**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()

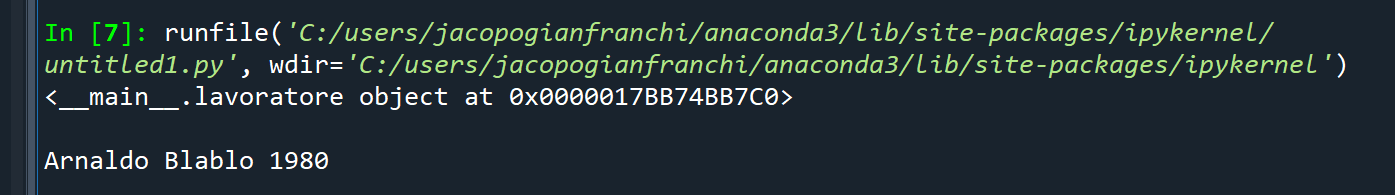
nome\_lavoratore = lavoratore.nome

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"

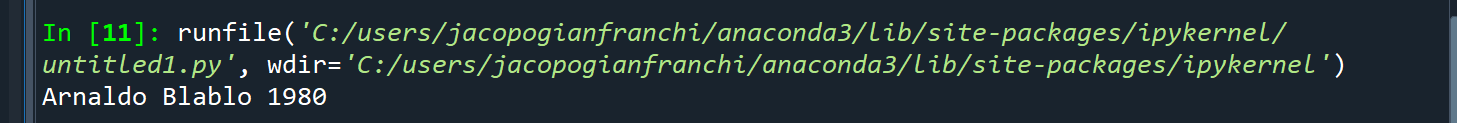
        self.cognome="Blablo"

        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

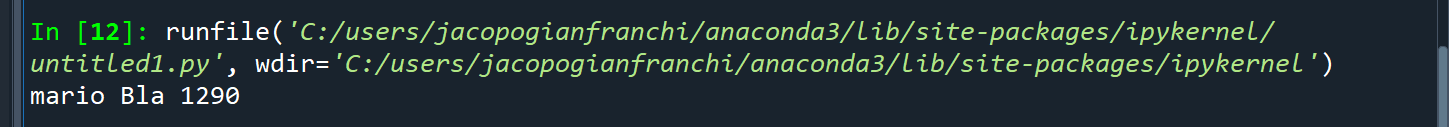
        self.cognome=cognome

        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:

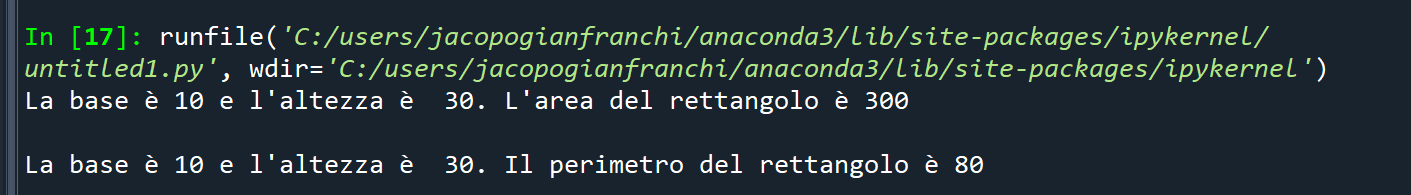
print(rettangolo1.area\_rettangolo())

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}"

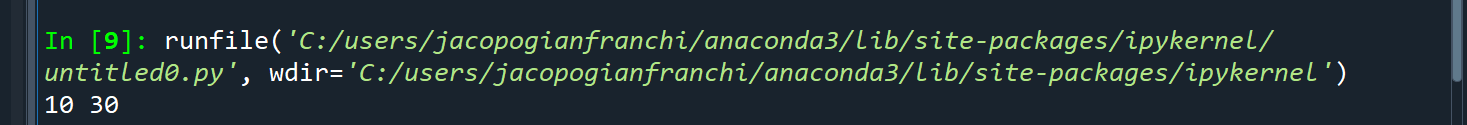
    def perimetro\_rettangolo(self):

        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}"

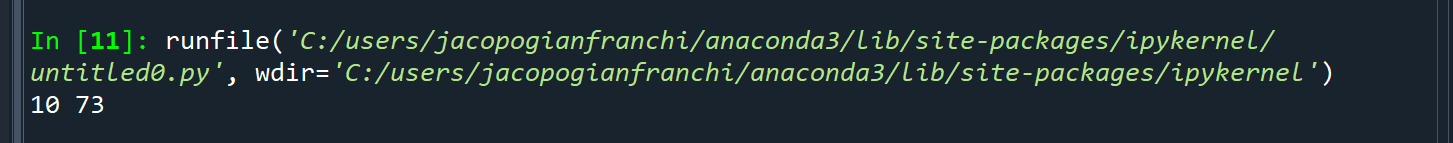
    def perimetro\_rettangolo(self):

        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}"

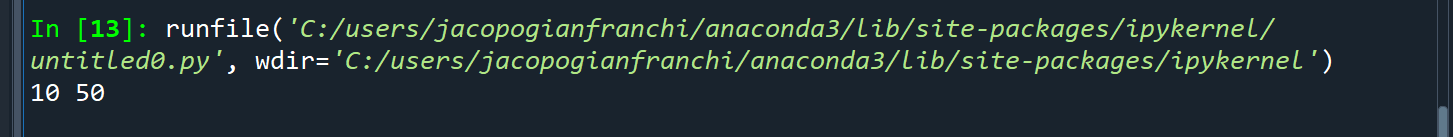
    def perimetro\_rettangolo(self):

