Programmazione (scientifica) in python

Liceo Scientifico e Sportivo Statale "A. Tassoni"

19/02/2018

Conversioni tra tipi di dati

Si utilizzano particolari funzioni (`int`, `float`, `str`):

```
print(int(124.741251))
print(int("ciao!"))
print(int("38198"))
print(int(True), int(False))
```

```
print(float("ciao!"))
print(float("13e-4"))
print(float(12))
print(float(True), float(False))
```

```
print(str(139))
print(str(31e42))
print(str(True), str(False))
```

Liste

Una lista è una sequenza di valori, non necessariamente omogenei come tipologia di dati

```
x = [231, -85e-23, "Rosso di sera", False, True, 0, 11]
z = []
```

Si può creare una lista contenente liste:

```
y = [[31,"bel tempo si spera", 100.3], ['il lupo perde il pelo', True]]
```

Liste

Una lista è una sequenza di valori, non necessariamente omogenei come tipologia di dati

```
x = [231, -85e-23, "Rosso di sera", False, True, 0, 11]
z = []
```

Si può creare una lista contenente liste:

```
y = [[31,"bel tempo si spera", 100.3], ['il lupo perde il pelo', True]]
```

Gli elementi delle liste si richiamano con le `[]`:

```
print(x)
print(x[4])
print(y[1])
print(y[0][3])
# ---
print(x[15])
```

Liste

Come le stringhe, le liste si possono "affettare".

```
print(x[2:4])
print(x[:3:2])
print(x[::-1])
```

A differenza delle stringhe, le liste sono *mutabili*:

```
x[0] = "domani c'è il sole"
print(x)
```

Liste

Come le stringhe, le liste si possono "affettare".

```
print(x[2:4])
print(x[:3:2])
print(x[::-1])
```

A differenza delle stringhe, le liste sono *mutabili*:

```
x[0] = "domani c'è il sole"
print(x)
```

Le liste si possono "spacchettare" (occhio al numero di elementi):

```
a, b, c, d, e, f, g = x
print(b, c)
```

Primi elementi di progammazione

Cicli `for`

Un **ciclo** `for` ripete una istruzione o una sequenza di istruzioni per tutti gli elementi di una lista, utilizzando una variabile apposita come indice

```
for i in [ 1, "Del bel Panaro", True, -3.3e-3, [0, 1] ]:
    print(i)
```

Ciò che identifica la sequenza di istruzioni da eseguire a ogni ciclo è la spaziatura, o **indentazione** (tipicamente di 4 spazi, spesso automaticamente impostata dal notebook)

```
for i in x:
    print(i)
    print(i and True)
```

Primi elementi di progammazione

Cicli `for`

La variabile del ciclo **non** può essere utilizzata per modificare i contenuti di una lista

```
for i in x:
    i = 15
print(x)
```

I cicli possono essere anche "nidificati" (nested):

```
for i in [1,2,3,4,5,6,7,8,9]:
    print("------- tabellina del: ", i)
    for j in [1,2,3,4,5,6,7,8,9]:
        print(i, " * ", j, " = ", i * j)
```

L'importante è che il livello di indentazione sia costante per ogni blocco di istruzioni

La funzione `range`

Si utilizza per eseguire un ciclo per una sequenza numerica

```
for i in range(10):
    print(i)
for j in range(3,15):
    print(j)
for k in range(2,19,2):
    print(k)
```

La funzione `range` **non** costruisce una lista. Per costruire una lista contenente la sequenza di numeri desiderata si usa la funzione `list`

```
seq = list( range(15) )
print(seq)
```

La funzione `range`

Si utilizza per eseguire un ciclo per una sequenza numerica

```
for i in range(10):
    print(i)
for j in range(3,15):
    print(j)
for k in range(2,19,2):
    print(k)
```

La funzione `range` **non** costruisce una lista. Per costruire una lista contenente la sequenza di numeri desiderata si usa la funzione `list`

```
seq = list( range(15) )
print(seq)
```

La funzione `range` può essere utilizzata per modificare gli elementi di una lista:

```
x = [1,2,3,4,5,6]
for i in range(len(x)):
    x[i] = x[i] * 2
print(x)
```

Primi elementi di progammazione

Istruzioni condizionali

Permettono di eseguire porzioni di codice in base al valore di verità di una particolare istruzione:

```
x = True
if x:
    print("x è vero!")
else:
    print("x è è falso!")
```

L'insieme di istruzioni da eseguire in ciascuno dei due casi va **indentato** come per i cicli.

Primi elementi di progammazione

Istruzioni condizionali

Per esempio, se vogliamo determinare se un numero *x* qualsiasi è pari o dispari, possiamo usare una istruzione del tipo:

```
x = 319
if (x % 2 == 0):
    print("x è pari!")
else:
    print("x è dispari!")
```

Primi elementi di progammazione

Istruzioni condizionali

Se si vuole verificare più di una condizione, si utilizza la sintassi `elif`, abbreviativa di *else if*:

```
y = 39 # 65 o 120
if y > 100:
    print("y è maggiore di 100")
elif y > 50:
    print("y è compreso tra 50 e 100")
else:
    print("y è minore di 50")
```

N.B.: una volta che una delle condizioni è verificata, vengono eseguite le espressioni corrispondenti, e le altre condizioni non vengono verificate

Compiti

Nella cartella `notebooks`, apri la cartella `compiti`, ed apri il notebook `esercizi_lezione2a.ipynb`.