Tutorial per principianti in Python

Josh Cogliati

22 agosto 2003

Copyright(c) 1999-2002 Josh Cogliati.

Permission is granted to anyone to make or distribute verbatim copies of this document as received, in any medium, provided that the copyright notice and permission notice are preserved, and that the distributor grants the recipient permission for further redistribution as permitted by this notice.

Permission is granted to distribute modified versions of this document, or of portions of it, under the above conditions, provided also that they carry prominent notices stating who last altered them.

All example python source code in this tutorial is granted to the public domain. Therefore you may modify it and relicense it under any license you please.

Sommario

Il Tutorial per principianti in Python è un documento pensato per essere una introduzione alla programmazione in Python, è destinato infatti a chi non ha esperienze con la programmazione.

Se qualcuno di voi ha già programmato con altri linguaggi, vi raccomando il Python Tutorial scritto da Guido van Rossum.

Questo documento è disponibile in LaTeX, HTML, PDF e Postscript. Potete reperirlo in tutti i formati elencati presso http://www.honors.montana.edu/~jjc/easytut/.

Se qualcuno di voi ha domande da pormi, contattatemi al mio indirizzo jjc@iname.com, (nicholas_wieland@yahoo.it per la presente versione tradotta) commenti e suggerimenti riguardo questo documento sono i benvenuti. Proverò a rispondere a qualsiasi domanda meglio che potrò.

I ringraziamenti vanno a James A. Brown per aver scritto la maggior parte della sezione dedicata all'installazione sotto Windows. Grazie anche ad Elizabeth Cogliati per le proteste :) circa la prima stesura del tutorial, (quasi inutilizzabile da un principiante) per averlo revisionato ed aver fornito molte idee e commenti. Grazie anche a Joe Oppergaard per aver scritto tutti gli esercizi. Grazie anche a tutti quelli che ho dimenticato.

Versione in lingua italiana del Tutorial per principianti in Python a cura di Nicholas Wieland e Ferdinando Ferranti.

Per quanto riguarda eventuali aggiornamenti della versione Italiana rivolgersi a: nicholas_wieland@yahoo.it o zappa-galattica@inwind.it, i successivi aggiornamenti di questa traduzione saranno reperibili presso il sito locale di Python: http://www.python.it.

Dedicato a Elizabeth Cogliati

INDICE

1 Introduzione **1**
- 1.1 La prima cosa . . . 1
- 1.2 Installare Python . . . 1
- 1.3 Modo interattivo . . . 1
- 1.4 Creare ed eseguire programmi . . . 2
- 1.5 Usare Python da riga di comando . . . 2

2 Hello, World **3**
- 2.1 Cosa dovresti già sapere . . . 3
- 2.2 Stampare . . . 3
- 2.3 Espressioni . . . 4
- 2.4 Parlare agli umani (e ad altri esseri intelligenti) . . . 6
- 2.5 Esempi . . . 6
- 2.6 Esercizi . . . 7

3 Chi va là? **9**
- 3.1 Input e variabili . . . 9
- 3.2 Esempi . . . 11
- 3.3 Esercizi . . . 13

4 Conta da 1 a 10 **15**
- 4.1 Cicli While . . . 15
- 4.2 Esempi . . . 17

5 Decisioni **19**
- 5.1 L'istruzione If . . . 19
- 5.2 Esempi . . . 20
- 5.3 Esercizi . . . 23

6 Debugging **25**
- 6.1 Cos'è il Debugging? . . . 25
- 6.2 Cosa dovrebbe fare il programma? . . . 25
- 6.3 Cosa fa il programma? . . . 26
- 6.4 Come posso riparare ad un errore nel programma? . . . 30

7 Definire le Funzioni **31**
- 7.1 Creare funzioni . . . 31
- 7.2 Variabili nelle funzioni . . . 32
- 7.3 Analizzare le funzioni . . . 34
- 7.4 Esempi . . . 36

7.5 Esercizi 38

8 Liste **39**
8.1 Variabili con più di un valore 39
8.2 Altre funzioni delle liste 40
8.3 Esempi 44
8.4 Esercizi 46

9 Cicli For **47**

10 Espressioni booleane **51**
10.1 Esempi 53
10.2 Esercizi 54

11 Dizionari **55**

12 Usare i moduli **63**
12.1 Esercizio 64

13 Ancora sulle liste **65**

14 La rivincita delle stringhe **71**
14.1 Esempi 75

15 File IO **77**
15.1 Esercizi 81

16 Occuparsi dell'imperfetto (o come gestire gli errori) **83**
16.1 Esercizi 84

17 Fine **85**

18 FAQ **87**

CAPITOLO

UNO

Introduzione

1.1 La prima cosa

Se state leggendo questo tutorial non avete mai programmato in vita vostra. Proseguite nella lettura e tenterò di insegnarvi come si programma. Innanzitutto chiarezza: c'è una sola via da percorrere per imparare a programmare, **dovete** leggere codice - scrivere codice e io ve ne farò leggere e scrivere tantissimo. Per questo dovrete sempre scrivere il codice degli esempi ed eseguirlo per poi vedere che cosa succede: giocate con il codice, modificatelo come volete, la cosa peggiore che può capitarvi è che il programma non funzioni. Per riconoscere il codice dalla normale scrittura userò questo formato:

```
# Python è facile da imparare
print "Hello, World!"
```

Facile da distinguere dal resto del testo no? Appunto per confondervi scriverò così anche l'output del computer :=)

Un'altra cosa importante: per programmare in Python vi serve Python. Se non avete ancora il software necessario dirigetevi verso http://www.python.org/download e scaricatevi la versione adatta alla vostra piattaforma, leggete le istruzioni e installatela.

1.2 Installare Python

Innanzitutto scaricatevi il file appropriato: Python 2.2, il Windows Installer nel caso usiate Windows; la versione rpm o i sorgenti da compilare se avete un sistema Unix.

Scaricando il Windows Installer avrete un file che basterà cliccare due volte per iniziare la procedura di installazione.

Scaricando i sorgenti Unix assicuratevi di compilare con l'estensione Tk per usare IDLE.

1.3 Modo interattivo

Aprite IDLE, la GUI di Python. Dovreste vedere una finestra di questo tipo:

```
Python 2.2.2 (#1, Mar 21 2003, 23:01:54)
[GCC 3.2.3 20030316 (Debian prerelease)] on linux2
Type "copyright", "credits" or "license" for more information.
IDLE 0.8 -- press F1 for help
>>>
```

Il >>> è il modo che ha Python per informarvi che siete in modo interattivo, dove i comandi digitati sono immediatamente eseguiti. Provate a digitare `1+1` e Python vi risponderà immediatamente `2`. In questa modalità potete provare Python e vedere come reagisce ai vari comandi. Usatela quando sentirete il bisogno di prendere confidenza con i comandi Python.

1.4 Creare ed eseguire programmi

Andate in modo interattivo se non ci siete già. Selezionate `File` dal menu dopodiché selezionate `New Window`. Digitate nella nuova finestra il seguente testo (codice):

```
print "Hello, World!"
```

Innanzitutto salvate il programma sempre selezionando dal menu `File` e poi `Save`. Salvatelo come 'hello.py' nella directory predefinita o nella directory che preferite. Ora che avete salvato potete eseguire il programma.

Selezionate dal menu la voce `Edit` e poi `Run script`. Questa azione vi restituirà l'output richiesto dal programma 'hello.py' nella finestra `*Python Shell*` che saluterà allegramente con un: `Hello, World!`.

Confusi? Provate questo tutorial per l'IDLE: http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html.

1.5 Usare Python da riga di comando

Se preferite non utilizzare Python da riga di comando non siete obbligati, usate IDLE. Per entrare in modo interattivo da riga di comando dovete semplicemente digitare `python`. Se volete eseguire un programma che avete scritto con un editor di testo (ad esempio Emacs è un ottimo editor per Python) non dovete fare altro che usare la sintassi `python nome programma`.

CAPITOLO

DUE

Hello, World

2.1 Cosa dovresti già sapere

Dovresti sapere come visualizzare un programma in un editor di testo, salvarlo, (su floppy o disco fisso) ed eseguirlo una volta salvato.

2.2 Stampare

Sin dall'inizio dei tempi i tutorial sono sempre iniziati con un semplice programma chiamato 'Hello World'. Eccolo:

```
print "Hello, World!"
```

Se state usando la riga di comando per eseguire i programmi, inserite la stringa di testo in un editor e salvate il file con il nome 'hello.py', eseguitelo con il comando "python hello.py".

Altrimenti, entrate in IDLE, aprite una nuova finestra (New window) e create il programma come spiegato nella sezione 1.4.

Ecco cosa stampa sullo schermo il programma quando viene eseguito:

```
Hello, World!
```

Non ve lo dirò ogni volta ma vi consiglio vivamente di ripetere tutti gli esercizi che vi mostro, questo vi aiuterà a comprenderli meglio, anch'io imparo di più quando scrivo, probabilmente anche voi ...

Tentiamo un programma un po' più complicato:

```
print "Jack and Jill went up a hill"
print "to fetch a pail of water;"
print "Jack fell down, and broke his crown,"
print "and Jill came tumbling after."
```

Quando eseguirete il programma l'output sul monitor sarà questo:

```
Jack and Jill went up a hill
to fetch a pail of water;
Jack fell down, and broke his crown,
and Jill came tumbling after.
```

Quando il computer esegue questo programma vede innanzitutto la prima linea:

```
print "Jack and Jill went up a hill"
```

Ed esegue l'ordine, ovvero stampa:

```
Jack and Jill went up a hill
```

Dopodiché il computer prosegue a leggere il codice e passa alla linea successiva:

```
print "to fetch a pail of water;"
```

Il risultato è la stampa di:

```
to fetch a pail of water;
```

Il computer continua a scendere di linea in linea seguendo e svolgendo le istruzioni che voi stessi gli ordinate finché non raggiunge la fine del programma.

2.3 Espressioni

Ecco qui un'altro programma:

```
print "2 + 2 is", 2+2
print "3 * 4 is", 3 * 4
print 100 - 1, " = 100 - 1"
print "(33 + 2) / 5 + 11.5 = ",(33 + 2) / 5 + 11.5
```

E qui l'output che questo programma produce:

```
2 + 2 is 4
3 * 4 is 12
99  = 100 - 1
(33 + 2) / 5 + 11.5 =  18.5
```

Come puoi vedere Python può trasformare il vostro costosissimo computer in una normale calcolatrice :-)

Python ha sei operatori basilari:

Operatore	Simbolo	Esempio
Elevamento a potenza	`**`	`5 ** 2 == 25`
Moltiplicazione	`*`	`2 * 3 == 6`
Divisione	`/`	`14 / 3 == 4`
Resto	`%`	`14 % 3 == 2`
Addizione	`+`	`1 + 2 == 3`
Sottrazione	`-`	`4 - 3 == 1`

Osservate come la divisione segua la regola per cui se nel dividendo e nel divisore non sono presenti decimali anche il risultato non conterrà decimali (questo però cambierà in Python 2.3). Il seguente programma dimostra la regola appena enunciata:

```
print "14 / 3 = ",14 / 3
print "14 % 3 = ",14 % 3
print
print "14.0 / 3.0 =",14.0 / 3.0
print "14.0 % 3.0 =",14 % 3.0
print
print "14.0 / 3 =",14.0 / 3
print "14.0 % 3 =",14.0 % 3
print
print "14 / 3.0 =",14 / 3.0
print "14 % 3.0 =",14 % 3.0
print
```

Con l'output:

```
14 / 3 =  4
14 % 3 =  2

14.0 / 3.0 = 4.66666666667
14.0 % 3.0 = 2.0

14.0 / 3 = 4.66666666667
14.0 % 3 = 2.0

14 / 3.0 = 4.66666666667
14 % 3.0 = 2.0
```

Python da risposte differenti in base alla presenza o meno di numeri decimali.

L'ordine delle operazioni è lo stesso che nella matematica:

1. parentesi `()`
2. elevamento a potenza `**`
3. moltiplicazione `*`, divisione \ e resto `%`
4. addizione + e sottrazione −

2.4 Parlare agli umani (e ad altri esseri intelligenti)

Vi capiterà sicuramente, quando sarete più esperti, di dover programmare applicazioni molto complesse e lunghe. Difficilmente rileggendo il codice dopo qualche tempo riuscirete a ricordarvi tutti i passaggi e tutti i ragionamenti fatti, per questo è meglio che prendiate da subito la buona abitudine di commentare il vostro lavoro. Ad esempio:

```
# Non è esattamente pi greco, ma un'incredibile simulazione.
print 22.0/7.0
```

Come potete vedere il commento inizia con il simbolo #. Un commento è semplicemente una nota, per altri programmatori ma anche per voi stessi, che spiega il programma nei punti salienti.

2.5 Esempi

Ogni capitolo conterrà esempi delle proprietà di programmazione introdotte nel capitolo stesso. Dovreste almeno dare un'occhiata al codice per vedere se riuscite a capirlo. Nel caso alcuni passaggi non fossero chiari potete scrivere il codice ed eseguirlo, per tentare di capirlo meglio o addirittura apportare delle modifiche per vedere cosa succede.

'Denmark.py'

```
print "Something's rotten in the state of Denmark."
print "                -- Shakespeare"
```

Output:

```
Something's rotten in the state of Denmark.
                -- Shakespeare
```

'School.py'

```
# Questo non è esattamente vero al di fuori degli USA, ed è basato su
# di un vago ricordo dei miei trascorsi giovanili.
print "Firstish Grade"
print "1+1 =",1+1
print "2+4 =",2+4
print "5-2 =",5-2
print
print "Thirdish Grade"
print "243-23 =",243-23
print "12*4 =",12*4
print "12/3 =",12/3
print "13/3 =",13/3," R ",13%3
print
print "Junior High"
print "123.56-62.12 =",123.56-62.12
print "(4+3)*2 =",(4+3)*2
print "4+3*2 =",4+3*2
print "3**2 =",3**2
print
```

Output:

```
Firstish Grade
1+1 = 2
2+4 = 6
5-2 = 3

Thirdish Grade
243-23 = 220
12*4 = 48
12/3 = 4
13/3 = 4  R  1

Junior High
123.56-62.12 = 61.44
(4+3)*2 = 14
4+3*2 = 10
3**2 = 9
```

2.6 Esercizi

Scrivete un programma che stampa su schermo il vostro nome e cognome in due stringhe separate.

Scrivete un programma che mostra l'utilizzo delle 6 operazioni matematiche.

CAPITOLO

TRE

Chi va là?