        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}"

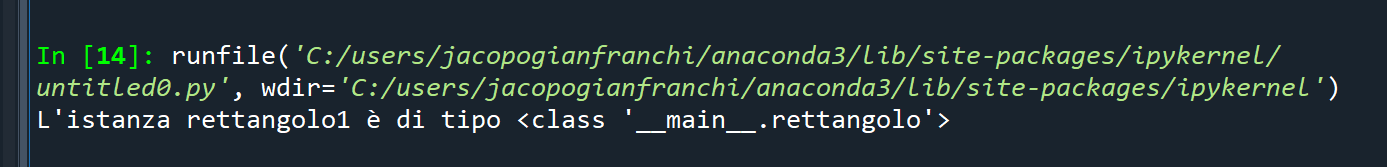
    def perimetro\_rettangolo(self):

        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()

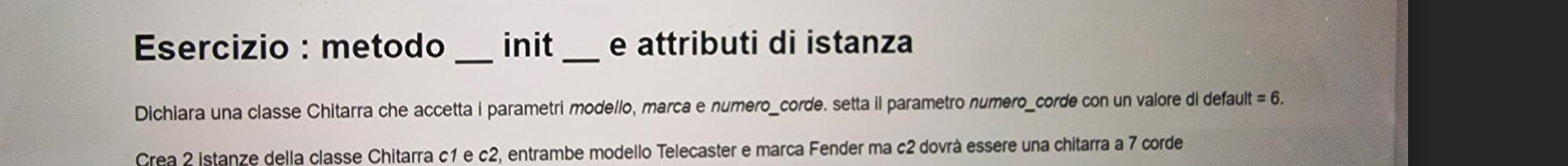
print(f"{h} è un oggetto di tipo {type(h)}. Questo significa che h è un'istanza della classe {type(h)}")

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")

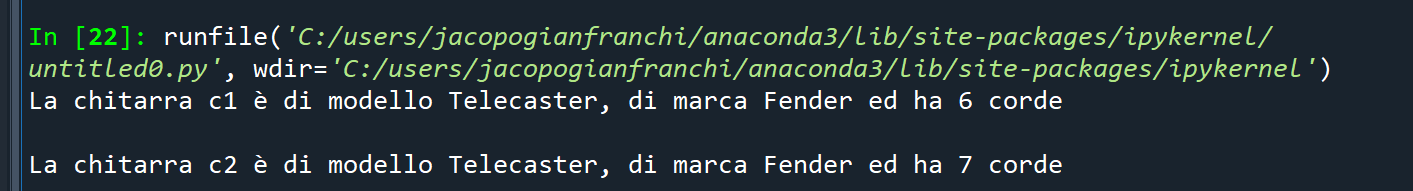
c2 = chitarra("Telecaster","Fender",7)

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:



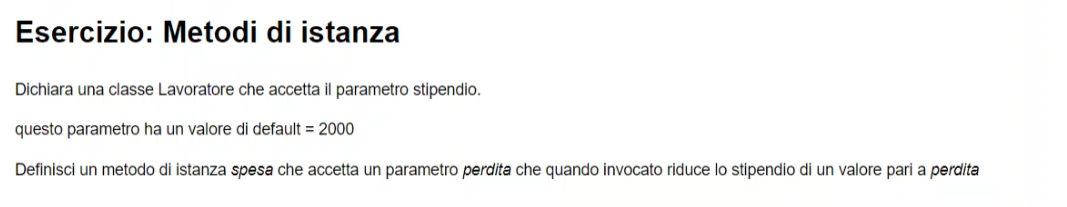
Al momento della creazione dell’oggetto calcolatrice passiamo i tre numeri da inserire nel costruttore: numero1 = 19, numero2 = 15, numero3 = 5.

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.

**ESERCIZIO**



Codice Python:

class lavoratore:

    def \_\_init\_\_(self,stipendio=2000):

        self.stipendio=stipendio

    def spesa(self,perdita):

        self.stipendio =  self.stipendio - perdita

        if self.stipendio < 0:

            return "Il tuo stipendio è 0. Non puoi più effettuare spese!"

        else:

            return self.stipendio

lavoratore\_mario = lavoratore(15000)

dopo\_spesa1 = lavoratore\_mario.spesa(160)

print(dopo\_spesa1)

dopo\_spesa2 = lavoratore\_mario.spesa(175.14)

print(dopo\_spesa2)

dopo\_spesa3 = lavoratore\_mario.spesa(55.90)

print(dopo\_spesa3)

dopo\_spesa4 = lavoratore\_mario.spesa(490)

print(dopo\_spesa4)

dopo\_spesa5 = lavoratore\_mario.spesa(138.68)

print(dopo\_spesa5)

dopo\_spesa6 = lavoratore\_mario.spesa(89.22)

print(dopo\_spesa6)

dopo\_spesa7 = lavoratore\_mario.spesa(300)

print(dopo\_spesa7)

dopo\_spesa7 = lavoratore\_mario.spesa(13000)

print(dopo\_spesa7)

dopo\_spesa7 = lavoratore\_mario.spesa(1000)

print(dopo\_spesa7)

