PROGRAMMARE IN PYTHON

LA GUIDA DI PROGRAMMAZIONE WEB STEP BY STEP PER PRINCIPIANTI

BRYAN HARRIS

PROGRAMMARE IN PYTHON

LA GUIDA DI PROGRAMMAZIONE WEB STEP BY STEP PER PRINCIPIANTI

BRYAN HARRIS

PROGRAMMARE IN PYTHON

Sommario

Premessa

Ogni programmatore ha una storia su come ha imparato a scrivere il suo primo programma o, in generale, su come è nata la sua passione per l'informatica.

Personalmente, ho iniziato a studiare informatica da bambino quando mio padre usava un vecchio PC per disegni tecnici e, non avendo nessuna console di gioco, usavo il suo PC per giocare. Ben presto i giochi prestati dai miei amici iniziarono a portare dei virus nel PC, mettendo a rischio il lavoro di mio padre. Da qui è nata la mia passione per l'informatica e ricorderò sempre quanto mi sentii soddisfatto nel costruire il mio primo programma. C'è una vera soddisfazione nel costruire qualcosa con uno scopo, qualcosa che risolva un problema. Il software che scrivo ora soddisfa un'esigenza più significativa rispetto ai miei sforzi dell'infanzia, ma il senso di soddisfazione che provo dalla creazione di un programma che funziona è sostanzialmente lo stesso.

L'obiettivo di questo ebook è di farti conoscere Python il più rapidamente possibile in modo da poter creare programmi che funzionino e risolvano dei problemi, giochi, visualizzare dei dati o creare applicazioni web. Nel frattempo, porremo le basi di programmazione che ti serviranno per il resto della tua vita, a prescindere dal linguaggio di programmazione. Questo ebook è per coloro che non hanno mai programmato in Python prima o che non hanno mai programmato in alcun linguaggio. Se vuoi imparare rapidamente le basi della programmazione in modo da poterti concentrare su progetti interessanti e ti piace mettere alla prova i concetti che hai appreso risolvendo problemi significativi, questo ebook fa per te.

Il mio scopo è renderti un buon programmatore in generale e, in particolare, un buon programmatore Python. Imparerai in modo efficiente e adotterai delle buone abitudini man mano che apprendi i concetti di programmazione generale. Imparerai a conoscere diversi tipi di dati e i modi in cui puoi archiviare i dati all'interno dei tuoi programmi. Imparerai a creare raccolte di dati e a lavorare su tali raccolte in modo efficiente. Imparerai ad usare cicli e if per testare determinate condizioni in modo da poter eseguire specifiche sezioni di codice quando le condizioni sono vere ed eseguire

altre sezioni quando non lo sono, una tecnica che aiuta notevolmente ad automatizzare i processi. Scoprirai come scrivere funzioni per rendere riutilizzabili parti del tuo programma, scrivendo blocchi di codice che eseguono determinate azioni una sola volta, riutilizzandole tutte le volte che vuoi.

Capitolo 1
Panoramica

Python è un linguaggio incredibilmente efficiente: i tuoi programmi saranno più concisi e più potenti in meno righe di codice rispetto a quante ne richiederebbero altri linguaggi.

La sintassi di Python ti aiuterà anche a scrivere codice cosiddetto "pulito" ovvero il tuo codice sarà facile da leggere, facile da capire, facile per eseguire il debug e facile da integrare rispetto ad altri linguaggi. I programmatori usano Python per molti scopi: per creare giochi, creare applicazioni web, risolvere problemi aziendali o sviluppare degli strumenti interessanti. Python è anche ampiamente utilizzato in campo scientifico per la ricerca accademica e il lavoro applicato.

Uno dei motivi più importanti per cui Python è molto usato è la sua community, che comprende un gruppo di persone incredibilmente vario ed accogliente. La community è essenziale per i programmatori perché la programmazione non diventi una ricerca solitaria infatti molti di noi, anche i programmatori più esperti, hanno bisogno di chiedere consigli ad altri che hanno già risolto problemi simili. Avere una community ben organizzata è fondamentale per aiutarti a risolvere i problemi e per aiutare le persone come te che stanno imparando Python come primo linguaggio di programmazione. Python è un ottimo linguaggio da imparare perché è semplice ed intuitivo, quindi iniziamo!

Capitolo 2
Installazione

Oggi sono disponibili due versioni di Python: Python 2 e il più recente Python 3. Ogni linguaggio di programmazione si evolve man mano che emergono nuove idee e tecnologie e, gli sviluppatori di Python, hanno continuamente reso il linguaggio più versatile e potente. La maggior parte delle modifiche è incrementale e appena percettibile, ma in alcuni casi il codice scritto per Python 2 potrebbe non funzionare correttamente su sistemi che utilizzano Python 3.

In questo libro sottolineo le differenze significative tra Python 2 e Python 3, quindi qualunque sia la versione che usi, sarai in grado di seguire le istruzioni. Se entrambe le versioni sono installate sul tuo sistema o se devi installare Python, usa Python 3. Se Python 2 è l'unica versione sul tuo sistema e preferisci passare alla scrittura del codice invece di installare Python, puoi iniziare con Python 2 ma ti conviene passare a Python 3 dato che la fine del supporto per Python 2 è stata fissata alla fine del 2019. Alla luce di ciò, prima installi Python 3, meglio è, in modo da poter lavorare con la versione più recente.

Python viene fornito con un interprete che viene eseguito in un terminale, che consente di provare codice Python senza dover salvare ed eseguire un intero programma.

Un concetto molto vecchio nel mondo della programmazione è quello di stampare un messaggio Hello World! sullo schermo affinché il tuo primo programma in un nuovo linguaggio ti porti fortuna. In Python, ti basta digitare:

```
print("Hello World!")
```

Un programma così semplice ha uno scopo davvero importante: se funziona correttamente sul tuo sistema, qualsiasi altro programma Python dovrebbe funzionare.

Python è un linguaggio di programmazione multipiattaforma, il che significa che funziona su tutti i principali sistemi operativi. Qualsiasi

programma Python potrebbe essere eseguito su qualsiasi computer su cui sia installato Python. Tuttavia, i metodi per configurare Python sui diversi sistemi operativi variano leggermente. In questa sezione imparerai come configurare Python ed eseguire il programma Hello World sul tuo sistema.

Per prima cosa controlleremo se Python è già installato sul tuo sistema e, nel caso non lo sia, lo installeremo. Userai un semplice editor di testo e salverai un file Python vuoto chiamato hello_world.py. Infine, eseguirai il programma Hello World e risolverai eventuali problemi. Ti guiderò attraverso questo processo per ciascun sistema operativo, quindi avrai un ambiente di programmazione Python a tua disposizione.

Windows

Windows non viene sempre fornito con Python, quindi probabilmente dovrai scaricarlo e installarlo, quindi scaricare e installare un editor di testo. Innanzitutto, controlliamo se Python è installato sul tuo sistema. Apri un terminale dal menu Start e nella finestra del terminale, digita python in minuscolo. Se ricevi un prompt di Python (>>>), Python è installato sul tuo sistema. Con maggiore probabilità, purtroppo, vedrai un messaggio di errore che ti informa che python non è un comando riconosciuto.

In tal caso, è necessario scaricare il programma di installazione di Python per Windows. Dall'indirizzo https://www.python.org/downloads/ dovresti vedere un pulsante per scaricare Python 3. Fai clic sul pulsante e dovrebbe iniziare automaticamente a scaricare il programma di installazione corretto per il tuo sistema. Dopo aver scaricato il file, esegui il programma di installazione e assicurati di selezionare l'opzione "Aggiungi Python al PATH", in modo da semplificare la corretta configurazione del tuo sistema.

