Programmazione in Python
per le scienze della vita

Salvatore Mancarella

Riepilogo delle funzioni

Capitolo 3 - Il linguaggio Python

Le principali funzioni della libreria *random*:

Funzione	Descrizione
random()	Genera un numero reale nell'intervallo [0,1) con una distribuzione di probabilità uniforme, cioè ogni valore di tale intervallo ha la stessa probabilità di essere "estratto".
randint(x,y)	Genera un numero interno all'intervallo [x,y) con una distribuzione di probabilità uniforme e gli argomenti devono essere numeri interi.
uniform(x,y)	Genera un numero qualsiasi nell'intervallo [x,y).

Le principali funzioni della libreria *math:*

Funzione	Descrizione
cos(x)	coseno dove x deve essere espresso in radianti
sin(x)	seno dove x deve essere espresso in radianti
tan(x)	tangente dove x deve essere espresso in radianti
acos(x)	arco-coseno dove x deve essere nell'intervallo [-1,1]
asin(x)	arco-seno dove x deve essere nell'intervallo [-1,1]

radians(x)	converte in radianti un angolo espresso in gradi
degrees(x)	converte in gradi un angolo espresso in radianti
exp(x)	e^x
log(x)	lnx
log(x,b)	$log_b x$
log10(x)	$log_{10} x$
pow(x,y)	x^y
sqrt(x)	$\sqrt{x}$

Le liste

```
>>> lista = [0, 1, 2, 3]
```

Il metodo *Len*:

```
>>> len (lista)
3
```

Il metodo *extend*:

```
>>> listan = [4,5]
>>> lista.extend(listan)
>>> lista
[-2, -1, 2, 12, [0, 'salvi', [1, 2, 3]], 4, 5]
```

Il metodo *insert*:

```
>>> lista = [4, 5]
>>> lista.insert(1,"ciao").
>>> lista
[4, 'ciao', 5]
```

Il metodo *del*:

```
>>> listan
[4, 'ciao', 'luca', 5]
>>> del listan[0]
>>> listan
['ciao', 'luca', 5]
```

Il metodo *reverse* inverte una lista:

```
>>> lista
[1, 2, 4, 5, 6, 5, 3]
>>> lista.reverse()
>>> lista
[3, 5, 6, 5, 4, 2, 1]
```

Ordinamento delle liste

Il metodo *sort:*

```
>>> lista
[3, 5, 6, 5, 4, 2, 1]
>>> lista.sort()
>>> lista
[1, 2, 3, 4, 5, 5, 6]
```

Il metodo *sorted*:

```
>>> lista = [3, 4, 61, 2, 9, 6, 9]
>>> sorted(lista)
[2, 3, 4, 6, 9, 9, 61]
>>> lista
[3, 4, 61, 2, 9, 6, 9]
```

Il metodo *sort()*:

```
>>> lista = [1,5,8,2]
>>> lista.sort(reverse=True)
>>> lista
[8, 5, 2, 1]
```

Operazioni sulle liste

Il metodo *in:*

```
>>> lista = [8, 5, 2, 1]
>>> 8 in lista
True
>>> 10 in lista
False
```

Per concatenare due liste:

```
>>> lista = [5,8,9] + [8,9]
>>> lista
[5, 8, 9, 8, 9]
>>> lista = lista + [10,11]
>>> lista
[5, 8, 9, 8, 9, 10, 11]
```

Per inizializzare una lista con una stringa:

```
>>> lista = ["Salvi"] * 4
>>> lista
['Salvi', 'Salvi', 'Salvi', 'Salvi']
```

Per inizializzare la lista con valori numerici:

```
>>> lista = [0] * 4
>>> lista
[0, 0, 0, 0]
```

Le funzioni *min* e *max:*

```
>>> lista = [5, 8, 9, 8, 9]
>>> min(lista)
5
>>> max(lista)
9
```

Il metodo *index:*

```
>>> lista = [5, 8, 9, 8, 9]
>>> lista.index(9)
2
```

Il metodo *count:*

```
>>> lista.count(5)
1
>>> lista.count(9)
2
```

Riepilogo delle funzioni principali che operano sulle liste:

Funzioni	Descrizione	Esempio
[ ]	Crea una lista	*lista = [1,"Salvo",5]*
len	Restituisce la lunghezza della lista	*len(lista)*
extend	Aggiunge una lista alla fine di una lista	*lista.extend([1, "Salvo",5]*
append	Inserisce un elemento alla fine di una lista	*lista.append(10)*
insert	Inserisce un nuovo elemento nella posizione indicata di una lista	*lista.insert(2,"Salvo")*
del	Elimina uno o più elementi da una lista	*del(lista[2])*
remove	Elimina un elemento nella lista	*lista.remove("Salvo")*
reserve	Inverte una lista	*lista.reserve()*
sort	Ordina una lista	*lista.sort()*
+	Concatena due liste	*lista1 + lista2*
*	Replica un valore in una lista	*Lista =["Salvo"]*5*
min	Trova il valore minimo	*min(lista)*
max	Trova il valore più grande	*min(lista)*
index	Restituisce la posizione di un elemento in una lista	*lista.index["Salvo"]*
count	Conta il numero di volte che un elemento si ripete in una lista	*lista.count["Salvo"]*
sum	Somma gli elementi numeri in una lista	*sum(lista)*
in	Restituisce la presenza o meno di un elemento in una lista	*"Salvo" in lista*