Console:

Immagine che contiene testo, software, Carattere, Software multimediale

Descrizione generata automaticamente

**ATTRIBUTI DI CLASSE**

Gli attributi visti finora sono definiti **attributi di istanza.** Questi attributi variano da istanza a istanza. Vediamo un banale esempio per ripassare

Codice Python:

class macchina:

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

macchina1 = macchina("Wolksvagen","Rossa")

macchina2 = macchina("Audi","Rossa")

macchina3 = macchina("Wolksvagen","Blu")

macchina4 = macchina("Fiat","Rossa")

macchina5 = macchina("Chevrolet","Gialla")

print(f"La {macchina1.modello} è {macchina1.colore}")

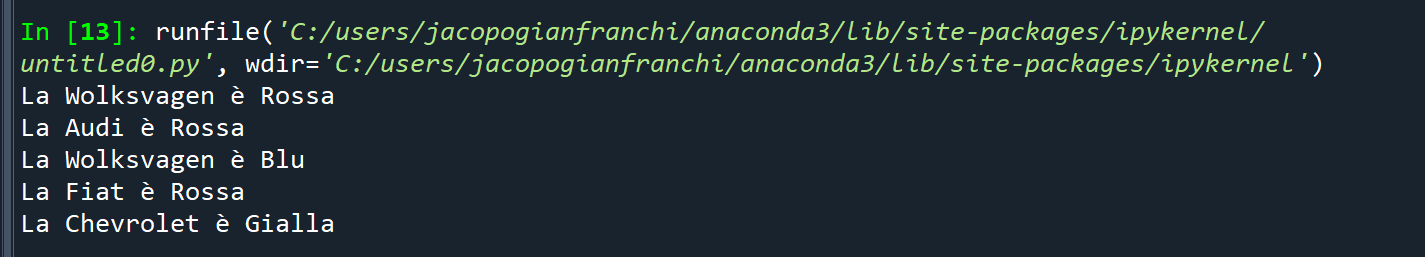
print(f"La {macchina2.modello} è {macchina2.colore}")

print(f"La {macchina3.modello} è {macchina3.colore}")

print(f"La {macchina4.modello} è {macchina4.colore}")

print(f"La {macchina5.modello} è {macchina5.colore}")

Console:



Al contrario degli attributi di istanza, che nel nostro esempio erano modello e colore, gli **attributi di classe** sono attributi disponibili per tutte le istanze. Vediamo anche in questo caso un banale esempio, considerando la classe precedente macchina:

Codice Python:

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

macchina1 = macchina("Wolksvagen","Rossa")

macchina2 = macchina("Audi","Rossa")

macchina3 = macchina("Wolksvagen","Blu")

macchina4 = macchina("Fiat","Rossa")

macchina5 = macchina("Chevrolet","Gialla")

print(f"La {macchina1.modello} è {macchina1.colore}. La {macchina1.modello} ha {macchina.numero\_ruote} ruote.")

print(f"La {macchina2.modello} è {macchina2.colore}. La {macchina2.modello} ha {macchina.numero\_ruote} ruote.")

print(f"La {macchina3.modello} è {macchina3.colore}. La {macchina3.modello} ha {macchina.numero\_ruote} ruote.")

print(f"La {macchina4.modello} è {macchina4.colore}. La {macchina4.modello} ha {macchina.numero\_ruote} ruote.")

print(f"La {macchina5.modello} è {macchina5.colore}. La {macchina5.modello} ha {macchina.numero\_ruote} ruote.")

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

La sintassi per estrarre un attributo di classe, quindi, è molto semplice. E’ sufficiente scrivere il nome della classe punto nome della variabile di classe: nel nostro esempio, macchina.numero\_ruote.

E’ anche possibile selezionare la nostra variabile di classe attraverso un’istanza: Nel nostro esempio precedente:

Codice Python:

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

macchina1 = macchina("Wolksvagen","Rossa")

macchina2 = macchina("Audi","Rossa")

macchina3 = macchina("Wolksvagen","Blu")

macchina4 = macchina("Fiat","Rossa")

macchina5 = macchina("Chevrolet","Gialla")

print(f"La {macchina1.modello} è {macchina1.colore}. La {macchina1.modello} ha {macchina1.numero\_ruote} ruote.")

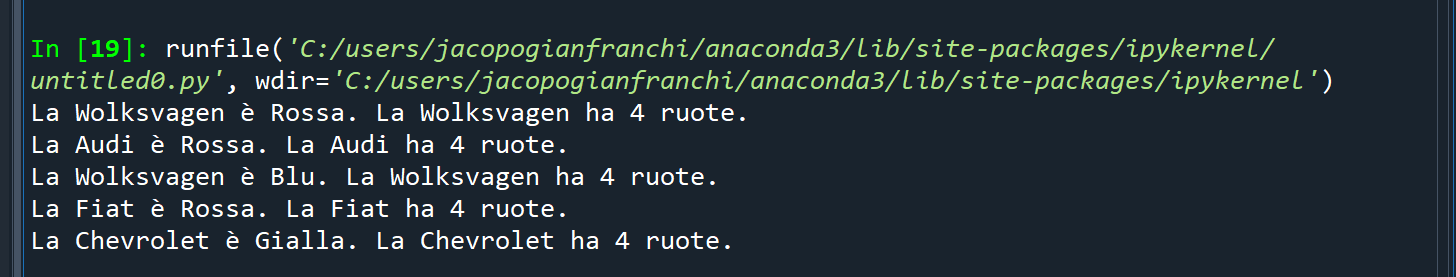
print(f"La {macchina2.modello} è {macchina2.colore}. La {macchina2.modello} ha {macchina2.numero\_ruote} ruote.")