L'impostazione dell'editor di testo sarà semplice se si configura il sistema per eseguire Python nella sessione di un terminale. Apri una finestra di comando e digita python in minuscolo. Se ricevi un prompt di Python (>>>), Windows ha trovato la versione di Python che hai appena installato e il risultato indicherà quale versione è stata installata. Adesso sei pronto per il tuo programma che stampa a video "Hello World!", dal terminale che hai già aperto digita:

```
>>> print("Hello World!")
```

```
Hello World!
```

Se vuoi chiudere la sessione del terminale in Windows potrai premere CTRL+Z o digitare il comando exit().

Linux

I sistemi Linux sono progettati per la programmazione, quindi Python è già installato sulla maggior parte dei computer Linux. La community che ruota attorno a Linux si aspetta che tu possa programmare fin da subito e ti incoraggino a farlo. Per questo motivo c'è davvero poco da installare e pochissime impostazioni da cambiare per iniziare con il tuo primo programma.

Apri una finestra del terminale eseguendo l'applicazione Terminale sul tuo sistema (in Ubuntu, puoi premere CTRL + ALT + T). Per scoprire se Python è installato, digita python3 in minuscolo. Dovresti vedere l'output che ti dice quale versione di Python è installata con un prompt >>> da cui puoi iniziare a inserire i comandi Python.

A questo punto sei pronto per il tuo primo programma, ti basta digitare come segue:

```
>>> print("Hello World!");

Hello World!
```

Nella remota eventualità in cui Python non risulti installato sul tuo sistema, puoi usare il comando sudo apt-get install python3.8 dal terminale per installarlo ed iniziare a programmare.

macOS

Python è già installato sulla maggior parte dei sistemi OS X. Non appena avrai verificato che Python è installato, dovrai installare un editor di testo e assicurarti che sia configurato correttamente.

È sufficiente aprire un terminale da Applicazioni -> Utility -> Terminale e digitare python3, in modo simile a Linux. Se l'output mostra che hai installato Python3, sarai in grado di usare Python 3 senza ulteriori step da eseguire.

In alternativa, qualora Python non fosse installato, potrai installarlo semplicemente tramite Brew con il comando:

brew install python

Una volta che Python è installato puoi eseguire il tuo primo programma digitando:

>>> print("Hello World!");

Hello World!

Dovresti vedere il tuo messaggio stampato direttamente nella finestra del terminale. Ricorda che puoi chiudere l'interprete Python premendo CTRL + D o digitando il comando exit().

Editor di testo

Per programmare in Python ti consiglio di usare un editor di testo (detto IDE) che è fondamentalmente un pacchetto software costituito da apparecchiature utilizzate per lo sviluppo e il test del software. Un IDE aiuta ad automatizzare i compiti di uno sviluppatore riducendo gli sforzi manuali e combina tutte le componenti in un quadro comune. Se non usi un IDE, dovrai eseguire manualmente le selezioni, le integrazioni e il processo di distribuzione di codice. L'IDE è stato sostanzialmente sviluppato per semplificare lo sviluppo, riducendo ed evitando errori di battitura. Alcuni IDE hanno la capacità di eseguire il codice tramite un click ed eseguire il debug del codice in modo molto facile.

I più famosi editor di testo sono PyCharm, Idle, Sublime Text, Atom o Visual Studio Code. Ti ho consigliato alcuni degli editor ma non tutti, tra questi alcuni sono a pagamento e altri gratuiti ma ti consiglio vivamente di usare un IDE per la programmazione.

La maggior parte dei programmi che scrivi nell'editor di testo lo eseguirai direttamente dall'editor tramite un click ma, a volte, è utile eseguire programmi dal terminale. Ad esempio, potresti voler eseguire un programma esistente senza aprirlo. Puoi farlo su qualsiasi sistema con

Python installato se sai come accedere alla directory in cui hai memorizzato il tuo file di programma. Per provare ciò, assicurati di aver salvato il file hello_world.py nella cartella imparoPy sul desktop.

Adesso ti basterà digitare il comando:

python hello_world.py

Potrai eseguire questo comando solo dopo aver navigato tra le cartelle, in Linux e macOS userai:

cd Desktop/imparoPy

In Windows, allo stesso modo, dovrai digitare:

cd Desktop\imparoPy

Capitolo 3
Le variabili

Diamo un'occhiata più da vicino a ciò che fa Python quando esegui il file hello_world.py. A quanto pare, Python esegue un bel carico di lavoro, anche quando esegue un semplice programma.

Quando esegui il file hello_world.py, l'estensione .py indica che il file è un programma Python. L'editor quindi esegue il file attraverso l'interprete Python, che legge il programma e determina il significato di ogni parola nel programma stesso. Ad esempio, quando l'interprete vede la parola print, stampa sullo schermo tutto ciò che è racchiuso tra le parentesi.

Mentre scrivi i tuoi programmi, il tuo editor evidenzia diverse parti del tuo codice in diversi modi, ad esempio con colori diversi o con un carattere leggermente in grassetto. Ad esempio, riconosce che print è il nome di una funzione, visualizzerà quella parola in blu, se riconosce che "Hello World!" non è un codice Python, visualizzerà quella frase in arancione. Questa funzione degli editor si chiama evidenziazione della sintassi ed è abbastanza utile quando inizi a scrivere i tuoi programmi.

Possiamo anche memorizzare il messaggio "Hello World" in una variabile con il nome messaggio:

```
messaggio = "Hello World!"
print ( messaggio)
```

Ogni variabile contiene un valore ovvero le informazioni associate a quella variabile. In questo caso il valore è il testo "Hello World!" e l'aggiunta di una variabile aggiunge un po' più di lavoro per l'interprete Python. Quest'ultimo, infatti, quando elabora la prima riga, associa il testo "Hello World!" con la variabile messaggio. Quando raggiunge la seconda riga, stampa il valore associato alla variabile sullo schermo. Espandiamo questo programma modificando hello_world.py per stampare un secondo messaggio. Aggiungi una riga vuota a hello_world.py, quindi aggiungi due nuove righe di codice, in modo che il tuo file sia simile a questo:

```
messaggio = "Hello World!"
print ( messaggio)

messaggio = "Hello Python World!"
print ( messaggio)
```

Quando esegui questo codice vedrai due messaggi sullo schermo ma, in realtà, abbiamo capito che puoi modificare il valore di una variabile nel tuo programma in qualsiasi momento e Python terrà sempre traccia del suo valore corrente.

Quando usi le variabili in Python, devi rispettare alcune regole e linee guida. La violazione di una regola causerà degli errori mentre le linee guida ti aiutano a scrivere codice più facile da leggere e capire.

Assicurati di rispettare le seguenti regole per le variabili:

- I nomi delle variabili possono contenere solo lettere, numeri e caratteri di sottolineatura. Possono iniziare con una lettera o un trattino basso, ma non con un numero. Ad esempio, puoi chiamare una variabile messaggio_1 ma non 1_messaggio;
- Gli spazi non sono consentiti nei nomi delle variabili, ma il carattere underscore (_) può essere usato per separare le parole nei nomi delle variabili. Ad esempio, messaggio_prova è un nome valido invece messaggio prova causerà errori;
- Evita di usare parole chiave e nomi di funzioni Python come nomi di variabili; ovvero, non usare le parole che Python ha riservato per un particolare scopo programmatico, come la parola print;
- I nomi delle variabili devono essere brevi ma descrittivi. Ad esempio, nome è meglio di n, nome_studente è migliore di n_s e lung_nome è migliore di lunghezza_nome_di_persona;
- Presta attenzione quando si usano la lettera minuscola l e la lettera maiuscola O perché potrebbero essere confuse con i numeri 1 e 0. Può essere necessaria una certa pratica per imparare a creare buoni nomi di variabili, soprattutto quando i programmi diventano più interessanti e più complicati. Man

mano che scrivi più programmi e inizi a leggere il codice di altre persone, migliorerai nel trovare nomi significativi.

