Introduzione alla programmazione in Python: Parte II

corso tenuto da:

Francesco Grigoli

organizzato da:

Associazione Next
Studio Mirabilia

con la collaborazione di:

ANFE, Sportello multifunzionale di Bagheria

Sommario

Funzioni

Definire le funzioni

La parola chiave *def* definisce una funzione:

```
>>> def potenze_di_due(n):
...         risultati = []
...         for i in range(n):
...             risultati.append( 2**i )
...         return risultati
...
>>> print potenze_di_due(10)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Argomenti di default e da tastiera

Dando un argomento di default rende il parametro opzionale:

```
def chiedi_ok( risp, prove=4, lamentela='si o no, per piacere!'):
    while True:
        ok = raw_input(risp)
        if ok in ('s', 'si'): return True
        if ok in ('n', 'no'): return False
        prove = prove - 1
        if prove < 0:
            raise IOError, 'errore di inserimento!'
            print lamentela

chiedi_ok('Vuoi realmente uscire?')
chiedi_ok('vuoi sovrascrivere il file?', 2)
chiedi_ok('Caffe?', lamentela='Digitare solo Si o No!')
```

Variabili Input e Output

Le funzioni posso restituire più di un valore:

```
def coord_sferiche( x, y, z ):
# ...
return r, theta, phi
r, theta, phi = coord_sferiche( x, y, z )
```

Documentazione

```
def my_function():
    """Non fa niente, ma serve a documentare la funzione.
    In questo modo è più facile usare il codice.
    """
    pass
    print my_function.__doc__

help(my_function)
```

1. Breve sintesi
2. Riga vuota
3. Descrizione completa

Chiamate per valore (C)

Nel linguaggio di programmazione C i valori di input alle funzioni sono passati per valore:

```
void f(int j) {
j++;
printf("dentro: %i\n", j);
}
main() {
int i;
i = 1;
printf("prima: %i\n", i);
f(i);
printf("dopo: %i\n", i);
}
```

```
prima: 1
dentro: 2
dopo: 1
```

Chiamate per riferimento (C)

Nel linguaggio di programmazione C si posso usare dei puntatori per fare una *chiamate per riferimento*:

```
void f(int *j) {
(*j)++;
printf("dentro: %i\n", *j);
}
main() {
int i;
i = 1;
printf("prima: %i\n", i);
f(&i);
printf("dopo: %i\n", i);
}
```

