mlampis.dev

Python

Manuale per le ripetizioni di Informatica

Marco Lampis

28 dicembre 2022

Indice

0 Informazioni generali

Ciao! Sono **Marco**, sono uno studente magistrale in *Software Engineering* al Politecnico di Torino e mi sono laureato in Ingegneria Informatica all'università di Pisa. Nella vita sono un programmatore, uno smanettone e amante di videogiochi! Amo quello che studio, e per questo motivo fornisco ripetizioni di informatica con particolare attenzione a:

- programmazione (Java, C++, C, Python, C#, Javascript, PHP)
- algoritmi e strutture dati
- basi di Dati
- più o meno tutto quello che riguarda l'informatica!

Sia per studenti delle scuole superiori che per l'università.

0.1 Come svolgo le lezioni

La mia metodologia è innovativa e interattiva: utilizzo una piattaforma apposita per lavorare contemporaneamente sullo stesso file con gli studenti, come se fossimo accanto. In questo modo, le lezioni sono più coinvolgenti e divertenti, e gli studenti possono imparare in modo facile ed efficiente. Inoltre, le lezioni includono sia esercitazioni che approfondimenti teorici, aiuto compiti e revisione.

Per aiutare gli studenti a prepararsi al meglio e a esercitarsi anche a casa, ho creato un sito web dedicato con risorse utili come teoria, esercizi e slide. Su questo sito, gli studenti possono trovare tutto il materiale di cui hanno bisogno, e il materiale è sempre disponibile gratuitamente per i miei studenti. Inoltre, uso un iPad per prendere appunti durante le lezioni e poi condividerli con gli studenti, in modo che possano rivedere tutto ciò che è stato trattato.

Se volete migliorare le vostre conoscenze in informatica e avere un supporto personalizzato e professionale, non esitate a contattarmi. Sarò felice di fornirvi maggiori informazioni e di fissare una lezione con voi.

1 Introduzione

Python è un linguaggio di programmazione e ad alto livello, il che significa che è facile da leggere, da scrivere e ha una sintassi semplice da comprendere. Questo lo rende ideale per chiunque, dai principianti agli esperti, che vogliano imparare a programmare.

Una delle cose che rende Python così popolare è la sua versatilità. Può essere utilizzato per sviluppare molti tipi diversi di software, come applicazioni desktop, applicazioni web, giochi e perfino sistemi operativi. Inoltre, ci sono una vasta gamma di librerie e framework disponibili per Python, che possono essere utilizzati per aggiungere funzionalità extra al tuo codice.

Per iniziare a lavorare con Python, tutto ciò di cui hai bisogno è un semplice editor di testo e una conoscenza di base della sintassi del linguaggio. Ecco un esempio di codice Python che stampa il testo “Ciao, mondo!” sulla finestra del terminale:

```
# Questa é una riga di commento, che viene ignorata dall'interprete
   Python
# Quando viene eseguito il codio. I commenti sono utili per spiegare
# il codice e renderlo più facile da comprendere.

# Stampa il testo "Ciao, mondo!" sulla finestra del terminale
print("Ciao, mondo!")
```

La prima linea è un commento, che viene ignorato dall’interprete Python. La seconda linea utilizza la funzione print() per stampare il testo `Ciao, mondo!` sulla finestra del terminale.

In Python, le righe di codice vengono eseguite in ordine, dall’alto verso il basso. Ciò significa che nel nostro esempio, la riga che contiene la funzione print() viene eseguita dopo la riga di commento.

2 Tipi

In Python, i tipi di dato sono le categorie in cui vengono suddivisi i valori che possono essere memorizzati e manipolati in un programma. Ogni tipo di dato ha un insieme specifico di proprietà e metodi che possono essere utilizzati per lavorare con i valori di quel tipo.

