Programmazione a oggetti

Primi passi con classi e oggetti

In [1]:
```
class Dataset:
    #Metodo inizializzatore
    def __init__(self, data):
        self.data = data  # attributo dell'oggetto
        self.numero_righe = len(data) # attributo dell'oggetto

    #Metodo media
    def calcolo_media(self):
        media =  sum(self.data) / len(self.data)
        self.media =  media
        return media

    #Metodo media primi n elementi
    def calcolo_media_primi_n(self, n):
        media =  sum(self.data[0:n]) / len(self.data[0:n])
        return media
```

In [2]:
```
# Creazione di due oggetti
data1 = Dataset([10, 20, 30])
data2 = Dataset([10, 20])
```

Interroghiamo gli attributi degli oggetti

In [3]:
```
data1.data
```

Out[3]:
```
[10, 20, 30]
```

In [4]:
```
data2.numero_righe
```

Out[4]:
```
2
```

il prossimo codice restituirà un errore

In [5]:
```
data1.media
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[5], line 1
----> 1 data1.media

AttributeError: 'Dataset' object has no attribute 'media'
```

Applichiamo il metodo mean a data1

In [6]:
```
risultato = data1.calcolo_media()
```

L'istruzione precedente non restituisce più l'errore

In [7]:
```
data1.media
```

Out[7]:
```
20.0
```

inoltre risultato è valorizzato come la variabile di output specificata nel return

In [8]:
```
risultato
```

Out[8]:
```
20.0
```

Proviamo ad eseguire calcolo_media_primi_n con il parametro n=2

In [9]:
```
data1.calcolo_media_primi_n(n = 2)
```

Out[9]:
```
15.0
```

Vediamo che l'attributo media non è stato modificato

In [10]:
```
data1.media
```

Out[10]:
```
20.0
```

Print di un oggetto

Proviamo ad eseguire print dell'oggetto precedente

In [11]:
```
print(data1)
```

```
<__main__.Dataset object at 0x000001B927C37CB0>
```

Ricreiamo la classe Dataset aggiungendo i metodi **str** e **repr**

In [12]:
```
class Dataset:
    #Metodo inizializzatore
    def __init__(self, data):
        self.data = data  # attributo dell'oggetto
        self.numero_righe = len(data) # attributo dell'oggetto

    #Metodo media
    def calcolo_media(self):
        media =  sum(self.data) / len(self.data)
        self.media =  media
        return media

    #Metodo per print
    def __str__(self):
        return (
            f"Elenco di dati di lunghezza {self.numero_righe}\n"
            f"Il primo elemento è {self.data[0]}\n"
            f"L'ultimo elemento è {self.data[-1]}\n"
        )

    #Metodo per debug
    def __repr__(self):
        return f"lista di {self.numero_righe}: {self.data}"
```

Instanziamo la classe in un oggetto ed eseguiamo il print

In [13]:
```
data1 = Dataset([10, 20, 30])
```

Eseguiamo la print

In [14]:
```
print(data1)
```

```
Elenco di dati di lunghezza 3
Il primo elemento è 10
L'ultimo elemento è 30
```

eseguiamo semplicemente data1

In [15]:
```
data1
```

Out[15]:
```
lista di 3: [10, 20, 30]
```

Documentazione delle Classi

La docstring serve per documentare classe e metodi, utile in progetti collaborativi.

