**OBJECT ORIENTED PROGRAMMING**

**INTRODUZIONE**

La programmazione OOP è un tipo di programmazione più comodo della programmazione procedurale, ossia della programmazione vista finora.

**CLASSI**

Per creare una classe in python si utilizz la keyword class. Al suo interno si crea un costruttore, contenente le VARIABILI della classe. Tale costruttore si inizializza tramite la keyword def, keyword già utilizzata per l’introduzione delle funzioni. Vediamo un esempo banale di classe.

Codice Python:

class lavoratore:

    def \_\_init\_\_(self):

        self.nome = "Arnaldo"

        self.cognome="Blablo"

        self.data\_di\_nascita = 1980

Per accedere alla classe, si utilizza la seguente sintassi strana

Codice Python:

class lavoratore:

    def \_\_init\_\_(self):

        self.nome = "Arnaldo"

        self.cognome="Blablo"

        self.data\_di\_nascita = 1980

lavoratore = lavoratore()

print(lavoratore)

print()

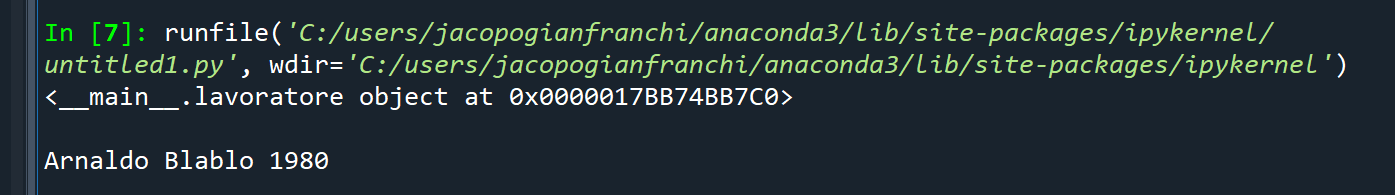
nome\_lavoratore = lavoratore.nome

cognome\_lavoratore = lavoratore.cognome

nascita\_lavoratore = lavoratore.data\_di\_nascita

print(nome\_lavoratore,cognome\_lavoratore,nascita\_lavoratore)

Console:



Solitamente i valori non vengono definiti all’interno della classe, ma vengono passati come parametri.

Codice Python:

    def \_\_init\_\_(self,nome,cognome,data\_di\_nascita):

        self.nome = "Arnaldo"

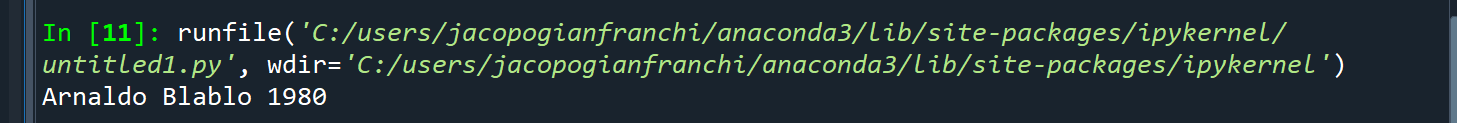
        self.cognome="Blablo"

        self.data\_di\_nascita = 1980

lavoratore1 = lavoratore("mario","Bla",1290)

print(lavoratore1.nome,lavoratore1.cognome,lavoratore1.data\_di\_nascita)

Console:



Si noti che, anche se abbiamo passato dei valore per i parametri nome, cognome, età, sono stati stampati i valori di default. Questo è normale, perché abbiamo forzato il valore all’interno della classe. Per rendere dinamici i valori, si passano gli elementi del costruttore dopo self.valore = :

Codice Python

class lavoratore:

    def \_\_init\_\_(self,nome,cognome,data\_di\_nascita):

        self.nome = nome

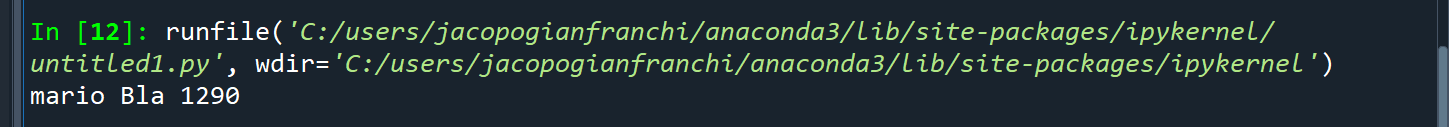
        self.cognome=cognome

        self.data\_di\_nascita = data\_di\_nascita

lavoratore1 = lavoratore("mario","Bla",1290)

print(lavoratore1.nome,lavoratore1.cognome,lavoratore1.data\_di\_nascita)

Console:



Finora abbiamo inserito solo variabili all’interno della classe.

