Laboratorio 05

Programmazione - CdS Matematica

Marco Virgulin
2 dicembre 2014

Script

- Aprire idle dal terminale (ricordarsi la & per poter utilizzare lo stesso terminale con idle in esecuzione):

```
idle &
```

- Aprire l'editor dal menu File → New window
- Salvare il file (es: *lab5.py*).
- Per eseguire lo script utilizzare il comando *da terminale*:

```
python lab5.py
```

Oppure premere F5 all'interno dell'editor.

Funzioni I

La sintassi generale della definizione di una funzione è:

```
def nome_funzione(<parametri>): # i 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 (nello stesso script delle definizioni):

```
saluta()
saluta_qualcuno("Marco")
```

Dall'esecuzione dello script da console si ottiene:

```
Ciao!
Ciao Marco!
```

Funzioni III

Valori di default per i parametri:

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

```
saluta_qualcuno()
saluta_qualcuno("Marco")
```

Output:

```
Ciao Nessuno!
Ciao Marco!
```

Funzioni IV

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

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

```
print valore_assoluto(1)
print valore_assoluto(-10)
type(valore_assoluto(-10))
type(saluta())
```

```
1
10
<type 'int'>
Ciao!
<type 'NoneType'>
```

Esercizio

Esercizio

Scrivere una funzione “chiedi_positivo()” che chiede all’utente di inserire un intero positivo. Se l’utente non inserisce un intero positivo la funzione continua a richiederlo, se invece viene inserito un intero positivo la funzione ritorna l’intero stesso.

Esercizio

Esercizio

Scrivere una funzione “chiedi_positivo()” che chiede all’utente di inserire un intero positivo. Se l’utente non inserisce un intero positivo la funzione continua a richiederlo, se invece viene inserito un intero positivo la funzione ritorna l’intero stesso.

```
def chiedi_positivo():
  n = int(raw_input("Inserire un intero positivo: "))
  while n < 1:
     n = int(raw_input("Inserire un intero positivo: "))
  return n
```

Esercizio

Soluzioni equivalenti?

```
def chiedi_positivo():
  n = int(raw_input("Inserire un intero positivo: "))
  while n < 1:
     n = int(raw_input("Inserire un intero positivo: "))
  return n
```

```
def chiedi_positivo():
  n = 0
  while n < 1:
     n = int(raw_input("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 chiede e ritorna la mossa dell'utente (l'utente dovrebbe poter scrivere sia "`caRTa`" che "`CARta`")
- `mossa_calc()`, che ritorna la mossa del calcolatore
- `vincitore(utente, calc)`, che stampa il vincitore date le due mosse

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

# Per giocare una partita si esegue l'istruzione:
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:

- Utilizzare la funzione `chiedi_positivo()` (definita prima) per chiedere all'utente il numero di round
- Definire la funzione `gioca_partita(num_round)` che gestisce i round (quale struttura dati per le statistiche durante i vari round?)
- Modificare la funzione `vincitore(mossa_utente, mossa_calc)` affinché *ritorni* il risultato del round invece di *stampare* un messaggio

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()

var += "-mod"
f()
```

```
glob loc
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()

print x
```

```
2
1
```

Esercizio

Che cosa stampa? (ragionateci prima di provare)

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

f()

f(2)

x = 3
f()
```

Esercizio

Che cosa stampa? (ragionateci prima di provare)

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

f()

f(2)

x = 3
f()
```

```
2
3
2
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
def f(x):
    return x**2
def g(x):
    return 2*x
def applica(lista, h):
    r = []
    for e in lista:
        r.append((e, h(e)))
    return r
print applica([1, 2, 3, 4], f)
print applica([1, 2, 3, 4], g)
```

Funzioni come parametri

Funzioni possono essere parametri di funzioni

```
def f(x):
    return x**2
def g(x):
    return 2*x
def applica(lista, h):
    r = []
    for e in lista:
        r.append((e, h(e)))
    return r
print applica([1, 2, 3, 4], f)
print applica([1, 2, 3, 4], g)
```

