli attributi creati in `.__init__()` sono chiamati **instance attributes**. Il valore di un instance attribute è specifico di una particolare istanza della classe.

C'è poi un altro tipo di attributi, chiamati **class attributes**, i cui valori sono gli stessi per tutte le istanze della classe:

```
In [6]: class Dog:
        species = 'Canis lupus familiaris'
        def __init__(self, name, age):
            self.name = name
            self.age = age
```

class attribute



CLASSI E OGGETTI

CLASS E INSTANCE ATTRIBUTES

Per istanziare oggetti della classe **Dog** che abbiamo definita è necessario fornire i valori per i parametri **name** e **age**:

```
In [7]: buddy = Dog('Buddy', 9)
```

```
In [8]: miles = Dog('Miles', 4)
```

In tal modo creiamo due nuove istanze.

Da notare che il metodo `.__init__()` ha tre parametri, mentre nelle istruzioni precedenti gli abbiamo passato solo due parametri.

Nota: Quando si istanzia un oggetto **Dog**, Python crea una nuova istanza e la passa al primo parametro di **init**. Questo essenzialmente rimuove il parametro **self**; quindi ci si deve solo preoccupare dei parametri **name** e **age**.

CLASSI E OGGETTI

CLASS E INSTANCE ATTRIBUTES

Una volta create le istanze di Dog, possiamo accedere ai loro attributi mediante la **dot notation**:

```
In [9]: buddy.name
```

```
Out[9]: 'Buddy'
```

```
In [10]: buddy.age
```

```
Out[10]: 9
```

CLASSI E OGGETTI

CLASS E INSTANCE ATTRIBUTES

modifica di un instance attribute

```
In [11]: buddy.age = 10  
buddy.age
```

Out[11]: 10

accesso ad un class attribute

```
In [12]: miles.species
```

Out[12]: 'Canis lupus familiaris'

modifica di un class attribute

```
In [13]: miles.species = 'Felis silvestris'  
miles.species
```

Out[13]: 'Felis silvestris'

CLASSI E OGGETTI

INSTANCE METHODS

Gli **Instance methods** sono funzioni definite all'interno di una classe e che possono essere invocati da un'istanza di tale classe.

Come abbiamo già visto per il metodo `.__init__()`, il primo parametro di un instance method è sempre **self**.

CLASSI E OGGETTI

INSTANCE METHODS

Vediamo un paio di esempi per la classe **Dog**:

```
In [14]: class Dog:
          species = 'Canis lupus familiaris'

          def __init__(self, name, age):
              self.name = name
              self.age = age

          # Instance method
          def description(self):
              return f'{self.name} ha {self.age} anni'

          # Altro instance method
          def speak(self, sound):
              return f'{self.name} dice {sound}'
```

CLASSI E OGGETTI

INSTANCE METHODS

Abbiamo due **instance methods**:

.description() restituisce una stringa che mostra i valori degli attributi **name** e **age** del cane.

.speak() ha un parametro chiamato **sound** e restituisce una stringa contenente il nome del cane e il verso che emette.

CLASSI E OGGETTI

INSTANCE METHODS

Creiamo una istanza chiamata **miles** e vediamo come invocare i metodi:

```
In [15]: miles = Dog('Miles', 4)
```

chiamata del metodo
.description()

```
In [16]: miles.description()
```

```
Out[16]: 'Miles ha 4 anni'
```

chiamata del metodo
.speak()

```
In [17]: miles.speak('Bau Bau')
```

```
Out[17]: 'Miles dice Bau Bau'
```

chiamata del metodo
.speak()

```
In [18]: miles.speak('Woof Woof')
```

```
Out[18]: 'Miles dice Woof Woof'
```

CLASSI E OGGETTI

INSTANCE METHODS

Il metodo `.description()` della classe **Dog** ci restituisce una stringa contenente informazioni sull'istanza **miles**.

Quando scriviamo le nostre classi, è una buona idea quella di avere un metodo che ci restituisca una stringa contenente informazioni utili su una istanza della classe.

Tuttavia, `.description()` non è il modo migliore per ottenere ciò in Python (in gergo si usa dire che non è “Pythonic”).