Le tuple

```
>>> tup= ("Salvo", 1,4)
>>> tup
('Salvo', 1, 4)
```

Le funzioni *min* e *max:*

```
tup= (2, 1, 4)
max(tup)
4
min(tup)
1
```

Assegnazione tuple:

```
>>> (a,b,c) = (1,2,3)
>>> a
1
>>> b
2
>>> c
3
```

Scambio dei valori delle variabili:

```
temp=a
a=b
b=temp
basta scrivere:
a,b=b,a
```

oppure con delle operazioni:

```
c= a+b
a=b
b=c

diventa:
a,b=b, a+c
```

I dizionari

```
>>> diz = { }
```

```
>>> diz = {1,4,6,7,8}
>>> diz
{1, 4, 6, 7, 8}
```

Aggiungere dei valori:

```
>>> diz = {}
>>> diz
{}
>>> diz[0] = 1
>>> diz
{0: 1}
>>> diz[1] = 2
>>> diz
{0: 1, 1: 2}
```

Elemento : chiave di ricerca

```
>>> diz
{0: 1, 1: 2}
```

```
>>> diz["nome"] = "Salvo"
>>> diz
{0: 1, 1: 2, 'nome': 'Salvo'}
```

Una rubrica telefonica:

```
diz_rub = {'Mancarella_Salvatore': 3333333, 'Rossi_Salvatore': 4444444}
diz_rub["Mancarella_Salvatore"]
33333333
diz_rub["Rossi_Salvatore"]
4444444
```

Il metodo *keys* e il metodo *values:*

```
>>> diz = {0: 1, 1: 2, 'nome': 'Salvo'}
>>> diz.keys()
dict_keys([0, 1, 'nome'])
>>> diz.values()
dict_values([1, 2, 'Salvo'])
```

Il metodo *items():*

```
diz.items()
dict_items([(0, 1), (1, 2), ('nome', 'Salvo')])
```

Il metodo *del:*

```
>>> diz ={0: 1, 1: 2, 'nome': 'Salvo'}
>>> del diz[0]
>>> diz
{1: 2, 'nome': 'Salvo'}
>>> del diz["nome"]
diz
{1: 2}
```

Il metodo *get()*:

```
>>> diz = {0: 1, 1: 2, 'nome': 'Salvo'}
>>> diz.get ("nome", "Non presente")
'Salvo'
>>> diz.get ("cognome", "Non presente")
'Non presente'
```

Il metodo *setdefault*:

```
>>> diz
{0: 1, 1: 2, 'nome': 'Salvo"}
>>> diz.setdefault("cognome", "Rossi")
'Rossi'
>>> diz
{0: 1, 1: 2, 'nome': 'Salvo', 'cognome': 'Rossi'}
```

Il metodo *copy*:

```
>>> diz = {0: 1, 1: 2, 'nome': 'Salvo'}
>>> dizc = diz.copy()
>>> dizc
{0: 1, 1: 2, 'nome': 'Salvo'}
```

Il metodo *update*:

```
>>> diz = {0: 1, 1: 2, 'nome': 'Salvo'}
>>> diz2 = {0: 5, 1: 6, 'nome': 'Salvo'}
>>> diz.update(diz2)
>>> diz
{0: 5, 1: 6, 'nome': 'Salvo'}
```

Nella tabella sono riportate alcune delle principali funzioni che operano sui dizionari:

Funzioni	Descrizione	Esempio
{}	Crea un dizionario.	diz = {}
len()	Restituisce il numero di elementi presenti nel dizionario.	len(diz)
keys()	Restituisce un elenco delle chiavi presenti nel dizionario.	diz.keys()
values()	Restituisce un elenco dei valori presenti nel dizionario.	diz.values()
items()	Restituisce un elenco dei valori presenti nel dizionario.	diz.items()
del()	Elimina un elemento nel dizionario.	del(diz[“chiave”])
in	Restituisce se un elemento è presente nel dizionario.	“Salvo” in diz
get()	Restituisce il valore di una chiave se presente, altrimenti un valore di default.	diz.get(“nome”,”Non presente”)
setdefault()	Restituisce il valore se la chiave è presente, in alternativa aggiunge la chiave e il valore.	diz.setdefault(“cognome”,”Rossi”)
copy()	Crea una copia superficiale di un dizionario.	dizc = diz.copy()
update()	Aggiorna le voci di un dizionario.	diz.update(diz2)