Possiamo anche creare metodi e richiamarli all’esterno.   
Costruiamo quindi una classe che presenta sia variabili sia metodi, e richiamiamo i metodi al loro interno:

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10,30)

# Calcola area del rettangolo:

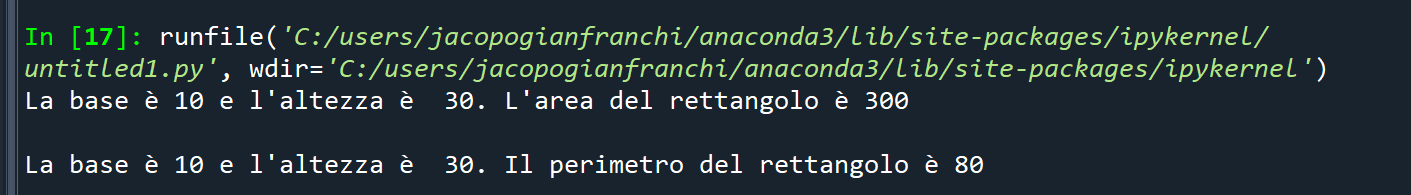
print(rettangolo1.area\_rettangolo())

print()

# Calcola area del rettangolo:

print(rettangolo1.perimetro\_rettangolo())

Console:



Nell’esempio precedente abbiamo considerato un solo rettangolo. Il bello della programmazione ad oggetti è che possiamo calcolare perimetro ed area di quanti rettangolo vogliamo!

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

print("Area e perimetro del rettangolo 1")

rettangolo1 = rettangolo(10,30)

# Calcola area del rettangolo:

print(rettangolo1.area\_rettangolo())

print()

# Calcola area del rettangolo:

print(rettangolo1.perimetro\_rettangolo())

print()

print()

print("Area e perimetro del rettangolo 2")

rettangolo2 = rettangolo(1,12.8)

# Calcola area del rettangolo:

print(rettangolo2.area\_rettangolo())

print()

# Calcola area del rettangolo:

print(rettangolo2.perimetro\_rettangolo())

print()

print()

print("Area e perimetro del rettangolo 3")

rettangolo3 = rettangolo(7,13)

# Calcola area del rettangolo:

print(rettangolo3.area\_rettangolo())

print()

# Calcola area del rettangolo:

print(rettangolo3.perimetro\_rettangolo())

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

RICAPITOLANDO:

Una classe è composta da attributi e metodi. Gli attributi vengono definiti STATI, e nel nostro esempio sono base e altezza. I metodi definiscono il comportamento delle istanze, e, nel nostro esempio, sono area\_rettangolo e perimetro\_rettangolo.

**VALORI DI DEFAULT NELLE CLASSI**

Riprendiamo la nostra classe “rettangolo”:

class rettangolo:

    def \_\_init\_\_(self,base,altezza):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

I parametri sono base e altezza. Ricordiamo che self è un parametro obbligatorio, che rappresenta i valori inseriti nel momento del richiamo di una classe.

Ricordiamo che, per creare un oggetto di tipo “rettangolo”, l’utente deve passare due parametri: Il primo indica la base, e il secondo indica l’altezza:

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

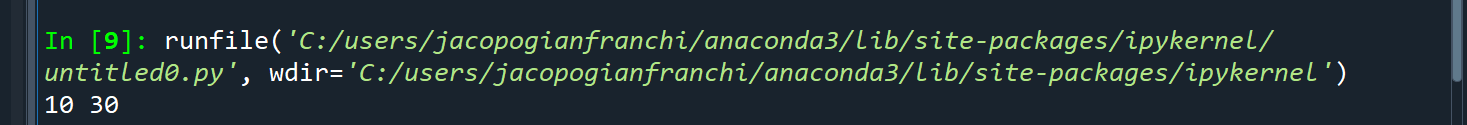
    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10,30)

print(rettangolo1.base, rettangolo1.altezza)

Console:



Cosa succede se passiamo un numero di parametri inferiore a 2? Python restituirà errore:

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10)

print(rettangolo1.base)

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Per risolvere questo problema possiamo inserire un valore di default all’interno della firma del costruttore. In questo modo, se l’ultimo parametro non viene passato, in automatico sarà passato il valore di default.