print(f"La {macchina3.modello} è {macchina3.colore}. La {macchina3.modello} ha {macchina3.numero\_ruote} ruote.")

print(f"La {macchina4.modello} è {macchina4.colore}. La {macchina4.modello} ha {macchina4.numero\_ruote} ruote.")

print(f"La {macchina5.modello} è {macchina5.colore}. La {macchina5.modello} ha {macchina5.numero\_ruote} ruote.")

Console:



CVD.

Supponiamo ora di creare un metodo all’interno della nostra classe che va a modificare la nostra variabile di classe.

Supponiamo di trovarci in un mondo nel quale sia possibile aumentare o diminuire le ruote di un’auto (Che cavolata!!!). Creiamo nella nostra classe macchina un metodo per aumentare le ruote ed uno per decrementarlo.

Codice Python:

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

Il codice precedente mostra che per utilizzare una variabile di classe all’interno di un metodo bisogna inserirlo all’interno di esso con il nome stesso della classe.

Ricordiamo, inoltre, che i metodi qui scritti sono metodi di istanza: Non è possibile quindi accedere ad essi utilizzando direttamente il nome della classe, ma bisogna accedere utilizzando una sua istanza.

Per prima cosa accediamo alla nostra variabile di classe: come sappiamo, possiamo accedervi sia tramite la classe sia tramite una sua istanza:

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

macchina1 = macchina("Fiat","Nera")

numero\_ruote\_macchina = macchina.numero\_ruote

# Oppure

numero\_ruote\_macchina1 = macchina1.numero\_ruote

print(numero\_ruote\_macchina, numero\_ruote\_macchina)

Console:



Cerchiamo ora di accedere al nostro metodo di istanza “diminuisci\_ruote”.

Essendo un metodo di istanza, non è possibile accedere ad esso attraverso il nome della classe, come accade con le variabili di classe.

Proviamo infatti ad accedere al metodo diminuisci\_ruote tramite la nostra classe: otterremo un messaggio di errore.

Codice Python

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

        return macchina.diminuisci\_ruote

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

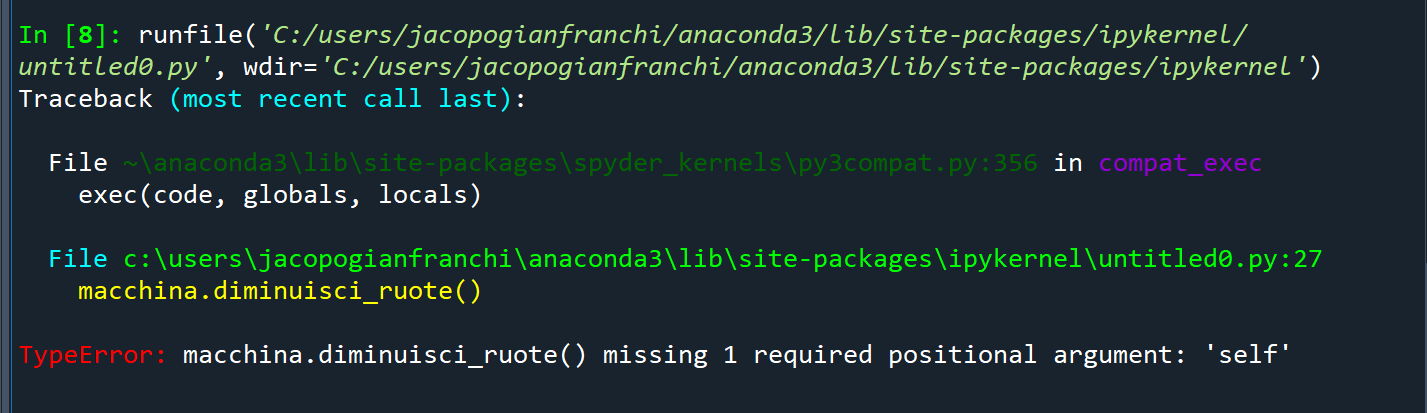
        return macchina.diminuisci\_ruote

macchina1 = macchina("Fiat","Nera")

macchina.diminuisci\_ruote()

print(macchina.numero\_ruote, macchina1.numero\_ruote)

Console:



Da quanto si evince dal codice, il metodo di istanza diminuisci\_ruote prende la variabile di classe numero\_ruote e la decrementa di 1. Quello che abbiamo fatto è stato tentare di accedere a tale metodo utilizzando il nome della classe: in altre parole, abbiamo considerato questo metodo come metodo di classe. Purtroppo, il metodo diminuisci\_ruote è un metodo di istanza 🡺 possiamo accedere ad esso solo tramite le istanza. Una volta richiamato questo metodo tramite un’istanza, la variabile numero\_ruote subirà un decremento. A questo punto, sarà possibile accedere al nuovo valore di numero\_ruote sia tramite classe sia tramite istanza. Il codice qui sotto chiarirà il tutto.