I set

```
>>> ins= set([5,7,9,2,5,6,5])
>>> ins
{2, 5, 6, 7, 9}
```

Il metodo *add*:

```
>>> ins.add(10)
>>> ins
{2, 5, 6, 7, 9, 10}
>>> ins.remove(6)
>>> ins
{2, 5, 7, 9, 10}
```

Il metodo *in*:

```
>> 2 in ins
True
>>1 in ins
False
```

La funzione *frozenset*:

```
>>> ins= set([10,8,2])
>>> inz_frozen = frozenset(ins)
>>> inz_frozen
frozenset({8, 10, 2})
>>> inz.remove(8)
Traceback (most recent call last):
  File "<pyshell#207>", line 1, in <module>
    inz.remove(8)
NameError: name 'inz' is not defined. Did you mean: 'ins'?
>>> inz.add(10)
Traceback (most recent call last):
  File "<pyshell#208>", line 1, in <module>
    inz.add(10)
NameError: name 'inz' is not defined. Did you mean: 'ins'?
```

Gestione dei file

La funzione *open():*

```
open(<nome del file>, <mode>)
```

dove:

- <*nome del file*> si riferisce al nome del file, più precisamente al percorso dove si trova il file (path), che può essere espresso in modo assoluto o relativo;
- <*mode*> indica le modalità di apertura del file, cioè gli interventi che possiamo eseguire:
  - "r", read, indica la modalità di sola lettura del file;
  - "a", append, modalità di inserimento dei dati al file, senza sovrascrivere le informazioni già presenti;
  - "w", write, modalità di scrittura nel file, dove i dati sono sovrascritti alle informazioni già presenti;
  - "x", create, modalità di creazione di un file.

```
<nome della variabile tipo file> = open (<nome del file>, <mode>)
```

Ad esempio, apriamo il file prova.txt in modalità lettura e copiamo l'oggetto file nella variabile *nome_oggetto_files*:

```
nome_oggetto_files = open("prova.txt","r")
```

Gli oggetti file hanno i seguenti attributi:

- *nome*, il nome del file;
- *close*, ci permette di chiudere il file se è stato valorizzato a *True*, altrimenti è valorizzato a *False*;
- *mode*, la modalità usata per aprire il file.

Di seguito la sintassi:

```
<nome_oggetto_files>.<attributo>
```

Leggiamo e stampiamo il contenuto del file:

```
f = open(“prova.txt”)
print(f.read())
```

```
output è
Salvatore Mancarella via Rossi 3
Rossi Verdi via roma 10
Giovanni Bianchi via lecce 23
```

Il metodo *close():*

```
< nome_oggetto_files>.close()
Nell’esempio f.close()
```

Il metodo *readline()*:

```
>>> f = open("rubrica.txt","r")
>>> f.readline()
    'Salvatore Mancarella via Rossi 3\n'
```

Il metodo *readlines()*:

```
<nome_oggetto_files>.readlines([size])
```

dove:

- *<nome_oggetto_files>* rappresenta il nome dell’oggetto *file,* nell’esempio *f;*
- *size*, elemento opzionale ed indica il numero di byte da leggere di seguito un esempio:

```
>>> f = open("rubrica.txt","r")
>>> print(f.readlines())
    ['Salvatore Mancarella via Rossi 3\n', 'Rossi Verdi via roma 10\n', 'Giovanni Bianchi via lecce 23\n']
```

Il metodo *write()* e il metodo *append()*.

```
>>> f = open("rubrica.txt","a")
>>> f.write("\n Giallo Rossi via Napoli 33")
>>> f.close()
```

Eseguire due operazioni contemporaneamente:

```
f = open(“nuovo_file.txt”, “w+”)  permette di leggere e scrivere in un file
f = open(“nuovo_file.txt”, “a+”)   permette di leggere e aggiungere in un file
f = open(“nuovo_file.txt”, “r+”)   permette di leggere e scrivere in un file
```

Il metodo *remove()*:

```
>>> import os
>>> os.remove("rubrica.txt")
```

Gestore di contesto usato per i file:

```
with open (<nome del file>, "<mode>") as <nome della variabile>:
```

Gli errori più comuni:

- **FileNotFoundError**
- **PermissionError**
- **IsADirectoryError**

```
try:
    # prova ad eseguire questo codice
except <tipo di eccezione>
    # se si verifica un'eccezione di questo tipo, interrompi il programma ed
    esegui questo blocco di codice
```

L'errore "FileNotFoundError":

```
>>> try:
...     f =  open("indirizzi.txt")
... except FileNotFoundError:
...     print("Il file non esiste")
...
...
Il file non esiste
```

```
try:
    # prova ad eseguire questo codice
except <tipo di eccezione>:
    # se si verifica un'eccezione di questo tipo, interrompi il programma ed
      esegui questo blocco di codice
finally:
    # esegui questo codice, anche se non si dovesse verificare l'eccezione
```