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza = 73):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

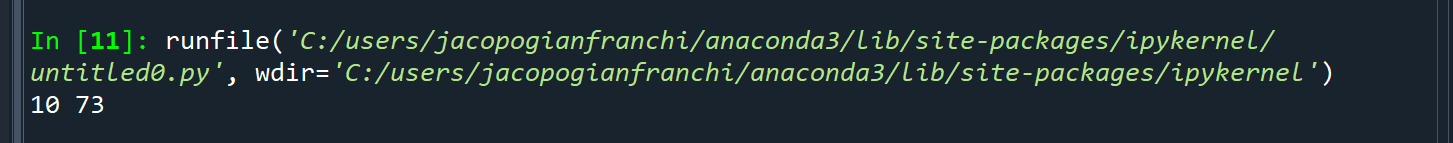
    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10)

print(rettangolo1.base,rettangolo1.altezza)

Console:



CVD.

Ovviamente, se modificassim il valore di altezza della nostra classe, il valore di default non verrebbe considerato.

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza = 73):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

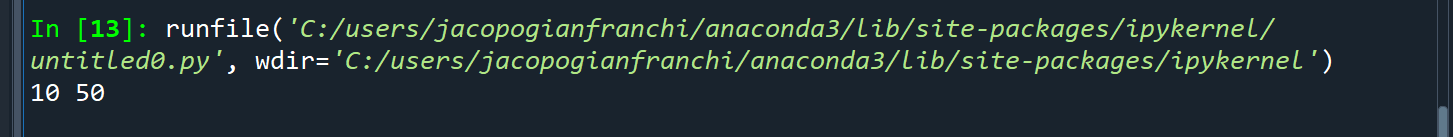
    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10,50)

print(rettangolo1.base,rettangolo1.altezza)

Console:



**PASSAGGIO DEI VALORI IN ORDINE DIVERSO**

Abbiamo finora passato i valori da passare alla nostra istanza seguendo lo stesso ordine presente nel costruttore. In realtà, se specifichiamo il nome dei parametri al momento della creazione del nostro oggetto (istanza), possiamo anche passare gli argomenti in ordine diverso. Vediamo un semplice esempio.

Codice Python:

class chitarra:

    def \_\_init\_\_(self,modello,marca,numero\_corde):

        self.modello=modello

        self.marca=marca

        self.numero\_corde=numero\_corde

c1 = chitarra(numero\_corde = 19,modello="Fender", marca="Telecaster")

c2 = chitarra("Telecaster","Fender",10)

print(c1.numero\_corde,c1.modello,c1.marca)

print(c1.modello,c1.marca,c1.numero\_corde)

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**TIPOLOGIA DELLE ISTANZE**

Ovviamente, la tipologia delle istanze di una classe è uguale alla classe stessa. Riprendiamo il nostro esempio e vediamo questo fenomeno nel concreto:

Codice Python:

class rettangolo:

    def \_\_init\_\_(self,base,altezza = 73):

        self.base=base

        self.altezza=altezza

    def area\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. L'area del rettangolo è {self.base\*self.altezza}"

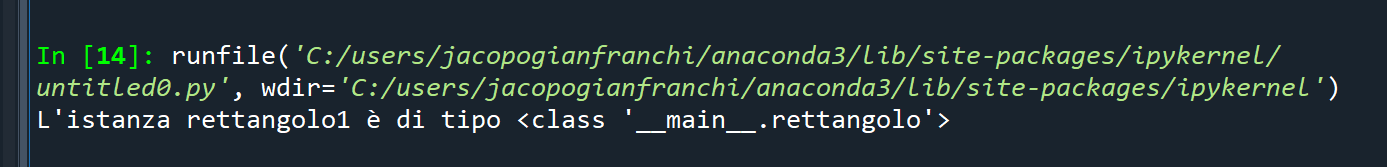
    def perimetro\_rettangolo(self):

        return f"La base è {self.base} e l'altezza è  {self.altezza}. Il perimetro del rettangolo è {self.base\*2 + self.altezza\*2}"

rettangolo1 = rettangolo(10,50)

print(f"L'istanza rettangolo1 è di tipo {type(rettangolo1)}")

Console:



Quanto visto finora espande i concetti di variabili e valori visti finora. Consideriamo i seguenti esempi banali:

Codice Python:

x = 4

y = "Ciao"

z = True

h = 8.3

print(f"{x} è un oggetto di tipo {type(x)}. Questo significa che x è un'istanza della classe {type(x)}")

print()

print()

print(f"{y} è un oggetto di tipo {type(y)}. Questo significa che y è un'istanza della classe {type(y)}")

print()

print()

print(f"{z} è un oggetto di tipo {type(z)}. Questo significa che z è un'istanza della classe {type(z)}")

print()

print()