In Python, ci sono diversi tipi di dati built-in, cioè tipi di dati che vengono forniti con il linguaggio e possono essere utilizzati senza dover importare alcuna libreria aggiuntiva. Ecco alcuni esempi di tipi di dato built-in in Python:

stringa: una sequenza di caratteri, come “Ciao, mondo!” o “12345”. Le stringhe possono essere create utilizzando apici singoli o doppi, ad esempio ‘Ciao, mondo!’ o “12345”. intero: un numero intero, come -5, 0 o 10. Gli interi possono essere positivi, negativi o zero. decimale: un numero con una parte decimale, come 3.14 o -2.718. I decimali possono essere positivi, negativi o zero. booleano: un valore logico che può essere vero o falso. In Python, i valori veri sono rappresentati dalla parola chiave True e i valori falsi dalla parola chiave False. Ecco alcuni esempi di come puoi creare variabili di questi tipi di dato in Python:

```
1  # Dichiara una variabile chiamata "nome" e assegnale il valore "Mario"
       (stringa)
2  nome = "Mario"
3
4  # Dichiara una variabile chiamata "anni" e assegnale il valore 25 (
       intero)
5  anni = 25
6
7  # Dichiara una variabile chiamata "pi" e assegnale il valore 3.14 (
       decimale)
8  pi = 3.14
9
10 # Dichiara una variabile chiamata "vero" e assegnale il valore True (
       booleano)
11 vero = True
```

Per creare una variabile di un determinato tipo di dato, basta assegnarle un valore di quel tipo. Ad esempio, per creare una variabile di tipo stringa, basta assegnarle una stringa tra apici singoli o doppi. Per creare una variabile di tipo intero o decimale, basta assegnarle un numero intero o decimale. E per creare una variabile di tipo booleano, basta assegnarle il valore True o False.

Una volta che hai creato una variabile, puoi utilizzare il suo valore nel tuo codice per eseguire operazioni o stamparlo sulla finestra del terminale, come vedremo successivamente

Inoltre, puoi utilizzare le variabili per eseguire operazioni matematiche. Ad esempio, puoi utilizzare la variabile pi per calcolare il perimetro di un cerchio di raggio 5 come segue:

```
# Dichiara una variabile chiamata "pi" e assegnale il valore 3.14 (decimale)
pi = 3.14

# Dichiara una variabile chiamata "raggio" e assegnale il valore 5 (intero)
raggio = 5

# Dichiara una variabile chiamata "perimetro" e assegnale il risultato del calcolo del perimetro del cerchio
perimetro = 2 * pi * raggio
```

In questo esempio, la variabile perimetro viene assegnata il valore 31.4, che è il risultato del calcolo del perimetro di un cerchio di raggio 5.

In sintesi: i tipi di dato in Python sono le categorie in cui vengono suddivisi i valori che possono essere utilizzati in un programma. Ogni tipo di dato ha un insieme specifico di proprietà e metodi che possono essere utilizzati per lavorare con i valori di quel tipo.

3 Variabili

In Python, le variabili sono utilizzate per memorizzare i valori che possono essere utilizzati in un programma. Una variabile può contenere una stringa, un numero intero o decimale, una booleana (cioè un valore vero o falso) o perfino un oggetto.

Per creare una variabile in Python, devi assegnare un valore a un nome. Ad esempio, puoi creare una variabile chiamata nome e assegnarle il valore “Mario” come segue:

```
# Dichiara una variabile chiamata "nome" e assegnale il valore "Mario"
nome = "Mario"
```

Una volta che hai creato una variabile, puoi utilizzarla nel tuo codice per accedere al valore che contiene. Ad esempio, puoi utilizzare la variabile nome per stampare il suo valore sulla finestra del terminale utilizzando la funzione print() come segue:

```
# Dichiara una variabile chiamata "nome" e assegnale il valore "Mario"
nome = "Mario"

# Stampa il valore della variabile "nome" sulla finestra del terminale
print(nome)
```

In questo esempio, la funzione print() stampa il valore della variabile nome, che è “Mario”.

Inoltre, puoi utilizzare le variabili per eseguire operazioni matematiche. Ad esempio, puoi utilizzare la variabile nome per eseguire un’operazione aritmetica come segue:

```
# Dichiara due variabili chiamate "a" e "b" e assegnale i valori 10 e 5
    rispettivamente
a = 10
b = 5

# Dichiara una variabile chiamata "risultato" e assegnale il risultato
   della somma di "a" e "b"
risultato = a + b

# Stampa il valore della variabile "risultato" sulla finestra del
   terminale
print(risultato)
```

In questo esempio, la variabile risultato viene assegnata il valore 15, che è il risultato della somma di a e b. Quindi, la funzione print() stampa il valore della variabile risultato, che è 15.