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

        return macchina.diminuisci\_ruote

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

        return macchina.diminuisci\_ruote

macchina1 = macchina("Fiat","Nera")

macchina2 = macchina("Chevrolet","Rossa")

print(f"Valore dell'attributo \"numero\_ruote\" in caso di accesso tramite classe macchina: {macchina.numero\_ruote}.", f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina1 della classe macchina: {macchina1.numero\_ruote}.",f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina2 della classe macchina: {macchina2.numero\_ruote}.",)

print()

print("----------------------------------------------------")

print()

print("Decrementiamo ora la variabile \"numero\_ruote\" accedendo al metodo diminusci\_ruote attraverso l'istanza macchina1 della classe macchina. Si ricorda che accedere al metodo \"diminuisci\_ruote\" tramite l'istanza macchina1 o macchina2 è indifferente: ")

print()

print()

macchina1.diminuisci\_ruote()

print(f"Valore dell'attributo \"numero\_ruote\" in caso di accesso tramite classe macchina: {macchina.numero\_ruote}.", f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina1 della classe macchina: {macchina1.numero\_ruote}.",f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina2 della classe macchina: {macchina2.numero\_ruote}.",)

print()

print("----------------------------------------------------")

print()

print("Decrementiamo ora ulteriormente la variabile numero\_ruote accedendo al metodo \"diminusci\_ruote\" attraverso l'istanza macchina2 della classe macchina. Si ricorda che accedere al metodo \"diminuisci\_ruote\" tramite l'istanza macchina1 o l'istanza macchina2 è indifferente': ")

print()

print()

macchina2.diminuisci\_ruote()

print(f"Valore dell'attributo \"numero\_ruote\" in caso di accesso tramite classe macchina: {macchina.numero\_ruote}.", f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina1 della classe macchina: {macchina1.numero\_ruote}.",f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina2 della classe macchina: {macchina2.numero\_ruote}.",)

print()

print("----------------------------------------------------")

print()

print("Infine, incrementiamo la variabile \"numero\_ruote\" accedendo al metodo \"aumenta\_ruote\" attraverso l'istanza macchina2 della classe macchina. Si ricorda che accedere al metodo \"aumenta\_ruote\" tramite l'istanza macchina1 o macchina12 è indifferente: ")

print()

print()

macchina2.aumenta\_ruote()

print(f"Valore dell'attributo \"numero\_ruote\" in caso di accesso tramite classe macchina: {macchina.numero\_ruote}.", f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina1 della classe macchina: {macchina1.numero\_ruote}.",f"\n\nValore dell'attributo \"numero\_ruote\" in caso di accesso tramite istanza macchina2 della classe macchina: {macchina2.numero\_ruote}.",)

Console:

