Laboratorio 05

Programmazione - CdS Matematica

Andrea Burattin
26 novembre 2013

Funzioni I

La sintassi generale di una funzione è:

```
def nome_funzione(<parametri>) : # parametri sono
    opzionali
   ''' documentazione della funzione ''' # opzionale
   <corpo della funzione>
```

Esempi di definizione di funzioni:

```
def saluta() :
   print("Ciao!")
```

```
def saluta_qualcuno(chi) :
   print "Ciao %s!" % chi
```

Funzioni II

Invocazione di funzioni:

```
>>> saluta()
Ciao!
>>> saluta_qualcuno("Michele")
Ciao Michele!
```

Valori di default per i parametri:

```
def saluta_qualcuno(chi = "senza nome") :
    print "Ciao %s!" % chi
```

```
>>> saluta_qualcuno()
Ciao senza nome!
```

Funzioni III

Tutte le funzioni *ritornano* un valore. Se non è specificato il comando `return`, viene "aggiunto" un `return None`.

```
def valore_assoluto(x) :
    if x < 0:
        return -x
    return x
```

Esempio:

```
>>> valore_assoluto(1)
1
>>> valore_assoluto(-10)
10
>>> type(valore_assoluto(-10))
<type 'int'>
```

Esercizio

Esercizio

Scrivere una funzione che restituisce un valore intero positivo chiesto in input all'utente. Se l'utente non inserisce un intero positivo, la funzione continua a richiederlo.

Esercizio

Esercizio

Scrivere una funzione che restituisce un valore intero positivo chiesto in input all'utente. Se l'utente non inserisce un intero positivo, la funzione continua a richiederlo.

```
def chiedi_numero(msg):
    return int(raw_input(msg))

def chiedi_positivo():
    n = chiedi_numero("Inserire un intero positivo: ")
    while n < 0:
        n = chiedi_numero("Inserire un intero positivo: ")
    return n
```

Esercizio

Soluzioni equivalenti?

```
def chiedi_positivo():
    n = chiedi_numero("Inserire un intero positivo: ")
    while n < 0:
        n = chiedi_numero("Inserire un intero positivo: ")
    return n
```

```
def chiedi_positivo():
    n = -1
    while n < 0:
        n = chiedi_numero("Inserire un intero positivo: ")
    return n
```

Esercizio

Esercizio

Scrivere un videogioco per giocare a *morra cinese* contro il calcolatore. In particolare: *(i)* il sasso spezza le forbici, *(ii)* le forbici tagliano la carta, *(iii)* la carta avvolge il sasso.

Esercizio

Esercizio

Scrivere un videogioco per giocare a *morra cinese* contro il calcolatore. In particolare: *(i)* il sasso spezza le forbici, *(ii)* le forbici tagliano la carta, *(iii)* la carta avvolge il sasso.

Suggerimenti, funzioni da definire:

- `mossa_utente()`, che ritorna la mossa dell'utente (l'utente può scrivere sia "`caRTa`" che "`CARta`")
- `mossa_calc()`, che genera la mossa del calcolatore
- `vincitore(mossa_utente, mossa_calc)`, che stampa il vincitore

Soluzione I

```
import random

def mossa_utente() :
    mosse_legali = ["carta", "sasso", "forbice"]
    l = raw_input("Inserisci la tua mossa: ").lower()
    while l not in mosse_legali:
        l = raw_input("Inserisci la tua mossa: ").lower()
    return l

def mossa_calc() :
    mosse_legali = ["carta", "sasso", "forbice"]
    return mosse_legali[random.randint(0,2)]
```

Soluzione II

```
def vincitore(mossa_utente, mossa_calc) :
   print "Utente: %s\nCalcolatore: %s" % (mossa_utente,
       mossa_calc)
   if (mossa_utente == mossa_calc) :
      print "Pareggio!"
   elif (mossa_utente=="carta" and mossa_calc=="sasso")
       or (mossa_utente=="sasso" and mossa_calc=="forbice
       ") or (mossa_utente=="forbice" and mossa_calc=="
       carta"):
      print "Hai vinto! :)"
   else:
      print "Hai perso! :("

# Chiamata alla partita vera e propria
vincitore(mossa_utente(), mossa_calc())
```

Esercizio

Esercizio

Estendere l'esercizio precedente, chiedendo, come prima cosa, il numero di round (intero positivo) da effettuare. Il vincitore finale è colui che vince più round.

Suggerimenti:

- Funzione `chiedi_positivo()`
- Funzione `gioca_partita(num_round)`
- Quale struttura dati per le statistiche?

Soluzione

```
# ...
def vincitore(mossa_utente, mossa_calc) :
    if (mossa_utente == mossa_calc) :
        return "pareggio"
    elif # ...
        return "vittoria"
    else:
        return "sconfitta"

def gioca_partita(num_round):
    stato = {"pareggio":0, "vittoria":0, "sconfitta":0}
    for r in range(0, num_round):
        print "ROUND %d" % r
        stato[vincitore(mossa_utente(),mossa_calc())] += 1
    return stato

print gioca_partita(chiedi_positivo())
```

Visibilità delle variabili I

Variabili *locali* $\rightarrow$ definite dentro alle funzioni (*locali* alla funzione)
Variabili *globali* $\rightarrow$ definite fuori da tutte le funzioni

```
>>> var = "glob"
>>> def f() :
...     var2 = "loc"
...     print var + " " +  var2
...

>>> f()
glob loc

>>> var += "-mod"
>>> f()
glob-mod loc
```

Visibilità delle variabili II

Risoluzione nomi:
variabili locali $\rightarrow$ funzioni esterne $\rightarrow$ globali $\rightarrow$ built-in

Esempio:

```
>>> x = 1
>>> def f():
...     x = 2
...     print x
...


>>> f()
2
>>> x
1
```

Esercizio

Costa stampa?

```
>>> def f(x = 1):
...     print x + 1
...

>>> f()
```

Esercizio

Costa stampa?

```
>>> def f(x = 1):
...     print x + 1
...

>>> f()
2

>>> f(2)
```

Esercizio

Costa stampa?

```
>>> def f(x = 1):
...     print x + 1
...

>>> f()
2

>>> f(2)
3

>>> x = 3
>>> f()
```

Esercizio

Costa stampa?

```
>>> def f(x = 1):
...     print x + 1
...

>>> f()
2

>>> f(2)
3

>>> x = 3
>>> f()
2
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
>>> def f(x):
...     return x**2
...
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
>>> def f(x):
...     return x**2
...
>>> def g(x):
...     return x*2
...
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
>>> def f(x):
...     return x**2
...
>>> def g(x):
...     return x*2
...
>>> def applica(l, h):
...     r = []
...     for i in l:
...             r.append((i, h(i)))
...     return r
...
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
>>> def f(x):
...     return x**2
...
>>> def g(x):
...     return x*2
...
>>> def applica(l, h):
...     r = []
...     for i in l:
...             r.append((i, h(i)))
...     return r
...
>>> applica([1, 2, 3], f)
[(1, 1), (2, 4), (3, 9)]
>>> applica([1, 2, 3], g)
[(1, 2), (2, 4), (3, 6)]
```