Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Iterazioni 3

Le seguenti slide sono tratte principalmente dal lavoro di Giorgio Fumera (Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi (Sapienza) e Domenico Lembo (Sapienza). Alcune sono invece una rielaborazione di slide analoghe di Diego Calvanese (Univ. di Bolzano).

Istruzione `Continue`

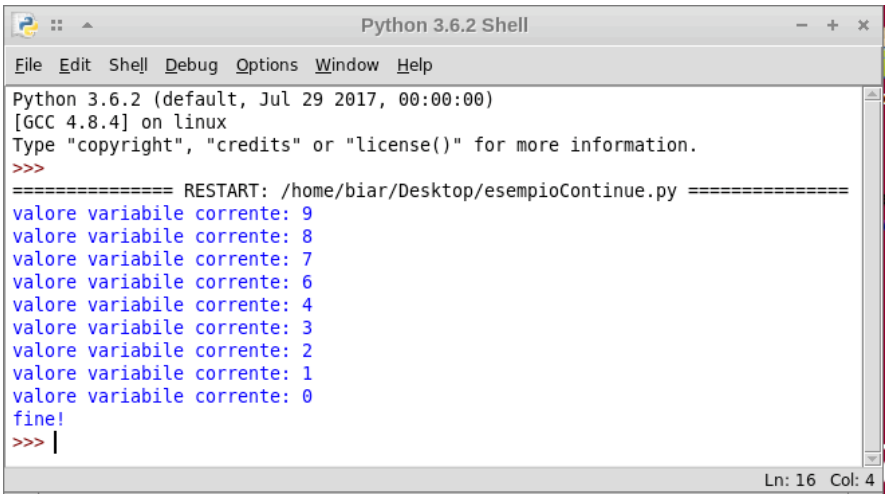
Analogamente all'istruzione `break`, l'istruzione `continue` può essere solo usata all'interno di un ciclo, e normalmente è scritta all'interno di un'istruzione condizionale presente nel ciclo per concludere l'esecuzione di una iterazione nel caso in cui si verifichi una data condizione.

Quindi, diversamente dal `break`, l'istruzione `continue` non causa l'interruzione del ciclo, spostando quindi il controllo alla istruzione immediatamente dopo il ciclo (se presente), ma interrompe solo l'esecuzione dell'iterazione in corso, e sposta il controllo all'inizio del ciclo stesso. Dopo l'esecuzione dell'istruzione `continue` l'iterazione sarà quindi valutata nuovamente.

Come l'istruzione `break`, non aggiunge espressività al linguaggio, ma in alcuni casi permette di scrivere un codice più leggibile e compatto

Istruzione `Continue`: Esempio

```
var = 10
while var > 0:
    var = var -1
    if var == 5:
        continue
    print("valore variabile corrente:", var)
print("fine!")
```



Istruzione iterativa FOR

L'istruzione iterativa `for`

Come altri linguaggi, anche Python include una versione alternativa dell'istruzione iterativa `while`: l'istruzione `for`.

Nel caso di Python l'istruzione `for` consente di esprimere un solo tipo di iterazione che consiste nell'accedere a **tutti** gli elementi di una sequenza, come, ad esempio una stringa, un range di interi o una lista (che vedremo più avanti). Tale sequenza è anche detta **contenitore**, cioè è un oggetto che contiene una raccolta di elementi.

L'accesso avviene dal primo all'ultimo elemento (non è possibile modificare tale ordine).

In questo tipo di operazione il vantaggio dell'istruzione `for` rispetto a `while` è la maggiore concisione, come si vedrà tra poco.

L'istruzione iterativa `for`: sintassi

```
for v in s
    sequenza di istruzioni
```

- v deve essere il nome di una variabile
- s deve essere un'espressione avente come valore una sequenza (una lista o una stringa)
- sequenza di istruzioni è una sequenza di una o più istruzioni qualsiasi, che devono essere scritte rispettando la stessa regola sui rientri già vista per l'istruzione `while` e per l'struzione `if`

L'istruzione iterativa `for`: semantica

La sequenza di istruzioni viene eseguita per un numero di volte pari alla lunghezza di s.

Nella *i* -esima iterazione, prima dell'esecuzione di sequenza di istruzioni, alla variabile v viene assegnato il valore dell'*i*-esimo elemento di s; la sequenza di istruzioni può quindi elaborare tale valore accedendo a v

Si noti che con l'istruzione `for` non è necessario usare esplicitamente gli indici per accedere agli elementi di una sequenza (ad esempio gli indici usati per le posizioni dei caratteri in una stringa).