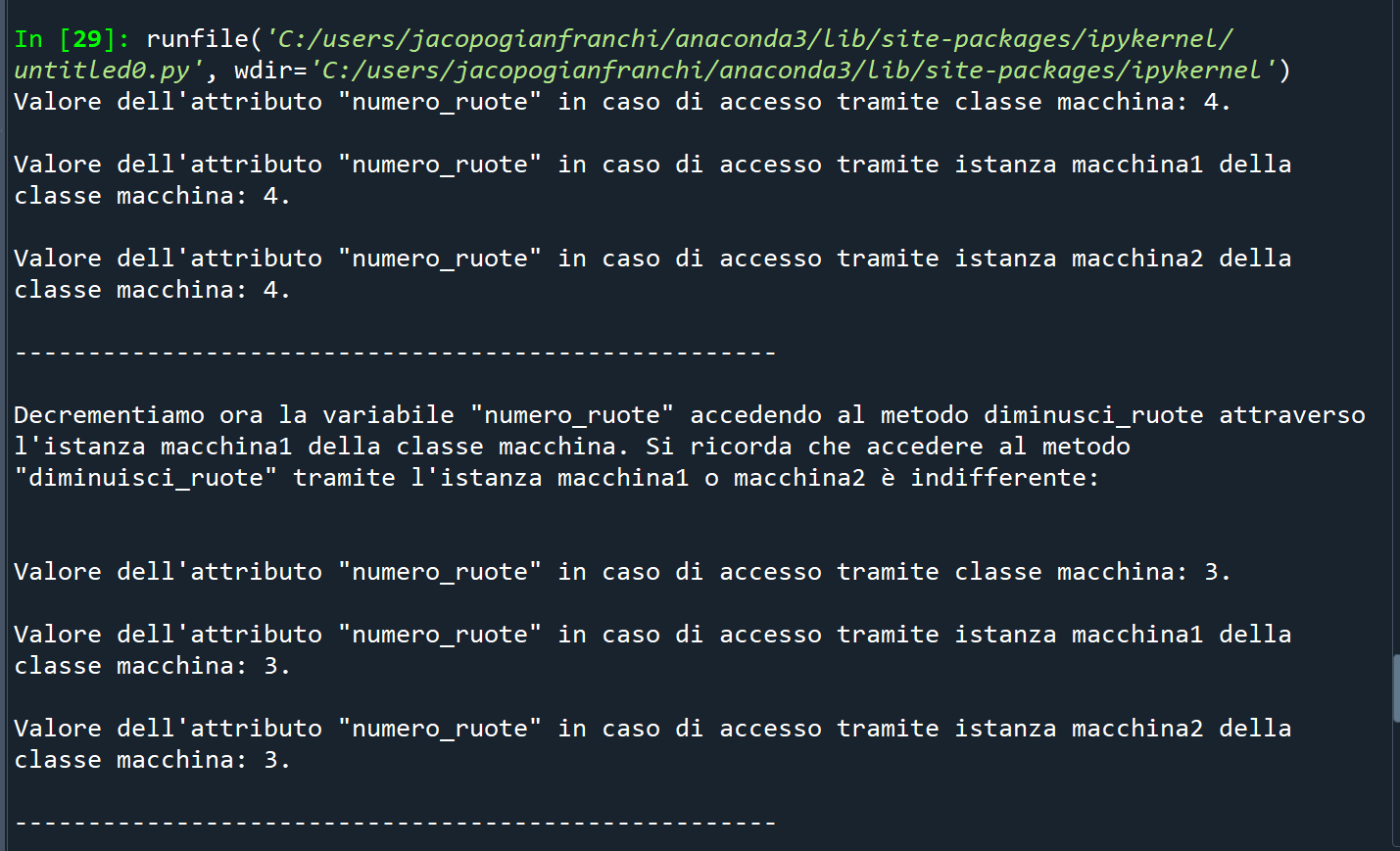


Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Ritorniamo alla nostra classe:

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

        return macchina.diminuisci\_ruote

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

        return macchina.diminuisci\_ruote

Abbiamo finora modificato la variabile “numero\_ruote” accedendo ai metodi aumenta\_ruote e diminuisci\_ruote. Cosa succede se forziamo il valore di numero\_ruote?

Dipende da come vi accediamo. Poiché non è specificato che si accede tramite la classe stessa, la modifica della variabile numero\_ruote ha valore in base all’istanza, ma non alla classe. Un esempio renderà il tutto più chiaro.

Codice Python:

class macchina:

    numero\_ruote=4

    def \_\_init\_\_(self,modello,colore):

        self.modello=modello

        self.colore=colore

    def aumenta\_ruote(self):

        macchina.numero\_ruote +=1

        return macchina.diminuisci\_ruote

    def diminuisci\_ruote(self):

        macchina.numero\_ruote -=1

        return macchina.diminuisci\_ruote

macchina1 = macchina("Fiat","Nera")

macchina2 = macchina("Chevrolet","Rossa")

macchina.numero\_ruote = 10

print(macchina.numero\_ruote, macchina1.numero\_ruote,macchina2.numero\_ruote)

macchina1.numero\_ruote = 8

print(macchina.numero\_ruote, macchina1.numero\_ruote,macchina2.numero\_ruote)

Console:



Dal codice precedente si evince che accedendo alla nostra variabile tramite la classe e modificandola, la modifica verrà “presa” anche da tutte le istanze. Se invece modifichiamo la nostra variabile di classe tramite un’istanza, allora la modifica varrà solo per quell’istanza.

Vediamo ora un altro esempio e creiamo un contatore.

Consideriamo la classe carrello\_della\_spesa. Supponiamo che per ogni prodotto comprato si incrementi di 1 un contatore in modo da contare il numero di prodotti acquistati. Per ogni prodotto consideriamo il nome e il prezzo. La classe sarà la seguente:

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class carrello\_della\_spesa:

    numero\_prodotti=0

    def \_\_init\_\_(self,nome\_prodotto,prezzo):

        self.nome\_prodotto=nome\_prodotto

        self.prezzo=prezzo

    def incrementa\_prodotti(self):

        carrello\_della\_spesa.numero\_prodotti +=1

        return carrello\_della\_spesa.numero\_prodotti

prodotto1 = carrello\_della\_spesa("Banane",2.21)

prodotto1.incrementa\_prodotti()

print(carrello\_della\_spesa.numero\_prodotti)

prodotto2 = carrello\_della\_spesa("Zucchine",5.16)

prodotto2.incrementa\_prodotti()

print(carrello\_della\_spesa.numero\_prodotti)

prodotto3 = carrello\_della\_spesa("Cetrioli",3.40)

prodotto3.incrementa\_prodotti()

print(carrello\_della\_spesa.numero\_prodotti)

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Quanto visto è molto utile, ma è anche molto macchinoso.

Ogni volta che inseriamo un nuovo prodotto nel carrello, iinfatti, dobbiamo richiamare il metodo di istanza per amentare il conteggio dei prodotti presenti nel carrello.

Piccolo recap: I METODI DI ISTANZA SI POSSONO RICHIAMARE SOLO TRAMITE LE ISTANZE!!! NON TRAMITE LA CLASSE STESSA. QUESTO SARA’ POSSIBILE QUANDO STUDIEREMO I METODI DI CLASSE.

Per risolvere il problema di dover richiamare ogni volta il nostro metodo di istanza ogni volta che inseriamo un nuovo prodotto all’interno del carrello, è possibile inserire l’incremento del numero dei prodotti direttamente nel nostro costruttore. In questo modo il conteggio dei nuovi prodotti avverrrà in automatico ogni volta che si crea una istanza della nostra classe carrello\_della\_spesa.