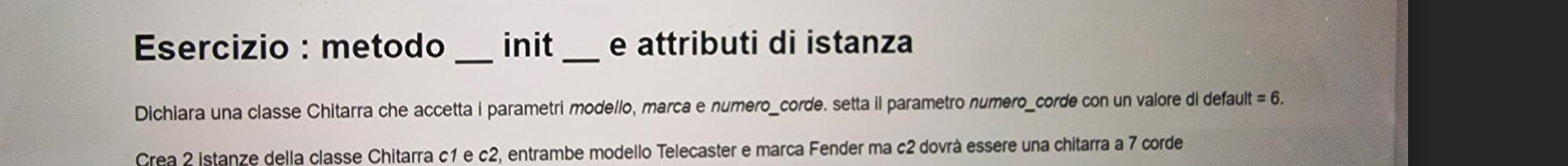
print(f"{h} è un oggetto di tipo {type(h)}. Questo significa che h è un'istanza della classe {type(h)}")

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

ESERCIZIO



Codice Python:

class chitarra:

    def \_\_init\_\_(self,modello,marca,numero\_corde=6):

        self.modello=modello

        self.marca=marca

        self.numero\_corde=numero\_corde

c1 = chitarra("Telecaster","Fender")

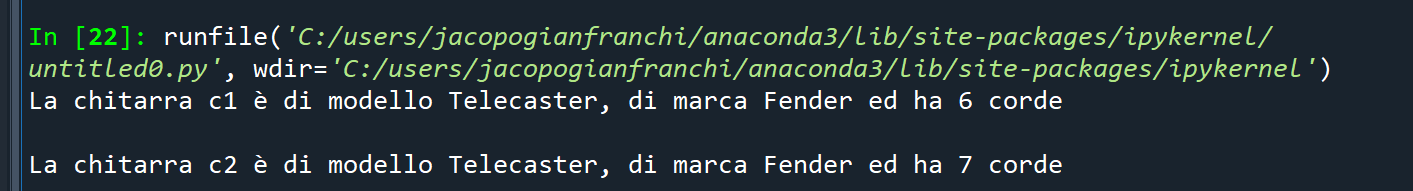
c2 = chitarra("Telecaster","Fender",7)

print(f"La chitarra c1 è di modello {c1.modello}, di marca {c1.marca} ed ha {c1.numero\_corde} corde")

print()

print(f"La chitarra c2 è di modello {c2.modello}, di marca {c2.marca} ed ha {c2.numero\_corde} corde")

Console:



**METODI E PASSAGGIO ARGOMENTI DALL’ESTERNO**

Abbiamo finora visto delle classi con metodi che accettano parametri presi dalla classe stessa. Come sappiamo, una classe è formata da attributi e metodi. Il parametro passato all’interno dei nostri metodi all’interno delle nostri classi è stato finora solo uno, ossia il parametro self. In questo modo, i metodi interni alle nostre classi possono processare e/o ritornare solo valori riguardanti le variabili all’interno del nostro costruttore.

Creiamo ora una nuova classe, e creiamo al suo interno un metodo che accetta sia i valori interni alla classe stessa (ossia i parametri inizianti con self) sia valori inseriti dall’utente. L’esempio qui sotto chiarirà il tutto.

Codice Python:

class calcolatrice:

    def \_\_init\_\_(self,numero1,numero2,numero3):

        self.numero1=numero1

        self.numero2=numero2

        self.numero3=numero3

    def somma\_numeri(self,numero\_esterno):

        return self.numero1 + self.numero2 + self.numero3 + numero\_esterno

calcolatrice = calcolatrice(19,15,5)

print(calcolatrice.somma\_numeri(50))

Console:



Al momento della creazione dell’oggetto calcolatrice passiamo i tre numeri da inserire nel costruttore: numero1 = 19, numero2 = 15, numero3 = 5.

Il metodo somma\_numeri accetta tutti i parametri passati come argomento del costruttore, ossia i parametri passati al momento di creazione dell’oggetto, più un altro parametro chiamato numero\_esterno. Questo parametro si passa al momento del richiamo della funzione somma\_numeri.

Nel nostro esempio, la somma sarà data dalla somma dei tre numeri passati nel costruttore, ossia 19+5+15, più il numero passato come argomento della funzione somma\_numeri, ossia di 50.

La somma generale sarà quindi 89.

**ERSERCIZIO**

