University of Ferrara

Python

Argomenti da linea di comando

Argparse

Il modulo di analisi della riga di comando consigliato nella libreria standard di Python è **`argparse`**:

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()

$ python3 prog.py
$ python3 prog.py --help
usage: prog.py [-h]

optional arguments:
  -h, --help  show this help message and exit
$ python3 prog.py --verbose
usage: prog.py [-h]
prog.py: error: unrecognized arguments: --verbose
$ python3 prog.py foo
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

Argparse

- L'esecuzione dello script senza alcuna opzione comporta che nulla verrà visualizzato sullo stdout. Non molto utile.
- Il secondo esempio inizia a mostrare l'utilità del modulo argparse . Non abbiamo fatto quasi nulla, ma riceviamo un messaggio di aiuto.
- L’opzione --help, che può anche essere abbreviata in –h, è l'unica opzione che otteniamo gratuitamente (cioè non è necessario specificarlo).
- Specificare qualcos'altro genera un errore. Ma anche in questo caso, riceviamo, gratuitamente, un messaggio di utilizzo utile.

Argparse

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
And running the code:

$ python3 prog.py
usage: prog.py [-h] echo
prog.py: error: the following arguments are required: echo
$ python3 prog.py --help
usage: prog.py [-h] echo

positional arguments:
  echo

optional arguments:
  -h, --help  show this help message and exit
$ python3 prog.py foo
foo
```

Argparse

- Abbiamo aggiunto il metodo add_argument() , che è quello che usiamo per specificare quali opzioni della riga di comando il programma è disposto ad accettare. In questo caso, l'ho chiamato echo in modo che sia in linea con la sua funzione.
- La chiamata al nostro programma ora richiede di specificare un'opzione.
- Il metodo parse_args() restituisce effettivamente alcuni dati dalle opzioni specificate, in questo caso, echo.

Argparse

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
parser.add_argument("echo1")
args = parser.parse_args()
print(args.echo)
print(args.echo1)
% python arg.py ciao
usage: arg.py [-h] echo echo1
arg.py: error: the following arguments are required: echo1
% python arg.py ciao ciao1
ciao
ciao1
```

strip

- strip() rimuove i caratteri sia da sinistra che da destra in base all'argomento (una stringa che specifica l'insieme di caratteri da rimuovere).
- strip() restituisce una copia della stringa con entrambi i caratteri iniziali e finali eliminati.
- Quando la combinazione di caratteri nell'argomento chars non corrisponde al carattere della stringa a sinistra, smette di rimuovere i caratteri iniziali.
- Allo stesso modo, quando la combinazione di caratteri nell'argomento chars non corrisponde al carattere della stringa a destra, smette di rimuovere i caratteri finali.

Esempio

```
string = ' xoxo love xoxo   '

# Leading whitepsace are removed
print(string.strip())

xoxo love xoxo
print(string.strip(' xoxoe'))
lov
```

Esempio

```
# L’argomento non contiene i caratteri indicati
# Nessun carattere viene rimosso.
print(string.strip('sti'))
 xoxo love xoxo

string = 'android is awesome'
print(string.strip('an'))
droid is awesome
```

split

- Dividi una stringa in un elenco in cui ogni parola è un elemento dell'elenco:

```
demo_ref_string_split.py:
txt = "welcome to the jungle"

x = txt.split()

print(x)

python demo_string_split.py
['welcome', 'to', 'the', 'jungle']
```

split

- `str.split(`*`sep=None, maxsplit=-1`*`)`
- Restituisce un elenco delle parole nella stringa, utilizzando sep come stringa di delimitazione. Se viene fornito maxsplit , vengono eseguiti al massimo maxsplit split (quindi, l'elenco avrà al massimo maxsplit+1 elementi). Se maxsplit non è specificato o -1, non c'è limite al numero di suddivisioni (vengono effettuate tutte le suddivisioni possibili).
- Se viene fornito sep, i delimitatori consecutivi non vengono raggruppati e si ritiene che delimitano le stringhe vuote (ad esempio, '1,,2'.split(',') restituisce ['1', '', '2']). L'argomento sep può essere composto da più caratteri (ad esempio, '1<>2<>3'.split('<>') restituisce ['1', '2', '3']). La divisione di una stringa vuota con un separatore specificato restituisce [''].

split

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

split

- Se sep non è specificato o è None, viene applicato un algoritmo di suddivisione diverso: le sequenze di spazi bianchi consecutivi vengono considerate come un singolo separatore e il risultato non conterrà stringhe vuote all'inizio o alla fine se la stringa ha spazi bianchi iniziali o finali

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> ' 1 2 3 '.split()
['1', '2', '3']
```

Basato su slide dell'Ing. Riccardo Zese
riccardo.zese@unife.it