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class carrello\_della\_spesa:

    numero\_prodotti=0

    def \_\_init\_\_(self,nome\_prodotto,prezzo):

        self.nome\_prodotto=nome\_prodotto

        self.prezzo=prezzo

        carrello\_della\_spesa.numero\_prodotti +=1

    def incrementa\_prodotti(self):

        carrello\_della\_spesa.numero\_prodotti +=1

        return carrello\_della\_spesa.numero\_prodotti

prodotto1 = carrello\_della\_spesa("Banane",2.21)

print(carrello\_della\_spesa.numero\_prodotti)

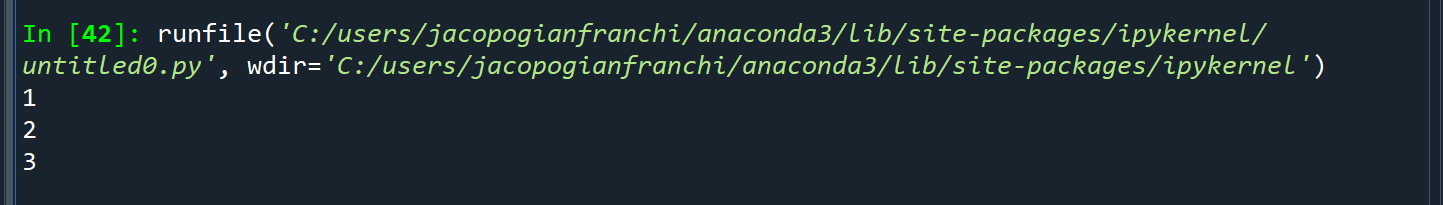
prodotto2 = carrello\_della\_spesa("Zucchine",5.16)

print(carrello\_della\_spesa.numero\_prodotti)

prodotto3 = carrello\_della\_spesa("Cetrioli",3.40)

print(carrello\_della\_spesa.numero\_prodotti)

Console:



Per concludere l’argomento sulle classi, vediamo cosa succede se inseriamo una variabile sia nel nostro costruttore sia come variabile di classe.

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class carrello\_della\_spesa:

    colore\_prodotto = "Rosso"

    def \_\_init\_\_(self,nome\_prodotto,prezzo,colore\_prodotto):

        self.nome\_prodotto=nome\_prodotto

        self.prezzo=prezzo

        self.colore\_prodotto = colore\_prodotto

    def descrizione\_prodotto(self):

        return f"Il prodotto {self.nome\_prodotto} costa {self.prezzo} ed è di colore {self.colore\_prodotto}"

print()

prodotto1 = carrello\_della\_spesa("Cereali","12€/Kg","Marroni")

print(prodotto1.descrizione\_prodotto())

print()

prodotto2 = carrello\_della\_spesa("Uova","3€/Kg","Arancioni")

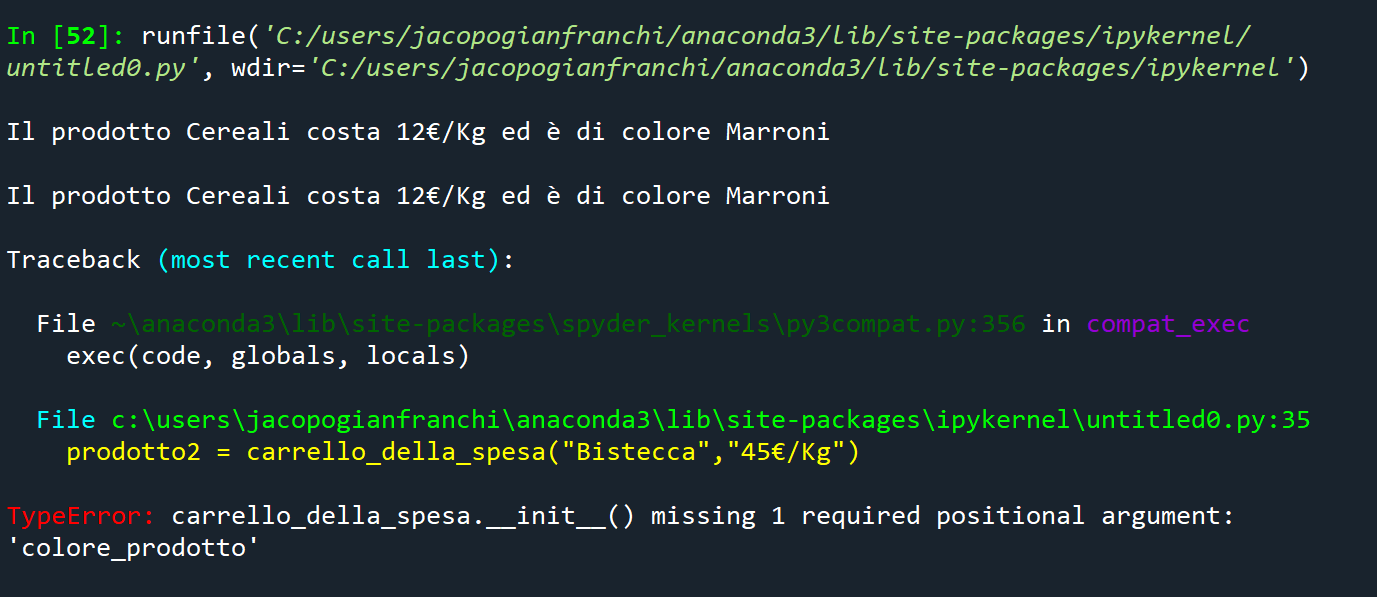
print(prodotto1.descrizione\_prodotto())

print()

prodotto2 = carrello\_della\_spesa("Bistecca","45€/Kg")

print(prodotto1.descrizione\_prodotto())

Console:



Nel caso di prodotto1 e prodotto2 il richiamo del metodo descrizione\_prodotto è andato a buon fine, perché abbiamo passato tutti e tre gli argomenti relativi ai parametri nel costruttore.

