Laboratorio 05b

Programmazione - CdS Matematica

Lauriola Ivano

13 dicembre 2016

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: *lab5b.py*).
- Per eseguire lo script utilizzare il comando *da terminale*:

```
python lab5b.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(chi = None):
   print"Ciao %s" % chi if chi != None else "Ciao mondo"
```

Funzioni II

Invocazione di funzioni (nello stesso script delle definizioni):

```
saluta()
saluta("Ivan")
```

Dall'esecuzione dello script da console si ottiene:

```
Ciao mondo
Ciao Ivan
```

Funzioni III

Un semplice esempio su come catturare un valore ritornato da una funzione

```
def incr(x):
    return x+1
```

```
print incr(10)
ris = incr(5)
print ris
```

```
11
6
```

Invocazione di una funzione

Una funzione ed il risultato dell'invocazione della stessa sono due cose diverse

```
def foo():
    return 1

ris = foo()     # ris è un intero
print ris
ris = foo       # ris è una funzione
print ris
print ris()
```

```
1
<function foo at 0x7f3899972848>
1
```

Visibilità delle variabili

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

```
x = 5              # riferimento globale
def foo(y):
    z = 4          # y e z sono locali a foo
    return y+z
print z            # errore: non esiste z
```

Visibilità locale

```
x,y = 5,2
def foo(x):
    x += 3          # modifico x
    y = 4           # dichiaro un altro riferimento
    print 'x: %d, y: %d' % (x,y)
foo(x)              # eseguo foo()
print 'x: %d, y: %d' % (x,y)
```

Visibilità locale

```
x,y = 5,2
def foo(x):
    x += 3          # modifico x
    y = 4           # dichiaro un altro riferimento
    print 'x: %d, y: %d' % (x,y)
foo(x)              # eseguo foo()
print 'x: %d, y: %d' % (x,y)
```

Le modifiche ad `x` ed `y` all'interno di foo non si ripercuotono all'esterno

```
x: 8, y: 4
x: 5, y: 2
```

Visibilità globale

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

```
x = 5              # globale
def foo():
    print x        # x non è dichiarato in foo
    y = x + 1
    print y
foo()
```

Visibilità globale

Risoluzione nomi:
variabili locali → funzioni esterne → globali → built-in

```
x = 5              # globale
def foo():
    print x        # x non è dichiarato in foo
    y = x + 1
    print y
foo()
```

```
5
6
```

Siccome all'interno di `foo` si vuole solo leggere il contenuto di `x`, allora è possibile accedervi senza problemi.

Modifica di una variabile globale

```
x = 5
def foo():
    x += 3       # modifico x globale
```

```
Traceback (most recent call last):
  File "lab5b.py", line 5, in <module>
    foo()
  File "lab5b.py", line 3, in foo
    x += 3
UnboundLocalError: local variable 'x' referenced before
    assignment
```

Siccome `x` è un riferimento globale ma immutabile, allora non è possibile modificarlo direttamente

Modifica di una variabile globale

```
x = 5
def foo():
    global x      # inserisco x nello scope di foo
    x += 3        # modifico x

foo()
print x
```

```
8
```

Con l'utilizzo della keyword `global`, è possibile modificare una variabile globale. Attenzione ai side-effect

Oggetti mutabili

```
x = [1,2,3,4]
def foo(y):
    y[0] = 5
    y = [0] * 4
foo(x)
print x
```

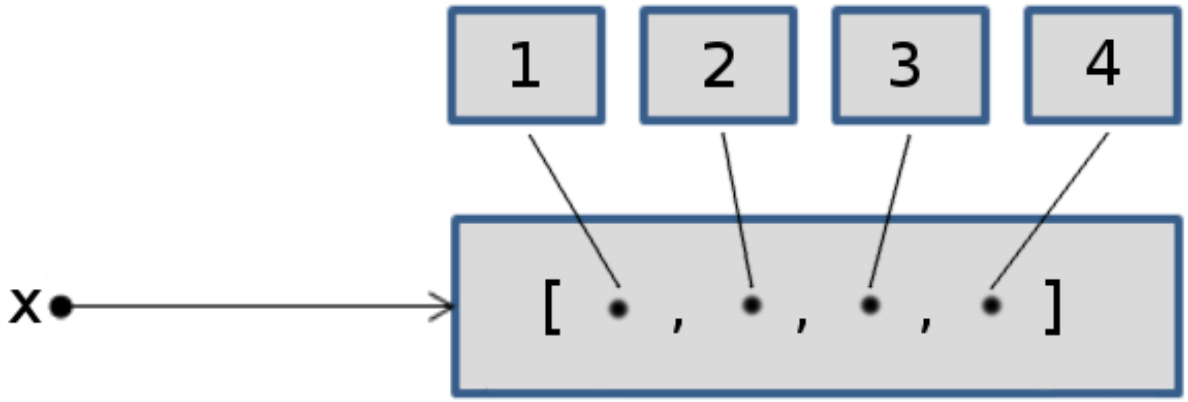
```
[5,2,3,4]
```

Cosa succede?

- Con `y[0]` accedo al primo elemento di `y`. Siccome `y` ed `x` sono due riferimenti che puntano alla stessa lista, ne segue che la modifica ad `y[0]` si ripercuote.
- Nella seconda istruzione invece, ridefinisco `y`, facendolo puntare ad una nuova lista.

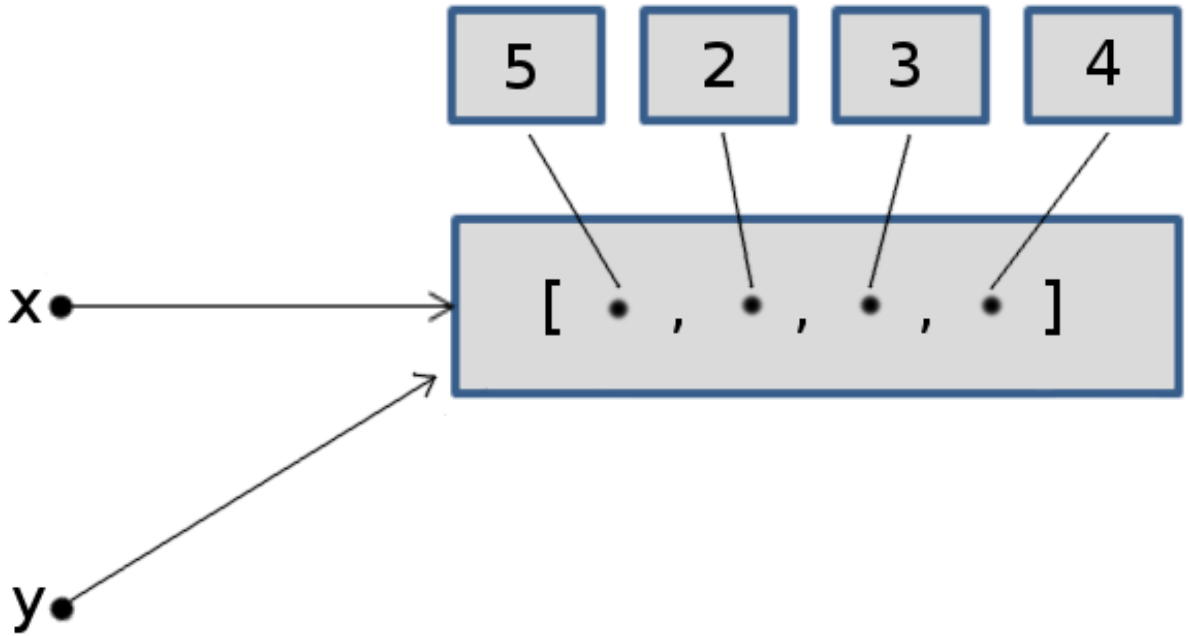
Cosa succede?

```
x = [1,2,3,4]
```



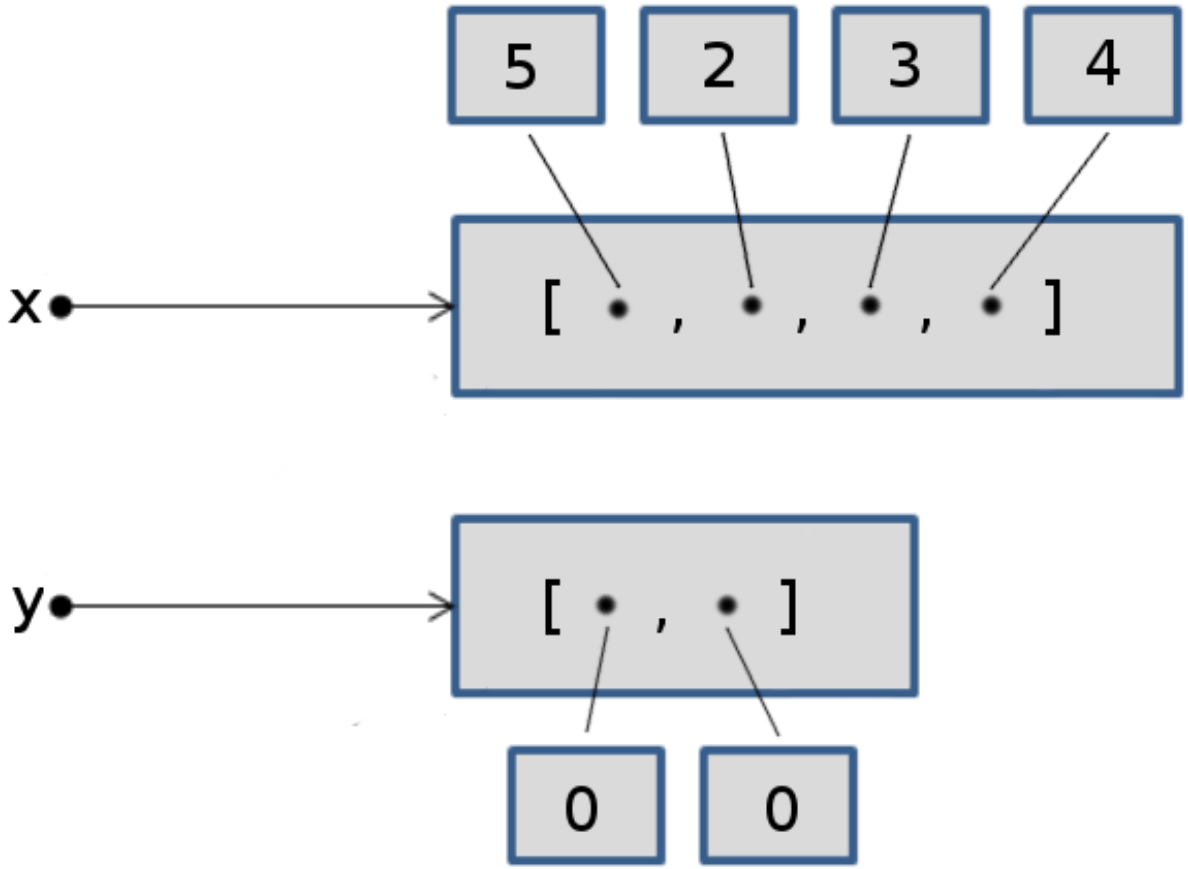
Cosa succede?

```
y[0] = 5
```



Cosa succede?

```
y = [0] * 4
```



Oggetti mutabili - modifica

All'interno di un blocco è possibile modificare un oggetto mutabile globale senza l'utilizzo della keyword `global`

```
x = [1,2,3,4]
def foo1():
    x = [0,0]    # creo un nuovo riferimento
def foo2():
    x[0] = 5     # x globale
foo1()
print x
foo2()
print x
```

```
[1,2,3,4]
[5,2,3,4]
```

Funzioni annidate

Cosa stampa?

```
def foo():
    x = 1

    def inner(x = 0):
        return x+2

    x += inner(x)
    return x

print foo()
```

Funzioni annidate

Cosa stampa?

```
def foo():
    x = 1

    def inner(x = 0):
        return x+2

    x += inner(x)
    return x

print foo()
```

```
4
```

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):
    return [(e, h(e)) for e in lista]

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):
    return [(e, h(e)) for e in lista]

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, k = 3):
    return crypt_char(c, -k)

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'
```

Esempi di ricorsione

Somma definita per ricorsione utilizzando solo operazioni di incremento e decremento

```
# versione ricorsiva terminale
def rsum(x,y):
    if y == 0:          # caso base
        return x
    else:               # caso ricorsivo
        return rsum(x+1, y-1)

# versione ricorsiva non terminale
def rsum(x,y):
    if y == 0:
        return x
    else
        return 1 + rsum(x,y-1)
```

Esempi di ricorsione

O in maniera più compatta...

```
# versione ricorsiva terminale
def rsum(x,y):
    return rsum(x+1,y-1) if y > 0 else x