In sintesi: le variabili sono una parte essenziale della programmazione in Python. Ti **consentono di memorizzare i valori** che puoi utilizzare e modificare nel tuo codice, rendendo più facile e flessibile scrivere programmi complessi.

4 print

In Python, la funzione `print()` viene utilizzata per stampare una stringa (cioè una sequenza di caratteri) sulla finestra del terminale. Ad esempio, è possibile utilizzare la funzione `print()` per stampare il testo `Ciao, mondo!` come segue:

```
print("Ciao, mondo!")
```

La funzione `print()` può anche essere utilizzata per stampare il valore di una variabile. Ad esempio, se si ha una variabile chiamata `nome` che contiene un nome, è possibile utilizzare la funzione per stamparlo sulla finestra del terminale come segue:

```
# Dichiara una variabile chiamata "nome" e assegnale il tuo nome
nome = "Mario"

# Stampa il valore della variabile "nome" sulla finestra del terminale
print(nome)
```

In questo esempio, la funzione print() stampa il valore della variabile nome, che è `Mario`.

Può anche essere utilizzata per stampare il risultato di un'espressione. Ad esempio, è possibile utilizzare stampare il risultato di un'operazione aritmetica come segue:

```
# Dichiara due variabili chiamate "a" e "b" e assegnale i valori 10 e 5
    rispettivamente
a = 10
b = 5

# Stampa il risultato della somma di "a" e "b" sulla finestra del
   terminale
print(a + b)
```

In questo esempio stampa il risultato della somma di `a` e `b`, che è `15`.

5 input

La funzione input in Python è una funzione che consente agli utenti di immettere dati dalla tastiera. La sintassi generale per utilizzare la funzione input è la seguente:

```
input([prompt])
```

Dove prompt è una stringa che viene visualizzata per invitare l'utente a immettere un input. Ad esempio, potresti utilizzare la funzione input come segue:

```
nome = input("Inserisci il tuo nome: ")
print("Ciao, " + nome + "!")
```

In questo esempio, verrebbe visualizzata la stringa `Inserisci il tuo nome:` e l'utente potrebbe immettere un valore come `Mario` o `Lucia`. Quando l'utente preme il tasto Invio, il valore immesso viene assegnato alla variabile nome, che viene quindi utilizzata dalla funzione print per stampare un messaggio di benvenuto.

La funzione input può anche essere utilizzata per chiedere all'utente di immettere un numero, ad esempio un intero o un numero in virgola mobile. In questo caso, il valore immesso dall'utente verrà automaticamente convertito in un valore numerico, che potrà essere utilizzato in operazioni matematiche o in altre parti del programma. Ad esempio:

```
anni = input("Inserisci la tua età: ")
anni = int(anni)  # converte il valore immesso in un intero
print("Tra 10 anni avrai " + str(età + 10) + " anni.")
```

In questo esempio, viene chiesto all'utente di immettere la propria età come un numero intero. Il valore immesso viene assegnato alla variabile `anni` e quindi convertito in un intero con la funzione int(). Infine, viene utilizzata la funzione print per stampare un messaggio che indica quanti anni avrà l'utente tra 10 anni.

In sintesi: la funzione input in Python è una funzione molto utile per consentire agli utenti di immettere dati dalla tastiera e utilizzarli all'interno del programma.

6 Commenti

In Python, i commenti sono linee di codice che vengono ignorate dall'interprete Python. I commenti vengono utilizzati per inserire note o spiegazioni sul codice, in modo che sia più facile per altre persone comprenderlo.

Per inserire un commento in Python, si utilizza il carattere di commento # all'inizio della linea. Ad esempio:

```
1 # Questa é una linea di commento
```

7 if

In Python `if` è una parola chiave che viene utilizzata per eseguire il controllo del flusso nei programmi. In altre parole, if ci permette di eseguire determinate azioni solo se una determinata condizione viene soddisfatta.

Ad esempio, supponiamo di voler scrivere un semplice programma che controlli se un numero è maggiore o minore di 10. Possiamo farlo utilizzando if nel seguente modo:

```
numero = 5

if numero > 10:
    print("Il numero é maggiore di 10")
else:
    print("Il numero é minore o uguale a 10")
```

In questo caso, il programma stamperà “Il numero è minore o uguale a 10”, poiché 5 non è maggiore di 10.