```
[(1, 1), (2, 4), (3, 9), (4, 16)]
[(1, 2), (2, 4), (3, 6), (4, 8)]
```

Esercizio I

Esercizio

Scrivere una funzione `crypt` che possa sia cifrare che decifrare una stringa passando un carattere alla volta alle funzioni (parametro) `crypt_char` e `decrypt_char` che codificano e decodificano singoli caratteri con chiave `k`.

Esempio di invocazione

```
def crypt_char(c, k = 3):
    return " " if c == " " else chr(ord("a") + ((ord(c) -
        ord("a") + k) % 26))
def decrypt_char(c):
    return crypt_char(c, -3)

s = "stringa"
enc = crypt(s, crypt_char)
dec = crypt(enc, decrypt_char)
```

Soluzione I

```
def crypt(s, f):
    enc = ""
    for c in s:
        enc += f(c)
    return enc
```

```
enc = crypt("test cifratura", crypt_char)
print enc
dec = crypt(enc, decrypt_char)
print dec
```

```
'whvw fliudwxud'
'test cifratura'
```

Funzioni ricorsive

Funzioni che dipendono dal risultato della funzione stessa su valori "più semplici".

Funzioni ricorsive

Funzioni che dipendono dal risultato della funzione stessa su valori "più semplici".

Il fattoriale di un numero:

$$n! = \begin{cases} 1 & \text{se } n \leq 1 \rightarrow \text{ detto } \textit{caso base} \\ n(n-1)! & \text{se } n > 1 \end{cases}$$

Funzioni ricorsive

Funzioni che dipendono dal risultato della funzione stessa su valori "più semplici".

Il fattoriale di un numero:

$$n! = \begin{cases} 1 & \text{se } n \leq 1 \rightarrow \text{ detto } \textit{caso base} \\ n(n-1)! & \text{se } n > 1 \end{cases}$$

```
def fatt(n):
    if n <= 1:
        return 1
    else:
        return n * fatt(n-1)
```

Funzioni ricorsive

```
def fatt(n):
  if n <= 1:
    return 1
  else:
    return n*fatt(n-1)
```

```
fatt(4)
```

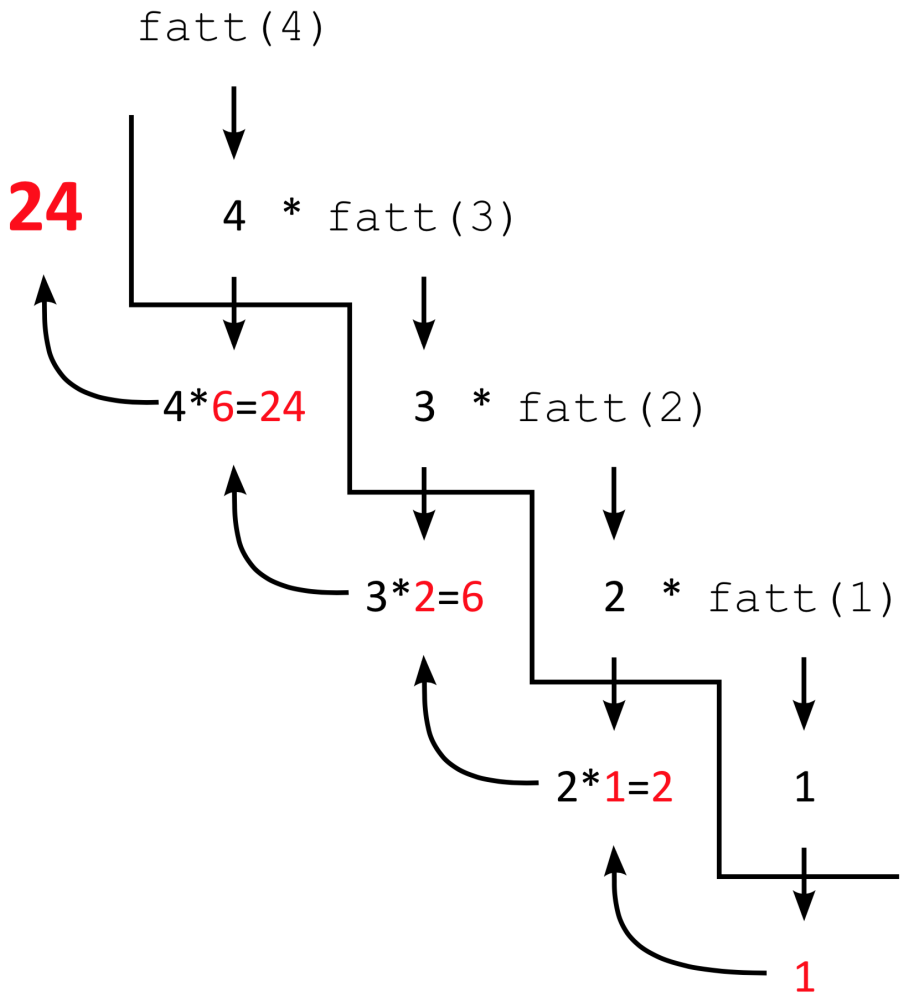
```
24
```

Funzioni ricorsive