Del resto se usiamo una `print()`, questo è ciò che accade:

```
In [19]: print(miles)
```

```
<__main__.Dog object at 0x1042de080>
```

CLASSI E OGGETTI

INSTANCE METHODS

Possiamo però definire un instance method speciale chiamato `.__str__()`:


```
In [20]: class Dog:
          species = 'Canis lupus familiaris'

          def __init__(self, name, age):
              self.name = name
              self.age = age

          def __str__(self):
              return f'{self.name} ha {self.age} anni'

          def speak(self, sound):
              return f'{self.name} dice {sound}'
```

sostituisce il metodo `.description()`



CLASSI E OGGETTI

INSTANCE METHODS

Adesso, con l'istruzione `print(miles)`, otteniamo un output più significativo:

```
In [21]: miles = Dog('Miles', 4)
```

```
In [22]: print(miles)
```

```
Miles ha 4 anni
```

I metodi come `.__init__()` e `.__str__()` sono chiamati **dunder methods** (dunder: double underscore)

ALTRO ESEMPIO DI CLASSE

Definiamo la seguente classe, relativa a prodotti finanziari:

```
In [24]: class FinancialInstrument():  
         def __init__(self, symbol, price):  
             self.symbol = symbol  
             self.price = price
```

ALTRO ESEMPIO DI CLASSE

Definiamo poi un oggetto, istanza di tale classe:

```
In [25]: aapl = FinancialInstrument('AAPL', 100)
```

```
In [26]: aapl.symbol
```

← accesso a un
instance attribute

```
Out[26]: 'AAPL'
```

```
In [27]: aapl.price
```

← accesso a un
instance attribute

```
Out[27]: 100
```

```
In [28]: aapl.price = 105  
aapl.price
```

← aggiornamento di un
instance attribute

```
Out[28]: 105
```


ENCAPSULATION

GETTERS E SETTERS

Encapsulation ha generalmente l'obiettivo di nascondere i dati all'utente che utilizza una classe.

I metodi **getter** e **setter** contribuiscono a questo scopo:

```
In [29]: class FinancialInstrument():  
    def __init__(self, symbol, price):  
        self.symbol = symbol  
        self.price = price  
    def get_price(self):  
        return self.price  
    def set_price(self, price):  
        self.price = price
```

ENCAPSULATION

GETTERS E SETTERS

```
In [30]: fi = FinancialInstrument('AAPL', 100)
```

```
In [31]: fi.get_price()
```

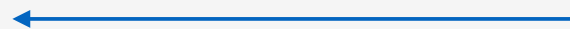
```
Out[31]: 100
```

```
In [32]: fi.set_price(105)
```

```
In [33]: fi.get_price()
```

```
Out[33]: 105
```

```
In [34]: fi.price
```



accesso diretto a un
instance attribute

```
Out[34]: 105
```

ENCAPSULATION

PRIVATE INSTANCE ATTRIBUTES

Abbiamo appena visto che l'uso dei metodi **getter** e **setter** non impedisce all'utente di accedere e manipolare direttamente gli attributi.

E' qui che entrano in gioco i **private** instance attributes. Essi sono definiti da due underscore iniziali:

```
In [35]: class FinancialInstrument():
          def __init__(self, symbol, price):
              self.symbol = symbol
              self.__price = price
          def get_price(self):
              return self.__price
          def set_price(self, price):
              self.__price = price
```

ENCAPSULATION

PRIVATE INSTANCE ATTRIBUTES

Il metodo `.get_price()` restituisce il valore:

```
In [36]: fi = FinancialInstrument('AAPL', 100)
```

```
In [37]: fi.get_price()
```

```
Out[37]: 100
```

ENCAPSULATION

PRIVATE INSTANCE ATTRIBUTES

Se proviamo ad accedere direttamente all'attributo otteniamo un messaggio di errore:

```
In [38]: fi.__price
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-38-e3138907ce6c> in <module>  
----> 1 fi.__price  
  
AttributeError: 'FinancialInstrument' object has no attribute '__price'
```

ENCAPSULATION

PRIVATE INSTANCE ATTRIBUTES

Tuttavia, l'accesso diretto è ancora possibile se mettiamo un underscore prima del nome della classe:

```
In [39]: fi._FinancialInstrument__price
```

```
Out[39]: 100
```

```
In [40]: fi._FinancialInstrument__price = 105
```

```
In [41]: fi.get_price()
```

```
Out[41]: 105
```

```
In [42]: fi.set_price(100)
```

```
In [43]: fi.get_price()
```

```
Out[43]: 100
```

RIFERIMENTI

Amos, D., Bader, D., Jablonski, J., Heisler, F. *Python Basics*, fourth edition, Real Python, 2021.

Hilpisch, Y. *Python for Finance*, 2nd edition, O'Reilly, 2019.

Lubanovic, B. *Introducing Python*, O'Reilly, 2020.

Horstmann, C., Necaie, R.D. *Python for Everyone*, John Wiley & Sons, 2014.

Hu, Y. *Easy learning Data Structures & Algorithms Python*, second edition, 2021.