In [16]:
```
class Dataset:
    """
    Classe per rappresentare un dataset semplice.

    Attributi:
        data (list): lista di valori numerici

    Metodi:
        mean(): calcola la media dei valori
    """
    def __init__(self, data):
        self.data = data

    def mean(self):
        """Calcola la media dei valori nel dataset"""
        return sum(self.data) / len(self.data)
```

In [17]:
```
help(Dataset)
```

```
Help on class Dataset in module __main__:

class Dataset(builtins.object)
 |  Dataset(data)
 |
 |  Classe per rappresentare un dataset semplice.
 |
 |  Attributi:
 |      data (list): lista di valori numerici
 |
 |  Metodi:
 |      mean(): calcola la media dei valori
 |
 |  Methods defined here:
 |
 |  __init__(self, data)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  mean(self)
 |      Calcola la media dei valori nel dataset
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables
 |
 |  __weakref__
 |      list of weak references to the object
```

Ereditarietà

Ricreiamo la classe Dataset

In [18]:
```
class Dataset:
    def __init__(self, data):
        self.data = data
        self.numero_righe = len(data)

    def mean(self):
        return sum(self.data) / len(self.data)
```

Creiamo ora una sotto-classe o classe derivata

In [19]:
```
class AdvancedDataset(Dataset):
    def median(self):
        sorted_data = sorted(self.data)
        n = len(sorted_data)
        mid = n // 2
        if n % 2 != 0:
            mediana =  sorted_data[mid]
        else:
            mediana = (sorted_data[mid-1]+sorted_data[mid])/2
        self.mediana = mediana
        return mediana
```

Questa classe eredita tutti i metodi di Dataset e ne aggiunge di nuovi

In [20]:
```
ad = AdvancedDataset([0, 20, 30, 40])
```

In [21]:
```
print(ad.mean())   # eredita mean()
print(ad.median()) # 25.0
```

```
22.5
25.0
```

Se nella sotto-classe usiamo un metodo con lo stesso nome della sovra-classe (anche __init) esso verrà sovrascritto

In [22]:
```
class AdvancedDataset(Dataset):
    def __init__(self, data):
        self.data = data
        self.sorted_data = sorted(data)
```

In [23]:
```
ad = AdvancedDataset([10, 20, 30, 40])
```

In [24]:
```
ad.data, ad.sorted_data
```

Out[24]:
```
([10, 20, 30, 40], [10, 20, 30, 40])
```

Nella prossima istruzione avrò un errore

In [25]:
```
ad.numero_righe
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[25], line 1
----> 1 ad.numero_righe

AttributeError: 'AdvancedDataset' object has no attribute 'numero_righe'
```

Gli altri metodo sono eridati

In [26]:
```
ad.mean()
```

Out[26]:
```
25.0
```

All'interno di un metodo della sotto-classe posso lanciare uno della sopra-classe. Molto utile se voglio aggiungere alcune passaggi

In [27]:
```
class AdvancedDataset(Dataset):
    def __init__(self, data):
        super().__init__(data)
        self.sorted_data = sorted(data)
```

In [28]:
```
ad = AdvancedDataset([10, 20, 30, 40])
```

Ora ho tutti gli attributi!

In [29]:
```
ad.data, ad.sorted_data, ad.numero_righe
```

Out[29]:
```
([10, 20, 30, 40], [10, 20, 30, 40], 4)
```

Esempio

In [30]:
```
class Persona():
    def __init__(self,nome,cognome,eta):
        self.nome = nome
        self.cognome = cognome
        self.eta = eta
        self.denominazione = f"{nome} {cognome}"
        self.pianeta = "Terra"

    def modifica_eta_compleanno(self):
        self.eta = self.eta + 1
        return "eta modificata"

    def valorizza_squadra_del_cuore(self, squadra):
        self.squadra = squadra

    def __str__(self):
        return f"{self.denominazione}"
```

In [31]:
```
class DataAnalyst(Persona):
    def __init__(self,nome,cognome,eta,competenza):
        super().__init__(nome,cognome,eta)
        self.competenza = competenza
        self.pianeta = "Giove"

    def modifica_competenza(self,materia):
        self.competenza = materia
```

In [32]:
```
NicolaDA = DataAnalyst(nome = "Nicola", cognome="Iantomasi", eta = 3, competenza = "SQL")
```

In [33]:
```
NicolaDA.pianeta
```

Out[33]:
```
'Giove'
```

In [ ]: