Input e output

November 6, 2020

1 Input e Output

In python e in molti altri linguaggi è possibile definire come verranno messi i valori in output e come possono essere immessi in input dall'utente al fine di poter definire come vengano gestiti i dati e i file su cui si voglia lavorare.

1.1 Input dall'utente

Qualora per esempio volessimo definire dei valori che sono passati dall'utente attraverso la console python fornisce il comando `input`.

[1]:
```
v = input('Inserire un numero:')
print('il numero che  è stato inserito è:',v)
```

```
Inserire un numero: 5
```

```
il numero che  è stato inserito è: 5
```

Qualora volessimo inserire più numeri possiamo usare `split` per dividerli usando come separatore uno spazio o raggrupparli in una lista con una lista comprehension.

1.1.1 Split

[2]:
```
a, b = input('Inserire due numeri separati da uno spazio:').split()
print('Il primo numero è:', a)
print('Il secondo numero è:', b)
```

```
Inserire due numeri separati da uno spazio: 23 56
```

```
Il primo numero è: 23
Il secondo numero è: 56
```

Nel caso in cui volessimo usare un diverso separatore come ad esempio un altro carattere diverso da , basterà inserire dentro le parentesi di split `split('carattere')`.

[3]:
```
a, b = input('Inserire due numeri separati da una virgola:').split(',')
print('Il primo numero è:', a)
print('Il secondo numero è:', b)
```

```
Inserire due numeri separati da una virgola: 13,42
```

```
Il primo numero è: 13
Il secondo numero è: 42
```

1.1.2 Liste

Nel caso in cui usassimo una lista per salvare i dati è necessario usare una comprensione di liste e convertire gli elementi in interi o altri formati poiché essi verranno salvati in stringhe.

[4]:
```
#int(x) converte i numeri immessi in numeri interi
l = [int(x) for x in input('Inserisci dei valori separati dallo spazio').
 ↪split()]
print('I valori inseriti convertiti come elementi della lista sono:',l)
```

```
Inserisci dei valori separati dallo spazio 54 76 91 43
```

```
I valori inseriti convertiti come elementi della lista sono: [54, 76, 91, 43]
```

1.2 Output

Qualora volessimo formattare il nostro output in un particolare modo è possibile utilizzare diversi tipi di formattazione delle stringhe oltre a quello visto fino ad adesso.

1.2.1 Valori letterali stringa formattati

I valori letterali in stringa permettono di formattare le stringhe contenenti valori precedendo la stringa con una `f` o `F` e racchiudendo le variabili all'interno di `{}` permettendo anche la loro conversione.

[5]:
```
v = 3 #valore intero
print(f'Il valore di v è {v}')
print(f'Il valore di v in numero decimale è {v:.2f}')
```

```
Il valore di v è 3
Il valore di v in numero decimale è 3.00
```

Come possiamo vedere sono riuscito a formattare v in un valore decimale e scrivedo :.2f sono riuscito a farmi considerare solo i primi due valori decimali dopo la virgola, esistono altri formati che potete trovare **qui**.

1.2.2 Metodo format

Il metodo format è analogo a quello abbena visto con la differenza che in questo caso nelle parentesi `{}` è possibile specificare nulla o un ordine con cui appaiono le variabili o le stringhe dopo la frase che si vuole printare seguita da `.format(valori)`.

[6]:
```
v1 = 5
v2 = 3
print('Il valore di v è {}, v2 è {:.2f}'.format(v1, v2))
```

```
Il valore di v è 5, v2 è 3.00
```

Come possiamo vedere il risultato è uguale alla formattazione precedente, solo con una diversa sintassi, per quanto riguarda l'ordine con cui sono stampati è possibile usare delle indici che partano da zero in modo da definire un diverso ordine.

[7]:
```
print('Il valore di v è {0}, v2 è {1:.2f}'.format(v1, v2)) #riproduce il␣
 ↪precedente
#definisco l'ordine con 0 e 1 definendo in che ordine venga stampato
print('Il valore di v2 è {1:.2f}, v è {0}'.format(v1, v2))
```

```
Il valore di v è 5, v2 è 3.00
Il valore di v2 è 3.00, v è 5
```

Per cambiare l'ordine definito in format è possibile usare degli indici come in analogia a una lista, notare che la formattazione in float è definito dentro `{}`. Possiamo definire anche un etichetta per variabile e associarli un valore qualora ci servisse.

[8]:
```
#possibile definire anche un identificativo per alcuni elementi
print('Il primo numero è {0}, il secondo {1}, il nome è {name}'.format(v, v2,␣
 ↪name='Matteo'))
```

```
Il primo numero è 3, il secondo 3, il nome è Matteo
```

1.2.3 Vecchio metodo di formattazione

Python 2 aveva un diverso modo di formattare le stringhe, per completezza vedremo anche questo che è ancora un metodo valido, ma datato. La differenza principale è la presena dell'operatore modulo `%` sia dentro la stringa che fuori.

[9]:
```
#il valore di v e la sua formattazione vengono preceduti da %
print('Il valore di v è %.2f' %v)
```

```
Il valore di v è 3.00
```

1.3 Lettura e scrittura file

In python sono presenti funzionalità anche di lettura e scrittura dei file, se per esempio volessimo creare, leggere e modificare dei file python ha delle funzioni apposite.

1.3.1 Creazione e scrittura file

Partiamo dalla creazione di un file e la sua scrittura, se ad esempio vogliamo creare un file dobbiamo prima aprirlo usando il conda `open` con al suo interno un nome racchiuso tra `'` e la modalità in cui lo si vuole aprire che nel caso di lettura è `'w'` che significa write.

[10]:
```
#questo creerà un file qualora non ce ne sia già uno di nome esempio.txt
#notare che è stato aperto in modalità write
f = open('esempio.txt', 'w')
```

Per scrivere una frase basterà usare la funzione `write` e scrivere all'interno in formato stringa la frase.

[11]:
```
f.write('questa è una frase\n')
```

[11]: 19

Qualora voi provaste ad aprire il file noterete che esso sarà vuoto e questo è dovuto al fatto che non abbiamo salvato le modifiche e chiuso il file, questo è possibile farlo usando il comando `close`.

[12]:
```
f.close()
```

Come potete vedere ora aprendo il file avrete la vostra frase!

1.3.2 Lettura file

Qualora sia necessario invece leggere il file è possibile aprire il file in modalità `r` o read usando in contemporanea il comando `with` che permetterà di associare al file uno pseudonimo e scorrere su tutto il file.

[13]:
```
f = open('esempio.txt', 'r')
```

[14]:
```
f.read()
```

[14]: 'questa è una frase\n'

[15]:
```
f.close()
```

Aggiungiamo una nuova riga usando la modalità `a` per impedire la modifica dei dati originali, ma l'aggiunta di nuovi elementi attraverso l'appensione.

[16]:
```
# questo appenderà le modifiche dopo il contenuto originale
f = open('esempio.txt', 'a')
f.write('questa è la seconda frase\n')
```

[16]: 26

[17]:
```
f.close()
```

Possiamo usare una formula ricorsiva per leggere il file.

[19]:
```
f = open('esempio.txt', 'r')
for line in f: # considera tutte le linee
    print(line, end = '') #legge fino alla fine della frase
```

```
questa è una frase
questa è la seconda frase
```

[21]:
```
f.close()
```

Oppure usare il comando `readlines`.

[22]:
```
f = open('esempio.txt', 'r')
f.readlines()
```

[22]:
```
['questa è una frase\n', 'questa è la seconda frase\n']
```

[23]:
```
f.close()
```

O ancora in maniera ripetitiva **readline** che dopo aver letto la linea andrà alla successiva.

[24]:
```
f = open('esempio.txt', 'r')
f.readline()
```

[24]:
```
'questa è una frase\n'
```

[25]:
```
f.readline()
```

[25]:
```
'questa è la seconda frase\n'
```

[26]:
```
f.close()
```

Esiste anche la modalità r+ che permette la lettura e scrittura del file e la modalità b per aprire il file in modalità binaria a posto della modalità di testo.

COMPLIMENTI AVETE COMPLETATO LA LEZIONE DI INPUT E OUTPUT!