```
prima: 1
dentro: 2
dopo: 2
```

Chiamate per riferimento (Fortran)

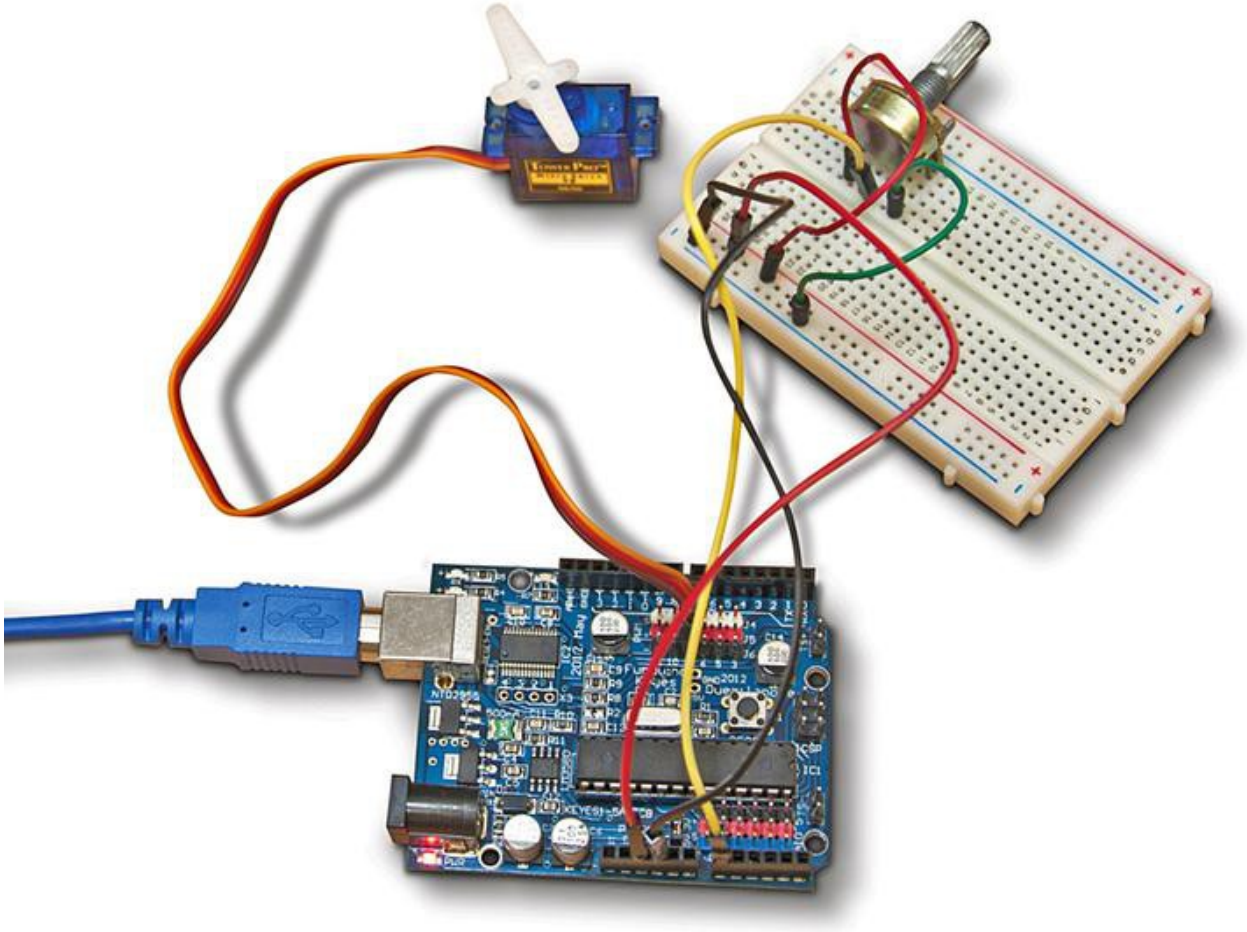
Fortran fa sempre *chiamate per riferimento:*

```
subroutine f(j)
integer :: j
j = j+1
write (*,*) "dentro:", j
end subroutine

program reference
integer :: i = 1
write (*,*) "prima:", i
call f(i)
write (*,*) "dopo: ", i
end program
```

```
prima: 1
dentro: 2
dopo: 2
```

Input e Output



Apertura di un file

Usa *open(filename,mode)* per aprire un file:

```
f = open('/tmp/file', 'w')
```

Alcuni dei modi possibili:

- r: Apre un file di testo per leggere.
- w: Apre un file di testo per scrivere.
- a: Apre un file di testo per aggiungere .
- rb: Apre un file binario per leggere.
- wb: Apre un file binario per scrivere.

Il comando Open riporta un File Object.

Per chiudere il file,usa:

```
f.close()
```

File oggetto predefiniti

- sys.stdin: Standard input
- sys.stdout: Standard output
- sys.stderr: Standard error

L'uso di *stderr* mentre si sta usando *stdout* per qualsiasi output è una buona pratica!

Lettura di un file oggetto

- Lettura di quantità dei dati da un file:

```
s = f.read( size ) # size: numero di bye da leggere
```

- Lettura di un intero file:

```
s = f.read()
```

- Lettura di una linea di un file:

```
s = f.readline()
```

- Mette tutte le linee del file in una lista:

```
list = f.readlines()
```

- Eseguire iterazioni su ogni riga del file:

```
for line in f:
print line
```

Scrittura di un file oggetto

Scrivere una stringa nel file:

f.write(stringa)

Scrivere diverse stringhe nel file:

f.writelines(sequenza)

Esempio: Un cat primitivo

```
import sys
if len(sys.argv) > 1:
    for nomefile in sys.argv[1:]:
        f = open(nomefile, 'r')
        for linea in f:
            sys.stdout.write(linea)
        f.close()
else:
sys.stdout.writelines( sys.stdin )
```

Uso: cat.py [FILE]...

Concatenare FILE(s), o standard input, a standard output.

Il modulo *pickle*

Il salvataggio di strutture di dati arbitrari di Python è banale:

```
>>> import pickle
>>> x = [’a’, [’struttura’, ’dati’, ’annidata’,(1, 2, 3)]]
>>> f = open("my_file", "w")
>>> pickle.dump( x, f )
>>> f.close()
>>> f = open("my_file", "r")
>>> x = pickle.load( f )
>>> f.close()
>>> print x
[’a’, [’struttura’, ’dati’, ’annidata’, (1, 2, 3)]]
```

Moduli

Importazione di un modulo

Approccio normale e sicuro:

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Importazione diretta nel *namespace* locale (da evitare):

```
>>> from math import *
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
```

Attenzione con *from module import **

Consideriamo diversi moduli che forniscono funzioni con lo stesso nome.

Esempio di conflitto tra nomi:

```
>>> from os import *
>>> f = open( "nomefile", "r" ) # fallisce!
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: an integer is required
# open() è stato mappato a os.open() !!!
The builtin open() could still be accessed though:
>>> f = __builtins__.open("lesson5.tex","r")
```

Meno tipizzazione per evitare collisioni

Legandosi a nomi locali:

```
>>> import math
>>> cos = math.cos
>>> pi = math.pi
>>> cos(pi)
```

Importa un modulo con un nome diverso/più corto:

```
>>> import math as m
>>> m.cos(m.pi)
```

Importa solo quello che serve:

```
>>> from math import pi, cos
>>> cos(pi)
```

Scrivere il proprio modulo

test.py:

```
"""Una funzione di generica utilità."""
def  visualizza_messaggio(messaggio ):
"""Visualizza il messaggio dato come input."""
print messaggio
return None

def quadrato( a ):
"""Calcola il quadrato di b=a**2 """
    b=a**2
return b
```

L'utilizzo del tuo modulo

Come con ogni altro modulo:

```
>>> import test
>>> test.visualizza('ciao')
ciao
>>> risultato=test.quadrato(2)
>>> print risultato
```

Introspezione

Si possono guardare i contenuti di ogni modulo:

```
>>> import test
>>> dir(test)
['__builtins__', '__doc__', '__file__', '__name__',
'visualizza', 'quadrato']
# dir without argument looks at local namespace:
>>> dir()
['__builtins__', '__doc__', '__name__', 'test']
```

Sette moduli essenziali

sys	argv, stdin, stdout, stderr, exit()
os	Gestione delle directory dei file e dei processi
math	Funzioni matematiche standard
random	Generatori di numeri random
time	Manipolazione del tempo e della data, sleep()
re	Espressioni regolari
subprocess	Management dei processi

Ulteriori letture

- A. B. Downey, Think Python, O'Reilly
- M. Lutz, Learning Python, O'Reilly
- W. McKinney, Python for data analysis, O'Reilly
- www.numpy.org

Ringraziamenti

Desidero ringraziare Sebastian Heimann per avermi fornito le slides da cui trarre spunto e Lidia Di Blasi per averle tradotte. Ringrazio inoltre Giuseppe Gallo per aver organizzato il corso e il team dello sportello multifunzionale ANFE di Bagheria per aver messo a disposizione i loro locali.

…. Infine, grazie a tutti voi per aver partecipato...