3.1 Input e variabili

Ora proviamo un programma un po' più complicato:

```
print "Halt!"
s = raw_input("Who Goes there? ")
print "You may pass,", s
```

Quando **io** lo eseguo, questo appare sul **mio** schermo:

```
Halt!
Who Goes there? Josh
You may pass, Josh
```

Ovviamente quando proverete anche voi l'esecuzione il vostro output sarà differente, grazie all'istruzione `raw_input`. Avviando il programma (perché lo state provando vero?) avrete notato che richiede l'inserimento del vostro nome e poi la pressione di Enter per continuare l'esecuzione; a quel punto viene visualizzato un messaggio seguito dal vostro nome. Questo è un esempio di input: il programma viene eseguito fino ad un certo punto, dopodiché attende un input di dati dall'utente.

Naturalmente un input sarebbe inutile se non avessimo nessun posto dove mettere l'informazione ottenuta, ecco quindi che entrano in gioco le variabili. Nel programma precedente `s` è, appunto, una variabile. Le variabili sono come delle scatole in cui si possono immagazzinare dati. Ecco un programma che ne spiega l'utilizzo:

```
a = 123.4
b23 = 'Spam'
first_name = "Bill"
b = 432
c = a + b
print "a + b is", c
print "first_name is", first_name
print "Sorted Parts, After Midnight or",b23
```

E questo è l'output:

```
a + b is 555.4
first_name is Bill
Sorted Parts, After Midnight or Spam
```

Come ho detto: le variabili immagazzinano dati. Nel programma qui sopra ad esempio sono `a`, `b23`, `first_name`, `b` e `c`. Le due tipologie base di variabili sono stringhe e numeri. Le stringhe sono sequenze di lettere, numeri e altri caratteri: nell'esempio `b23` e `first_name` sono variabili contenenti stringhe. `Spam`, `Bill`, `a + b is`, e `first_name is` sono stringhe, come potete notare sono sempre racchiuse da o `'`. Gli altri tipi di variabili presenti sono numeri.

Ora, abbiamo queste scatole chiamate variabili ed anche dati da metterci dentro. Il computer vedrà una linea come `first_name = "Bill"` che leggerà come "Metti la stringa `Bill` nella scatola (o variabile) `first_name`". Più tardi vedrà l'espressione `c = a + b` che leggerà come "Metti `a + b` o `123.4 + 432` o `555.4` all'interno della variabile `c`".

Ecco un'altro esempio sull'utilizzo delle variabili:

```
a = 1
print a
a = a + 1
print a
a = a * 2
print a
```

E naturalmente ecco l'output:

```
1
2
4
```

Anche se in tutti i casi elencati la variabile ha lo stesso nome, il computer la rilegge ogni volta: prima trova i dati da immagazzinare e poi decide dove immagazzinarli.

Un'ultimo esempio prima di chiudere il capitolo:

```
num = input("Type in a Number: ")
str = raw_input("Type in a String: ")
print "num =", num
print "num is a ",type(num)
print "num * 2 =",num*2
print "str =", str
print "str is a ",type(str)
print "str * 2 =",str*2
```

Ecco l'output ottenuto:

```
Type in a Number: 12.34
Type in a String: Hello
num = 12.34
num is a  <type 'float'>
num * 2 = 24.68
str = Hello
str is a  <type 'string'>
str * 2 = HelloHello
```

Osserva il modo in cui ho ottenuto dall'utente le due variabili: `input` per la variabile `numerica` e `raw_input` per la variabile `stringa`. Quando volete che l'utente inserisca un numero o una stringa utilizzate, rispettivamente, `input` o `raw_input`.

La seconda parte del programma utilizza la funzione `type` che vi informa sul tipo di variabile: le varibili numeriche sono di tipo `'int'` (numero 'intero') o `'float'` (numero 'decimale'), le stringhe sono di tipo `string`. Interi e decimali possono essere utilizzati per operazioni matematiche, le stringhe no. Tuttavia quando una stringa viene moltiplicata per un intero vengono aggiunte alla stringa copie di se stessa: ad es. `str * 2 = HelloHello`.

Le operazioni con le stringhe differiscono leggermente da quelle con i numeri. I seguenti, sono alcuni esempi che mostrano tali differenze.

```
>>> "This"+" "+"is"+" joined."
'This is joined.'
>>> "Ha, "*5
'Ha, Ha, Ha, Ha, Ha, '
>>> "Ha, "*5+"ha!"
'Ha, Ha, Ha, Ha, Ha, ha!'
>>>
```

Questa è una lista di alcune operazioni con le stringhe:

Operazione	Simbolo	Esempio
Ripetizione	*	`"i"*5 == "iiiii"`
Concatenazione	+	`"Hello, "+"World!" == "Hello, World!"`

3.2 Esempi

'Velocita_tempo.py'

```
# Questo programma calcola lo spazio, data velocità e tempo.
print "Input a rate and a distance"
rate = input("Rate:")
distance = input("Distance:")
print "Time:",distance/rate
```

Esecuzione:

```
> python velocita_tempo.py
Input a rate and a distance
Rate:5
Distance:10
Time: 2
> python velocita_tempo.py
Input a rate and a distance
Rate:3.52
Distance:45.6
Time: 12.9545454545
```

‘Area.py’

```
# Questo programma calcola perimetro ed area del rettangolo.
print "Calculate information about a rectangle"
length = input("Length:")
width = input("Width:")
print "Area",length*width
print "Perimeter",2*length+2*width
```

Esecuzione:

```
> python area.py
Calculate information about a rectangle
Length:4
Width:3
Area 12
Perimeter 14
> python area.py
Calculate information about a rectangle
Length:2.53
Width:5.2
Area 13.156
Perimeter 15.46
```

‘temperature.py’

```
# Converte Fahrenheit in Celsius.
temp = input("Farenheit temperature:")
print (temp-32.0)*5.0/9.0
```

Esecuzione:

```
> python temperature.py
Farenheit temperature:32
0.0
> python temperature.py
Farenheit temperature:-40
-40.0
> python temperature.py
Farenheit temperature:212
100.0
> python temperature.py
Farenheit temperature:98.6
37.0
```

3.3 Esercizi

Scrivete un programma che prenda due variabili stringa e due numeriche intere dall'utente, le concateni (unisca le due stringhe senza spazi) e le visualizzi sullo schermo, infine moltiplichi i due numeri interi su una nuova linea.

CAPITOLO

QUATTRO

Conta da 1 a 10

4.1 Cicli While

Ecco qui finalmente la nostra prima struttura di controllo. Solitamente il computer legge il nostro programma cominciando dalla prima linea per poi scendere da lì fino alla fine del codice. Le strutture di controllo influiscono sul programma cambiando l'ordine d'esecuzione dei comandi o decidendo se un determinato comando verrà eseguito o meno. A voi il sorgente di un primo esempio che utilizza la struttura di controllo while:

```
a = 0
while a < 10:
        a = a + 1
        print a
```

Quindi il risultato dell'esecuzione:

```
1
2
3
4
5
6
7
8
9
10
```

Cosa fa il programma? Prima di tutto vede la linea `a = 0` e assegna il valore zero alla variabile numerica a. Dopodiché vede il comando `while a < 10:` che ordina a Python di controllare se la variabile a è minore di 10, in questo caso a corrisponde al valore zero quindi è minore di 10, per questo motivo Python eseguirà tutte le istruzioni rientrate sotto la struttura while. In poche parole finché la variabile numerica a è minore di dieci Python esegue tutte le istruzioni tabulate sotto while.

Ecco un'altro esempio dell'uso di `while`:

```
a = 1
s = 0
print 'Enter Numbers to add to the sum.'
print 'Enter 0 to quit.'
while a != 0 :
        print 'Current Sum:',s
        a = input('Number? ')
        s = s + a
print 'Total Sum =',s
```

Appena eseguito questo script l'output è stato questo:

```
  File "sum.py", line 3
    while a != 0
                ^
SyntaxError: invalid syntax
```

Ho dimenticato il `:` dopo il while. Python avverte dell'errore e mostra all'utente dov'è il problema, segnando la linea di codice incriminata con un utile `^`. Dopo aver risolto il problema il programma funziona a meraviglia ed ecco finalmente il risultato:

```
Enter Numbers to add to the sum.
Enter 0 to quit.
Current Sum: 0
Number? 200
Current Sum: 200
Number? -15.25
Current Sum: 184.75
Number? -151.85
Current Sum: 32.9
Number? 10.00
Current Sum: 42.9
Number? 0
Total Sum = 42.9
```

Come puoi vedere `print 'Total Sum =',s` è eseguito solamente alla fine. Questo perché la struttura di controllo `while` influisce solamente sulle istruzioni tabulate (rientrate, indentate). Il simbolo `!=` significa diverso: `while a != 0 :` finché a è diverso da zero, vengono eseguite le istruzioni tabulate sotto il `while`.

Ora che abbiamo capito il ciclo `while`, possiamo programmare uno script che venga eseguito all'infinito. Una via molto facile per farlo è, ad esempio, questa:

```
while 1 == 1:
     print "Help, I'm stuck in a loop."
```

L'output di questo programma sarà una continua ripetizione della frase `Help, I'm stuck in a loop.` all'infinito, a meno che non lo fermiate premendo i tasti Control (o 'Ctrl') e 'c' (la lettera) contemporaneamente. Questo manderà un segnale di terminazione al programma (Nota: Talvolta è necessario premere anche enter dopo il Control-c).

4.2 Esempi

‘Fibonacci.py’

```
# Questo programma calcola la sequenza di Fibonacci.
a = 0
b = 1
count = 0
max_count = 20
while count < max_count:
    count = count + 1
    # Occorre tenere traccia finché ci sono cambiamenti
    old_a = a
    old_b = b
    a = old_b
    b = old_a + old_b
    # Attenzione che la virgola alla fine di un istruzione print
    # prosegue la stampa sulla stessa linea.
    print old_a,
print
```

Output:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

‘Password.py’

```
# Attende sino a quando non viene inserita la giusta password.
# Usate Control-C per fermare il programma senza password.

# Notate che se non viene inserita la giusta password, il ciclo
# while prosegue all'infinito.
password = "foobar"

# Notate il simbolo != (diverso).
while password != "unicorn":
    password = raw_input("Password:")
print "Welcome in"
```

Esecuzione:

```
Password:auo
Password:y22
Password:password
Password:open sesame
Password:unicorn
Welcome in
```

CAPITOLO

CINQUE

Decisioni

5.1 L'istruzione If

Credo sia meglio iniziare il capitolo con un esempio a caldo: ecco quindi un programma che calcola il valore assoluto di un numero:

```
n = input("Number? ")
if n < 0:
        print "The absolute value of",n,"is",-n
else:
        print "The absolute value of",n,"is",n
```

Output di due esecuzioni:

```
Number? -34
The absolute value of -34 is 34

Number? 1
The absolute value of 1 is 1
```

Quindi cosa fa il computer quando esegue questo codice? Prima di tutto chiede all'utente un numero grazie alla linea `n = input(Number? )` dopodiché legge la linea `if n < 0:`, se `n` è minore di zero, Python esegue la linea `print The absolute value of,n,is,-n`, altrimenti (cioè se il numero è maggiore di zero) esegue la linea `print The absolute value of,n,is,n`.

Più semplicemente, Python decide se l'*affermazione* `n < 0` è vera o falsa. Un'istruzione `if` è sempre seguita da un *blocco* di altre istruzioni indentate che vengono eseguite nel caso l'affermazione sia vera. In caso contrario vengono eseguiti i comandi indentati sotto l'istruzione `else`, ovvero quando l'affermazione `if` risulta falsa.

Python presenta svariati operatori associabili al confronto di una variabile, eccone una lista completa:

Operatore	Funzione
<	minore di
<=	minore o uguale a
>	maggiore di
>=	maggiore o uguale a
==	uguale
!=	diverso da
<>	variante di diverso da

Un'altra proprietà dell'istruzione `if` è la funzione `elif` . È un'abbreviazione di else if e significa: esegui il blocco di istruzioni tabulato sotto elif se la condizione necessaria al primo `if` è falsa e la condizione `elif` è vera. Ecco un esempio:

```
a = 0
while a < 10:
        a = a + 1
        if a > 5:
                print a," > ",5
        elif a <= 7:
                print a," <= ",7
        else:
                print "Neither test was true"
```

Questo è l'output:

```
1  <=  7
2  <=  7
3  <=  7
4  <=  7
5  <=  7
6  >  5
7  >  5
8  >  5
9  >  5
10  >  5
```

Osservate come la linea `elif a <= 7` entri in azione solamente quando la prima istruzione `if` risulta falsa. Come potete vedere, l'istruzione `elif` consente di effettuare test multipli all'interno di una singola istruzione if.

5.2 Esempi

'High_low.py'

```
# Giocate ad indovinare il numero alto-basso
# (originariamente scritto da Josh Cogliati, migliorato da Quique)

# Questo dovrebbe attualmente essere qualcosa che è semi casuale, come
# l'ultimo carattere del tempo o qualcos'altro, ma dovrete aspettare
# i prossimi capitoli (Esercizio supplementare, modificate con random,
# come richiesto nel capitolo sull'uso dei moduli).

number = 78
guess = 0

while guess != number :
    guess = input ("Guess a number: ")

    if guess > number :
        print "Too high"

    elif guess < number :
            print "Too low"

print "Just right"
```

Esecuzione:

```
Guess a number:100
Too high
Guess a number:50
Too low
Guess a number:75
Too low
Guess a number:87
Too high
Guess a number:81
Too high
Guess a number:78
Just right
```

'even.py'

```
# Chiedi un numero.
# Stampa se è pari o dispari.

number = input("Tell me a number: ")
if number % 2 == 0:
    print number,"is even."
elif number % 2 == 1:
    print number,"is odd."
else:
    print number,"is very strange."
```

Esecuzione:

```
Tell me a number: 3
3 is odd.

Tell me a number: 2
2 is even.

Tell me a number: 3.14159
3.14159 is very strange.
```

‘average1.py’

```
# Continua a chiedere numeri finché non viene immesso 0.
# Stampa la media dei valori.

count = 0
sum = 0.0
number = 1 # imposta un valore, altrimenti il ciclo
           # while terminerebbe immediatamente.

print "Enter 0 to exit the loop"

while number != 0:
    number = input("Enter a number:")
    count = count + 1
    sum = sum + number

count = count - 1 # Togli un’unità all’ultimo numero.
print "The average was:",sum/count
```

Esecuzione:

```
Enter 0 to exit the loop
Enter a number:3
Enter a number:5
Enter a number:0
The average was: 4.0

Enter 0 to exit the loop
Enter a number:1
Enter a number:4
Enter a number:3
Enter a number:0
The average was: 2.66666666667
```

‘average2.py’

```
# Continua a chiedere numeri finché count li richiede.
# Stampa la media dei valori.

sum = 0.0

print "This program will take several numbers than average them"
count = input("How many numbers would you like to sum:")
current_count = 0

while current_count < count:
    current_count = current_count + 1
    print "Number ",current_count
    number = input("Enter a number:")
    sum = sum + number

print "The average was:",sum/count
```

Esecuzione:

```
This program will take several numbers than average them
How many numbers would you like to sum:2
Number  1
Enter a number:3
Number  2
Enter a number:5
The average was: 4.0

This program will take several numbers than average them
How many numbers would you like to sum:3
Number  1
Enter a number:1
Number  2
Enter a number:4
Number  3
Enter a number:3
The average was: 2.66666666667
```

5.3 Esercizi

Scrivete un programma che chieda all'utente di indovinare una password, ma che dia al giocatore solamente 3 possibilità, fallite le quali il programma terminerà, stampando "È troppo complicato per voi".