Ogni programmatore commette errori e la maggior parte commette errori ogni giorno. Sebbene i bravi programmatori possano creare errori, sanno anche come identificare e risolvere tali errori in modo efficiente. Diamo un'occhiata ad un errore che potresti fare presto e scopriamo come risolverlo. Scriveremo un codice che genera un errore di proposito. Scrivi il codice seguente, incluso il messaggio con errori ortografici mostrato in grassetto:

```
messaggio = "Hello World!"
print ( messagio)
```

Quando si verifica un errore nel programma, l'interprete Python fa del suo meglio per aiutarti a capire dove si trova il problema. L'interprete fornisce un **traceback** quando un programma non può essere eseguito correttamente. Un traceback è un record relativo a dove l'interprete ha riscontrato dei problemi durante l'esecuzione del codice. Ecco un esempio di traceback fornito da Python dopo aver erroneamente sbagliato il nome di una variabile:

Traceback (most recent call last):

File "hello_world.py", line 2, in <module>

print(messagio)

NameError: name 'messagio' is not defined

Il traceback segnala che si è verificato un errore nella riga 2 del file hello_world.py. L'interprete mostra questa riga per aiutarci ad individuare rapidamente l'errore e informa sul tipo di errore ha trovato.

In questo caso ha rilevato un errore sul nome e segnala che la variabile in fase di stampa, messagio, non è stata definita. Un errore sul nome (NameError) di solito significa che ci siamo dimenticati di impostare il valore di una variabile prima di usarlo oppure abbiamo fatto un errore di ortografia durante l'inserimento del nome della variabile.

Naturalmente, in questo esempio abbiamo omesso la lettera g della variabile messaggio. L'interprete Python non esegue il controllo ortografico del codice ma garantisce che i nomi delle variabili siano scritti coerentemente. Ad esempio, guarda cosa succede quando scriviamo il nome della variabile in modo errato anche in un altro punto del codice:

```
messagio = "Hello World!"
print ( messagio)
```

In questo caso il codice verrà eseguito senza alcun problema. I computer sono severi ma in alcuni casi, come nell'ambito della programmazione, ignorano il modo corretto di scrivere una parola. Di conseguenza, non è necessario prendere in considerazione le regole di ortografia e grammatica quando si tenta di creare nomi di variabili o scrivere codice. Molti errori di programmazione sono semplici errori di battitura, se stai impiegando molto tempo a cercare uno di questi errori, sappi che sei in buona compagnia. Molti programmatori esperti e di talento passano ore a cercare questo tipo di piccoli errori. Non ti preoccupare, sono problemi comuni e accadrà spesso durante la tua vita.

Stringhe

Poiché la maggior parte dei programmi definisce e raccoglie dei dati e li elabora, è utile aiutare l'elaboratore a classificare i diversi tipi di dati. Il primo tipo di dati che vedremo sarà string. Le stringhe sono piuttosto semplici a prima vista, ma puoi usarle in molti modi diversi. Una stringa è semplicemente una serie di caratteri quindi qualsiasi carattere racchiuso tra virgolette (singole o doppie) è considerata una stringa in Python.

Abbiamo già visto le stringhe:

```
"Hello World"
```

Poiché è possibile usare sia virgolette singole che doppie, automaticamente è possibile inserire delle citazioni all'interno di una stringa:

```
'Eraclito disse: "Tutto scorre"'
```

Uno dei compiti più semplici che puoi fare con le stringhe è trasformarle da maiuscole a minuscole e viceversa. Esamina il seguente codice e prova a determinare cosa sta succedendo:

```
nome = "eraclito di efeso"
print ( nome. title())
# Eraclito Di Efeso
```

In questo esempio, la stringa minuscola "eraclito di efeso" è memorizzata nella variabile nome. Il metodo title() appare dopo la variabile ma nell'istruzione print(). Un metodo è un'azione che Python può eseguire su una porzione di dati e il punto (.) dopo nome indica a Python di far agire il metodo title() sulla variabile. Ogni metodo è seguito da una coppia di parentesi, poiché i metodi spesso richiedono informazioni aggiuntive per svolgere il proprio lavoro. Tali informazioni sono fornite tra parentesi.

La funzione title() non ha bisogno di ulteriori informazioni quindi le parentesi sono vuote. Questa funzione rende l'iniziale di ogni parola una lettera maiuscola.

Allo stesso modo, puoi usare le funzioni upper() e lower() per modificare la stringa rispettivamente con tutti i caratteri maiuscoli o minuscoli. Il metodo lower() è particolarmente utile per l'archiviazione dei dati infatti molte volte non vorrai fidarti di ciò che è stato digitato dai tuoi utenti quindi convertirai le stringhe in lettere minuscole prima di memorizzarle.

```
nome = "Eraclito Di Efeso"
print ( nome. lower ())
# eraclito di efeso

print ( nome. upper())
# ERACLITO DI EFESO
```

Spesso è utile unire le stringhe, ad esempio, potresti voler memorizzare nome e un cognome in variabili separate e combinarle quando desideri visualizzare il nome completo di una persona:

```
nome = "marco"
```

```
cognome = "rossi"
nome_completo = nome + " " + cognome
print ( nome_completo)
```

Questo metodo di combinazione delle stringhe si chiama **concatenazione** . Puoi utilizzare la concatenazione per comporre messaggi completi utilizzando le informazioni che hai archiviato nelle variabili.

```
nome = "marco"
cognome = "Rossi"
nome_completo = nome + " " + cognome
print ( "Ciao, " + nome_completo. title() + "!" )
```

In questo esempio abbiamo unito alcune delle nostre conoscenze su Python fino ad ora. In questo modo, si vedrà sullo schermo il seguente messaggio:

Ciao, Marco Rossi!

Nella programmazione, lo spazio bianco si riferisce a qualsiasi carattere non stampabile, come spazi, tabulazioni e simboli di fine riga. Puoi utilizzare gli spazi bianchi per organizzare l'output in modo che sia più facile da leggere per gli utenti. Per aggiungere un TAB al testo, usa la combinazione di caratteri \t come mostrato:

```
>>> print("Python")
```

Python

```
>>> print("\tPython")
```

Python

Per aggiungere una nuova riga è sufficiente usare il carattere \n:

```
print ( "Linguaggi di programmazione:\nPython\nJava\nJavaScript\nC" )
```

Linguaggi di programmazione**:**
Python

Java
JavaScript
C

È inoltre possibile combinare TAB e nuove righe in un'unica stringa. La stringa \n\t indica a Python di passare a una nuova riga e di iniziare la riga successiva con un TAB.

Talvolta, gli spazi vuoti possono essere fonte di confusione nei tuoi programmi. Per i programmatori 'python' e 'python ' sembrano praticamente uguali ma per un programma, sono due stringhe diverse. Python rileva lo spazio extra in 'python ' e lo considera significativo.

È importante pensare agli spazi bianchi perché spesso vorrai confrontare due stringhe per determinare se sono uguali o diverse. Per fortuna, Python semplifica l'eliminazione di spazi bianchi dai dati infatti può cercare spazi bianchi sia sul lato destro che su quello sinistro di una stringa. Per assicurarti che non vi siano spazi bianchi all'estremità destra di una stringa, utilizza il metodo rstrip():

```
linguaggio = 'python  '
# 'python  '

linguaggio. rstrip()
# 'python'
```

Allo stesso modo, puoi usare lstrip() per rimuovere gli spazi vuoti presenti in testa alla stringa o puoi affidarti al metodo strip() per rimuovere gli spazi bianchi presenti a sinistra e a destra di una stringa.

```
linguaggio = '  python  '
# '  python  '

linguaggio. rstrip()
# '  python'

linguaggio. lstrip()
# 'python  '

linguaggio. strip()
```

```
# 'python'
```

In Python 2 la sintassi di print è leggermente diversa infatti le parentesi non sono necessarie attorno alla frase che si desidera stampare in Python 2. Tecnicamente, print è una funzione in Python 3, motivo per cui ha bisogno di parentesi. Fondamentalmente, quando guardi del codice scritto in Python 2, puoi trovare alcune istruzioni print con le parentesi e altre senza.

Numeri

I numeri sono usati molto spesso nella programmazione per diversi motivi: il punteggio in un gioco, archiviare le informazioni nelle applicazioni Web e così via. Python tratta i numeri in diversi modi, a seconda di come vengono utilizzati. Diamo prima un'occhiata a come Python gestisce gli interi, perché sono i più semplici con cui lavorare.

Come in buona parte dei linguaggi di programmazione è consentito usare l'operatore + per addizioni, - per sottrazioni, * per moltiplicazioni e / per divisioni.

```
>>> 5 + 5
10

>>> 10 - 6
4

>>> 10 * 3
30

>>> 10 / 2
5
```

A differenza di alcuni linguaggi che usano il simbolo ^ per indicare gli esponenti, Python usa l'operatore **:

```
>>> 5 ** 0
1

>>> 10 ** 2
100
```

Python definisce un qualsiasi numero con virgola come **float** . Questo termine viene utilizzato nella maggior parte dei linguaggi di programmazione e si riferisce al fatto che un punto decimale può apparire in qualsiasi posizione in un numero. Ogni linguaggio di programmazione

deve essere progettato con cura per gestire correttamente i numeri decimali in modo che si comportino in modo appropriato a prescindere da dove appare il punto decimale.

Inserisci semplicemente i numeri che desideri utilizzare e Python molto probabilmente farà ciò che ti aspetti:

```
>>> 0.1 + 0.1
0.2

>>> 2 * 0.1
0.2

>>> 0.2 + 0.1
0.30000000000000004
```

Ti aspettavi qualcosa di diverso? Purtroppo, questo problema si verifica in molti linguaggi. Python cerca di trovare un modo per rappresentare il risultato con la maggior precisione, il che a volte è difficile, dato che i computer devono rappresentare i numeri al loro interno. Sappi che esistono delle funzioni, ad esempio round(), che ti consentono di risolvere questo problema con pochissimo sforzo.

Spesso, capita di usare il valore di una variabile all'interno di una stringa. Ad esempio, supponi di voler informare qualcuno della sua posizione in coda. Puoi scrivere il codice in questo modo:

```
posizione = 5
messaggio = "Ci sono " + posizione- 1 + " persone prima di te"
print ( messaggio)
```

Ti aspetti un messaggio come abbiamo fatto prima ma in realtà otterrai un errore, in particolare, un errore di tipo. Significa che Python non è in grado di riconoscere il tipo di informazioni che stai utilizzando infatti in questo esempio vede che stai usando una variabile che ha un valore intero ma non è sicuro di come interpretare quel valore.

Quando si utilizzano numeri interi all'interno di stringhe come questa, è necessario specificare esplicitamente che si desidera una conversione da numero a stringa. Questa operazione è possibile grazie alla funzione str(), che indica a Python di rappresentare i valori come stringhe:

```
posizione = 5
messaggio = "Ci sono " + str( posizione- 1 ) + " persone prima di te"
print ( messaggio)
```

Adesso il risultato sarà quello che ci aspettavamo ovvero verrà stampato il messaggio:

Ci sono 4 persone prima di te

Lavorare con i numeri in Python è molto semplice per la maggior parte del tempo. Se stai ottenendo dei risultati imprevisti, verifica se Python sta interpretando i tuoi numeri nel modo desiderato, sia come valore numerico che come stringa.

Come abbiamo già visto, Python 2 segue una strada diversa infatti la divisione 5/2 restituirà 2 come risultato. La divisione di numeri interi in Python 2 genera un numero intero e il resto viene troncato. Nota bene che il risultato non è un numero intero arrotondato; il resto è semplicemente omesso. Per ottenere anche il resto devi usare i numeri in formato decimale quindi 5.0/2.0 restituirà 2.5 in Python 2.

Commenti

Probabilmente hai già notato in qualche codice precedente il simbolo cancelletto (#) accompagnato dal colore verde. In quei casi abbiamo definito un commento su una singola riga.

I commenti sono estremamente utili nella maggior parte dei linguaggi di programmazione. Man mano che i tuoi programmi diventano più lunghi e più complicati, dovresti aggiungere delle note all'interno dei tuoi programmi che descrivano il tuo approccio generale al problema che stai risolvendo. Un commento ti consente di scrivere qualsiasi cosa all'interno dei tuoi programmi.

Il motivo principale per scrivere commenti è spiegare cosa dovrebbe fare il tuo codice e come funziona. Quando stai lavorando a un progetto, se è commentato, capisci meglio come si incastrano tutti i vari pezzi. Se il codice non è commentato, puoi sempre studiare il codice per capire come dovrebbero funzionare le varie parti ma scrivere buoni commenti può farti risparmiare tempo riassumendo il tuo approccio generale.

Se vuoi diventare un programmatore professionista o collaborare con altri programmatori, dovresti scrivere commenti significativi. Oggi, la maggior parte dei software è scritta in modo collaborativo, sia da un gruppo di dipendenti di una società sia da un gruppo di persone che lavorano insieme su un progetto open source. I programmatori esperti si aspettano di vedere i commenti nel codice, quindi è meglio iniziare ora ad aggiungere commenti descrittivi ai tuoi programmi.

Scrivere commenti chiari e concisi nel codice è una delle migliori abitudini che puoi apprendere come nuovo programmatore. Quando decidi se scrivere un commento, chiediti se devi prendere in considerazione diversi approcci prima di trovare un modo ragionevole per far funzionare qualcosa; in tal caso, scrivi un commento sulla tua soluzione. È molto più semplice eliminare in seguito i commenti extra piuttosto che tornare indietro e scrivere commenti per un programma scarsamente commentato.

Capitolo 4
Liste

Una lista è una raccolta di articoli in un ordine particolare. Puoi creare una lista che includa le lettere dell'alfabeto, le cifre da 0 a 9 o i nomi di tutte le persone della tua famiglia. Puoi inserire tutto ciò che desideri in una lista e gli elementi contenuti non devono essere correlati in alcun modo particolare.

Poiché una lista di solito contiene più di un elemento, è consigliabile rendere plurale il nome della lista, ad esempio, puoi usare lettere, cifre o nomi. In Python, le parentesi quadre ([]) indicano una lista e i singoli elementi nell'elenco sono separati da virgole. Ecco un semplice esempio di una lista che contiene alcuni tipi di alberi:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
print ( alberi)
# Il risultato sarà ['quercia', 'abete', 'palma']
```

Poiché questo non è l'output che desideri venga visualizzato dai tuoi utenti, impariamo come accedere ai singoli elementi in una lista.

Le liste sono raccolte ordinate, quindi puoi accedere a qualsiasi elemento in un elenco indicando a Python la posizione o l'indice dell'elemento desiderato. Per accedere a un elemento in una lista, scrivi il nome della lista seguito dall'indice dell'elemento racchiuso tra parentesi quadre. Ad esempio, estraiamo il primo albero dell'elenco:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
print ( alberi[ 0 ])
# quercia
```

Questo è il risultato che vuoi che i tuoi utenti vedano: pulito e ben formattato. Puoi anche utilizzare le funzioni per le stringhe che abbiamo visto in precedenza per modificare il tuo risultato.

Probabilmente ti starai chiedendo ben altro. Python considera il primo elemento in una lista in posizione 0, non in posizione 1. Questo è vero per la maggior parte dei linguaggi di programmazione e il motivo ha a che fare con il modo in cui le operazioni dell'elenco vengono implementate a basso livello.