```
def fatt(n):
  if n <= 1:
    return 1
  else:
    return n*fatt(n-1)
```

```
fatt(4)
```

```
24
```



Esercizio

Esercizio

Scrivere una funzione ricorsiva per il calcolo dell'n-esimo numero nella serie di Fibonacci. Dove $F_n = F_{n-1} + F_{n-2}$.
Si assume $F_1 = 0$ e $F_2 = 1$.
I primi termini della serie sono quindi: $0, 1, 1, 2, 3, 5, 8, \ldots$

Esercizio

Esercizio

Scrivere una funzione ricorsiva per il calcolo dell'n-esimo numero nella serie di Fibonacci. Dove $F_n = F_{n-1} + F_{n-2}$.
Si assume $F_1 = 0$ e $F_2 = 1$.
I primi termini della serie sono quindi: $0, 1, 1, 2, 3, 5, 8, \ldots$

```
def fib(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Esercizio

Esercizio

Riscrivere la funzione `chiedi_positivo()` senza usare **`while`** (versione *iterativa*) sfruttando invece la *ricorsione*.

Esercizio

Esercizio

Riscrivere la funzione `chiedi_positivo()` senza usare **`while`** (versione *iterativa*) sfruttando invece la *ricorsione*.

```
def chiedi_positivo():
    n = int(raw_input("Inserisci un intero positivo: "))
    if n < 1:
        return chiedi_positivo()
    return n
```

```
print chiedi_positivo()
```

```
Inserisci un intero positivo: -5
Inserisci un intero positivo: 10
10
```

Esercizio

Esercizio

Definire una funzione ricorsiva che, data una stringa, ritorna `True` se questa è palindroma, `False` altrimenti.
Non potete usare alcuna funzione delle stringhe, ad eccezione di `len` e dello *slicing* (non si può usare *striding*).

Esercizio

Esercizio

Definire una funzione ricorsiva che, data una stringa, ritorna `True` se questa è palindroma, `False` altrimenti.
Non potete usare alcuna funzione delle stringhe, ad eccezione di `len` e dello *slicing* (non si può usare *striding*).

```
def palindroma(s):
    if not s:
        return True
    elif s[0] == s[len(s)-1]:
        return palindroma(s[1:len(s)-1])
    else:
        return False
```

```
print palindroma("osso")
```

```
True
```

Esercizio

Estensione esercizio

Estendere l'esercizio precedente al trattamento di **frasi** palindrome (senza considerare spazi, segni di punteggiatura e la distinzione maiuscole/minuscole).