Scrivete un programma che chieda due numeri. Se la somma dei due numeri supera 100, stampate "Numero troppo grande".

Scrivete un programma che chieda all'utente il nome. Se viene inserito il vostro nome, il programma dovrà rispondere con un "Questo è un bel nome", se il nome inserito è John Cleese o Michel Palin il programma dovrà rispondere con una battuta ;) mentre in tutti gli altri casi l'output del programma sarà un semplice "Tu hai un bel nome!".

CAPITOLO

SEI

Debugging

6.1 Cos'è il Debugging?

> Poco dopo aver iniziato a programmare ci siamo accorti di come non fosse affatto facile far fare ai programmi quello che volevamo. Avevamo scoperto il Debugging. Posso ricordare il preciso momento in cui ho realizzato che avrei passato buona parte della mia vita a scoprire e correggere gli errori nei miei stessi programmi.
>
> – Maurice Wilkes scopre il debugging, 1949

Se finora avete ripetuto tutti i programmi degli esempi vi sarete certamente accorti che a volte (spesso) il programma da voi scritto assume comportamenti differenti da quelli che avevate previsto. È molto comune. Il Debugging è il processo grazie al quale portate il programma a svolgere le funzioni per cui è stato scritto correggendo gli errori, e vi assicuro, può essere un'operazione lunga e snervante. Una volta ho impiegato un'intera settimana per correggere un bug dovuto allo scambio di una `x` con una `y`.

Questo capitolo sarà più astratto dei precedenti. Vi prego di dirmi se vi è sembrato utile o meno.

6.2 Cosa dovrebbe fare il programma?

La prima cosa da fare (mi sembra ovvio) è pensare a cosa dovrebbe fare il programma se fosse corretto. Iniziate ad eseguire qualche test per vedere che cosa succede. Ad esempio, diciamo che ho scritto un programma che calcola il perimetro di un rettangolo (la somma dei quattro lati) e che ho intenzione di testarne il funzionamento immettendo i seguenti casi:

Larghezza	Altezza	Perimetro
3	4	14
2	3	10
4	4	16
2	2	8
5	1	12

Ora avvierò il mio programma ed inserirò i dati dei test per vedere se restituisce i risultati che mi aspetto. In caso contrario dovrò scoprire cosa sta facendo il computer.

Comunemente alcuni casi restituiranno il risultato che voglio, mentre altri risulteranno sbagliati, quindi dovrete vedere cos'hanno in comune i casi funzionanti. Ecco ad esempio l'output del programma che calcola il perimetro del rettangolo:

```
Height: 3
Width: 4
perimeter =  15


Height: 2
Width: 3
perimeter =  11


Height: 4
Width: 4
perimeter =  16


Height: 2
Width: 2
perimeter =  8


Height: 5
Width: 1
perimeter =  8
```

Il programma restituisce un risultato errato nei primi due casi, corretto nei seguenti due, per sbagliare di nuovo nell'ultimo. Provate ad immaginarvi cos'hanno in comune i casi funzionanti. Quando avrete un'idea sul motivo del malfunzionamento, trovarne la causa sarà più semplice. Con un vostro programma dovreste testare i casi che vi interessano.

6.3 Cosa fa il programma?

La prossima cosa da fare è rileggere il codice sorgente. Una delle cose più importanti da fare mentre si programma è leggere e rileggere il codice sorgente. Innanzitutto analiziamo il codice linea per linea, comportandoci esattamente come farebbe il programma.

Grazie alle strutture di controllo, ovvero i cicli `while` e le istruzioni `if`, alcune linee di codice possono non essere eseguite, mentre altre possono essere eseguite più di una volta. L'importante è che immaginiate ad ogni linea il comportamento di Python.

Iniziamo con il semplice programma per calcolare il perimetro. Non scrivetelo, dovete leggerlo e basta. Questo è il sorgente:

```
height = input("Height: ")
width = input("Width: ")
print "perimeter = ",width+height+width+width
```

Domanda: Qual'è la prima linea che Python esegue?

Risposta: La prima linea è sempre la prima. In questo caso è: `height = input("Height: ")`.

Domanda: Qual'è il suo effetto?

Risposta: Stampa `Height:` sullo schermo e attende l'inserimento di un numero da parte dell'utente per assegnarlo alla variabile height.

Domanda: Qual'è la prossima linea che Python esegue?

Risposta: In generale, è la linea successiva, che è `width = input("Width: ")`.

Domanda: Qual'è il suo effetto?

Risposta: Stampa `Width:`, attende che l'utente inserisca un numero, quindi assegna il valore alla variabile width.

Domanda: Qual'è la prossima linea che eseguirà?

Risposta: La linea successiva è questa: `print "perimeter = ",width+height+width+width`. Si può anche eseguire la funzione dalla corrente linea, ma lo vedremo meglio nel prossimo capitolo.

Domanda: Qual'è il suo effetto?

Risposta: Per prima cosa stampa `perimeter =` e poi stamperà `width+height+width+width`.

Domanda: Questa stringa `width+height+width+width` calcola il perimetro?

Risposta: Vediamo: il perimetro di un rettangolo si calcola sommando la base (width) più il lato sinistro (height), più il lato superiore (widht), più il lato destro (huh?). L'ultimo elemento, (width) dovrebbe essere il lato destro (height) e non la base (width).

Domanda: È per questo motivo che alcune volte il perimetro risultava corretto?

Risposta: Certo, risultava corretto perché tutti i lati erano uguali.

Nel prossimo programma vedremo come verrà analizzato il codice, supponendo che il programma stampi 5 punti sullo schermo. Dunque, questo è l'output del programma:

```
. . . .
```

E questo il programma:

```
number = 5
while number > 1:
    print ".",
    number = number - 1
print
```

Questo programma sarà un po' più complesso, perché, come potete notare, nel codice è presente una porzione indentata (o una struttura di controllo). Bando alle ciance, rimbocchiamoci le maniche e cominciamo.

Domanda: Qual'è la prima linea ad essere eseguita?

Risposta: La prima linea del file: `number = 5`

Domanda: Qual'è il suo effetto?

Risposta: Assegna il valore 5 alla variabile number.

Domanda: Qual'è la prossima linea?

Risposta: La prossima linea è: `while number > 1:`

Domanda: Qual'è il suo effetto?

Risposta: La struttura `while` verifica che l'istruzione indentata che segue sia vera o falsa. Nel primo caso esegue il codice indentato sotto, nel secondo caso lo salta per passare al codice indentato al suo stesso livello.

Domanda: Quindi in questo caso come si comporta?

Risposta: Se la condizione `number > 1` è vera allora saranno eseguite le prossime due linee.

Domanda: Allora `number` è maggiore di uno?

Risposta: L'ultimo valore assegnato a `number` è `5` e quindi cinque è maggiore di uno.

Domanda: Qual'è la prossima linea?

Risposta: Siccome la condizione `while` è vera, la linea successiva sarà `print ".",`.

Domanda: Qual'è il suo effetto?

Risposta: Stamperà un punto, e siccome alla fine dell'istruzione compare una virgola (`","`), lo stamperà sulla stessa linea del precedente.

Domanda: Qual'è la prossima linea?

Risposta: `number = number - 1` visto che la linea successiva non modifica il livello d'indentazione.

Domanda: Qual'è il suo effetto?

Risposta: Calcola `number - 1`, dato che il valore corrente di `number` è 5, gli viene sottratta una unità, creando così il nuovo valore. Sostanzialmente il valore di `number` viene modificato da 5 a 4.

Domanda: Qual'è la prossima linea?

Risposta: Dato che il livello d'indentazione decrementa, dobbiamo pensare che tipo di struttura di controllo stiamo analizzando. Preso atto che è un ciclo `while`, dobbiamo tornare alla linea `while number > 1:`

Domanda: Qual'è il suo effetto?

Risposta: Occorre controllare il valore della variabile number, che è 4 e confrontarla con 1, siccome `4 > 1` il ciclo while prosegue.

Domanda: Qual'è la prossima linea?

Risposta: Siccome il ciclo while è vero, la prossima linea sarà sempre: `print ".",`

Domanda: Qual'è il suo effetto?

Risposta: Stamperà il secondo punto sulla linea.

Domanda: Qual'è la prossima linea?

Risposta: L'indentazione non è cambiata e quindi: `number = number - 1`

Domanda: Ed ora cosa succede?

Risposta: Al corrente valore di number (4), sottraiamo 1, che restituisce 3, quindi il nuovo valore assegnato alla variabile number sarà 3.

Domanda: Qual'è la prossima linea?

Risposta: Siccome è verificata la condizione del ciclo while, la linea successiva sarà sempre `print ".",`.

Domanda: Qual'è la prossima linea?

Risposta: Siccome l'indentazione non è modificata, il che causerebbe l'interruzione del ciclo, la prossima linea sarà sempre: `while number > 1:`.

Domanda: Qual'è il suo effetto?

Risposta: Compara il valore corrente di number (3) con 1. `3 > 1` perciò il ciclo while continua.

Domanda: Qual'è la prossima linea?

Risposta: Siccome è verificata la condizione del ciclo while, la linea successiva sarà sempre `print ".",`.

Domanda: Ed ora cosa succede?

Risposta: Un altro punto sulla linea.

Domanda: Qual'è la prossima linea?

Risposta: È sempre: `number = number - 1`

Domanda: Qual'è il suo effetto?

Risposta: Prende il valore corrente di number (3), gli sottrae 1 e imposta a 2 il nuovo valore.

Domanda: E adesso cosa succede?

Risposta: Si ricomincia da capo: `while number > 1:`

Domanda: Qual'è il suo effetto?

Risposta: Confronta il valore corrente di number (2) con 1. Siccome `2 > 1` il ciclo while continua.

Domanda: Qual'è la prossima linea?

Risposta: Il ciclo while prosegue: `print "."`,

Domanda: Qual'è il suo effetto?

Risposta: Scoprirete il senso della vita, l'universo sarà vostro. Sto scherzando (così sono sicuro che vi sveglierete). Stampa il quarto punto sullo schermo.

Domanda: Qual'è la prossima linea?

Risposta: È: `number = number - 1`

Domanda: Qual'è il suo effetto?

Risposta: Prende il valore corrente di number (2) gli sottrae 1 e imposta ad 1 il nuovo valore.

Domanda: Qual'è la prossima linea?

Risposta: Torniamo al ciclo while: `while number > 1:`

Domanda: Qual'è il suo effetto?

Risposta: Compara il corrente valore di number (1) con 1. Siccome `1 > 1` è falso (uno non è più grande di uno), il ciclo while termina.

Domanda: Qual'è la prossima linea?

Risposta: Siccome la condizione del ciclo while è falsa, la prossima linea analizzata, dopo l'uscita dal ciclo, sarà `print`.

Domanda: Qual'è il suo effetto?

Risposta: Stampa sullo schermo un'altra linea, vuota.

Domanda: Ma il programma non doveva stampare 5 punti?

Risposta: Il ciclo termina troppo presto e non stampa 1 punto.

Domanda: Come posso riparare questo difetto?

Risposta: Creare un ciclo che termina dopo aver stampato l'ultimo punto.

Domanda: E come posso realizzarlo?

Risposta: Ci sono molte possibilità. Una via è modificare il ciclo così: `while number > 0:`. Un'altra è questa: `number >= 1`. Esistono molti altri metodi.

6.4 Come posso riparare ad un errore nel programma?

Dovete capire cosa sta facendo il programma. Immaginarvi cosa dovrebbe fare il programma, confrontare e capire le differenza dei metodi che provate. Il Debugging si impara grazie all'esperienza. Se non riuscite a trovare il difetto dopo un'ora o due, fate una pausa e parlatene con qualcuno. Quando ci ritornerete sopra probabilmente avrete nuove idee per risolvere il problema. Buona fortuna.

CAPITOLO

SETTE

Definire le Funzioni

7.1 Creare funzioni

Inizierò questo capitolo mostrandovi, con un esempio, cosa potreste, ma non dovreste fare (quindi non scrivete):

```
a = 23
b = -23

if a < 0:
    a = -a

if b < 0:
    b = -b

if a == b:
    print "The absolute values of", a,"and",b,"are equal"
else:
    print "The absolute values of a and b are different"
```

Con questo output:

```
The absolute values of 23 and 23 are equal
```

Il programma sembra un po' ripetitivo (e i programmatori odiano ripetere le cose, altrimenti i computer a cosa servono??). Fortunatamente Python permette di creare funzioni per rimuovere i duplicati. Ecco l'esempio riscritto:

```
a = 23
b = -23

def my_abs(num):
    if num < 0:
        num = -num
    return num

if my_abs(a) == my_abs(b):
    print "The absolute values of", a,"and",b,"are equal"
else:
    print "The absolute values of a and b are different"
```

Con questo output:

```
The absolute values of 23 and -23 are equal
```

L'elemento chiave di questo programma è l'istruzione `def`. `def` (abbreviazione di define, definisci) inizializza la definizione di una funzione. `def` è seguita dal nome della funzione `my_abs`. Poi troviamo una "`(`" seguita dal parametro `num` (`num` viene passato dal programma ad ogni chiamata di funzione). Le istruzioni dopo il `:` vengono eseguite ogni volta che la funzione viene usata dal programma e si possono distinguere dalle normali righe di codice perché sono indentate sotto la funzione. Quando l'indentazione torna al livello della funzione o quando incontra l'istruzione `return` la funzione termina. L'istruzione `return` ritorna il valore assunto nel punto in cui è chiamata nella funzione.

Osservate che i valori `a` e `b` non sono cambiati. Le funzioni, naturalmente, possono essere usate anche per ripetere azioni che non ritornano alcun valore. Eccone alcuni esempi:

```
def hello():
    print "Hello"

def area(width,height):
    return width*height

def print_welcome(name):
    print "Welcome",name

hello()
hello()

print_welcome("Fred")
w = 4
h = 5
print "width =",w,"height =",h,"area =",area(w,h)
```

Con questo output:

```
Hello
Hello
Welcome Fred
width = 4 height = 5 area = 20
```

Questo esempio mostra solamente un po' più in profondità in quanti differenti modi si possono utilizzare le funzioni. Osservate attentamente, potete anche non dare argomenti ad una funzione, così come potete dargliene più d'uno. Notate anche che return è opzionale.