È importante notare che l’istruzione else nell’esempio precedente viene eseguita solo se la condizione specificata dopo if non viene soddisfatta. In altre parole, else viene utilizzato per eseguire un’azione quando la condizione specificata dopo if è False.

È inoltre possibile utilizzare più istruzioni if all’interno di un singolo blocco di codice, come illustrato di seguito:

```
numero = 5

if numero > 10:
    print("Il numero é maggiore di 10")
elif numero < 10:
    print("Il numero é minore di 10")
else:
    print("Il numero é uguale a 10")
```

In questo caso, il programma stamperà “Il numero è minore di 10”, poiché 5 non è né maggiore né uguale a 10.

8 for

In Python, il costrutto for è utilizzato per eseguire un blocco di codice ripetutamente per ogni elemento di una sequenza di dati. Ad esempio, si può utilizzare il costrutto for per stampare tutti gli elementi di una lista di numeri:

```
# Crea una lista di numeri
numeri = [1, 2, 3, 4, 5]

# Per ogni numero nella lista, stampa il numero
for numero in numeri:
    print(numero)
```

9 while

In Python, il costrutto while è utilizzato per eseguire un blocco di codice ripetutamente finché una determinata condizione viene soddisfatta. Ad esempio, si può utilizzare il costrutto while per stampare una sequenza di numeri fino a quando l'utente non inserisce il numero 0:

```
# Inizializza una variabile per memorizzare il numero inserito dall'
    utente
numero = 1

# Continua a chiedere all'utente di inserire un numero finché non
    inserisce 0
while numero != 0:
    # Chiede all'utente di inserire un numero
    numero = int(input("Inserisci un numero (0 per uscire): "))

    # Se il numero inserito non é 0, stampa il numero
    if numero != 0:
        print("Hai inserito:", numero)
```

10 Turtle

La libreria **turtle** di Python è un modulo che fornisce un'interfaccia grafica per il disegno di forme e figure geometriche utilizzando un "tartaruga" che si muove sullo schermo. La turtle si può muovere avanti e indietro, girare a destra e a sinistra, sollevare e abbassare il pennello per disegnare o non disegnare, e cambiare il colore del pennello per disegnare con diverse tonalità.

Per utilizzare la libreria turtle, è necessario prima importarla con il comando import turtle. Una volta importata, è possibile utilizzare i metodi della classe Turtle per creare un'istanza della tartaruga e muoverla sullo schermo. Ad esempio:

```
import turtle

t = turtle.Turtle()  # crea un'istanza della tartaruga

t.forward(100)  # muovi la tartaruga in avanti di 100 pixel
t.right(90)  # gira la tartaruga a destra di 90 gradi
t.forward(50)  # muovi la tartaruga in avanti di 50 pixel
```

In questo esempio, la tartaruga viene creata e poi viene fatta muovere in avanti di 100 pixel, quindi viene fatta girare a destra di 90 gradi e poi muovere ancora in avanti di 50 pixel. Il risultato sarà una linea di 150 pixel di lunghezza che forma un angolo di 90 gradi.

La libreria turtle offre molti altri metodi per muovere la tartaruga e disegnare forme e figure geometriche. Ad esempio, è possibile utilizzare il metodo circle() per disegnare un cerchio, il metodo penup() per sollevare il pennello e spostare la tartaruga senza disegnare, e il metodo pendown() per abbassare il pennello e continuare a disegnare. Ecco un esempio che mostra alcuni di questi metodi in azione:

```
import turtle

t = turtle.Turtle()

t.pencolor("red")  # cambia il colore del pennello in rosso
t.circle(50)  # disegna un cerchio di raggio 50 pixel

t.penup()  # solleva il pennello
t.goto(100, 0)  # sposta la tartaruga a 100 pixel a destra
t.pendown()  # abbassa il pennello

t.pencolor("blue")  # cambia il colore del pennello in blu
```

```
13 t.circle(50)  # disegna un cerchio di raggio 50 pixel
```

In questo esempio, la tartaruga disegna prima un cerchio di raggio 50 pixel in rosso, quindi solleva il pennello e si sposta a 100 pixel a destra senza disegnare, quindi abbassa di nuovo il pennello e disegna un altro cerchio di raggio 50 pixel in blu. Il risultato sarà due cerchi, uno rosso e uno blu, separati di 100 pixel.

