Complementi di Programmazione

Python: Costrutti di base

CdL Informatica - Università degli studi di Modena e Reggio Emilia
AA 2023/2024

Filippo Muzzini

Blocco di codice

In Python un blocco di codice è caratterizzato da:

- Inizia con **:** (seguito dal ritorno a capo)
- è indentato dall'inizio alla fine
  - La fine dell'indentazione indica la fine del blocco
- In altri linguaggi si usano le {}

```
while b<0:
    b = b+1
    b = b+2
b = 0
```

Statement ;

In Python non si usa **;** per indicare la fine di un’istruzione.

La fine di un’istruzione viene identificata dal ritorno a capo.

b=0

b=b+1

Si può usare per mettere più istruzioni sulla stessa riga

b=0; b=b+1

Statement ,

In Python è possibile utilizzare , per assegnamenti multipli.

b, a = 1, 2

a -> 2

b -> 1

Moduli e import

In Python è possibile importare moduli esterni (librerie) per poterle utilizzare all’interno del proprio codice

```
import pandas

a = pandas.DataFrame()
```

Moduli e import

In Python è possibile importare moduli esterni (librerie) per poterle utilizzare all’interno del proprio codice

import pandas as pd

a = pd.DataFrame()

Si rinomina il modulo per comodità o evitare conflitti

Moduli e import

In Python è possibile importare moduli esterni (librerie) per poterle utilizzare all’interno del proprio codice

from pandas import DataFrame

a = DataFrame()

Si importa una funzione specifica e la si utilizza senza inserire il riferimento al modulo

Moduli e import

In Python è possibile importare moduli esterni (librerie) per poterle utilizzare all’interno del proprio codice

from pandas import *

a = DataFrame()

Si importano tutte le funzioni del modulo (e le si utilizza senza specificare il modulo)

Commenti

In Python si commenta riga per riga utilizzando il **#**

Non vi è il commento per blocchi

Costrutti condizionali

Classico **if then else**

if condizione:

    codice

else:

    codice

Costrutti condizionali

Ulteriore condizione **elif.** Se falsa **condizione1** si guarda **condizione2**

if condizione1:

    codice

elif condizione2:

    codice

else:

    codice

Ciclo while

Ciclo che esegue finché la condizione è vera

```
while b<10:
    b=b+1
```

Ciclo while

Ciclo che esegue finché la condizione è vera, quando esce esegue l’else (se vi è)

```
while b<10:
    b=b+1
else:
    b=100
```

Ciclo for

Ciclo che esegue su tutti gli elementi nell’elenco

```
for a in elenco:
    print(a)
```

Ciclo for

Ciclo che esegue su tutti gli elementi nell’elenco. Quando esce esegue l’else (se esiste)

```
for a in elenco:
    print(a)
else:
    print(‘vuoto’)
```

Comandi di salto

- Comando break:
  - Interrompe un ciclo for/while
- Comando continue:
  - Salta all'iterazione for/while successiva
- Clausola else:
  - Può essere inserita alla fine di un blocco relativo ad un ciclo
  - Viene eseguita (una volta sola) se un ciclo termina le sue iterazioni o quando la condizione del ciclo è valutata False
  - **Non viene eseguita in caso di break**

Ciclo for… diversi elenchi diverso comportamento

Il ciclo for può iterare su qualsiasi elenco di oggetti: stringhe, tuple, liste ecc..

In base al tipo di elenco gli elementi su cui si itera saranno differenti

Tipico uso (come in altri linguaggi) iterare su interi ordinati (for i=0; i<10; i++)

Si usa la funzione **range(10)**

```
for i in range(10):
    print(i)
```

range(inizio, fine, intervallo)
range(0,10,2) -> 0,2,4,6,8

range(inizio, fine)
range(0,5) -> 0,1,2,3,4

range(fine)
range(5) -> 0,1,2,3,4

Ciclo for… diversi elenchi diverso comportamento

Ciclo for su liste: ad ogni iterazione vi è il riferimento ad un oggetto della lista.

```
a = [‘a’, ‘b’]
for i in a:
    print(i)
```

risultato:

a

b

Ciclo for… diversi elenchi diverso comportamento

Ciclo for su stringhe: ad ogni iterazione vi è il riferimento ad un carattere.

```
a = ‘cd’

for i in a:

    print(i)
```

risultato:

c

d

Ciclo for… diversi elenchi diverso comportamento

Ciclo for su tuple: come per le liste.

```
a = (‘a’, ‘b’)

for i in a:

    print(i)
```

risultato:

a

b

Ciclo for… diversi elenchi diverso comportamento

Ciclo for su set: ad ogni iterazione vi è il riferimento ad un elemento dell'insieme.

Ricordarsi che in questo caso non è garantito l'ordine!

```
a = set(['a', 'b'])
for i in a:
    print(i)
```

risultato:

b

a

Ciclo for… diversi elenchi diverso comportamento

Ciclo for su dictionary: ad ogni iterazione vi è il riferimento ad una chiave dell'insieme.

Ricordarsi che in questo caso non è garantito l'ordine!

```
a = {'k1':1, 'k2':2}
for i in a:
    print(i)
```

risultato:

k1

k2

Ciclo for… funzione **enumerate()**

La funzione **enumerate** enumera la sequenza su cui si itera

Ritorna una tupla (indice, oggetto)

```
a = ‘abc’
for i in enumerate(a):
        print(i)
```

risultato:

(0,’a’)

(1,’b’)

(2,’c)

Ciclo for… funzione **enumerate()**

La funzione **enumerate** enumera la sequenza su cui si itera

Ritorna una tupla (indice, oggetto). E' possibile dividere i due valori in due variabili

```
a = 'abc'
for i,o in enumerate(a):
        print(i)
```

risultato:

0

1

2

Ciclo for… come funziona

In pratica possiamo iterare su moltissimi tipi di elenchi.

L’importante e che tali elenchi siano **iterabili**

Ciclo for… come funziona

In pratica possiamo iterare su moltissimi tipi di elenchi.

L’importante e che tali elenchi siano **iterabili**

l’elenco deve avere un metodo __iter__() che ritorna un iterabile

(duck typing)

Ciclo for… come funziona

l’elenco deve avere un metodo **__iter__**() che ritorna un iterabile

(duck typing)

A sua volta un oggetto per essere un **iterabile** deve:

- avere un metodo **__next__**() che ritorna l'elemento successivo dell'insieme

Ciclo for… come funziona

In pratica lo statement **for** chiama **__iter__**() e poi **__next__**() sull'oggetto ritornato

a = [1,2]

i = a.__iter__()

i.__next__() -> 1

i.__next__() -> 2

i.__next__() -> ?

Ciclo for… come funziona

In pratica lo statement **for** chiama **__iter__**() e poi **__next__**() sull'oggetto ritornato

a = [1,2]

i = a.__iter__()

i.__next__() -> 1

i.__next__() -> 2

i.__next__() -> Eccezzione StopIteration

Ciclo for… come funziona

In pratica lo statement **for** chiama **__iter__**() e poi **__next__**() sull'oggetto ritornato

a = [1,2]

i = a.__iter__()

i.__next__() -> 1

i.__next__() -> 2

i.__next__() -> Eccezione StopIteration

for cattura questa eccezione per uscire dal ciclo

Ciclo for… come funziona

Vi sono wrapper più comodi per __iter__() e __next__():

- iter(a) -> i
- next(i)

Queste funzioni semplicemente chiamano i rispettivi metodi sull'oggetto passato come argomento.