7.2 Variabili nelle funzioni

Naturalmente troverete spesso variabili nel codice duplicato che andrete ad eliminare. Queste variabili vengono trattate da Python in modo speciale. Fino ad ora tutte le variabili che abbiamo visto erano variabili globali. Le funzioni usano variabili particolari chiamate variabili locali. Queste variabili esistono solamente per la durata della funzione. Quando una variabile locale ha lo stesso nome di una variabile globale, la variabile locale nasconde l'altra variabile. Siete confusi? Beh, il prossimo esempio dovrebbe chiarire un po' le cose.

```
a_var = 10
b_var = 15
e_var = 25

def a_func(a_var):
    print "in a_func a_var = ",a_var
    b_var = 100 + a_var
    d_var = 2*a_var
    print "in a_func b_var = ",b_var
    print "in a_func d_var = ",d_var
    print "in a_func e_var = ",e_var
    return b_var + 10

c_var = a_func(b_var)

print "a_var = ",a_var
print "b_var = ",b_var
print "c_var = ",c_var
print "d_var = ",d_var
```

L'output è:

```
in a_func a_var =  15
in a_func b_var =  115
in a_func d_var =  30
in a_func e_var =  25
a_var =  10
b_var =  15
c_var =  125
d_var =
Traceback (innermost last):
  File "separate.py", line 20, in ?
    print "d_var = ",d_var
NameError: d_var
```

In questo esempio le variabili `a_var`, `b_var` e `d_var` sono variabili locali quando sono all'interno della funzione `a_func`. Dopo che l'istruzione `return b_var + 10` viene eseguita, non esisteranno più. La variabile `a_var` è automaticamente una variabile locale, dato che è un nome di parametro. Le variabili `b_var` e `d_var` sono variabili locali, dato che compaiono nella parte sinistra di un assegnamento, all'interno della funzione nelle istruzioni `b_var = 100 + a_var` e `d_var = 2*a_var`.

All'interno della funzione `a_var` è 15 finché la funzione viene chiamata con `a_func(b_var)`. Visto che fino a quel punto `b_var` corrisponde a 15, la chiamata alla funzione è `a_func(15)`, che di conseguenza assegna 15 ad `a_var` all'interno di `a_func`.

Come potete vedere, nel momento in cui la funzione finisce di essere eseguita, le variabili locali `a_var` e `b_var`, che fino a quel momento hanno nascosto le variabili globali con lo stesso nome, sono svanite. Quindi l'istruzione `print "a_var = ",a_var` restituisce il valore `10` piuttosto che `15`, proprio perché la variabile locale non esiste più.

Un'altra cosa da notare è il `NameError` che avviene alla fine. Questo errore è dovuto al fatto che la variabile `d_var` non esiste più. Tutte le varibili locali vengono cancellato quando si esce dalla `a_funzione` in cui sono state definite. Se volete ritornare un valore da una funzione dovete usare `return qualcosa`.

L'ultimo particolare è che il valore di `e_var` rimane invariato all'interno di `a_func`, visto che non è un parametro e non viene ridefinito all'interno della funzione `a_func`. Quando si accede ad una variabile globale dall'interno di una funzione, è come accedervi dall'esterno, rimane invariata.

Le funzioni permettono alle variabili locali di esistere solo ed unicamente all'interno della funzione stessa, e possono nascondere le altre variabili definite all'esterno della funzione.

7.3 Analizzare le funzioni

Ora esamineremo il seguente programma linea per linea:

```
def mult(a,b):
    if b == 0:
        return 0
    rest = mult(a,b - 1)
    value = a + rest
    return value

print "3*2 = ",mult(3,2)
```

Il programma crea una funzione che moltiplica un intero positivo (il che è nettamente più lento rispetto alla funzione di moltiplicazione built-in), dopodiché ne dimostra l'utilizzo.

Domanda: Qual'è la prima cosa che fa il programma?

Risposta: La prima cosa che fa è definire la funzione mult() con queste linee:

```
def mult(a,b):
    if b == 0:
        return 0
    rest = mult(a,b - 1)
    value = a + rest
    return value
```

Queste righe realizzano una funzione che prende due parametri e ritornano un valore. Più tardi questa funzione sarà eseguita.

Domanda: Ed ora cosa succede?

Risposta: Viene eseguita la linea successiva dopo la funzione, `print "3*2 = ",mult(3,2)`.

Domanda: E cosa fa?

Risposta: Stampa `3*2 =` e ritorna il valore di `mult(3,2)`

Domanda: E cosa ritorna `mult(3,2)`?

Risposta: Esaminiamo la funzione `mult` passo per passo per scoprirlo. Alla variabile `a` viene assegnato il valore 3 mentre alla variabile `b` viene assegnato il valore 2.

Domanda: E poi?

Risposta: La linea `if b == 0:` viene eseguita. Visto che `b` ha il valore 2 la condizione è falsa e `return 0` viene saltato.

Domanda: E poi?

Risposta: La linea `rest = mult(a,b - 1)` viene eseguita. Questa linea assegna la variabile locale `rest` al

valore ritornato da 'mult(a,b - 1). Il valore di a è 3 e il valore di b è 2 quindi la chiamata di funzione è mult(3,1).

Domanda: Quindi qual'è il valore di mult(3,1)?

Risposta: Dobbiamo eseguire la funzione mult con i parametri 3 ed 1.

Domanda: Ovvero?

Risposta: Le variabili locali nella *nuova* esecuzione della funzione hanno valori differenti: a ha il valore 3, mentre b ha il valore 1. Dato che sono variabili locali esse non hanno nulla a che vedere con i precedenti valori di a e b.

Domanda: E quindi?

Risposta: Visto che b ha il valore 1 la condizione if è falsa, quindi la linea successiva diventa rest = mult(a,b - 1).

Domanda: Cosa fa questa linea?

Risposta: Questa linea assegnerà il valore di 'mult(3,0) a rest.

Domanda: E qual'è il suo valore?

Risposta: Dobbiamo calcolare la funzione un'altra volta, questa volta con i valori 3 per a e 0 per b.

Domanda: Cosa succede ora?

Risposta: La prima linea della funzione ad essere eseguita è if b == 0:. b ha il valore 0 e quindi la condizione risulta vera. Quindi la prossima linea da eseguire è return 0.

Domanda: E cosa fa la linea return 0?

Risposta: Ritorna il valore 0 all'esterno della funzione.

Domanda: Quindi?

Risposta: Quindi ora sappiamo che il risultato di mult(3,0) è 0. Ora che abbiamo un valore di ritorno possiamo tornare a mult(3,1), assegnando alla variabile rest il valore 0.

Domanda: Che linea viene eseguita ora?

Risposta: La linea value = a + rest, dove a=3 e rest=0, quindi value=3.

Domanda: Cosa succede adesso?

Risposta: La linea return value viene eseguita. Questa linea ritorna il valore 3 ed esce dalla funzione mult(3,1). Dopo che return viene eseguito, torniamo alla funzione mult(3,2).

Domanda: Dove eravamo in mult(3,2)?

Risposta: Avevamo le variabili a=3, b=2 e stavamo esaminando la linea rest = mult(a,b - 1).

Domanda: Cosa succede ora?

Risposta: La variabile rest assume il valore 3. La prossima linea value = a + rest assegna a value il valore di 3+3, ovvero 6.

Domanda: E ora?

Risposta: La prossima linea viene eseguita, ovvero il valore 6 viene ritornato dalla funzione. Ora siamo tornati alla linea print "3*2 = ",mult(3,2) e il risultato finale, 6, può essere stampato sullo schermo.

Domanda: Cos'è successo in definitiva?

Risposta: Praticamente, dando per scontato che ogni numero moltiplicato per zero dia a sua volta 0, e che un numero moltiplicato per un'altro numero è sempre uguale al primo numero + il primo numero per il secondo - 1 (x * y = x + x * (y - 1)), abbiamo moltiplicato due numeri. Ciò che accade è che 3*2 viene inizialmente convertito in 3 + 3*1. Quindi 3*1 è convertito in 3 + 3*0. Sapendo che ogni numero moltiplicato per 0 risulta 0, siamo tornati indietro ed abbiamo calcolato 3 + 3*0. Conoscendo il risultato di 3*1 possiamo calcolare che 3 + 3*1 è

`3 + 3`, che risulta 6.

Questo è come si presenta il tutto:

```
3*2
3 + 3*1
3 + 3 + 3*0
3 + 3 + 0
3 + 3
6
```

Queste due ultime sezioni sono state scritte di recente. Se aveste dei commenti o riscontraste errori, siete pregati di spedirmi un'email. Vorrei sapere se ho passato del tempo a creare semplicemente cose incomprensibili. Se il resto del tutorial vi sembra che abbia senso, ma non questa sezione, è probabile che la colpa sia mia, mi piacerebbe saperlo. Grazie.

7.4 Esempi

'factorial.py'

```
# Definisce una funzione che calcola il fattoriale.

def factorial(n):
    if n <= 1:
        return 1
    return n*factorial(n-1)

print "2! = ",factorial(2)
print "3! = ",factorial(3)
print "4! = ",factorial(4)
print "5! = ",factorial(5)
```

Output:

```
2! =  2
3! =  6
4! =  24
5! =  120
```

'temperature2.py'

```
# Converte la temperatura in Fahrenheit o Celsius.

def print_options():
    print "Options:"
    print " 'p' print options"
    print " 'c' convert from celsius"
    print " 'f' convert from fahrenheit"
    print " 'q' quit the program"

def celsius_to_fahrenheit(c_temp):
    return 9.0/5.0*c_temp+32

def fahrenheit_to_celsius(f_temp):
    return (f_temp - 32.0)*5.0/9.0

choice = "p"
while choice != "q":
    if choice == "c":
        temp = input("Celsius temperature:")
        print "Fahrenheit:",celsius_to_fahrenheit(temp)
    elif choice == "f":
        temp = input("Fahrenheit temperature:")
        print "Celsius:",fahrenheit_to_celsius(temp)
    elif choice != "q":
        print_options()
    choice = raw_input("option:")
```

Semplice esecuzione:

```
> python temperature2.py
Options:
 'p' print options
 'c' convert from celsius
 'f' convert from fahrenheit
 'q' quit the program
option:c
Celsius temperature:30
Fahrenheit: 86.0
option:f
Fahrenheit temperature:60
Celsius: 15.5555555556
option:q
```

'area2.py'

```
# Di Amos Satterlee
print
def hello():
    print 'Hello!'

def area(width,height):
    return width*height

def print_welcome(name):
    print 'Welcome,',name

name = raw_input('Your Name: ')
hello(),
print_welcome(name)
print
print 'To find the area of a rectangle,'
print 'Enter the width and height below.'
print
w = input('Width:  ')
while w <= 0:
    print 'Must be a positive number'
    w = input('Width:  ')
h = input('Height: ')
while h <= 0:
    print 'Must be a positive number'
    h = input('Height: ')
print 'Width =',w,' Height =',h,' so Area =',area(w,h)
```

Semplice esecuzione:

```
Your Name: Josh
Hello!
Welcome, Josh

To find the area of a rectangle,
Enter the width and height below.

Width:  -4
Must be a positive number
Width:  4
Height: 3
Width = 4  Height = 3  so Area = 12
```

7.5 Esercizi

Riscrivete il programma 'area.py' della sezione 3.2, definendo funzioni separate per l'area del quadrato, del rettangolo e del cerchio (3.14 * raggio**2). Il programma deve includere anche un'interfaccia a menu.

CAPITOLO

OTTO

Liste

8.1 Variabili con più di un valore

Avete già visto le variabili ordinarie che immagazzinano un singolo valore. Tuttavia esistono altri tipi di variabili che possono contenere più di un valore. Queste variabili vengono chiamate liste. Ecco un esempio del'utilizzo delle liste:

```
which_one = input("What month (1-12)? ")
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',\
        'August', 'September', 'October', 'November', 'December']
if 1 <= which_one <= 12:
        print "The month is",months[which_one - 1]
```

E questo l'output dell'esempio:

```
What month (1-12)? 3
The month is March
```

In questo esempio la variabile `months` è una lista. `months` è definita dalla linea `months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',\ 'August', 'September', 'October', 'November', 'December']` (osserva nel codice che `\` permette di andare a capo quando la linea di codice è troppo lunga). Le parentesi quadrate `[` e `]` iniziano e finiscono la lista e la virgola (","") separa un elemento della lista dall'altro. La lista qui descritta viene usata nell'istruzione `months[which_one - 1]` e consiste in un insieme di elementi numerati partendo da 0. In altre parole, se cercate l'elemento January dovrete utilizzare `months[0]`. Ovvero, quando assegnate ad una lista un numero, utilizzando la sintassi appena descritta, essa ritornerà l'elemento immagazzinato nella locazione corrispondente.

L'istruzione `if 1 <= which_one <= 12:` sarà vera solo nel caso in cui `which_one` sia un numero tra 1 e 12 inclusi. In altre parole, segue le regole dell'algebra.

Le liste possono essere pensate come una serie di scatole. Ad esempio le scatole create da `demolist = ['life',42, 'the universe', 6,'and',7]` sarebbero all'incirca così:

box number	0	1	2	3	4	5
demolist	'life'	42	'the universe'	6	'and'	7

Ogni scatola ha un numero di riferimento, cosicché l'istruzione `demolist[0]` seleziona l'elemento `'life'` dalla lista, `demolist[1]` l'elemento `42`, `demolist[5]` l'elemento `7`.

8.2 Altre funzioni delle liste

Il prossimo esempio vi mostrerà moltissimi altri utilizzi per le liste (non mi aspetto che proviate ogni esempio, ma dovreste provarli finché non vi sentite a vostro agio nell'utilizzo delle liste). Iniziamo:

```
demolist = ['life',42, 'the universe', 6,'and',7]
print 'demolist = ',demolist
demolist.append('everything')
print "after 'everything' was appended demolist is now:"
print demolist
print 'len(demolist) =', len(demolist)
print 'demolist.index(42) =',demolist.index(42)
print 'demolist[1] =', demolist[1]
# Il prossimo ciclo analizza la lista.
c = 0
while c < len(demolist):
    print 'demolist[',c,']=',demolist[c]
    c = c + 1
del demolist[2]
print "After 'the universe' was removed demolist is now:"
print demolist
if 'life' in demolist:
    print "'life' was found in demolist"
else:
    print "'life' was not found in demolist"
if 'amoeba' in demolist:
    print "'amoeba' was found in demolist"
if 'amoeba' not in demolist:
    print "'amoeba' was not found in demolist"
demolist.sort()
print 'The sorted demolist is ',demolist
```

E questo è l'output:

```
demolist =  ['life', 42, 'the universe', 6, 'and', 7]
after 'everything' was appended demolist is now:
['life', 42, 'the universe', 6, 'and', 7, 'everything']
len(demolist) = 7
demolist.index(42) = 1
demolist[1] = 42
demolist[ 0 ]= life
demolist[ 1 ]= 42
demolist[ 2 ]= the universe
demolist[ 3 ]= 6
demolist[ 4 ]= and
demolist[ 5 ]= 7
demolist[ 6 ]= everything
After 'the universe' was removed demolist is now:
['life', 42, 6, 'and', 7, 'everything']
'life' was found in demolist
'amoeba' was not found in demolist
The sorted demolist is  [6, 7, 42, 'and', 'everything', 'life']
```

Questo esempio utilizza moltissime nuove funzioni. Ad esempio guardate come potete stampare (`print`) un'in-

tera lista per poi aggiungere nuovi elementi con la funzione `append`. `len` serve a contare quanti elementi sono presenti nella lista. Le entrate (`index`) valide di una lista (cioè quelle richiamabili grazie a nome_lista [numero dell'elemento]) vanno da 0 a `len - 1`. La funzione `index` serve a sapere la posizione di un elemento in una lista. Nel caso esistano elementi con lo stesso nome ritornerà la prima posizione dell'elemento. Osservate come l'istruzione `demolist.index(42)` ritorni il valore 1 e l'istruzione `demolist[1]` ritorni il valore 42. La linea `# Il prossimo ciclo analizza la lista` è un semplice commento usato per introdurre le prossime righe di codice:

```
c = 0
while c < len(demolist):
    print 'demolist[',c,']=',demolist[c]
    c = c + 1
```

Crea una variabile `c` inizialmente di valore 0 che viene incrementata finché non raggiunge l'ultimo elemento della lista. Nel frattempo l'istruzione `print` stampa ogni elemento della lista.