La libreria turtle offre anche la possibilità di personalizzare molti altri aspetti della tartaruga e del disegno, come il colore di sfondo, la velocità di disegno, la forma e la dimensione della tartaruga, e altro ancora. Ecco un esempio di come si possono personalizzare alcune di queste opzioni:

```
1  import turtle
2
3  t = turtle.Turtle()
4
5  t.screen.bgcolor("lightyellow")  # imposta il colore di sfondo
6  t.speed(10)  # imposta la velocità di disegno (da 1 a 10)
7  t.shape("turtle")  # imposta la forma della tartaruga
8  t.shapesize(2, 2)  # imposta la dimensione della tartaruga
9
10 t.pencolor("red")  # cambia il colore del pennello in rosso
11 t.circle(50)  # disegna un cerchio di raggio 50 pixel
12
13 t.penup()  # solleva il pennello
14 t.goto(100, 0)  # sposta la tartaruga a 100 pixel a destra
15 t.pendown()  # abbassa il pennello
16
17 t.pencolor("blue")  # cambia il colore del pennello in blu
18 t.circle(50)  # disegna un cerchio di raggio 50 pixel
```

In questo esempio, il colore di sfondo viene impostato su “lightyellow”, la velocità di disegno viene impostata su 10 (la massima), la forma della tartaruga viene impostata su “turtle” (la forma standard) e la dimensione viene impostata su 2x2 (il doppio della dimensione standard). Il risultato sarà lo stesso dell’esempio precedente, con due cerchi di raggio 50 pixel, uno rosso e uno blu, separati di 100 pixel, ma con il colore di sfondo impostato su “lightyellow” e la tartaruga che si muove più velocemente e ha una forma e una dimensione personalizzate.

10.1 Spostamenti

Prendendo in considerazione una variabile di tipo `turtle` di nome `t` e un generico valore intero `x`:

Sposta la tartaruga in avanti di `x` pixel:

```
1 t.forward(x)
```

Sposta la tartaruga indietro di `x` pixel:

```
```python
t.backward(x)
```

Ruota la tartaruga verso destra di `x` gradi:

```
```python
t.right(x)
```

Ruota la tartaruga verso sinistra di `x` gradi:

```
t.left(x)
```

11 list

In Python, una lista è una raccolta ordinata di elementi. Si può pensare ad una lista come ad una sequenza di oggetti che possono essere di tipi diversi. Ad esempio, una lista potrebbe contenere un intero, una stringa e un'altra lista come elementi.

Per creare una lista in Python, si utilizza la sintassi seguente:

```
lista = [elemento1, elemento2, elemento3]
```

Ad esempio, per creare una lista di numeri interi, potremmo scrivere:

```
numeri = [1, 2, 3]
```

Per accedere a un elemento della lista, si utilizza l'indice dell'elemento. L'indice di un elemento è il suo numero di posizione nella lista, iniziando da 0. Ad esempio, per accedere al primo elemento della lista numeri creata sopra, scriveremo:

```
primo_elemento = numeri[0]
```

12 Tuple

In Python, una tupla è simile ad una lista, ma è immutabile, cioè una volta creata non può essere modificata. Una tupla viene utilizzata quando si vuole rappresentare un insieme di valori che non dovranno mai cambiare.

Per creare una tupla in Python, si utilizza la sintassi seguente:

```
1 tupla = (elemento1, elemento2, elemento3)
```

Ad esempio, per creare una tupla di stringhe, potremmo scrivere:

```
1 colori = ("rosso", "verde", "blu")
```

Come per le liste, è possibile accedere agli elementi della tupla utilizzando gli indici. Ad esempio, per accedere alla prima stringa nella tupla colori, scriveremo:

```
1 primo_colore = colori[0]
```

13 Main

In Python, il termine "main" si riferisce alla funzione principale del programma, ovvero la funzione che viene eseguita quando il programma viene avviato. In molti linguaggi di programmazione, la funzione principale viene solitamente chiamata main o main().