Se ricevi risultati imprevisti, controlla se stai commettendo un errore relativo all'indice dell'elemento. Il secondo elemento in una lista ha un indice di 1. Utilizzando questo semplice sistema di conteggio, è possibile ottenere qualsiasi elemento desiderato da un elenco sottraendo uno dalla sua posizione nella lista. Ad esempio, per accedere al quarto elemento in un elenco, è necessario richiedere l'indice 3. Recuperiamo il secondo e l'ultimo elemento dalla nostra lista:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
print ( alberi[ 1 ])
# abete

print ( alberi[ 2 ])
# palma

print ( alberi[- 1 ])
# palma
```

Python ha una sintassi speciale per accedere all'ultimo elemento in una lista. Indicando l'indice -1, Python restituisce sempre l'ultimo elemento della lista come vedi nell'ultimo caso. Questa sintassi è abbastanza utile, perché spesso vorrai accedere agli ultimi elementi in una lista senza sapere esattamente quanto è lunga. Questa convenzione si estende anche ad altri valori con indice negativo infatti l'indice -2 restituisce il penultimo elemento della lista, l'indice -3 restituisce il terzultimo elemento e così via.

Modificare elementi

La maggior parte delle liste che crei sarà dinamica, il che significa che dopo averla creata potrai aggiungere e rimuovere elementi nel corso del programma. Ad esempio, potresti creare un elenco di fatture. È possibile memorizzare il set iniziale di fatture in una lista e successivamente rimuovere quelle pagate e aggiungere le nuove (ovvero le non pagate). La tua lista diminuirà e aumenterà in lunghezza durante l'esecuzione del programma.

La sintassi per la modifica di un elemento è simile alla sintassi per l'accesso ad un elemento. Per modificare un elemento, utilizza il nome della lista seguito dall'indice dell'elemento che desideri modificare, quindi fornisci il nuovo valore per quell'elemento. Modifichiamo il primo elemento della nostra lista:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
alberi[ 0 ] = 'betulla'
print ( alberi)
# Il risultato sarà ['betulla', 'abete', 'palma']
```

Aggiungere elementi

Potresti voler aggiungere un nuovo elemento ad una lista per diverse ragioni e Python offre diversi modi per aggiungere nuovi dati alle liste esistenti. Il modo più semplice per aggiungere un nuovo elemento ad una lista consiste nell'uso del metodo append(). In questo caso l'elemento viene aggiunto alla fine della lista:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
alberi. append( 'betulla' )
print ( alberi)
# Il risultato sarà ['quercia', 'abete', 'palma', 'betulla']
```

Costruire le liste in questo modo è molto comune, perché spesso non si conoscono i dati che gli utenti desiderano archiviare fino a quando il programma non è in esecuzione. Per avere il controllo dei tuoi utenti, puoi iniziare definendo una lista vuota che conterrà i loro valori, quindi, aggiungi ogni nuovo valore fornito alla lista appena creata.

È possibile aggiungere un nuovo elemento in qualsiasi posizione nella lista utilizzando il metodo insert(). Puoi specificare l'indice del nuovo elemento e il valore del nuovo elemento come segue:

```
alberi = [ 'quercia' , 'abete' , 'palma' ]
alberi. insert( 1 , 'betulla' )
print ( alberi)
# Il risultato sarà ['quercia', 'betulla', 'abete', 'palma']
```

In questo esempio, è stato inserito il valore 'betulla' nella seconda posizione dell'elenco. Il metodo insert() crea uno spazio nella posizione 1 e memorizza il valore 'betulla' in quella posizione. Questa operazione sposta ogni altro valore nell'elenco di una posizione a destra.

Rimuovere elementi

Così come hai bisogno di aggiungere o modificare elementi della lista, potresti avere la necessità di rimuovere un elemento o un insieme di elementi da una lista. Ad esempio, quando una fattura viene pagata molto probabilmente vorrai rimuoverla dall'elenco di fatture da pagare, oppure, quando un utente decide di cancellare il proprio account su un'applicazione web, è necessario rimuovere tale utente dall'elenco degli utenti attivi. È possibile rimuovere un elemento in base alla sua posizione nell'elenco o in base al suo valore:

```
alberi = [ 'quercia' , 'betulla' , 'abete' , 'palma' ]
del alberi[ 2 ]
print ( alberi)
# Il risultato sarà ['quercia', 'betulla', 'palma']
```

Puoi rimuovere un elemento da qualsiasi posizione in un elenco usando l'istruzione del, basta conoscere la posizione dell'elemento.

Questo approccio potrebbe non bastare, a volte, dopo aver rimosso dalla lista un elemento si vuole comunque continuare ad usarlo. Ad esempio, in un'applicazione Web, è possibile rimuovere un utente da un elenco di membri attivi e quindi aggiungerlo ad un elenco di membri inattivi. Il metodo pop() rimuove l'ultimo elemento in un elenco ma ti consente di lavorare con quell'elemento dopo averlo rimosso. Il termine pop deriva dal pensare ad una lista come una pila di oggetti e dal far apparire un oggetto in cima alla pila. In questa analogia, la parte superiore di una pila (anche detta **stack**) corrisponde alla fine di una lista.

```
alberi = [ 'quercia' , 'betulla' , 'abete' , 'palma' ]
el_rimosso = alberi. pop()
print ( alberi)
# Il risultato sarà ['quercia', 'betulla', 'abete']

print ( el_rimosso)
# palma
```

Abbiamo iniziato definendo l'elenco degli alberi, successivamente abbiamo rimosso un valore dall'elenco e lo abbiamo memorizzato nella variabile

el_rimosso. Infine, abbiamo stampato la lista e, in seguito, il valore che è stato rimosso dall'elenco. L'output mostra che il valore palma è stato rimosso dalla fine dell'elenco e ora è memorizzato nella variabile el_rimosso.

Come può essere utile questo metodo pop()? Immagina che i valori nell'elenco siano memorizzati in ordine cronologico in base all'ordine con cui piantare questi alberi. In questo caso, possiamo usare il metodo pop() per stampare una dichiarazione sull'ultimo tipo di albero che abbiamo piantato. Tale metodo, inoltre, consente di rimuovere un elemento dalla lista in qualsiasi posizione semplicemente specificando l'indice dell'elemento.

A volte, purtroppo, non si conosce la posizione del valore che si desidera rimuovere dalla lista. Se conosci solo il valore dell'elemento che desideri rimuovere, puoi utilizzare il metodo remove(). Ad esempio, supponiamo di voler rimuovere il valore "abete" dall'elenco degli alberi piantati.

```
alberi = [ 'quercia' , 'betulla' , 'abete' , 'palma' ]
alberi. remove( 'abete' )
print ( alberi)
# Il risultato sarà ['quercia', 'betulla', 'palma']
```

Questo codice indica a Python di capire dove appare "abete" nell'elenco e rimuovere quell'elemento.

Ordinamento e funzioni utili

Spesso, le tue liste vengono create in un ordine imprevedibile, perché non puoi sempre controllare l'ordine in cui i tuoi utenti forniscono i loro dati. Sebbene ciò sia inevitabile nella maggior parte dei casi, ti consigliamo spesso di presentare le tue informazioni in un ordine preciso. A volte vorrai preservare l'ordine originale della tua lista e altre volte vorrai cambiare tale ordine. Python offre diversi modi per organizzare le tue liste, a seconda della situazione.

Il metodo sort() di Python rende relativamente facile ordinare un elenco, immagina di avere una lista di auto e di voler cambiare l'ordine dell'elenco per memorizzarle in ordine alfabetico. Per semplificare l'attività, supponiamo che tutti i valori nell'elenco siano in minuscolo.

```
auto = [ 'bmw' , 'audi' , 'toyota' , 'subaru' ]
auto. sort()
print ( auto)
# Il risultato sarà ['audi', 'bmw', 'subaru', 'toyota']
```

Il metodo sort() modifica definitivamente l'ordine della lista. Le auto sono ora in ordine alfabetico e non possiamo più tornare all'ordinamento originale. È inoltre possibile ordinare questo elenco in ordine alfabetico inverso passando l'argomento reverse=True al metodo sort(). L'esempio seguente ordina l'elenco delle auto in ordine alfabetico inverso:

```
auto = [ 'bmw' , 'audi' , 'toyota' , 'subaru' ]
auto. sort( reverse= True )
print ( auto)
# Il risultato sarà ['toyota', 'subaru', 'bmw', 'audi']
```

Per mantenere l'ordine originale di un elenco ma presentarlo in modo ordinato, è possibile utilizzare la funzione sorted(). La funzione sorted() ti consente di visualizzare l'elenco in un ordine particolare ma non influisce sull'ordine effettivo della lista.

