Strutture blocco

- Indentazione e istruzione blocco
  - In Python il blocco di istruzioni è definito tramite il meccanismo dell'indentazione.
  - L'indentazione consiste nell'inserire un certo numero di spazi (o tab) prima del codice.
  - Tutte le istruzioni del blocco relativo ad uno statement composto devono essere indentate dello stesso numero di spazi.
  - Python individua i blocchi come istruzioni con la stessa indentazione.
  - Un'struzione con indentazione uguale a quella dell'intestazione indica la fine del blocco e l'inizio di un'istruzione al di fuori di esso.
  - E' possibile annidare arbitrariamente blocchi uno dentro l'altro, aumentando l'indentazione ad ogni annidamento.

Strutture di controllo - if

- Costrutto `if` – sintassi semplice:

```
if <condizione>:
        <istruzione>
```

- Esempio:

```
if x < 0:
   x = 0
   print 'Numero negativo cambiato in zero”
```

- `if / else` – sintassi:

```
if <condizione>:
        <istruzione_true>
else:
        <istruzione_false>
```

- Esempio:

```
if x < 0:
   x = 0
   print 'Numero negativo cambiato in zero”
else:
  print “Numero maggiore o uguale a zero”
```

Strutture di controllo - if

- Costrutto `if / elif / else` – sintassi

```
if <condizione_1>:
        <istruzione_1>
elif <condizione_2>:
        <istruzione_2>
….
else:
        <istruzione_false>
```

- Esempio:

```
x = int(raw_input("Introdurre un numero: "))
if x < 0:
        x = 0
        print 'Numero negativo cambiato in zero'
elif x == 0:
         print 'Zero
elif x == 1:
         print 'Uno ...
else:
        print 'Più di uno ...
```

- Possono essere presenti o meno, una o più parti `elif` e la parte `else` è facoltativa. La parola chiave `elif` è un'abbreviazione di else if, e serve ad evitare un eccesso di indentazioni. Una sequenza if ... elif ... elif ... sostituisce le istruzioni switch o case che si trovano in altri linguaggi.

Strutture di controllo - while

- `while` - Iterazione (ciclo) con controllo in testa.
- Sintassi

```
while <condizione>:
    <istruzione>
```

- Esempio:

```
while x != 0:
  x = int(raw_input("Inserire un numero (0 per uscire: "))
```

Strutture di controllo - while

- `break` serve per uscire dal ciclo, saltando le rimanenti istruzioni presenti nel ciclo.
- `continue` permette di saltare all'intestazione del ciclo corrente senza terminare la sequenza di istruzioni.
- Esempio:

```
while True:
   x = int(raw_input("Inserire un numero (0 per uscire: "))
   if x == 0:
     break
   print 'Numero inserito: ', x
```

- Esempio:

```
while True:
   x = int(raw_input("Inserire un numero (0 per uscire: "))
   if x != 0:
     print 'Numero inserito: ', x
     continue
   break
```

Strutture di controllo – while/else

- Spesso al termine di un ciclo è utile sapere se si è usciti “normalmente” per il fallimento della condizione, o a causa di un `break`
- Tale situazione può essere gestita in Python aggiungendo un `else` finale dopo il blocco annidato, che verrà eseguito solo se si esce per il fallimento della condizione.

- Esempio:

```
while s != “parolasegreta”:
  s = raw_input(“Inserire la parola segreta (exit per uscire)”)
  if s == “exit”:
    break
else:
 print “parola segreta indovinata!!!”
```

Strutture di controllo - condizioni

- Le condizioni usate nelle istruzioni `while` e `if` possono contenere oltre a quelli di confronto classici altri operatori.
- Gli operatori di confronto `not in` e `in` verificano se un valore compare (o non compare) in una sequenza.
- Gli operatori **is** ed **is not** confrontano due oggetti per vedere se siano in realtà lo stesso oggetto; questo ha senso solo per oggetti mutabili come le liste.
- Tutti gli operatori di confronto hanno la stessa priorità che è minore di quella di tutti gli operatori matematici.
- I confronti possono essere concatenati. Per esempio, a < b == c verifica se a sia minore di b e inoltre se b eguale a c.