L'istruzione `del` può essere usata per rimuovere un elemento dalla lista. Le linee successive utilizzano l'operatore `in` per sapere se un elemento è presente o meno nella lista.

La funzione `sort` ordina la lista ed è utile se avete bisogno di una lista ordinata dall'elemento più piccolo a quello più grande, oppure in ordine alfabetico. Notate che questo comporta elaborare nuovamente la lista.

Riassumendo possiamo compiere le seguenti operazioni:

esempio	spiegazione
`list[2]`	accesso all'elemento 2 dell'indice
`list[2] = 3`	imposta a 3 l'elemento 2 dell'indice
`del list[2]`	cancella l'elemento 2 dall'indice
`len(list)`	restituisci la lunghezza della lista
`value in list`	è vero se `value` è un elemento della lista
`value not in list`	è vero se `value` non è un elemento della lista
`list.sort()`	ordina la lista
`list.index(value)`	restituisce l'indice dove viene trovata la prima occorrenza di `value`
`list.append(value)`	aggiunge l'elemento `value` alla fine della lista

Il prossimo esempio applica queste funzioni per qualcosa di più utile:

```
menu_item = 0
list = []
while menu_item != 9:
        print "--------------------"
        print "1. Print the list"
        print "2. Add a name to the list"
        print "3. Remove a name from the list"
        print "4. Change an item in the list"
        print "9. Quit"
        menu_item = input("Pick an item from the menu: ")
        if menu_item == 1:
                current = 0
                if len(list) > 0:
                        while current < len(list):
                                print current,". ",list[current]
                                current = current + 1
                else:
                        print "List is empty"
        elif menu_item == 2:
                name = raw_input("Type in a name to add: ")
                list.append(name)
        elif menu_item == 3:
                del_name = raw_input("What name would you like to remove: ")
                if del_name in list:
                        item_number = list.index(del_name)
                        del list[item_number]
                        # Il codice precedente rimuove solamente le
                        # prime occorrenze di name. Quello successivo,
                        # da Gerald, rimuove tutto.
                        # while del_name in list:
                        #        item_number = list.index(del_name)
                        #        del list[item_number]
                else:
                        print del_name," was not found"
        elif menu_item == 4:
                old_name = raw_input("What name would you like to change: ")
                if old_name in list:
                        item_number = list.index(old_name)
                        new_name = raw_input("What is the new name: ")
                        list[item_number] = new_name
                else:
                        print old_name," was not found"
print "Goodbye"
```

Ecco una parte dell'output:

```
--------------------
1. Print the list
2. Add a name to the list
3. Remove a name from the list
4. Change an item in the list
9. Quit

Pick an item from the menu: 2
Type in a name to add: Jack

Pick an item from the menu: 2
Type in a name to add: Jill

Pick an item from the menu: 1
0 .  Jack
1 .  Jill

Pick an item from the menu: 3
What name would you like to remove: Jack

Pick an item from the menu: 4
What name would you like to change: Jill
What is the new name: Jill Peters

Pick an item from the menu: 1
0 .  Jill Peters

Pick an item from the menu: 9
Goodbye
```

È un programma abbastanza lungo, diamo un'occhiata al codice sorgente per capirlo meglio. La linea `list = []` crea una variabile `list` senza elementi al suo interno. La prossima linea importante è `while menu_item != 9:`. Questa linea inizia un ciclo che permette di ottenere il menu di questo programma. Le linee seguenti del programma visualizzano un menu per permettere di decidere quale parte del programma eseguire.

La sezione:

```
current = 0
if len(list) > 0:
        while current < len(list):
                print current,". ",list[current]
                current = current + 1
else:
        print "List is empty"
```

Scorre tutta la lista e visualizza ogni suo elemento. `len(list_name)` conta quanti elementi sono presenti nella lista, se `len` ritorna `0` significa che la lista è vuota.

Poche linee più sotto appare l'istruzione `list.append(name)` che aggiunge un elemento alla fine della lista. Saltate giù di un'altro paio di linee e osservate questa sezione di codice:

```
item_number = list.index(del_name)
del list[item_number]
```

Qui la funzione `index` viene utilizzata per trovare l'indice dell'elemento richiesto, per poi utilizzarlo nella sua cancellazione. `del list[item_number]` viene utilizzato appunto per rimuovere l'elemento dalla lista.

La prossima sezione:

```
old_name = raw_input("What name would you like to change: ")
if old_name in list:
        item_number = list.index(old_name)
        new_name = raw_input("What is the new name: ")
        list[item_number] = new_name
else:
        print old_name," was not found"
```

Usa `index` per cercare l'elemento `item_number` ed assegnare il nuovo valore desiderato, `new_name`, alla variabile `old_name`.

Congratulazioni, con le liste sotto il vostro controllo, conoscete abbastanza il linguaggio da poter compiere ogni genere di calcolo che il computer può svolgere (questo tecnicamente sottintende la conoscenza della macchina di Turing). Naturalmente ci sono ancora molte altre funzionalità che vi possono rendere la vita più semplice.

8.3 Esempi

'test.py'

```
## Questo programma verifica la vostra conoscenza.

true = 1
false = 0

# Prima ottieni il questionario, successivamente sarà modificato
# per l'uso del file IO.
def get_questions():
    # Nota come il dato viene magazzinato in una lista di liste.
    return [["What color is the daytime sky on a clear day?","blue"],\
            ["What is the answer to life, the universe and everything?","42"],\
            ["What is a three letter word for mouse trap?","cat"]]
```

```
# Questo verificherà una singola domanda, restituirà vero, se l'utente
# ha scritto la risposta corretta, altrimenti restituirà falso.
def check_question(question_and_answer):
    # Estrai la domanda e la risposta dalla lista.
    question = question_and_answer[0]
    answer = question_and_answer[1]
    # Poni la domanda all'utente.
    given_answer = raw_input(question)
    # Confronta le risposte dell'utente con quelle del test.
    if answer == given_answer:
        print "Correct"
        return true
    else:
        print "Incorrect, correct was:",answer
        return false



# Questa funzione effettuerà tutte le domande.
def run_test(questions):
    if len(questions) == 0:
        print "No questions were given."
        # Esce dalla funzione.
        return
    index = 0
    right = 0
    while index < len(questions):
        # Controlla la domanda.
        if check_question(questions[index]):
            right = right + 1
        # Vai alla prossima domanda.
        index = index + 1
    # Attenzione all'ordine dei conteggi, prima moltiplica, poi dividi.
    print "You got ",right*100/len(questions),"% right out of",len(questions)

# Adesso esegui la funzione sulle domande.
run_test(get_questions())
```

Semplice output:

```
What color is the daytime sky on a clear day?green
Incorrect, correct was: blue
What is the answer to life, the universe and everything?42
Correct
What is a three letter word for mouse trap?cat
Correct
You got  66 % right out of 3
```

8.4 Esercizi

Espandete il programma ‘test.py’ in modo che abbia un menu per selezionare le opzioni del test, visualizzi la lista delle domande ed un’opzione per terminare l’esecuzione. Aggiungete inoltre un’altra domanda: What noise does a truly advanced machine make ? la risposta sarà Ping.

CAPITOLO

NOVE

Cicli For

Qui il nuovo esercizio da scrivere per questo capitolo:

```
onetoten = range(1,11)
for count in onetoten:
        print count
```

E qui l'onnipresente output:

```
1
2
3
4
5
6
7
8
9
10
```

L'output è familiare ma il codice è diverso. La prima linea utilizza la funzione `range`, usa due argomenti, proprio come nell'esempio: `range(inizio,fine)`. `inizio` è il primo numero che viene prodotto, mentre `fine` è maggiore di 1 dell'ultimo numero prodotto. Questo programma avrebbe anche potuto essere scritto più brevemente:

```
for count in range(1,11):
        print count
```

Ecco a voi alcuni esempi per comprendere meglio il funzionamento di `range`:

```
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(-32, -20)
[-32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -21]
>>> range(5,21)
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> range(21,5)
[]
```

Avrete certamente notato la presenza di una nuova struttura di controllo, `for count in onetoten:`, gli esempi infatti usano la struttura `for`. La sintassi della struttura di controllo `for` è simile a `for variabile in lista:`.

La `lista` viene quindi analizzata dal primo elemento sino all'ultimo. Mentre il ciclo `for` compie il suo tragitto, ogni elemento viene inserito in una `variabile`. Questo consente alla `variabile` di essere usata successivamente, in qualsiasi momento, in ogni ciclo `for` attivo. Qui un'altro esempio (non dovete scriverlo) per dimostrarlo:

```
demolist = ['life',42, 'the universe', 6,'and',7,'everything']
for item in demolist:
    print "The Current item is:",
    print item
```

L'output è:

```
The Current item is: life
The Current item is: 42
The Current item is: the universe
The Current item is: 6
The Current item is: and
The Current item is: 7
The Current item is: everything
```

Osservate come il ciclo `for` scorra tutti gli elementi nella lista - provate ad esempio a togliere la virgola alla fine dell'istruzione `print` o a cambiare il testo all'interno della stessa... Quindi, a cosa serve il `for`? Beh, serve a scorrere uno ad uno tutti gli elementi di una lista e a fare qualcosa con ognuno di essi. Questo è un esempio dove sono sommati tutti gli elementi:

```
list = [2,4,6,8]
sum = 0
for num in list:
        sum = sum + num
print "The sum is: ",sum
```

Con questo semplice output:

```
The sum is:  20
```

Altrimenti potreste scrivere un semplice programma per trovare i duplicati in una lista, come in questo programma:

```
list = [4, 5, 7, 8, 9, 1,0,7,10]
list.sort()
prev = list[0]
del list[0]
for item in list:
        if prev == item:
                print "Duplicate of ",prev," Found"
        prev = item
```

Il cui output è ovviamente:

```
Duplicate of  7  Found
```

Va bene, allora come funziona? Scriviamo il programma in modo che ritorni un risultato passaggio per passaggio, qualcosa di molto simile al debugging:

```
l = [4, 5, 7, 8, 9, 1,0,7,10]
print "l = [4, 5, 7, 8, 9, 1,0,7,10]","\tl:",l
l.sort()
print "l.sort()","\tl:",l
prev = l[0]
print "prev = l[0]","\tprev:",prev
del l[0]
print "del l[0]","\tl:",l
for item in l:
        if prev == item:
                print "Duplicate of ",prev," Found"
        print "if prev == item:","\tprev:",prev,"\titem:",item
        prev = item
        print "prev = item","\t\tprev:",prev,"\titem:",item
```

L'output è cambiato:

```
l = [4, 5, 7, 8, 9, 1,0,7,10]   l: [4, 5, 7, 8, 9, 1, 0, 7, 10]
l.sort()        l: [0, 1, 4, 5, 7, 7, 8, 9, 10]
prev = l[0]     prev: 0
del l[0]        l: [1, 4, 5, 7, 7, 8, 9, 10]
if prev == item:        prev: 0         item: 1
prev = item             prev: 1         item: 1
if prev == item:        prev: 1         item: 4
prev = item             prev: 4         item: 4
if prev == item:        prev: 4         item: 5
prev = item             prev: 5         item: 5
if prev == item:        prev: 5         item: 7
prev = item             prev: 7         item: 7
Duplicate of  7  Found
if prev == item:        prev: 7         item: 7
prev = item             prev: 7         item: 7
if prev == item:        prev: 7         item: 8
prev = item             prev: 8         item: 8
if prev == item:        prev: 8         item: 9
prev = item             prev: 9         item: 9
if prev == item:        prev: 9         item: 10
prev = item             prev: 10        item: 10
```

La ragione per cui sono stati inseriti così tanti `print` nel codice è che in questo modo riusciamo a vedere il comportamento del programma linea per linea (quindi se non riuscite ad immaginare il problema di un programma malfunzionante tentate di mettere molti print per individuarlo con più facilità). Il programma inizia con la nostra solita noiosa e vecchia lista, ordinata nella seconda linea a seconda del valore degli elementi (dal minore al maggiore), in modo che ogni duplicato, se ce ne sono, sia vicino all'originale da cui proviene. Dopo questi primi passi viene inizializzata una variabile, prev(ious). Il primo elemento della lista (0) viene cancellato perché altrimenti sarebbe incorrettamente sembrato un duplicato della variabile prev. La linea successiva, come potete vedere, è un ciclo `for`. Ogni singolo elemento della lista viene in questo modo testato e confrontato con il precedente per riconoscere eventuali duplicati. Questo avviene memorizzando il valore del precedente elemento della lista nella variabile prev e confrontando quest'ultima con l'elemento corrente. prev viene quindi di volta in volta (ad ogni ciclo) riassegnato, in modo che l'elemento corrente venga sempre confrontato con quello appena precedente nella lista. Vengono rilevate 7 voci duplicate. Osservate anche come viene utilizzato `\t` per stampare una tabulazione.

Un'altro modo per utilizzare un ciclo `for` è ripetere la stessa azione un determinato numero di volte. Ecco il codice per visualizzare i primi 11 numeri della serie di Fibonacci:

```
a = 1
b = 1
for c in range(1,10):
        print a,
        n = a + b
        a = b
        b = n
```

Con il sorprendente output:

```
1 1 2 3 5 8 13 21 34
```

Tutto quello che potete fare con i cicli `for` potete farlo anche con `while` ma grazie ai cicli `for` è più semplice scorrere tra tutti gli elementi di una lista o compiere un'azione un determinato numero di volte.