La sintassi generale per definire la funzione principale in Python è la seguente:

```
if __name__ == "__main__":
    # codice da eseguire
```

In questo codice, la prima riga utilizza una costante predefinita di Python chiamata **name** per verificare se il programma viene eseguito direttamente o importato da un altro programma. Se il programma viene eseguito direttamente, cioè se viene avviato dalla riga di comando o dal prompt dei comandi, il valore della costante **name** sarà "main". In questo caso, il codice all'interno del blocco if verrà eseguito.

Ad esempio, il seguente programma Python definisce una funzione principale che stampa il messaggio "Ciao, mondo!" e quindi chiama la funzione saluta() che stampa il messaggio "Ciao, utente!":

```
def saluta():
    print("Ciao, utente!")

if __name__ == "__main__":
    print("Ciao, mondo!")
    saluta()
```

Se questo programma viene eseguito direttamente, verrà visualizzato il seguente output:

```
Ciao, mondo!
Ciao, utente!
```

In sintesi: la funzione principale in Python è una funzione che viene eseguita quando il programma viene avviato direttamente e non viene importato da un altro programma. La funzione principale può essere utilizzata per eseguire il codice principale del programma, come la definizione di variabili globali, la chiamata di altre funzioni e la gestione degli errori.

14 Programmazione Orientata agli Oggetti

La **programmazione orientata agli oggetti** (OOP, dall'inglese "Object-Oriented Programming") è un paradigma di programmazione che si basa sulla creazione di "oggetti", che rappresentano entità reali o astratte, dotati di proprietà (chiamate "attributi") e di comportamenti (chiamati "metodi"). In Python, come in molti altri linguaggi di programmazione, è possibile utilizzare l'OOP per creare classi che rappresentino gli oggetti e per istanziare oggetti a partire da queste classi.

Ecco un esempio di come si può definire una classe in Python:

```
class Automobile:
    def __init__(self, marca, modello, anno, colore):
        self.marca = marca
        self.modello = modello
        self.anno = anno
        self.colore = colore
        self.velocita = 0

    def accelera(self, incremento_velocita):
        self.velocita += incremento_velocita

    def frena(self, decremento_velocita):
        self.velocita -= decremento_velocita
```

In questo esempio, la classe Automobile rappresenta un'automobile generica, con alcune proprietà come la marca, il modello, l'anno e il colore. La classe definisce anche due metodi: `accelera()`, che aumenta la velocità dell'automobile di un determinato valore, e 'frena(), che diminuisce la velocità dell'automobile di un determinato valore.

Per istanziare un oggetto a partire da questa classe, si può utilizzare il costruttore **init**(). Ad esempio:

```
auto1 = Automobile("Ford", "Mustang", 1967, "Rosso")
auto2 = Automobile("Toyota", "Camry", 2010, "Grigio")
```

In questo modo, si sono create due istanze della classe Automobile, rappresentate dalle variabili auto1 e auto2. Ognuna di queste istanze ha le proprie proprietà e può utilizzare i metodi definiti nella classe. Ad esempio:

```
print(auto1.marca)  # "Ford"
print(auto2.modello)  # "Camry"
```

```
3
4  auto1.accelera(20)
5  auto2.frena(10)
6
7  print(auto1.velocita)  # 20
8  print(auto2.velocita)  # -10
```

Oltre a definire le proprietà e i metodi della classe, è anche possibile definire degli "attributi di classe", ovvero delle variabili condivise da tutte le istanze della classe. Gli attributi di classe vengono definiti all'interno della classe, ma non all'interno di un qualsiasi metodo, e vengono indicati utilizzando il prefisso self. Ecco un esempio:

```
1  class Automobile:
2      numero_ruote = 4  # attributo di classe
3
4      def __init__(self, marca, modello, anno, colore):
5          self.marca = marca
6          self.modello = modello
7          self.anno = anno
8          self.colore = colore
9          self.velocita = 0
10
11     def accelera(self, incremento_velocita):
12         self.velocita += incremento_velocita
13
14     def frena(self, decremento_velocita):
15         self.velocita -= decremento_velocita
16
17 auto1 = Automobile("Ford", "Mustang", 1967, "Rosso")
18 auto2 = Automobile("Toyota", "Camry", 2010, "Grigio")
19
20 print(auto1.numero_ruote)  # 4
21 print(auto2.numero_ruote)  # 4
```