Esempio

Un programma che stampa gli elementi di una stringa acquisita attraverso la tastiera :

```
# file stampaStringa.py

sequenza = input ("Inserire una stringa: ")
print ("I suoi elementi sono: ")
for elemento in sequenza :
    print (elemento)
```


Esempio scritto con un ciclo while

Un programma che stampa gli elementi di una stringa acquisita attraverso la tastiera :

```
sequenza = input ("Inserire una stringa: ")
print ("I suoi elementi sono: ")
i = 0
while (i < len(sequenza))
    print (sequenza[i])
    i = i + 1
```

Confronto tra `while` e `for`

L'istruzione `for` è un'alternativa conveniente all'istruzione `while` per **accedere agli elementi di una sequenza** ed elaborarne i valori, senza usare esplicitamente una variabile con il ruolo di indice.

In particolare, si consideri un'istruzione `for` avente la seguente struttura, dove v indica una variabile qualsiasi e s indica una stringa (o una lista):

```
for v in s
    istruzioni che elaborano v
```

Non è difficile vedere che tale istruzione svolge la stessa operazione della seguente sequenza di istruzioni, dove k indica una variabile usata come indice:

```
k = 0
while k < len(s)
    istruzioni che elaborano s[k]
    k = k + 1
```

Nota: con l'istruzione `for` non possiamo esprimere tutte le forme di iterazione che esprimiamo con il `while`. In particolare con il solo `for` non possiamo esprimere iterazioni indefinite (anche se con l'uso del `break` nel corpo del ciclo si può di fatto rendere indefinita anche una iterazione codificata con il `for` – si sconsiglia però di condificare cicli indefiniti in tal modo).

La funzione `range`

La funzione `range` genera una sequenza di numeri interi, che è generalmente usata in un ciclo come base per l'iterazione

La funzione `range` ha tre argomenti, ma solo uno è obbligatorio.
Sintassi:

```
range([start], stop[, step])
```

con `start, stop e step` argomenti di tipo intero, `stop` unico obbligatorio, e `step` presente solo se lo sono gli altri due.

- `start`: numero iniziale della sequenza.
- `stop`: numero fino a cui si genera la sequenza, senza includerlo
- `step`: differenza fra due numeri consecutivi nella sequenza (il default è `1`)

Se `range` è invocata con un solo intero come argomento, questo è il valore di `stop`, e la sequenza generata parte da `0` e si ferma a `stop-1`.
Invece `range(x,y)` genera una sequenza da `x` a `y-1`, tale che due interi consecutivi nella sequenza differiscono di `1`.
`range(x,y,z)` opera come `range(x,y)`, ma due numeri consecutivi nella sequenza differiscono di `z`

Esempi

```
>>> # Un parametro
>>> for i in range(3):
        print(i)

0
1
2
>>> # Due parametri
>>> for i in range(3, 6):
        print(i)

3
4
5
```

Esempi

```
>>> # Tre parametri
>>> for i in range(4, 10, 2):
        print(i)

4
6
8
>>> # Contare all'indietro
>>> for i in range(0, -10, -2):
        print(i)

0
-2
-4
-6
-8
```

Esempio: fattoriale

Programma che stampa a schermo il fattoriale di un numero non negativo acquisito da tastiera

```
# file fattoriale_3.py

n = int(input("Inserire un numero naturale non negativo: "))
fatt = 1
for i in range(1,n+1):
     fatt = fatt * i
print("Il fattoriale di", n, "e'", fatt)
```

Esempio: Somma numeri pari

Programma che acquisisce attraverso la tastiera una sequenza di interi di lunghezza qualsiasi e stampa a schermo la somma dei soli numeri pari inseriti.

```
# File sommaPari_2.py

n = int(input("Quanti numeri si vogliono immettere? "))
somma = 0
for k in range(n) :
    x = int(input("Prossimo valore: "))
    if (x % 2 == 0):
         somma = x + somma
print ("La somma e'", somma)
```

Esempio: Tabelline

Scrivere un programma Python che crea la tabellina di un numero fra 1 e 10 inserito in input dall'utente

```
# versione con il while

y = int(input("Introduci un intero da 1 a 10: "))
cont = 1
while (cont <= 10):
  print(y, "x", cont,"\t", cont*y)
  cont = cont + 1
```

```
# versione con il for

y = int(input("Introduci un intero da 1 a 10: "))
for cont in range(1,11):
  print(y, "x", cont,"\t", (cont)*y)
```