Suggerimento: può essere utile la funzione `str.isalpha()`.
Esempi di frasi palindrome:

- O mordo tua nuora o aro un autodromo
- I topi non avevano nipoti
- Avida diva
- Amo Roma
- Ettore evitava le madame lavative e rotte
- Eran i mesi di seminare
- Occorre pepe per Rocco
- Etna gigante
- Ave, Eva!
- Alla bisogna tango si balla
- I tropici, mamma. Mi ci porti?
- Alle carte t'alleni nella tetra cella

Soluzione

Possibile soluzione

```
def palindroma(s):
    if not s:
        return True
    if s[0].isalpha() == False:        # nuovo caso
        return palindroma(s[1:len(s)])
    if s[-1].isalpha() == False:       # nuovo caso
        return palindroma(s[0:len(s)-1])
    elif s[0].lower() == s[len(s)-1].lower():
        return palindroma(s[1:len(s)-1])
    else:
        return False
```

Direzione della ricorsione

Possibili "direzioni" della ricorsione

In avanti: chiamata ricorsiva è l'ultima istruzione

All'indietro: chiamata ricorsiva è prima istruzione (dopo caso base)

Direzione della ricorsione

Possibili "direzioni" della ricorsione

In avanti: chiamata ricorsiva è l'ultima istruzione

All'indietro: chiamata ricorsiva è prima istruzione (dopo caso base)

Esercizio

Scrivere una funzione ricorsiva `avanti(l)` che stampa gli elementi della lista `l` dal primo all'ultimo. Scrivere una funzione ricorsiva `indietro(l)` che stampa gli elementi della lista `l` dall'ultimo al primo.
`avanti` e `indietro` devono differenziarsi solo per la posizione della chiamata ricorsiva.

Soluzione I

```
def avanti(l):
    if not l: return
    print l[0]
    avanti(l[1:])

def indietro(l):
    if not l: return
    indietro(l[1:])
    print l[0]

l = [1,2]
avanti(l)
indietro(l)
```

```
1
2
2
1
```

Esercizio

Esercizio

Scrivere un generatore di frasi casuali (non di senso compiuto).

- Ciascuna `frase()` genera un `soggetto()` un `verbo()` e, se il verbo è transitivo, un `complemento()`
- Più frasi si possono concatenare con una `congiunzione()`
- Non si possono concatenare più di 4 frasi assieme
- Esecuzione di `frase()` devono produrre risultati differenti

Soluzione

```
import random
```

Soluzione

```
import random

def soggetto():
    l = ["Pippo", "Pluto", "Paperino", "un robot"]
    return l[random.randrange(len(l))]
```

Soluzione

```
import random

def soggetto():
    l = ["Pippo", "Pluto", "Paperino", "un robot"]
    return l[random.randrange(len(l))]

def verbo():
    l = [("corre", False), ("dorme", False),
         ("mangia", True), ("lancia", True)]
    verbo = l[random.randrange(len(l))]
    if (verbo[1]):
        return verbo[0] + " " + complemento()
    return verbo[0]
```

Soluzione

```
import random

def soggetto():
    l = ["Pippo", "Pluto", "Paperino", "un robot"]
    return l[random.randrange(len(l))]

def verbo():
    l = [("corre", False), ("dorme", False),
         ("mangia", True), ("lancia", True)]
    verbo = l[random.randrange(len(l))]
    if (verbo[1]):
        return verbo[0] + " " + complemento()
    return verbo[0]

def complemento():
    l = ["il libro", "la porta", "un panino"]
    return l[random.randrange(len(l))]
```

Soluzione

```
def congiunzione():
    l = ["e", "ma", "mentre"]
    return l[random.randrange(len(l))]
```

Soluzione

```
def congiunzione():
    l = ["e", "ma", "mentre"]
    return l[random.randrange(len(l))]

def frase(n = 2):
    if (n == 0 or random.random() > 0.5):
        return soggetto() + " " + verbo()
    else:
        return frase(n-1) + " " + congiunzione() + " " +
            frase(n-1)
```