CAPITOLO

DIECI

Espressioni booleane

Questo è un piccolo esempio di espressioni booleane (non dovete scriverlo):

```
a = 6
b = 7
c = 42
print 1, a == 6
print 2, a == 7
print 3,a == 6 and b == 7
print 4,a == 7 and b == 7
print 5,not a == 7 and b == 7
print 6,a == 7 or b == 7
print 7,a == 7 or b == 6
print 8,not (a == 7 and b == 6)
print 9,not a == 7 and b == 6
```

Questo è l'output:

```
1 1
2 0
3 1
4 0
5 1
6 1
7 0
8 1
9 0
```

Cosa succede? Il programma consiste in una serie di istruzioni `print`. Ogni istruzione `print` visualizza un numero e un'espressione. Il numero serve a farvi capire quale istruzione viene eseguita. Potete notare che ogni espressione consiste in uno 0 o in un 1. In Python equivalgono a falso (0) e a vero (1).

Le linee:

```
print 1, a == 6
print 2, a == 7
```

Restituiscono infatti vero e falso, esattamente come dovrebbero fare finché la prima affermazione è vera e la seconda è falsa. La terza istruzione `print` è un po' diversa: print `print 3,a == 6 and b == 7`. L'operatore `and` indica che se entrambe le affermazioni prima e dopo l'operatore logico sono vere tutta l'espressione è vera, altrimenti tutta l'espressione è falsa. L'espressione successiva, `print 4,a == 7 and b == 7`, dimostra che se una parte

dell'espressione and è falsa, tutta l'espressione sarà falsa. Il significato di and può essere riassunto come segue:

espressione	risultato
vero and vero	vero
vero and falso	falso
falso and vero	falso
falso and falso	falso

Potete notare che se la prima espressione è falsa Python non esegue un controllo sulla seconda espressione perché sa già che tutta l'espressione è falsa.

La linea successiva, `print 5,not a == 7 and b == 7`, utilizza l'operatore `not` che restituisce semplicemente l'opposto dell'espressione. L'espressione può infatti essere riscritta semplicemente come `print 5,a != 7 and b == 7`. Questa è la tabella:

espressione	risultato
not vero	falso
not falso	vero

Le due linee seguenti, `print 6,a == 7 or b == 7` e `print 7,a == 7 o b == 6`, utilizzano l'operatore logico `or` che ritorna vero se una delle affermazioni (o entrambe) è vera. Questa è la tabella:

espressione	risultato
vero or vero	vero
vero or falso	vero
falso or vero	vero
falso or falso	falso

Anche qui Python non esegue il controllo sulla seconda espressione se riconosce la prima come vera dato che anche se la seconda affermazione risultasse falsa l'intera espressione sarebbe comunque vera.

Le ultime due linee, `print 8,not (a == 7 and b == 6)` e `print 9,not a == 7 and b == 6`, mostrano come le parentesi possano raggruppare espressioni e forzarne l'esecuzione prima di altre al di fuori dalle parentesi. Potete osservare infatti che le parentesi cambiano il valore dell'espressione da falso a vero, visto che obbligano l'operatore `not` a valutare l'intera espressione anziché solamente la porzione `a == 7`.

Ecco un esempio sull'utilizzo delle espressioni booleane:

```
list = ["Life","The Universe","Everything","Jack","Jill","Life","Jill"]

# Crea una copia della lista. Vedi il capitolo ``Ancora sulle liste''
# per una spiegazione del costrutto [:].
copy = list[:]
# Ordina la copia.
copy.sort()
prev = copy[0]
del copy[0]

count = 0

# Esamina la lista per trovare una corrispondenza.
while count < len(copy) and copy[count] != prev:
    prev = copy[count]
    count = count + 1

# Se non viene trovata una corrispondenza allora count non può essere
# < len(copy) in quel momento, quindi usciamo dal ciclo while.
if count < len(copy):
    print "First Match: ",prev
```

Ecco l'output:

```
First Match:  Jill
```

Questo programma continua a scorrere la lista cercando duplicati (`while count < len(copy and copy[count]`). Quando uno dei due `contatori` è più grande dell'ultimo indice di `copy` o viene trovato un duplicato, l'operatore `and` non risulta più vero e si esce dal ciclo. `if` si occupa semplicemente di accertarsi che l'uscita dal ciclo `while` sia dovuta alla presenza di un duplicato.

Nell'esempio viene utilizzato un'altro 'trucco' dell'operatore `and`. Se guardate la tabella di `and` potete osservare che la terza espressione è falsa senza che Python esegua un controllo sul secondo elemento. Se `count >= len(copy)` (in altre parole `count < len(copy)` è falsa) allora `copy[count]` non viene processata. Questo avviene perché Python sa che se la prima accezione è falsa, lo sono entrambe. Questo piccolo trucco è utile se la seconda metà dell'`and` causa un errore. Ho usato la prima espressione (`count < len(copy)`) per eseguire un controllo su `count` e controllare se `count` sia un indice valido (se non mi credete, rimuovete 'Jill' e 'Life', eseguite il programma e vedete se funziona ancora, quindi invertite l'ordine di `count < len(copy) and copy[count] != prev` con `copy[count] != prev and count < len(copy)`).

Le espressioni booleane possono essere usate quando avete bisogno di verificare due o più elementi in una volta.

10.1 Esempi

'password1.py'

```
## Questo programma chiede ad un utente un nome ed una password,
#  poi controlla per accertarsi che gli sia consentito l'accesso.

name = raw_input("What is your name? ")
password = raw_input("What is the password? ")
if name == "Josh" and password == "Friday":
    print "Welcome Josh"
elif name == "Fred" and password == "Rock":
    print "Welcome Fred"
else:
    print "I don't know you."
```

Semplice esecuzione:

```
What is your name? Josh
What is the password? Friday
Welcome Josh

What is your name? Bill
What is the password? Money
I don't know you.
```

10.2 Esercizi

Scrivete un programma che spinga l'utente ad indovinare il vostro nome dandogli solamente 3 possibilità, dopo le quali il programma termina.

CAPITOLO

UNDICI

Dizionari

Questo capitolo tratta i dizionari. I dizionari hanno chiavi e valori. Le chiavi sono usate per trovare i valori. Ecco un esempio di un dizionario in uso:

```
def print_menu():
    print '1. Print Phone Numbers'
    print '2. Add a Phone Number'
    print '3. Remove a Phone Number'
    print '4. Lookup a Phone Number'
    print '5. Quit'
    print
numbers = {}
menu_choice = 0
print_menu()
while menu_choice != 5:
    menu_choice = input("Type in a number (1-5):")
    if menu_choice == 1:
        print "Telephone Numbers:"
        for x in numbers.keys():
            print "Name: ",x," \tNumber: ",numbers[x]
        print
    elif menu_choice == 2:
        print "Add Name and Number"
        name = raw_input("Name:")
        phone = raw_input("Number:")
        numbers[name] = phone
    elif menu_choice == 3:
        print "Remove Name and Number"
        name = raw_input("Name:")
        if numbers.has_key(name):
            del numbers[name]
        else:
            print name," was not found"
    elif menu_choice == 4:
        print "Lookup Number"
        name = raw_input("Name:")
        if numbers.has_key(name):
            print "The number is",numbers[name]
        else:
            print name," was not found"
    elif menu_choice != 5:
        print_menu()
```

E qui l'output:

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Quit

Type in a number (1-5):2
Add Name and Number
Name:Joe
Number:545-4464
Type in a number (1-5):2
Add Name and Number
Name:Jill
Number:979-4654
Type in a number (1-5):2
Add Name and Number
Name:Fred
Number:132-9874
Type in a number (1-5):1
Telephone Numbers:
Name:  Jill     Number:  979-4654
Name:  Joe      Number:  545-4464
Name:  Fred     Number:  132-9874

Type in a number (1-5):4
Lookup Number
Name:Joe
The number is 545-4464
Type in a number (1-5):3
Remove Name and Number
Name:Fred
Type in a number (1-5):1
Telephone Numbers:
Name:  Jill     Number:  979-4654
Name:  Joe      Number:  545-4464

Type in a number (1-5):5
```

Questo programma è simile a quello con la lista di nomi nel capitolo sulle liste. Ecco come funziona il programma. Innanzitutto viene definita la funzione `print_menu` che visualizza sullo schermo un menu più volte usato nel programma. A questo punto compare la linea `numbers = {}` che dichiara `numbers` come un dizionario. Le linee seguenti fanno funzionare il menu:

```
for x in numbers.keys():
    print "Name: ",x," \tNumber: ",numbers[x]
```

Tramite questo ciclo si possono visualizzare le informazioni contenute nel dizionario. La funzione `numbers.keys()` restituisce una lista che viene poi utilizzata dal ciclo `for`. Questa lista non ha un ordine particolare, quindi se la volete in ordine alfabetico la dovrete ordinare. Con la notazione `numbers[x]` potete accedere ai singoli membri del dizionario. Ovviamente in questo caso `x` è una stringa. Successivamente la linea `numbers[name] = phone` aggiunge un nome ed un numero di telefono al dizionario. Se `name` è stato già inserito nel dizionario `phone`, rimpiazza il valore precedente. Le linee successive:

```
if numbers.has_key(name):
    del numbers[name]
```

Controllano se una chiave è già presente nel dizionario, in tal caso la rimuovono. La funzione `numbers.has_key(name)` ritorna vero se `name` è presente in `numbers`, altrimenti ritorna falso. La linea del `del numbers[name]` rimuove la chiave `name` ed il valore ad essa associato. Le linee:

```
if numbers.has_key(name):
    print "The number is",numbers[name]
```

Controllano se nel dizionario è presente una determinata chiave, se la trovano, stampano il numero ad essa associato. Infine, se la scelta non è presente nel menu (quindi non è valida) il programma visualizza nuovamente il menu.

Ricapitolando: i dizionari hanno chiavi e valori. Le chiavi possono essere stringhe o numeri e puntano a valori. I valori puntati possono essere qualsiasi tipo di variabile, anche liste di dizionari (che possono contenere a loro volta dizionari e liste (paura eh? :=))). Questo è un esempio che utilizza una lista in un dizionario:

```
max_points = [25,25,50,25,100]
assignments = ['hw ch 1','hw ch 2','quiz   ','hw ch 3','test']
students = {'#Max':max_points}

def print_menu():
    print "1. Add student"
    print "2. Remove student"
    print "3. Print grades"
    print "4. Record grade"
    print "5. Print Menu"
    print "6. Exit"



def print_all_grades():
        print '\t',
        for i in range(len(assignments)):
            print assignments[i],'\t',
        print
        keys = students.keys()
        keys.sort()
        for x in keys:
            print x,'\t',
            grades = students[x]
            print_grades(grades)



def print_grades(grades):
    for i in range(len(grades)):
        print grades[i],'\t\t',
    print
```

```
print_menu()
menu_choice = 0
while menu_choice != 6:
    print
    menu_choice = input("Menu Choice (1-6):")
    if menu_choice == 1:
        name = raw_input("Student to add:")
        students[name] = [0]*len(max_points)
    elif menu_choice == 2:
        name = raw_input("Student to remove:")
        if students.has_key(name):
            del students[name]
        else:
            print "Student: ",name," not found"
    elif menu_choice == 3:
        print_all_grades()

    elif menu_choice == 4:
        print "Record Grade"
        name = raw_input("Student:")
        if students.has_key(name):
            grades = students[name]
            print "Type in the number of the grade to record"
            print "Type a 0 (zero) to exit"
            for i in range(len(assignments)):
                print i+1,' ',assignments[i],'\t',
            print
            print_grades(grades)
            which = 1234
            while which != -1:
                which = input("Change which Grade:")
                which = which-1
                if 0 <= which < len(grades):
                    grade = input("Grade:")
                    grades[which] = grade
                elif which != -1:
                    print "Invalid Grade Number"
        else:
            print "Student not found"
    elif menu_choice != 6:
        print_menu()
```

E questo è il semplice output:

```
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6):3
        hw ch 1         hw ch 2         quiz            hw ch 3         test
#Max    25              25              50              25              100



Menu Choice (1-6):6
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6):1
Student to add:Bill



Menu Choice (1-6):4
Record Grade
Student:Bill
Type in the number of the grade to record
Type a 0 (zero) to exit
1   hw ch 1     2   hw ch 2     3   quiz        4   hw ch 3     5   test
0               0               0               0               0
Change which Grade:1
Grade:25
Change which Grade:2
Grade:24
Change which Grade:3
Grade:45
Change which Grade:4
Grade:23
Change which Grade:5
Grade:95
Change which Grade:0



Menu Choice (1-6):3
        hw ch 1         hw ch 2         quiz            hw ch 3         test
#Max    25              25              50              25              100
Bill    25              24              45              23              95

Menu Choice (1-6):6
```

La variabile students è un dizionario le cui chiavi sono i nomi degli studenti, i valori sono i voti degli studenti. Le prime

due linee creano semplicemente due liste. La linea successiva `students = {'#Max':max_points}` crea un nuovo dizionario la cui chiave è `#Max` e il valore `[25,25,50,25,100]` (è il valore di `max_points` nel momento in cui il valore viene assegnato; viene usata la chiave `#Max` perché `#` è sempre ordinato sopra i caratteri alfabetici). Quindi viene definita la funzione `print_menu`. Le linee seguenti definiscono `print_all_grades`.