Esempio: Palindroma

Scrivere un programma Python che chiede di inserire in input una stringa e stampa a schermo "Palindroma!" se e solo se la stringa in input è palindroma

```
# versione con il while

parola = input("Inserisci la parola: ")

cont = 0
ret = ""
while (cont < len(parola)):
  ret = parola[cont] + ret
  cont = cont + 1
if ret == parola:
  print("Palindroma!")
else:
  print("Non Palindroma!")
```

Esempio: Palindroma

Scrivere un programma Python che chiede di inserire in input una stringa e stampa a schermo "Palindroma!" se e solo se la stringa in input è palindroma

```
# versione con il for

parola = input("Inserisci parola: ")

ret = ""
for cont in parola :
  ret = cont + ret
if ret == parola:
  print("Palindroma!")
else:
  print("Non Palindroma!")
```

Ciclo for annidato: esempio 1

Programma che stampa un rettangolo di asterischi di dimensione n x m (senza usare la moltiplicazione di stringa per intero)

```
# File rettangoloAsterischi.py

n = int(input("numero righe"))
m = int(input("numero colonne"))
for j in range(n) :
  riga = ""
  for i in range(m) :
    riga = riga + '*'
  print(riga)
```

Ciclo for annidato: esempio 1

Programma che stampa un rettangolo di asterischi di dimensione n x m (senza usare la moltiplicazione di stringa per intero)

```
# soluzione alternativa non usa la variabile
# riga ma stampa carattere per carattere

n = int(input("numero righe: "))
m = int(input("numero colonne: "))
for j in range(n) :
    for i in range(m) :
      print('*', end="")
    print()
```

Ciclo for annidato: esempio 2

Nell'esempio precedente, il numero di iterazioni del ciclo annidato non dipende dalla particolare iterazione del ciclo esterno in cui è eseguito. Ora vediamo invece un esempio in cui il ciclo annidato viene eseguito un numero di volte che dipende dall'iterazione esterna.

Esempio: Programma che legge un numero `n` da tastiera e stampa a schermo un triangolo di interi di `n` righe e della forma seguente

```
1
12
123
…
123456…n-1
123456….n-1n
```

Ciclo for annidato: esempio 2

```
# file triangoloNumeri.py

n = int(input("inserisci un numero : "))
for i in range(1,n+1) :
  for j in range(1,i+1):
    print(j, end="")
  print()
```

Ciclo for annidato: esempio 2

In realtà questo problema si poteva risolvere anche con un solo ciclo `for`, ed è stato mostrato con un ciclo annidato principalmente per scopi didattici. Di seguito viene proposta una soluzione alternativa che usa un solo ciclo `for`, e memorizza la riga stampata per usarla nell'iterazione successiva

```
n = int(input("inserisci un numero : "))
riga = ""
for i in range(1,n+1) :
  riga = riga + str(i)
  print(riga)
```

Ciclo for annidato: esempio 3

Presa in input una stringa, seguendo l'ordine dei caratteri nella stringa, per ogni carattere *C* si stampi gli indici relativi alla sua posizione nella sottostringa che inizia da *C*. Ad esempio, se la stringa in input è `abba`, stampa a schermo

```
posizioni del carattere 'a' a partire dall'indice 0: 0 3
posizioni del carattere 'b' a partire dall'indice 1: 1 2
posizioni del carattere 'b' a partire dall'indice 2: 2
posizioni del carattere 'a' a partire dall'indice 3: 3
```

```
# file posizioniCarattereInStringa.py

stringa = input("immetti una stringa: ")
for i in range(len(stringa)) :
  print("posizioni del carattere", "'"+stringa[i]+"'", \
        "a partire dall'indice", i,":", i, end=" ")
  for j in range(i+1, len(stringa)):
       if (stringa[i] == stringa[j]):
            print(j, end=" ")
  print()
```

`break` e `continue` nel ciclo `for`

Analogamente al ciclo `while`, anche nel ciclo `for` è possibile usare l'istruzione `break` e/o l'istruzione `continue`, con lo stesso significato visto per il `while`

Esempio:

```
# file MCD_5.py

m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
if m < n :
  start = m
else :
  start = n
for mcd in range(start,0,-1) :
    if m % mcd == 0 and n % mcd == 0  :
       break
print ("Il MCD di", m, "e", n, "e'", mcd)
```