Perché nel caso di prodotto3 abbiamo ricevuto errore? La variabile di classe colore\_prodotto non serve a niente?

In realtà si, ma essa viene chiamata solo se non abbiamo una variabile corrispondente all’interno dei parametri del costruttore! Rimuoviamo quindi la variabile colore\_prodotto dal nostro costruttore.

Codice Python:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Jun 15 09:30:16 2023

@author: jacopogianfranchi

"""

class carrello\_della\_spesa:

    colore\_prodotto = "Rosso"

    def \_\_init\_\_(self,nome\_prodotto,prezzo):

        self.nome\_prodotto=nome\_prodotto

        self.prezzo=prezzo

    def descrizione\_prodotto(self):

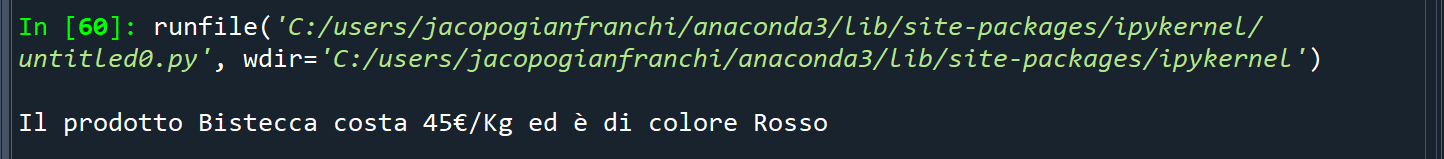
        return f"Il prodotto {self.nome\_prodotto} costa {self.prezzo} ed è di colore {self.colore\_prodotto}"

print()

prodotto3 = carrello\_della\_spesa("Bistecca","45€/Kg")

print(prodotto3.descrizione\_prodotto())

Console:



All’interno del metodo descrizione\_prodotto abbiamo quindi richiamato la variabile di clesse colore\_prodotto. Essa si trova nella classe e quindi non viene passata dall’esterno.

**METODI DI CLASSE**

Finora abbiamo utilizzato i metodi di istanza. Per accedere a questi metodi è necessario creare una istanza della nostra classe, e poi accedere al metodo. E’ possibile anche creare metodi di classe. Tramite il nome della classe, è possibile accedere direttamente a questi metodi.

Per creare un metodo di classe, bisogna considerare queste due regole.

1. Non si utilizza il parametro self nella creazione del metodo. Il parametro self serve per utilizzarlo all’interno del metodo nel caso in cui si desideri modificare/ritornare le variabili presenti nel costruttore. In questo caso si utilizza invece il parametro cls. Ovviamente questa è una convenzione. Potremmo tranquillamente passare il parametro self, ma si ricorda che self si riferisce alle variabili di istanza. cls si riferisce invece alle variabili di classe.
2. Essendo self o cls due nomi di convenzione, il cambio da self a cls da solo non serve a un tubo. Bisogna anche utilizzare il decoratore @classmethod subito sopra la firma del metodo. Un decoratore equivale ad una @annotaion in Java.

Un metodo creato con cls e decoratore @classmethod può riferirsi SOLO alle variabili di classe. Non può quindi trasformare/ritornare variabili di istanza. Non è quindi possibile creare metodi che utilizzano contemporaneamente variabili di istanza introdotte con self con variabili di classe introdotte con

Vediamo un semplice esempio.

Codice Python:

class carrello\_della\_spesa:

    colore\_prodotto = "Rosso"

    def \_\_init\_\_(self,nome\_prodotto,prezzo,colore\_prodotto):

        self.nome\_prodotto=nome\_prodotto

        self.prezzo=prezzo

        self.colore\_prodotto = colore\_prodotto

    @classmethod

    def colore\_dei\_prodotti(cls):

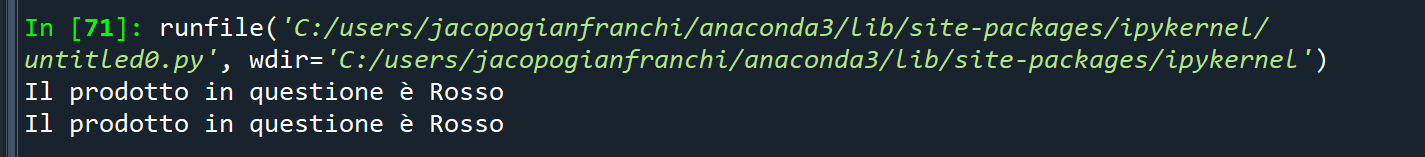
        return f"Il prodotto in questione è {cls.colore\_prodotto}"

prodotto1 = carrello\_della\_spesa("piselli","10.00€/Kg","Verde")

print(carrello\_della\_spesa.colore\_dei\_prodotti())

print(prodotto1.colore\_dei\_prodotti())

Console:



Così come avviene con le variabili di classe, anche con i metodi di classe è possibile accedervi sia tramite la classe, sia tramite una istanza di essa.