<Lorenzo Bruno>

PYTHON3 PER IL MACHINE LEARNING

Continue →

{01} Interazioni con l'utente

In questa sezione imparerete a parlare con un computer: familiarizzerete con la funzione input(), eseguirete conversioni di tipo e imparerete a usare gli operatori di stringa.

La funzione input()

Presentiamo una funzione completamente nuova, che sembra essere lo speculare della funzione print().

Perché? print() invia dati alla console, la nuova funzione ottiene dati da essa.

La funzione input() è in grado di leggere i dati inseriti dall'utente e di restituire gli stessi dati al programma in esecuzione.

Il programma può manipolare i dati, rendendo il codice veramente interattivo.

Praticamente tutti i programmi leggono ed elaborano dati. Un programma che non riceve l'input dell'utente è un programma deaf (sordo).

input(): funzionamento base

Abbiamo già detto come una variabile viene creata nel momento in cui viene dichiarata e le viene assegnato un valore.

La funzione input() viene **invocata** senza argomenti (questo è il modo più semplice di usare la funzione).

Quando chiamata la funzione converte la console in modalità di input; si vedrà un cursore lampeggiante e si potranno inserire alcuni tasti, terminando con il tasto Invio; tutti i dati inseriti saranno inviati al programma attraverso il risultato della funzione;

N.B.: è necessario assegnare il risultato a una variabile; questo passaggio è fondamentale - se lo si tralascia, i dati inseriti andranno persi.

input() con un argomento

La funzione input() può fare anche qualcos'altro. Può chiedere all'utente di rispondere senza l'aiuto di print(). Modifichiamo il prmo esempio, guardiamo il codice dell'esempio successivo. Il funzionamento è il seguente:

1. la funzione input() viene invocata con un solo argomento: una stringa contenente un messaggio;
2. il messaggio viene visualizzato sulla console prima che l'utente abbia la possibilità di inserire qualcosa;
3. input() farà quindi il suo normale lavoro.

Questa variante dell'invocazione di input() semplifica il codice e lo rende più chiaro.
Vediamo la documentazione della funzione input()

Risultato funzione input()

Lo abbiamo già detto, ma va ribadito senza ambiguità: il risultato della funzione input() è una stringa.

Una stringa contenente tutti i caratteri inseriti dall'utente dalla tastiera. Non è un intero o un float.

Ciò significa che non è possibile utilizzarla come argomento di alcuna operazione aritmetica, ad esempio non è possibile utilizzare questi dati per elevarli al quadrato, dividerli per qualcosa o dividere qualsiasi cosa per essi.

Analizziamo l'errore

L'ultima riga della frase spiega tutto: avete cercato di applicare l'operatore ** a 'str' (stringa) accompagnato da 'float'.

Questo operazione è proibita.

Questo dovrebbe essere ovvio: potete prevedere il valore di “Ciao sono Lorenzo” elevato alla potenza di 2?

Noi non possiamo e nemmeno Python può farlo.

Siamo in una situazione di stallo? Esiste una soluzione a questo problema? Certo che c'è.

Casting del tipo (conversione)

Python offre due semplici funzioni per specificare un tipo di dati e risolvere questo problema: int() e float().

I loro nomi si commentano da soli:

- la funzione int() prende un argomento (ad esempio, una stringa: int(stringa)) e cerca di convertirlo in un numero intero; se fallisce, fallisce anche l'intero programma (esiste una soluzione per questa situazione, ma ve la mostreremo poco più avanti);
- la funzione float() prende un argomento (ad esempio, una stringa: float(stringa)) e cerca di convertirlo in un float (il resto è lo stesso).

È molto semplice e molto efficace. Inoltre, è possibile invocare qualsiasi funzione passando direttamente i risultati di input(). Non è necessario utilizzare alcuna variabile come memoria intermedia.

Riuscite a immaginare come la stringa inserita dall'utente passa da input() a print()?
Provate a eseguire il codice modificato. Verificate alcuni valori diversi, piccoli e grandi, negativi e positivi. Anche lo zero è un buon input.

Altro su input() e il casting

Avere un trio input()-int()-float() apre molte nuove possibilità.

Ora sarete in grado di scrivere programmi completi, che accettano dati sotto forma di numeri, elaborandoli e visualizzando i risultati.

Naturalmente, questi programmi saranno molto primitivi e poco utilizzabili, non possono prendere decisioni e, di conseguenza, non sono in grado di reagire in modo diverso a situazioni diverse.

Il prossimo esempio si riferisce al programma precedente per trovare la lunghezza di un'ipotenusa. Eseguiamolo e facciamo in modo che sia in grado di leggere le lunghezze dei gambe dalla console.

Casting a stringa

Sapete già come usare le funzioni int() e float() per convertire una stringa in un numero.

Questo tipo di conversione non è a senso unico. È anche possibile convertire un numero in una stringa, il che è molto più semplice e sicuro. Questo tipo di operazione è sempre possibile.

La funzione in grado di farlo si chiama (ovviamente) str()

Replica di una stringa

Abbiamo visto come l'operatore + non è usato solo per la somma di valori ma anche per la concatenazione di stringhe.

Esiste anche un'altra operazione possibile sulle stringhe che sfrutta un operatore già visto. Stiamo parlando dell'asterisco

Il * ci permette di effettuare la "replication" di stringhe.

Vediamo nella pratica come usarlo

N.B.: A number less than or equal to zero produces an empty string.

Quiz n.1

Qual è l'output del seguente pezzo di codice?

```
1 | x = int(input("Enter a number: ")) # The user enters 2
2 | print(x * "5")
3 |
```

<PYTHON3> <Quiz: manipolare dati in input>

Quiz n.1

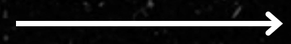
```python
x = int(input("Enter a number: ")) # The user enters 2
print(x * "5")

```

Output

```
55
```

Quiz n.2

```
1 | x = input("Enter a number: ") # The user enters 2
2 | print(type(x))
3 |
```

<PYTHON3> <Quiz: manipolare dati in input>

Quiz n.2

```python
x = input("Enter a number: ") # The user enters 2
print(type(x))

```

Output

```
<class 'str'>
```

{02} Condizioni

Un programmatore scrive un programma, il programma pone delle domande.

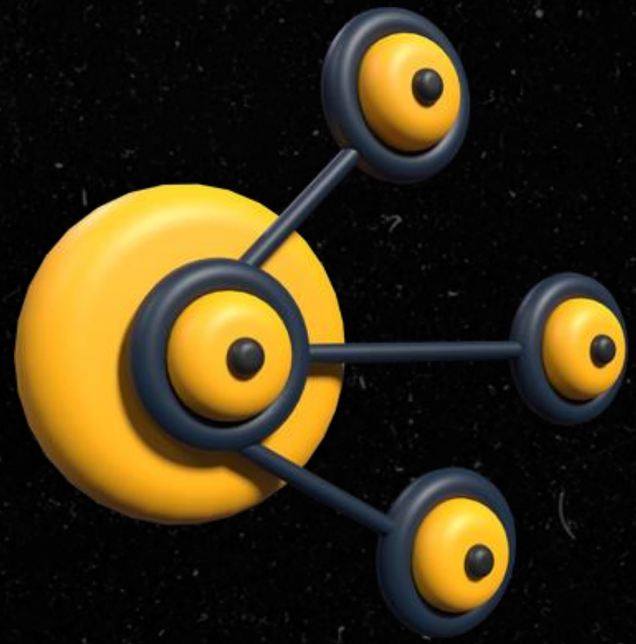
Un computer esegue il programma e fornisce le risposte.
Il programma deve essere in grado di reagire in base alle risposte ricevute.

Fortunatamente, i computer conoscono solo due tipi di risposte:

- sì, è vero;
- no, è falso.

Non si otterrà mai una risposta del tipo: "non lo so", o "Probabilmente sì, ma non ne sono sicuro".

Per porre domande, Python utilizza una serie di operatori speciali che abbiamo già avuto modo di vedere



Condizioni ed esecuzione condizionale

Sapete già come porre le domande a Python, ma non sapete ancora come fare un uso ragionevole delle risposte. Dovete avere un meccanismo che vi permetta di fare qualcosa se una condizione è soddisfatta, e di non farla se non lo è.

È proprio come nella vita reale: si fanno certe cose o non si fanno quando una specifica condizione è soddisfatta o meno, ad esempio si va a fare una passeggiata se il tempo è bello, o si resta a casa se è umido e freddo.

Per prendere tali decisioni, Python offre un'istruzione speciale. A causa della sua natura e della sua applicazione, viene chiamata selezione (o anche istruzione/struttura condizionale).

Ne esistono diverse varianti. Inizieremo con la più semplice, aumentando lentamente la difficoltà.