Strutture di controllo - condizioni

- Sono disponibili gli operatori booleani `and`, `or`, `not`; `and` e `or` sono dei cosiddetti operatori short-circuit; i loro argomenti vengono valutati da sinistra a destra e la valutazione si ferma non appena viene determinato il risultato. Per esempio se A e
- B è falso, A and B and C non valuta l'espressione C.
- È possibile assegnare il risultato di un confronto o di un'altra espressione booleana a una variabile.
- Si noti che in Python, a differenza del C, all'interno delle espressioni non può comparire un assegnamento.
- Ciò evita di sostituire = con == accidentalmente

Strutture di controllo - for

- L'istruzione FOR di Python differisce un po' da quella a cui si è abituati in C. Piuttosto che iterare sempre su una progressione aritmetica, o dare all'utente la possibilità di definire sia il passo iterativo che la condizione di arresto (come in C), in Python l'istruzione for compie un'iterazione sugli elementi di una qualsiasi sequenza (p.e. una lista o una stringa), nell'ordine in cui appaiono nella sequenza.
- Sintassi:

```
for <elemento> in <oggetto>:
        <istruzioni>
```

- Esempio1:

```
for i in [1,2,3,4,5]:
  sum += i
```

- Esempio2:

```
# Misura la lunghezza di alcune stringhe:
a = ['gatto', 'finestra', 'defenestrare']
for x in a:
   print x, len(x)
```

output:

```
gatto 5
finestra 8
defenestrare 12
```

Strutture di controllo - for

- Come il ciclo `while`, anche il ciclo `for` può contenere `break`, `continue` ed `else` finale.

```
for i in [1,-2,5,-4, 3]:
    sum += i
    if sum > max:
      break
else
  print sum < max
```

Strutture di controllo - for

- Il `for` permette di assegnare l'elemento successivo ad una tupla:

```
dict = {1234 : "aaabbcc", 567 : ”ddeeff"}
for (k,v) in dict.items():
     if v.find(”ee") > 0:
          print k
          break
```

Strutture di controllo - for

- Se è necessario iterare su una successione di numeri viene in aiuto la funzione built-in `range()`, che genera liste contenenti progressioni aritmetiche, ad esempio:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- L'estremo destro passato alla funzione non fa mai parte della lista range(10) genera una lista di 10 valori, esattamente gli indici leciti per una sequenza di lunghezza 10.
- E' possibile far partire l'intervallo da un altro numero, o specificare un incremento diverso:

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Strutture di controllo - for

- Per effettuare un'iterazione sugli indici di una sequenza, si possono usare in combinazione `range()` e `len()` come segue:

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print i, a[i]
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

Strutture di controllo - for

- Utilizzando un ciclo su due o più sequenze contemporaneamente le voci possono essere accoppiate con la funzione zip().

```
domande = ['nome', 'scopo', 'colore preferito’]
risposte = ['lancillotto', 'il santo graal', lL blu']
for q, a in zip(domande, risposte):
        print 'Qual'e` il tuo %s? E` %s.' % (q, a)
```

Output:

```
Qual'e` il tuo nome? E` lancillotto.
Qual'e` il tuo scopo? E` il santo graal.
Qual'e` il tuo colore preferito? E` il blu.
```

- Per eseguire un ciclo ordinato su di una sequenza si deve usare la funzione sorted() che restituisce una nuova lista ordinata finché rimane inalterata la sorgente dei dati da elaborare

```
basket = ['apple', 'orange', 'apple', 'pear', 'range', 'banana']
for f in sorted(set(basket)):
        print f
```

Output:

```
apple banana orange pear
```

Strutture di controllo – for e iteratori

- L'istruzione `for` itera su un oggetto tramite un iteratore.
- Un iteratore si crea tramite la funzione `iter()` utilizzando come parametro oggetti iterabili (per esempio una lista).
- Una volta creato l'iteratore, `for` usa il metodo `next()` per ottenere il prossimo elemento dell'oggetto su cui si itera.
- `next()` genera un'eccezione quando si arriva alla fine dell'oggetto e il `for`, a quel punto, termina il ciclo.
- L'istruzione `for` rende tali operazioni trasparenti al programmatore.

```
>>>lista = [1,2,3]
>>>it = iter(lista)
>>>it.next()
1
>>>it.next()
2
>>>it.next()
3
>>>it.next()
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
StopIteration
```

Strutture di controllo – for e iteratori

- Qualsiasi oggetto iterabile può essere attraversato con un `for`.
- Per esempio, un file è un oggetto iterabile e l'iteratore del file restituisce una riga per volta, fino alla fine del file.

```
for line in open(”esempio.txt"):
      print line
```

Output:

```
linea di testo 1
linea di testo 2
…
```