```
def print_all_grades():
        print '\t',
        for i in range(len(assignments)):
            print assignments[i],'\t',
        print
        keys = students.keys()
        keys.sort()
        for x in keys:
            print x,'\t',
            grades = students[x]
            print_grades(grades)
```

Notate come le chiavi vengano innanzitutto estratte dal dizionario `students` con la funzione di `keys` contenuta nella linea: `keys = students.keys()`. `keys` è una lista, quindi possono essere usate tutte le funzioni delle liste. Successivamente vengono ordinate le chiavi nella linea `keys.sort()` e viene utilizzato un ciclo `for` per scorrere tutte le chiavi. I voti sono immagazzinati come una lista all'interno del dizionario, in modo che l'assegnamento `grades = students[x]` assegni a `grades` la lista immagazzinata nella chiave `x`. La funzione `print_grades` visualizza semplicemente una lista ed è definita poche linee sotto.

Le ultime linee del programma implementano le varie opzioni del menu. La linea `students[name] = [0]*len(max_points)` aggiunge uno studente alla chiave corrispondente al nome. La notazione `[0]*len(max_points)` crea una array di zeri della stessa lunghezza della lista `max_points`.

La scelta `remove students` cancella uno studente in modo simile all'esempio precedente. La scelta `record grades` è più complessa. I voti vengono estratti nella linea `grades = students[name]` che ritorna un riferimento ai voti dello studente `name`. Un voto viene quindi registrato nella linea `grades[which] = grade`. Notate che `grades` non viene inserito nel dizionario `students`. Il motivo di questa mancanza è che `grades` è semplicemente un altro nome per `students[name]` quindi cambiare `grades` significa cambiare `student[name]`.

I dizionari rappresentano un modo semplice per collegare chiavi a valori. Possono essere usati facilmente per tenere traccia dei dati che sono attribuiti alle varie chiavi.

CAPITOLO

DODICI

Usare i moduli

Questo è l'esercizio del capitolo (chiamatelo 'cal.py'): [1]:

```
import calendar

year = input("Type in the year number:")
calendar.prcal(year)
```

E qui una parte dell'output:

```
Type in the year number:2001
                                  2001

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7                1  2  3  4                1  2  3  4
 8  9 10 11 12 13 14       5  6  7  8  9 10 11       5  6  7  8  9 10 11
15 16 17 18 19 20 21      12 13 14 15 16 17 18      12 13 14 15 16 17 18
22 23 24 25 26 27 28      19 20 21 22 23 24 25      19 20 21 22 23 24 25
29 30 31                  26 27 28                  26 27 28 29 30 31
```

Ho saltato parte dell'output, credo vi siate fatti un'idea. Cosa fa il programma? La prima linea `import calendar` usa un nuovo comando: `import`. Il comando `import` carica un modulo (in questo caso il modulo `calendar`). Per vedere i comandi disponibili nei moduli standard cercate nella "library reference" (se l'avete scaricata) o sul sito ufficiale `http://www.python.org/doc/current/lib/lib.html`. Il modulo calendar è descritto nel capitolo 5.9. Se leggete la documentazione vi accorgerete di una funzione chiamata `prcal` che visualizza il calendario di un anno. La linea `calendar.prcal(year)` usa la funzione descritta. Riassumendo, per utilizzare un modulo importatelo, quindi usate `nome_modulo.funzione` per utilizzare le funzioni del modulo. Un'altro modo per scrivere il programma:

```
from calendar import prcal

year = input("Type in the year number:")
prcal(year)
```

Questa versione importa una specifica funzione dal modulo. Creiamo un'altro programma utilizzando le funzioni della Python Library (chiamate il file 'clock.py')(premete i tasti 'Ctrl' e 'c' per chiudere il programma):

[1] Import cerca un file di nome 'calendar.py' e lo legge. Se il file chiamante si chiama 'calendar.py' e contiene 'import calendar' cerca di leggere in sé stesso con risultati quantomeno scarsi.

```
from time import time, ctime

prev_time = ""
while(1):
    the_time = ctime(time())
    if(prev_time != the_time):
        print "The time is:",ctime(time())
        prev_time = the_time
```

E questo l'output dell'esempio:

```
The time is: Sun Aug 20 13:40:04 2000
The time is: Sun Aug 20 13:40:05 2000
The time is: Sun Aug 20 13:40:06 2000
The time is: Sun Aug 20 13:40:07 2000
Traceback (innermost last):
  File "clock.py", line 5, in ?
    the_time = ctime(time())
KeyboardInterrupt
```

L'output è ovviamente infinito, così l'ho interrotto (l'output continua fino a quando non premete 'Ctrl+c'). Il programma entra semplicemente in un cliclo infinito ed esegue un controllo per vedere se l'orario è cambiato, nel qual caso visualizza il nuovo orario sul monitor. Fate attenzione a come import è utilizzato nella linea `from time import time, ctime` per richiamare più di una funzione.

La Python Library contiene molte funzioni utili che semplificano la programmazione in Python.

12.1 Esercizio

Riscrivete il programma 'high_low.py 'della sezione 5.2 usando le ultime due cifre della funzione `time` per rendere il numero 'random' (casuale).

CAPITOLO

TREDICI

Ancora sulle liste

Abbiamo già esaminato le liste e come vengono utilizzate. Ora che avete un background più corposo entrerò più in dettaglio. Innanzitutto scopriremo altri modi per estrarre gli elementi dalle liste, quindi vedremo un metodo per copiarle.

Questi sono alcuni esempi dell'utilizzo degli indici per accedere ai singoli elementi di una lista:

```
>>> list = ['zero','one','two','three','four','five']
>>> list[0]
'zero'
>>> list[4]
'four'
>>> list[5]
'five'
```

Questi esempi dovrebbero esservi familiari. Se volete estrarre il primo elemento di una lista dovrete richiamare l'elemento con indice 0, il secondo elemento avrà indice 1 e così via attraverso la lista. Come fare se volete l'ultimo elemento della lista ? Una via potrebbe essere quella di utilizzare la funzione `len` all'interno delle parentesi quadre: `list[len(list)-1]`. `len` ritorna sempre l'ultimo indice più uno. Allo stesso modo l'istruzione per avere il penultimo elemento della lista sarà `list[len(list)-2]`. Esiste una via più semplice: in Python l'ultimo elemento è sempre indicizzato come -1, il penultimo come -2 e così via. Ecco un'altro esempio:

```
>>> list[len(list)-1]
'five'
>>> list[len(list)-2]
'four'
>>> list[-1]
'five'
>>> list[-2]
'four'
>>> list[-6]
'zero'
```

Così ogni elemento in una lista può essere indicizzato in due modi differenti: dall'inizio della lista e dalla fine della lista.

Un'altro metodo molto funzionale per estrarre elementi da una lista sono le sezioni. Ecco un'altro esempio per avere un'idea di come usarle:

```
>>> list = [0,'Fred',2,'S.P.A.M.','Stocking',42,"Jack","Jill"]
>>> list[0]
0
>>> list[7]
'Jill'
>>> list[0:8]
[0, 'Fred', 2, 'S.P.A.M.', 'Stocking', 42, 'Jack', 'Jill']
>>> list[2:4]
[2, 'S.P.A.M.']
>>> list[4:7]
['Stocking', 42, 'Jack']
>>> list[1:5]
['Fred', 2, 'S.P.A.M.', 'Stocking']
```

Le sezioni sono utilizzate per estrarre parti di liste. La sintassi per estrarre sezioni è `list[primo_indice:ultimo_indice]`. La sezione selezionata va da `primo_indice` all'indice prima di `ultimo_indice`. È possibile utilizzare entrambi i metodi di indicizzazione:

```
>>> list[-4:-2]
['Stocking', 42]
>>> list[-4]
'Stocking'
>>> list[-4:6]
['Stocking', 42]
```

Un'altro trucco con le sezioni è non specificare l'indice. Se il primo indice non viene specificato, Python assumerà il primo indice della lista come indice da cui partire. Se il successivo indice non è specificato, si intenderà tutto il resto della lista. Ecco altri esempi:

```
>>> list[:2]
[0, 'Fred']
>>> list[-2:]
['Jack', 'Jill']
>>> list[:3]
[0, 'Fred', 2]
>>> list[:-5]
[0, 'Fred', 2]
```

Ancora un'altro programma d'esempio (copiate ed incollate, se volete, la definizione di poem):

```
poem = ["<B>","Jack","and","Jill","</B>","went","up","the","hill","to","<B>",\
"fetch","a","pail","of","</B>","water.","Jack","fell","<B>","down","and",\
"broke","</B>","his","crown","and","<B>","Jill","came","</B>","tumbling",\
"after"]

def get_bolds(list):
        true = 1
        false = 0
        ## Dichiara is_bold come falso, quindi stiamo guardando una
        ## porzione di testo non in grassetto.
        is_bold = false
        ## start_block rappresenta l'indice delle porzioni di testo in
        ## grassetto.
        start_block = 0
        for index in range(len(list)):
                ## TAG che rappresenta l'inizio del grassetto.
                if list[index] == "<B>":
                        if is_bold:
                                print "Error:  Extra Bold"
                        ##print "Not Bold:",list[start_block:index]
                        is_bold = true
                        start_block = index+1
                ## TAG che rappresenta la fine del grassetto.
                ## Ricordate che l'ultimo numero in una sezione
                ## rappresenta l'indice e dopo l'ultimo indice usato.
                if list[index] == "</B>":
                        if not is_bold:
                                print "Error: Extra Close Bold"
                        print "Bold [",start_block,":",index,"] ",\
                        list[start_block:index]
                        is_bold = false
                        start_block = index+1

get_bolds(poem)
```

E questo è l'output:

```
Bold [ 1 : 4 ]  ['Jack', 'and', 'Jill']
Bold [ 11 : 15 ]  ['fetch', 'a', 'pail', 'of']
Bold [ 20 : 23 ]  ['down', 'and', 'broke']
Bold [ 28 : 30 ]  ['Jill', 'came']
```

La funzione `get_bold` scorre una lista dividendola in parole e chiamate. Le chiamate che cerca sono `<B>` che inizia il testo in grassetto e `<\B>` che lo termina. La funzione `get_bold` ricerca l'inizio e la fine delle chiamate.

La prossima funzione delle liste è la copia. Se volete, potete provare questo semplice esempio:

```
>>> a = [1,2,3]
>>> b = a
>>> print b
[1, 2, 3]
>>> b[1] = 10
>>> print b
[1, 10, 3]
>>> print a
[1, 10, 3]
```

Probabilmente vi sorprenderà in quanto una modifica a `b` modifica anche `a`. Questo grazie all'istruzione `b = a` che restituisce `b` come *riferimento* ad `a`. Significa che `b` non è altro che un'altro nome per riferirsi ad `a`, il risultato è che una modifica a `b` è una modifica ad `a`. Ciò nonostante alcune tipologie di assegnamento non significano la creazione di un doppio nome per una lista:

```
>>> a = [1,2,3]
>>> b = a*2
>>> print a
[1, 2, 3]
>>> print b
[1, 2, 3, 1, 2, 3]
>>> a[1] = 10
>>> print a
[1, 10, 3]
>>> print b
[1, 2, 3, 1, 2, 3]
```

In questo caso `b` non è un riferimento ad `a` in quanto l'istruzione `a*2` crea una nuova lista. `b` quindi si riferisce ad `a*2` e non ad `a`. Tutti gli assegnamenti creano un riferimento. Quando passate una lista come argomento ad una funzione create un riferimento. La maggior parte delle volte non dovrete preoccuparvi di creare un riferimento anziché una copia. Tuttavia, quando dovete modificare una lista senza cambiarne un'altra, assegnata ad un nome differente, dovete assicurarvi di aver creato una copia e non un riferimento.

Esistono diversi modi per copiare una lista. Il modo più semplice è usare le sezioni poiché creano sempre una nuova lista perfino se la sezione comprende tutta la lista:

```
>>> a = [1,2,3]
>>> b = a[:]
>>> b[1] = 10
>>> print a
[1, 2, 3]
>>> print b
[1, 10, 3]
```

La sezione `[:]` crea una nuova copia della lista. In questo modo si può copiare una lista, ma qualsiasi sottolista creata in seguito si riferirà alla rispettiva sottolista della lista originale. Potete ovviare al problema copiando anche le sottoliste usando la funzione `deepcopy` del modulo `copy`:

```
>>> import copy
>>> a = [[1,2,3],[4,5,6]]
>>> b = a[:]
>>> c = copy.deepcopy(a)
>>> b[0][1] = 10
>>> c[1][1] = 12
>>> print a
[[1, 10, 3], [4, 5, 6]]
>>> print b
[[1, 10, 3], [4, 5, 6]]
>>> print c
[[1, 2, 3], [4, 12, 6]]
```

Innanzitutto notate che `a` è una lista di liste. L'istruzione `b[0][1] = 10` cambia sia la lista `b` che la lista `a`, mentre `c` rimane inalterata. Questo accade perché `b` continua ad essere un riferimento ad `a` se vengono utilizzate le sezioni; `c` invece è una copia completa, ottenuta grazie alla funzione `deepcopy`.

Perciò dovrete preoccuparvi dei riferimenti ogni volta che userete la funzione "="? La buona notizia è che dovrete occuparvi dei riferimenti solamente quando utilizzate dizionari e liste. Numeri e stringhe creano dei riferimenti, ma quando vengono modificati creano una copia, quindi non potrete mai modificarli inaspettatamente.

Adesso vi starete probabilmente chiedendo perché vengono usati i riferimenti. La motivazione, sostanzialmente, è la loro velocità. È molto più veloce fare un riferimento ad una lista di migliaia di riferimenti che copiarli tutti. Un'altra ragione è che permettono di avere una funzione che modifichi una lista o un dizionario. Tenete in considerazione tutto questo se vi troverete ad avere strani errori in relazione a dati modificati quando quest'ultimi non avrebbero dovuto subire alcun cambiamento.

CAPITOLO

QUATTORDICI

La rivincita delle stringhe

Ora presenteremo un trucco che può essere usato con le stringhe:

```
def shout(string):
    for character in string:
        print "Gimme a "+character
        print "'"+character+"'"

shout("Lose")

def middle(string):
    print "The middle character is:",string[len(string)/2]

middle("abcdefg")
middle("The Python Programming Language")
middle("Atlanta")
```

E l'output è:

```
Gimme a L
'L'
Gimme a o
'o'
Gimme a s
's'
Gimme a e
'e'
The middle character is: d
The middle character is: r
The middle character is: a
```

Questo programma dimostra che le stringhe sono per alcuni aspetti simili alle liste. La procedura `shout` dimostra che un ciclo `for` può essere usato con le stringhe esattamente nello stesso modo in cui veniva usato con le liste. La procedura `middle` mostra come possa essere utilizzato il `len` anche con le stringhe, così come gli indici e le sezioni. Molte funzionalità delle liste funzionano con le stringhe.

La prossima funzione utilizzerà alcune funzionalità specifiche delle stringhe:

```
def to_upper(string):
    ## Converte la stringa in maiuscolo.
    upper_case = ""
    for character in string:
        if 'a' <= character <= 'z':
            location = ord(character) - ord('a')
            new_ascii = location + ord('A')
            character = chr(new_ascii)
        upper_case = upper_case + character
    return upper_case

print to_upper("This is Text")
```

Con l'output che è:

```
THIS IS TEXT
```

Il motivo per cui questo codice funziona è che il computer rappresenta i caratteri delle stringhe come numeri da 0 a 255. Python ha una funzione chiamata `ord` (abbreviazione di ordinale) che ritorna un carattere come un numero. Esiste anche una funzione corrispondente `chr` che converte un numero in un carattere. Capito questo, il programma dovrebbe essere chiaro. Il primo dettaglio è la linea: `if 'a' <= character <= 'z':` che controlla se la lettera è minuscola. Se è così, allora prosegue alla linea successiva. La linea: `location = ord(character) - ord('a')` converte i numeri corrispondenti alle lettere nel codice ASCII, in numeri da 0 a 36, quindi la linea `new_ascii = location + ord('A')` converte la lettera minuscola in maiuscola.

Adesso, qualche esercizio interattivo da digitare:

```
>>> # Da intero a stringa.
...
>>> 2
2
>>> repr(2)
'2'
>>> -123
-123
>>> repr(-123)
'-123'
>>> # Da stringa a intero.
...
>>> "23"
'23'
>>> int("23")
23
>>> "23"*2
'2323'
>>> int("23")*2
46
>>> # Da decimale a stringa.
...
>>> 1.23
1.23
>>> repr(1.23)
'1.23'
>>> # Da decimale a intero.
...
>>> 1.23
1.23
>>> int(1.23)
1
>>> int(-1.23)
-1
>>> # Da stringa a decimale.
...
>>> float("1.23")
1.23
>>> "1.23"
'1.23'
>>> float("123")
123.0
```

Se non l'avete ancora indovinato, la funzione `repr` converte un intero in stringa e la funzione `int` converte una stringa in intero. La funzione `float` converte una stringa in un numero in virgola mobile. La funzione `repr` ritorna una rappresentazione stampabile di qualcosa. Ecco alcuni esempi di quanto detto:

```
>>> repr(1)
'1'
>>> repr(234.14)
'234.14'
>>> repr([4,42,10])
'[4, 42, 10]'
```

La funzione `int` prova a convertire una stringa (o un numero in virgola mobile) in un intero. Esiste anche una funzione

simile, `float` che converte un numero intero in numero in virgola mobile. Un'altra funzione di Python è `eval` che ritorna il tipo di dato che viene immesso. La funzione `eval` prende una stringa e ritorna i dati che Python rileva. Per esempio:

```
>>> v=eval('123')
>>> print v,type(v)
123 <type 'int'>
>>> v=eval('645.123')
>>> print v,type(v)
645.123 <type 'float'>
>>> v=eval('[1,2,3]')
>>> print v,type(v)
[1, 2, 3] <type 'list'>
```

Se usate la funzione `eval` dovete controllare il valore che ritorna per assicurarvi che sia quello che vi aspettate.

Un'altra funzione utile in string è `split`. Questo è un esempio:

```
>>> import string
>>> string.split("This is a bunch of words")
['This', 'is', 'a', 'bunch', 'of', 'words']
>>> string.split("First batch, second batch, third, fourth",",")
['First batch', ' second batch', ' third', ' fourth']
```

Notate come `split` converte una stringa in una lista di stringhe. La stringa viene divisa in corrispondenza di ogni spazio o di un secondo argomento (in questo caso la virgola).

14.1 Esempi

```
# Questo programma richiede un'eccellente conoscenza dei numeri
# decimali.

def to_string(in_int):
    "Converts an integer to a string"
    out_str = ""
    prefix = ""
    if in_int < 0:
        prefix = "-"
        in_int = -in_int
    while in_int / 10 != 0:
        out_str = chr(ord('0')+in_int % 10) + out_str
        in_int = in_int / 10
    out_str = chr(ord('0')+in_int % 10) + out_str
    return prefix + out_str

def to_int(in_str):
    "Converts a string to an integer"
    out_num = 0
    if in_str[0] == "-":
        multiplier = -1
        in_str = in_str[1:]
    else:
        multiplier = 1
    for x in range(0,len(in_str)):
        out_num = out_num * 10 + ord(in_str[x]) - ord('0')
    return out_num * multiplier

print to_string(2)
print to_string(23445)
print to_string(-23445)
print to_int("14234")
print to_int("12345")
print to_int("-3512")
```

L'output è:

```
2
23445
-23445
14234
12345
-3512
```

CAPITOLO

QUINDICI

File IO

Questo è un semplice esempio di File IO:

```
# Scrive un file.
out_file = open("test.txt","w")
out_file.write("This Text is going to out file\nLook at it and see\n")
out_file.close()

# Legge un file.
in_file = open("test.txt","r")
text = in_file.read()
in_file.close()

print text,
```

L'output ed il contenuto del file test.txt è:

```
This Text is going to out file
Look at it and see
```

Osservate come il programma scriva un file chiamato test.txt nella directory nella quale viene eseguito. Il `\n` nella stringa dice a Python di andare a capo (**n**ewline) nel punto in cui compare.

Una panoramica sul file IO:

1. Aprite un oggetto file con la funzione `open`
2. Leggete o scrivete nell'oggetto file (a seconda di come l'avete aperto)
3. Chiudetelo

Il primo passo è usare un oggetto file grazie alla funzione `open`. La sintassi è `oggetto_file = open (nome_file, modo)` dove `oggetto_file` è la variabile contenente l'oggetto file, `nome_file` la stringa con il nome del file, e `modo` la modalità di apertura del file: `"r"` in lettura (**r**ead), `"w"` in scrittura (**w**rite). Dopodiché potrete chiamare le funzioni dell'oggetto file. Le due funzioni più comuni sono `read` e `write`. La funzione `write` aggiunge una stringa alla fine del file. La funzione `read` legge il file e ne ritorna il contenuto sottoforma di stringa. Se non vi sono argomenti, ritorna l'intero file (come nell'esempio).

Questa è una nuova versione dell'agenda telefonica che abbiamo scritto precedentemente:

```
import string

true = 1
false = 0

def print_numbers(numbers):
    print "Telephone Numbers:"
    for x in numbers.keys():
        print "Name: ",x," \tNumber: ",numbers[x]
    print

def add_number(numbers,name,number):
    numbers[name] = number

def lookup_number(numbers,name):
    if numbers.has_key(name):
        return "The number is "+numbers[name]
    else:
        return name+" was not found"

def remove_number(numbers,name):
    if numbers.has_key(name):
        del numbers[name]
    else:
        print name," was not found"


def load_numbers(numbers,filename):
    in_file = open(filename,"r")
    while true:
        in_line = in_file.readline()
        if in_line == "":
            break
        in_line = in_line[:-1]
        [name,number] = string.split(in_line,",")
        numbers[name] = number
    in_file.close()

def save_numbers(numbers,filename):
    out_file = open(filename,"w")
    for x in numbers.keys():
        out_file.write(x+","+numbers[x]+"\n")
    out_file.close()


def print_menu():
    print '1. Print Phone Numbers'
    print '2. Add a Phone Number'
    print '3. Remove a Phone Number'
    print '4. Lookup a Phone Number'
    print '5. Load numbers'
    print '6. Save numbers'
    print '7. Quit'
    print
```

```
phone_list = {}
menu_choice = 0
print_menu()
while menu_choice != 7:
    menu_choice = input("Type in a number (1-7):")
    if menu_choice == 1:
        print_numbers(phone_list)
    elif menu_choice == 2:
        print "Add Name and Number"
        name = raw_input("Name:")
        phone = raw_input("Number:")
        add_number(phone_list,name,phone)
    elif menu_choice == 3:
        print "Remove Name and Number"
        name = raw_input("Name:")
        remove_number(phone_list,name)
    elif menu_choice == 4:
        print "Lookup Number"
        name = raw_input("Name:")
        print lookup_number(phone_list,name)
    elif menu_choice == 5:
        filename = raw_input("Filename to load:")
        load_numbers(phone_list,filename)
    elif menu_choice == 6:
        filename = raw_input("Filename to save:")
        save_numbers(phone_list,filename)
    elif menu_choice == 7:
        pass
    else:
        print_menu()
print "Goodbye"
```

Ora include anche il salvataggio e la lettura di file. Qui una parte dell'output dell'esecuzione ripetuta due volte:

```
> python tele2.py
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7):2
Add Name and Number
Name:Jill
Number:1234
Type in a number (1-7):2
Add Name and Number
Name:Fred
Number:4321
Type in a number (1-7):1
Telephone Numbers:
Name:  Jill     Number:  1234
Name:  Fred     Number:  4321

Type in a number (1-7):6
Filename to save:numbers.txt
Type in a number (1-7):7
Goodbye


> python tele2.py
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7):5
Filename to load:numbers.txt
Type in a number (1-7):1
Telephone Numbers:
Name:  Jill     Number:  1234
Name:  Fred     Number:  4321

Type in a number (1-7):7
Goodbye
```

Le nuove porzioni del programma sono:

```
def load_numbers(numbers,filename):
    in_file = open(filename,"r")
    while 1:
        in_line = in_file.readline()
        if len(in_line) == 0:
            break
        in_line = in_line[:-1]
        [name,number] = string.split(in_line,",")
        numbers[name] = number
    in_file.close()
```

```
def save_numbers(numbers,filename):
    out_file = open(filename,"w")
    for x in numbers.keys():
        out_file.write(x+","+numbers[x]+"\n")
    out_file.close()
```

Per prima cosa osserviamo la porzione del programma che esegue il salvataggio. Innanzitutto crea un oggetto file con il comando `open(filename,"w")`, dopodiché crea una nuova linea per ognuno dei numeri di telefono con il comando `out_file.write(x+","+numbers[x]+"\n")`. In questo modo scrive una linea contenente il nome, una virgola ed il numero, seguito da un 'a capo' (newline).

La funzione che effettua il caricamento in memoria è un po' più complessa. Inizia creando un oggetto file, quindi usa il ciclo `while 1:` finché non incontra un'istruzione `break`. Successivamente passa alla linea `in_line = in_file.readline()`. La funzione `readline` ritorna una stringa vuota `(len(string) == 0)` quando viene raggiunta la fine del file. L'istruzione `if` esegue un controllo sul `break` e se è il caso, uscirà dal ciclo `while`. Naturalmente, se la funzione `readline` non restituisse il newline all'estremità della linea, non potremmo stabilire se è una stringa vuota, una linea vuota o la fine del file, per questo inseriamo il newline che `readline` restituisce, solamente dop lo elimineremo.

La linea `in_line = in_line[:-1]` elimina l'ultimo carattere della linea (il newline che farebbe andare a capo la linea). Nelle restanti istruzioni viene trattata la stringa, dividendola in base alla virgola, in due parti: `[name,number] = string.split(in_line,",")`, nome e numero. Infine il numero viene inserito nel dizionario `numbers`.

15.1 Esercizi

Modificate il programma dei voti del Capitolo 11 per salvare su di un file la registrazione degli studenti.

CAPITOLO

SEDICI

Occuparsi dell'imperfetto (o come gestire gli errori)

Ora avete un programma perfetto, funziona senza crepe, ad eccezione di un dettaglio: va in crash se l'utente inserisce un input errato. Non abbiate paura, Python ha una speciale struttura di controllo per eliminare questi imprevisti. Sto parlando di `try`... try tenta di fare qualcosa. Segue un esempio di un programma che contiene un problema:

```
print "Type Control C or -1 to exit"
number = 1
while number != -1:
    number = int(raw_input("Enter a number: "))
    print "You entered: ",number
```

Se tentate di inserire qualcosa come `@#&` l'output sarà simile a questo:

```
Traceback (innermost last):
  File "try_less.py", line 4, in ?
    number = int(raw_input("Enter a number: "))
ValueError: invalid literal for int(): @#&
```

Come potete vedere la funzione `int` non funziona (e non deve funzionare) con il numero `@#&`. L'ultima linea mostra qual'è l'errore: Python ha trovato un `ValueError`. Come fare in modo che il nostro programma sappia gestire anche questo valore? Innanzitutto inserendo la parte di codice che restituisce l'errore in un blocco `try` e poi decidendo come Python debba trattare `ValueError`. Il seguente programma mostra come ottenere tale comportamento:

```
print "Type Control C or -1 to exit"
number = 1
while number != -1:
    try:
        number = int(raw_input("Enter a number: "))
        print "You entered: ",number
    except ValueError:
        print "That was not a number."
```

Ora, quando eseguiremo il nuovo programma ed inseriremo `@#&`, il programma risponderà con la frase "That was not a number." per poi continuare ciò che stava facendo prima.

Quando il vostro programma restituisce un errore che sapete come gestire, mettete il codice in un blocco `try` ed il modo per gestire l'errore nel successivo blocco `except`.

16.1 Esercizi

Aggiornate il programma della rubrica telefonica del capitolo precedente in modo che non si interrompa se l'utente non inserisce dati nel menu.

CAPITOLO

DICIASSETTE

Fine

Sto lavorando per aggiungere ulteriori sezioni a questo documento. Per ora vi raccomando il Python Tutorial di Guido Van Rossum, ormai dovreste essere in grado di capire la maggior parte degli argomenti.

Questo tutorial è ancora un lavoro in corso. Gradirei qualsiasi tipo di commento e/o richiesta di argomenti da aggiungere. Le vostre e-mail mi hanno portato ad ampliare e migliorare notevolmente il documento, ed io ho sempre tenuto in molta considerazione ogni vostro intervento :).

Buona programmazione, che possa cambiare la vostra vita e il mondo.

CAPITOLO

DICIOTTO

FAQ

Scherzetto ... le FAQ non sono state tradotte ;-)

Niente paura, semplicemente erano così stringate che, per ora, è stato ritenuto più opportuno impostarle in altro modo. In due parole ... suggerivano di usare una versione di Python 2.1 o superiori e davano indicazioni su dove trovare la versione originale di questo tutorial ed i formati disponibili.

In merito all'ubicazione del tutorial, per quanto riguarda la versione originale:

http://www.honors.montana.edu/~jjc/easytut/ (comprese le soluzioni degli esercizi...)

All'URL indicata è possibile reperire il documento in molteplici formati, html, ps, pdf, ps e pdf Book, il make per crearli e logicamente il sorgente in formato .tex. Come molti di voi avranno capito il sorgente è stato scritto in LaTeX, usando però la classe manual, presa direttamente da quella creata dal gruppo Python per scrivere la documentazione del linguaggio.

Inoltre è disponibile l'elenco delle versioni rilasciate dall'autore:

What and when was the last thing changed? 2000-Dec-16, added error handling chapter.
2000-Dec-22, Removed old install procedure.
2001-Jan-16, Fixed bug in program, Added example and data to lists section.
2001-Apr-5, Spelling, grammar, added another how to break programs, url fix for PDF version.
2001-May-13, Added chapter on debugging.
2001-Nov-11, Added exercises, fixed grammar, spelling, and hopefully improved explanations of some things.
2001-Nov-19, Added password exercise, revised references section.
2002-Feb-23, Moved 3 times password exercise, changed l to list in list examples question. Added a new example to Decisions chapter, added two new exercises.
2002-Mar-14, Changed abs to my_abs since python now defines a abs function.
2002-May-15, Added a faq about creating an executable. Added a comment from about the list example. Fixed typos from Axel Kleiboemer.
2002-Jun-14, Changed a program to use while true instead of while 1 to be more clear.
2002-Jul-5, Rewrote functions chapter. Modified fib program to hopefully be clearer.
2003-Jan-3, Added average examples to the decisions chapter.
2003-Jan-19, Added comment about value of a_var. Fixed mistake in average2.py program.

Per quanto riguarda la presente versione, tradotta in italiano potete trovare qui http://www.python.it/doc/index.html la pagina dalla quale raggiungere tutti i formati del documento.

A proposito del presente documento, sarebbe cosa gradita che chiunque volesse renderlo disponibile per il download sul proprio sito, quantomeno rilasciasse le versioni html, txt, ps, pdf e pdf Book, un file compresso contenente una directory che racchiuda i formati elencati ed il sorgente LaTeX.