Esecuzione condizionale: l'istruzione *if*

Uno sviluppatore Python è insonne e si addormenta solo quando conta 120 pecore. Il programma per indurre il sonno può essere implementata come una funzione speciale chiamata sleep_and_dream(), l'intero codice assume la forma seguente:

```
1  if sheep_counter >= 120: # Evaluate a test expression
2      sleep_and_dream() # Execute if test expression is True
3
```

Esecuzione condizionale: l'istruzione *if*

Si può leggere come: se *sheep_counter* è maggiore o uguale a *120*, allora addormentarsi e sognare (cioè, eseguire la funzione *sleep_and_dream*). Abbiamo detto che le istruzioni eseguite in modo condizionale devono essere indentate. Questo crea una struttura molto leggibile, che mostra chiaramente tutti i possibili percorsi di esecuzione del codice.

```
1  if sheep_counter >= 120: # Evaluate a test expression
2      sleep_and_dream() # Execute if test expression is True
3
```

```python
if sheep_counter >= 120:
    make_a_bed()
    take_a_shower()
    sleep_and_dream()
feed_the_sheepdogs()

```


Come si può vedere, rifare il letto, fare la doccia, addormentarsi e sognare vengono tutti eseguiti in modo condizionale, quando *sheep_counter* raggiunge il limite desiderato.

Dare da mangiare ai cani da pastore, invece, viene fatto sempre (cioè, la funzione dai da mangiare ai cani da pastore() non è indentata e non appartiene al blocco if, il che significa che viene sempre eseguita).

Ora discuteremo un'altra variante dell'istruzione condizionale, che consente di eseguire un'azione aggiuntiva quando la condizione non è soddisfatta. →

La struttura *if-else*

Abbiamo iniziato con una semplice frase: Se il tempo è buono, andremo a fare una passeggiata.

Nota: non c'è una parola su cosa succederà se il tempo è brutto. Sappiamo solo che non andremo all'aperto, ma non sappiamo cosa potremmo fare al suo posto. Potremmo voler pianificare qualcosa anche in caso di maltempo.

Possiamo dire, ad esempio: Se il tempo è bello, andremo a fare una passeggiata, altrimenti andremo al cinema.

Ora sappiamo cosa faremo se le condizioni sono soddisfatte e sappiamo cosa faremo se non tutto va come vorremmo. In altre parole, abbiamo un “piano B”.

AND e *OR*: l'uso di *and*

In Python, gli operatori logici and e or vengono utilizzati all'interno di istruzioni if per combinare più condizioni logiche e determinare il flusso di esecuzione del programma in base a queste condizioni.

L'operatore and viene utilizzato quando desideri verificare che tutte le condizioni siano vere. Solo se tutte le condizioni sono vere, l'intera espressione risulterà vera.

```
x = 10
y = 20

if x > 5 and y < 30:
    print("Entrambe le condizioni sono vere")
```

AND e *OR*: l'uso di *or*

In Python, gli operatori logici and e or vengono utilizzati all'interno di istruzioni if per combinare più condizioni logiche e determinare il flusso di esecuzione del programma in base a queste condizioni.

L'operatore or viene utilizzato quando desideri verificare che almeno una delle condizioni sia vera. L'intera espressione risulterà vera se almeno una delle condizioni è vera.

```
x = 10
y = 40

if x > 5 or y < 30:
    print("Almeno una delle condizioni è vera")
```

Cortocircuito

In programmazione, il cortocircuito (short-circuiting) in un'istruzione if si riferisce a un comportamento in cui l'esecuzione delle espressioni viene interrotta non appena il risultato finale dell'operazione logica è determinato. Questo comportamento è tipico degli operatori logici and (&&) e or (||).

Esempio: siamo dei controllori e dobbiamo verificare che una persona, per vedere un film vietato ai minori di 18 anni, soddisfi dei requisiti:

- abbia pagato il biglietto,
- non porti con se cibo o bevande,
- abbia effettivamente 18 anni.

Quale controllare prima?

L'istruzione *elif*

Vediamo un secondo caso speciale che introduce un'altra nuova parola chiave di Python: *elif*

Si tratta di una forma più breve di else if.

elif viene utilizzato per verificare più di una condizione e per fermarsi quando viene trovata la prima affermazione vera.

Il prossimo esempio assomiglia alla nidificazione, ma le somiglianze sono minime. Anche in questo caso, cambieremo i nostri piani e li esprimeremo come segue: Se il tempo è bello, andremo a fare una passeggiata, altrimenti se avremo i biglietti, andremo a teatro, altrimenti se ci sono tavoli liberi al ristorante, andremo a pranzo; se tutto il resto fallisce, resteremo a casa a giocare a scacchi.

Avete notato quante volte abbiamo usato la parola altrimenti? Questa è la fase in cui la parola chiave elif svolge il suo ruolo.

CHALLENGE

Le regole sono semplici:

- 1h di tempo
- 3 suggerimenti per gruppo
- Non cercare aiuto da Internet (non ci provate)

That's all folks!

Ci vediamo la prossima settimana!

Credits

Lorenzo Bruno

Happy coding!