```
auto = [ 'bmw' , 'audi' , 'toyota' , 'subaru' ]
print ( "Lista originale:" )
print ( auto)
# ['bmw', 'audi', 'toyota', 'subaru']

print ( "\nLista ordinata:" )
print ( sorted( auto))
# ['audi', 'bmw', 'subaru', 'toyota']

print ( "\nLista originale:" )
print ( auto)
```

```
# ['bmw', 'audi', 'toyota', 'subaru']
```

Infine, ma non meno importante, puoi verificare la grandezza di una lista con il metodo len(). Questa lista, per esempio, ha lunghezza pari a 4 perché contiene 4 elementi:

```
auto = [ 'bmw' , 'audi' , 'toyota' , 'subaru' ]
len( auto)
# 4
```

Troverai utile la funzione len() quando devi determinare la quantità di dati che devi gestire in un grafico o quando devi capire il numero di utenti registrati su un sito Web.

Capitolo 5
Cicli

Spesso vorrai esaminare tutte le voci in un elenco, eseguendo la stessa attività per ciascun elemento. Ad esempio, in un gioco potresti voler spostare tutti gli elementi sullo schermo di una unità o in un elenco di numeri potresti voler eseguire la stessa operazione statistica su ogni elemento. O forse vorrai visualizzare ogni titolo da un elenco di articoli su un sito Web.

Ciclo for

Quando vuoi eseguire la stessa azione con ogni elemento in una lista, puoi usare l'istruzione for di Python. Immaginiamo di avere una lista di nomi e che vogliamo stampare ogni nome nella lista. Potremmo farlo recuperando ogni nome individualmente ma questo approccio potrebbe causare diversi problemi. Innanzitutto, sarebbe ripetitivo farlo se si trattasse di una lunga lista di nomi, inoltre, dovremmo cambiare il nostro codice ogni volta che cambia la lunghezza della lista. Un ciclo for evita entrambi questi problemi consentendo a Python di gestirli internamente. Usiamo un ciclo for per stampare ogni nome in una lista:

```
nomi = [ 'andrea' , 'antonio' , 'bruno' ]
for nome in nomi:
print ( nome)
```

Iniziamo definendo una lista, proprio come nel capitolo precedente e, in seguito, definiamo un ciclo tramite la parola chiave for. Questa riga dice a Python di estrarre un nome della lista e di memorizzarlo nella variabile nome. Inoltre, indichiamo a Python di stampare il nome che è stato appena memorizzato nella variabile. Python quindi ripete le righe, una volta per ciascun nome nell'elenco. Potrebbe essere utile leggere questo codice come "Per ogni nome nell'elenco dei nomi, stampa il nome". L'output è una semplice stampa di ciascun nome nell'elenco:

```
# andrea
# antonio
# bruno
```

Il concetto di ciclo è importante perché è uno dei modi più comuni in cui un computer automatizza le attività ripetitive. Ad esempio, in un semplice ciclo come quello creato, Python inizialmente legge la prima riga del ciclo che indica a Python di recuperare il primo valore della lista e di memorizzarlo nella variabile nome. Questo primo valore è andrea, Python legge quindi la riga successiva che dice di stampare il valore corrente della variabile nome quindi verrà stampato andrea. Il processo si ripete fino a quando l'ultimo valore di nome è pari a bruno.

Dopo aver stampato l'ultimo elemento, poiché non ci sono più valori nell'elenco, Python passa alla riga successiva nel programma. In questo caso non viene eseguito nulla dopo il ciclo for, quindi il programma termina.

Quando si utilizzano i cicli per la prima volta, devi tenere presente che l'insieme di passaggi viene ripetuto una volta per ogni elemento nell'elenco, indipendentemente dal numero di elementi presenti. Se hai un milione di elementi nella lista, Python ripete questi passaggi un milione di volte e, di solito, molto rapidamente. Quando si scrive il proprio ciclo, considera che è possibile scegliere qualsiasi nome per la variabile temporanea che conterrà ciascun valore dell'elenco. Tuttavia, come sempre, è utile scegliere un nome significativo che rappresenti un singolo elemento dell'elenco. Ad esempio, se stai iterando su un gruppo di studenti potrai chiamare la variabile temporanea studente, se si tratta di un insieme di lettere, la variabile temporanea potrebbe chiamarsi lettera.

Queste convenzioni sul nome da usare possono aiutarti a seguire l'azione in corso su ciascun elemento all'interno di un ciclo for. L'uso di nomi singolari e plurali può aiutarti a identificare se una porzione di codice si riferisce ad un singolo elemento della lista o all'intera lista.

Puoi fare qualsiasi cosa con ogni elemento in un ciclo for. Usando l'esempio precedente potremmo stampare un messaggio per ogni nome, effettuando delle elaborazioni:

```
nomi = [ 'andrea' , 'antonio' , 'bruno' ]
for nome in nomi:
print ( "Ciao " + nome. title() + ", hai un nuovo messaggio!" )

# Ciao Andrea, hai un nuovo messaggio!
# Ciao Antonio, hai un nuovo messaggio!
# Ciao Bruno, hai un nuovo messaggio!
```

L'unica differenza in questo codice è dove componiamo il messaggio per ogni utente. La prima volta attraverso il ciclo il valore di nome è andrea, quindi Python scrive il primo messaggio con il nome "Andrea". La seconda volta nel messaggio userà "Antonio" e la terza volta nel messaggio userà "Bruno". L'output mostra un messaggio personalizzato per ciascun utente nell'elenco.

Cosa succede una volta che un ciclo for ha terminato l'esecuzione? Di solito, vorrai riassumere un blocco di output o far eseguire altri task al tuo programma. Qualsiasi riga di codice dopo il ciclo for che non è indentata viene eseguita una sola volta e senza ripetizione. Scriviamo un messaggio che invita gli utenti ad usare una nuova funzionalità. Per visualizzare questo messaggio dopo che tutti i singoli messaggi sono stati stampati, inseriamo il messaggio dopo il ciclo for senza indentazione:

```
nomi = [ 'andrea' , 'antonio' , 'bruno' ]
for nome in nomi:
print ( "Ciao " + nome. title() + ", hai un nuovo messaggio!" )
print ( "Prova la nuova interfaccia!" )

# Ciao Andrea, hai un nuovo messaggio!
# Ciao Antonio, hai un nuovo messaggio!
# Ciao Bruno, hai un nuovo messaggio!
# Prova la nuova interfaccia!
```

La prima istruzione print viene ripetuta una volta per ciascun utente nell'elenco, come visto in precedenza. Tuttavia, poiché l’ultima riga non è indentata, viene stampata una sola volta.

Quando elabori i dati utilizzando un ciclo for, scoprirai che questo è un buon modo per riassumere un'operazione eseguita su un intero set di dati. Ad esempio, è possibile utilizzare un ciclo for per inizializzare un gioco iterando su un elenco di personaggi e visualizzando ogni personaggio sullo schermo. Dopo questo ciclo potresti usare una funzione (senza indentazione) che visualizza un pulsante dopo che tutti i personaggi sono stati disegnati sullo schermo.

Python usa l'indentazione per determinare quando una riga di codice è connessa alla riga sopra di essa. Nell'esempio precedente, le righe che stampavano i messaggi ai singoli utenti facevano parte del ciclo for perché erano indentate. L'uso dell'indentazione da parte di Python rende il codice molto facile da leggere.

Fondamentalmente, Python utilizza gli spazi bianchi per costringerti a scrivere codice ben formattato con una chiara struttura visiva. Nei programmi Python più lunghi, noterai blocchi di codice indentati a livelli diversi. Questi livelli di indentazione ti aiutano ad ottenere un senso generale dell'organizzazione del programma. Quando inizi a scrivere codice ben formattato, dovrai controllare che tutti i cicli siano corretti. Ad esempio, a volte si indentano blocchi di codice che non devono essere indentati o si dimentica di indentare blocchi che devono esserlo. Presta attenzione a questo aspetto perché è una fonte comune di errori, soprattutto tra i principianti.

Le sezioni

Tornando ai cicli, puoi anche lavorare con un gruppo specifico di elementi in una lista che Python chiama **sezione** (o slice). Per creare una sezione, devi specificare l'indice del primo e dell'ultimo elemento con cui vuoi lavorare. Per produrre i primi tre elementi in un elenco, è necessario richiedere gli indici da 0 a 3, che restituiranno gli elementi in posizione 0, 1 e 2. Il seguente esempio prevede una lista di giocatori in una squadra:

```
giocatori = [ 'filippo' , 'michele' , 'martina' , 'elisa' , 'eva' ]
print ( giocatori[ 0 : 3 ])
```

Questo codice stampa una sezione di questo elenco, che include solo i primi tre giocatori. L'output mantiene la struttura della lista e include i primi tre giocatori nell'elenco:

```
# ['filippo', 'michele', 'martina']
```

Se si omette il primo indice in una sezione, Python avvia automaticamente la sezione all'inizio dell'elenco:

```
giocatori = [ 'filippo' , 'michele' , 'martina' , 'elisa' , 'eva' ]
print ( giocatori[: 4 ])
```

Il risultato di questo codice sarà:

```
# ['filippo', 'michele', 'martina', 'elisa']
```

Una sintassi simile è applicabile se si desidera una sezione che include la fine di un elenco. In tal caso si avrà:

```
print ( giocatori[ 4 :])
```

È possibile utilizzare una sezione in un ciclo for se si desidera iterare su un sottoinsieme degli elementi in un elenco. Nel prossimo esempio passiamo in rassegna i primi tre giocatori e stampiamo i loro nomi come parte di un semplice elenco:

```
giocatori = [ 'filippo' , 'michele' , 'martina' , 'elisa' , 'eva' ]
print ( "Ecco i primi tre giocatori del team:" )

for giocatore in giocatori[: 3 ]:
print ( giocatore. title())
```

Invece di scorrere l'intero elenco di giocatori, Python itera solo sui primi tre nomi quindi il risultato è simile al seguente:

```
# Ecco i primi tre giocatori del team:
```

Filippo
Michele
Martina

Le sezioni sono molto utili in diverse situazioni. Ad esempio, puoi aggiungere il punteggio finale di un giocatore ad una lista ogni volta che quel giocatore finisce di giocare. In seguito, puoi ottenere i primi tre punteggi di un giocatore ordinando la lista in ordine decrescente e considerando una sezione che include solo i primi tre punteggi. Quando lavori con i dati, puoi utilizzare le sezioni per elaborare i tuoi dati in blocchi di dimensioni specifiche. Oppure, quando si crea un'applicazione Web, è possibile utilizzare sezioni per visualizzare le informazioni in una serie di pagine con una quantità appropriata di informazioni su ciascuna pagina.

Capitolo 6
Istruzioni decisionali

La programmazione spesso comporta il verificare una serie di condizioni e prendere una decisione sulla base di tali condizioni. L'istruzione if di Python ti consente di esaminare lo stato corrente di un programma e di rispondere in modo appropriato a quello stato.

Il seguente breve esempio mostra come un if consente di rispondere correttamente a diverse situazioni. Immagina di avere un elenco di auto e di voler stampare il nome di ogni marchio. I nomi delle auto sono nomi propri, quindi dovrebbero essere stampati con la funzione title(). Tuttavia, il valore "bmw" deve essere stampato in maiuscolo. Il seguente codice itera su un elenco di nomi di auto e cerca il valore "bmw". Ogni volta che il valore è pari a "bmw", questo viene stampato in maiuscolo anziché avere solo l'iniziale maiuscola:

```
auto = [ 'audi' , 'bmw' , 'ferrari' , 'nissan' ]

for marchio in auto:
if marchio == 'bmw' :
print ( marchio. upper())
else :
print ( marchio. title())
```

Il ciclo in questo esempio controlla innanzitutto se il valore corrente è "bmw", in tal caso, il valore viene stampato in maiuscolo perché viene usata la funzione upper(). Se il valore di è diverso da "bmw", questo viene stampato usando la funzione title():

```
# Audi
# BMW
# Ferrari
# Nissan
```

Al centro di ogni istruzione if c'è un'espressione che può essere valutata come vera o falsa e viene chiamata **test condizionale** . Python utilizza i

valori True e False per decidere se eseguire il codice in un'istruzione if. Se un test condizionale restituisce True, Python esegue il codice seguendo l'istruzione if. Se il test restituisce False, Python ignora il codice che segue immediatamente l'istruzione if.

La maggior parte dei test condizionali confronta il valore corrente di una variabile con un valore specifico di interesse. Il test condizionale più semplice verifica se il valore di una variabile è uguale al valore di interesse ma bisogna prestare attenzione. Come hai visto, il confronto è ottenuto dall'operatore (==):

```
marchio = 'bmw'
marchio == 'bmw'
# True
```

La prima riga imposta il valore della variabile marchio su "bmw" usando un unico segno uguale, come hai già visto molte volte. La riga successiva verifica se il valore di marchio è "bmw" usando un doppio segno uguale (==). Questo operatore di uguaglianza restituisce True se i valori sul lato sinistro e destro dell'operatore corrispondono e False se non corrispondono. I valori in questo esempio corrispondono, quindi Python restituisce True; quando il valore del marchio è diverso da "bmw", questo test restituisce False.

Il test di uguaglianza è case sensitive in Python, ad esempio, due valori con lettere maiuscole diverse non sono considerati uguali:

```
marchio = 'Bmw'
marchio == 'bmw'
# False
```

Se la distinzione tra lettere maiuscole e minuscole è importante, questo comportamento è vantaggioso. Qualora la distinzione non sia importante e vuoi solo testare il valore di una variabile, puoi convertire il valore della variabile in minuscolo prima di effettuare il confronto:

```
marchio = 'Bmw'
marchio.lower() == 'bmw'
# True
```

Questo test restituisce True, indipendentemente dalla formattazione del valore "Bmw" poiché il test non è sensibile alle lettere maiuscole / minuscole. La funzione lower() non modifica il valore originale che è memorizzato in marchio, quindi puoi fare questo tipo di confronto senza influire sulla variabile originale.

Quando vuoi determinare se due valori non sono uguali, puoi combinare un punto esclamativo e un segno uguale (!=). Il punto esclamativo rappresenta l'operatore not, come in molti linguaggi di programmazione.

```
eta = 16
if ( eta != 18 ):
print ( "Non sei un diciottenne!" )
```

Se questi due valori non corrispondono, Python restituisce True ed esegue il codice che segue immediatamente l'istruzione if. Se i due valori corrispondono, Python restituisce False e non esegue il codice dopo l'istruzione if. In questo caso i valori non sono uguali e si vedrà il messaggio nella console di output.

Potresti voler controllare più condizioni nello stesso momento. Ad esempio, a volte potresti aver bisogno di due condizioni vere per eseguire un'azione, altre volte potresti essere soddisfatto se anche una sola condizione è vera. Le parole chiave **and** e **or** possono aiutarti in queste situazioni.

Per verificare se due condizioni sono entrambe True contemporaneamente, utilizza la parola chiave **and** per combinare i due test condizionali; se ogni test viene superato, l'espressione complessiva viene valutata come True. Se uno dei test fallisce o se entrambi i test falliscono, l'espressione viene valutata False. Ad esempio, puoi verificare se due persone hanno più di 21 anni usando il seguente test:

```
eta_0 = 22
eta_1 = 18
eta_0 >= 21 and eta_1 >= 21
# False

eta_1 = 25
```

```
eta_0 >= 21 and eta_1 >= 21
# True
```

Abbiamo definito due variabili e le abbiamo inizializzate con valori diversi. Nel primo caso eta_0 è maggiore di 21 ma eta_1 non lo è quindi Python valuta l'istruzione come False perché il test a sinistra di and è verificato ma non quello a destra.

In seguito, il valore di eta_1 è stato modificato diventando maggiore di 21, quindi entrambi i test individuali sono verificati, facendo sì che l'espressione condizionale complessiva venga valutata come True. Per migliorare la leggibilità, è possibile utilizzare le parentesi attorno ai singoli test ma non sono obbligatorie.

La parola chiave or consente di controllare anche più condizioni ma è verificata quando uno o entrambi i singoli test sono verificati. Un'espressione or fallisce solo quando entrambi i singoli test falliscono. Consideriamo di nuovo l'esempio precedente ma questa volta cercheremo solo una persona che abbia più di 21 anni:

```
eta_0 = 22
eta_1 = 18
eta_0 >= 21 or eta_1 >= 21
# True

eta_0 = 18
eta_0 >= 21 and eta_1 >= 21
# False
```

Un altro scenario che spesso si verifica durante lo sviluppo di codice Python consiste nel verificare se un elemento si trova in una lista. A volte è importante verificare se un elenco contiene un determinato valore prima di eseguire un'azione. Ad esempio, potresti voler verificare se esiste già un nuovo nome utente in un elenco di nomi utente prima di completare la registrazione di qualcuno su un sito Web. Per scoprire se un determinato valore è già in un elenco, usa la parola chiave in. Consideriamo un sito Web con alcuni nomi utente dedicati agli amministratori, in base al valore

digitato dall'utente vuoi mostrare la console di amministrazione o quella per l'utente. In particolare, se il nome utente è contenuto nella lista dei nomi utente degli amministratori, mostrerai la console di amministrazione, in caso contrario mostrerai quella dell'utente.

```
nome_utente = 'pippo'
amministratori = [ 'root' , 'superuser' , 'master' ]

if ( nome_utente in amministratori):
print ( "console amministratore" )
else :
print ( "console utente" )
```

Questa tecnica è abbastanza potente perché puoi creare un elenco di valori essenziali e quindi verificare facilmente se il valore che stai testando corrisponde a uno dei valori nell'elenco. In questo caso verrà mostrata la console utente invece, sostituendo uno dei valori contenuti nella lista, verrà mostrata la console di amministrazione.

Molto spesso vorrai eseguire un'azione quando viene superato un test condizionale e un'azione totalmente diversa in tutti gli altri casi. La sintassi if-else di Python rende possibile tutto ciò, proprio come abbiamo fatto nell'esempio. Un blocco if-else è simile a un'istruzione if ma l'istruzione else consente di definire un'azione o un insieme di azioni eseguite quando il test condizionale ha esito negativo.

Ancora più frequentemente dovrai testare più di due possibili situazioni e per valutarle puoi usare la sintassi if-elif-else di Python. Python esegue ogni test condizionale in ordine fino a quando uno di questi è verificato. Quando viene superato un test, viene eseguito il codice che segue quel test e saltando tutti gli altri. Molte situazioni del mondo reale coinvolgono più di due possibili scenari. Ad esempio, considera un parco di divertimenti che addebita tariffe diverse per diverse fasce di età:

- Gratis sotto i 4 anni
- 5€ da 4 a 18 anni
- 10€ per i maggiorenni

Come possiamo utilizzare un'istruzione if per determinare quanto pagare? Il codice seguente verifica la fascia d'età di una persona e quindi stampa il prezzo di ammissione in base all'età:

```
age = 16

if age < 4 :
print ( "Entrata gratuita" )
elif age < 18 :
print ( "L'entrata ha un costo di 5€" )
else :
print ( "L'entrata ha un costo di 10€" )
```

Il test if verifica se una persona ha meno di 4 anni, in tal caso, viene stampato un messaggio appropriato e Python salta il resto dei test. La parola chiave elif è essenzialmente un altro if, che viene eseguito solo se il test precedente è fallito. A questo punto della catena, sappiamo che la persona ha almeno 4 anni perché il primo test ha fallito. Se la persona ha meno di 18 anni, viene stampato un messaggio appropriato e Python salta il blocco else. Se entrambi i test if ed elif falliscono, Python esegue il codice nel blocco else. In questo esempio il test verificato è il secondo quindi il blocco di codice in else non viene eseguito. L'output è una frase, che informa l'utente del costo di ammissione:

```
# L'entrata ha un costo di 5€
```

Puoi usare anche più blocchi elif nel tuo codice, ad esempio, se il parco divertimenti dovesse prevedere uno sconto per gli anziani, potresti aggiungere un altro test condizionale al codice per determinare se qualcuno ha diritto allo sconto senior:

```
age = 16

if age < 4 :
print ( "Entrata gratuita" )
elif age < 18 :
print ( "L'entrata ha un costo di 5€" )
```

```
elif age > 65 :
print ( "L'entrata ha un costo di 5€" )
else :
print ( "L'entrata ha un costo di 10€" )
```

In realtà puoi migliorare questo codice, rendendolo più compatto grazie all'operatore or. Due messaggi sono uguali quindi puoi raggrupparli in un'unica condizione if:

```
age = 16

if age < 4 :
print ( "Entrata gratuita" )
elif age < 18 or age > 65 :
print ( "L'entrata ha un costo di 5€" )
else :
print ( "L'entrata ha un costo di 10€" )
```

Conclusioni

Congratulazioni! Hai imparato le basi di Python e applicato le tue conoscenze a semplici esempi. Abbiamo affrontato alcuni tipici scenari della programmazione in Python. Da qui, puoi andare in diverse direzioni per continuare a sviluppare le tue capacità di programmazione, innanzitutto, dovresti continuare a lavorare su progetti significativi che ti interessano. La programmazione è più interessante quando risolvi problemi rilevanti e significativi e ora hai le competenze per impegnarti in una varietà di progetti. Puoi inventare un gioco ed implementarlo, scrivere una tua applicazione o un tuo programma per scrivere file. Potresti voler esplorare alcuni dati importanti per te e realizzare grafici che mostrino modelli e connessioni interessanti. Puoi creare la tua applicazione Web o provare ad emulare una delle tue app preferite.

Quando possibile, invita altre persone a provare a utilizzare i tuoi programmi, questo ti fornisce un feedback molto importante sul tuo lavoro. Se crei un gioco, lascia che gli altri lo provino e lascia che siano critici. Se crei un'app Web, distribuiscila online e invita altri utenti a provarla. Ascolta i tuoi utenti e cerca di incorporare il loro feedback nei tuoi progetti; diventerai sicuramente un programmatore migliore perché restare fermi sulla propria idea non sempre aiuta. Quando lavori ai tuoi progetti, ti imbatterai in problemi che sono difficili o addirittura impossibili da risolvere da soli. Trova dei modi per chiedere aiuto tramite i motori di ricerca o chiedi alla comunità Python.

Unisciti ad un gruppo di utenti Python della tua zona o esplora alcune community di Python online. Dovresti cercare di mantenere un equilibrio tra il lavoro su progetti che ti interessano e lo sviluppo delle tue abilità in Python in generale. Sono disponibili online molte fonti di apprendimento riguardo a Python e un gran numero di libri si rivolge a programmatori con un po' di esperienza, proprio come te. Molte di queste risorse ti saranno accessibili ora che conosci le basi e sai come applicare le tue skills.

Adesso puoi lavorare con i tutorial e altri libri su Python basandoti direttamente su ciò che hai imparato qui e approfondendo la tua comprensione della programmazione in generale e di Python in particolare.

Quindi, quando torni a lavorare sui progetti dopo esserti concentrato sull'apprendimento di Python, sarai in grado di risolvere una varietà più ampia di problemi in modo più efficiente. Congratulazioni per essere arrivato a questo punto e buona fortuna per il tuo continuo apprendimento!