In questo esempio, l'attributo di classe numero_ruote viene condiviso da entrambe le istanze della classe Automobile. È possibile modificare il valore di un attributo di classe utilizzando il nome della classe seguito dal punto e il nome dell'attributo. Ad esempio:

```
1  Automobile.numero_ruote = 3
2  print(auto1.numero_ruote)  # 3
3  print(auto2.numero_ruote)  # 3
```

Inoltre, è possibile definire dei "metodi di classe" utilizzando il decoratore @classmethod. I metodi di classe vengono chiamati utilizzando il nome della classe come primo argomento, invece che self, e possono essere utilizzati per effettuare operazioni che coinvolgono la classe stessa, anziché le istanze della classe.

Ecco un esempio di come si possono definire e utilizzare i metodi di classe in Python:

```python
class Automobile:
    numero_ruote = 4

    def __init__(self, marca, modello, anno, colore):
        self.marca = marca
        self.modello = modello
        self.anno = anno
        self.colore = colore
        self.velocita = 0

    def accelera(self, incremento_velocita):
        self.velocita += incremento_velocita

    def frena(self, decremento_velocita):
        self.velocita -= decremento_velocita

    @classmethod
    def numero_ruote_standard(cls):
        return cls.numero_ruote

auto1 = Automobile("Ford", "Mustang", 1967, "Rosso")
auto2 = Automobile("Toyota", "Camry", 2010, "Grigio")

print(Automobile.numero_ruote_standard())  # 4
```

In questo esempio, il metodo di classe numero_ruote_standard() viene chiamato utilizzando il nome della classe Automobile, e non l'istanza della classe. All'interno del metodo, il primo argomento cls rappresenta la classe stessa, e può essere utilizzato per accedere agli attributi di classe.

Infine, è possibile creare delle "relazioni tra classi" utilizzando l'ereditarietà. In Python, è possibile definire una classe figlia che eredita le proprietà e i metodi di una classe padre, e può anche aggiungere nuove proprietà e metodi o modificare quelli ereditati. Ecco un esempio:

```python
class Automobile:
    numero_ruote = 4

    def __init__(self, marca, modello, anno, colore):
        self.marca = marca
        self.modello = modello
        self.anno = anno
        self.colore = colore
        self.velocita = 0

    def accelera(self, incremento_velocita):
        self.velocita += incremento_velocita

    def frena(self, decremento_velocita):
        self.velocita -= decremento_velocita

```

```
    @classmethod
    def numero_ruote_standard(cls):
        return cls.numero_ruote

auto1 = Automobile("Ford", "Mustang", 1967, "Rosso")
auto2 = Automobile("Toyota", "Camry", 2010, "Grigio")

print(Automobile.numero_ruote_standard())  # 4
```

In questo esempio, il metodo di classe numero_ruote_standard() viene chiamato utilizzando il nome della classe Automobile, e non l'istanza della classe. All'interno del metodo, il primo argomento cls rappresenta la classe stessa, e può essere utilizzato per accedere agli attributi di classe.

Infine, è possibile creare delle "relazioni tra classi" utilizzando l'ereditarietà. In Python, è possibile definire una classe figlia che eredita le proprietà e i metodi di una classe padre, e può anche aggiungere nuove proprietà e metodi o modificare quelli ereditati. Ecco un esempio:

```
class Veicolo:
    numero_ruote = 0

    def __init__(self, marca, modello, anno, colore):
        self.marca = marca
        self.modello = modello
        self.anno = anno
        self.colore = colore
        self.velocita = 0

    def accelera(self, incremento_velocita):
        self.velocita += incremento_velocita

    def frena(self, decremento_velocita):
        self.velocita -= decremento_velocita

class Automobile(Veicolo):
    numero_ruote = 4

    def apri_tetto_convertibile(self):
        print("Apertura del tetto convertibile in corso...")

class Motocicletta(Veicolo):
    numero_ruote = 2

    def piega_specchietti(self):
        print("Piegatura degli specchietti in corso...")

auto = Automobile("Ford", "Mustang", 1967, "Rosso")
moto = Mot
```