# versione ricorsiva non terminale
def rsum(x,y):
    return 1rsum(x,y-1) if y > 0 else x
```

Ricorsione VS Iterazione

Ogni funzione ricorsiva può esser trasformata in una funzione iterativa e viceversa. Esempio:

```
# versione ricorsiva
def rsum(x,y):
    if y == 0:
        return x
    else:
        return rsum(x+1, y-1)

# versione iterativa
def isum(x,y):
    while (y>0):
        x +=1
        y -=1
    return x
```

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

Soluzione I

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

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

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

```
1
2
2
1
```

Esercizio

Esercizio

Definire una funzione ricorsiva che, data una lista di interi eventualmente vuota, ritorni `true` se e solo se tale lista contiene almeno uno 0

Esercizio

Esercizio

Definire una funzione ricorsiva che, data una lista di interi eventualmente vuota, ritorni `true` se e solo se tale lista contiene almeno uno 0

```
def rcheck(l):
    if not l:
        return False
    else:
        return l[0] == 0 or rcheck(l[1:])

print rcheck([50,70,2,3,9,4,30])    # false
```

Esercizio

Esercizio

Definire una funzione ricorsiva che, dato in input una lista di interi con almeno un elemento, ne restituisca il maggiore

Esercizio

Esercizio

Definire una funzione ricorsiva che, dato in input una lista di interi con almeno un elemento, ne restituisca il maggiore

```
def rmax(l):
    if len(l) == 1:
        return l[0]
    else:
        s = rmax(l[1:])
        return l[0] if l[0] > s else s

print rmax([50,70,2,3,9,4,30])    # 70
```

Esercizio

Esercizio

Modificare l'esercizio precedente in maniera tale che la funzione restituisca sia il maggiore sia il minore

Esercizio

Esercizio

Modificare l'esercizio precedente in maniera tale che la funzione restituisca sia il maggiore sia il minore

```
def rmax(l):
    if len(l) == 1:
        return {'max':l[0], 'min':l[0]}
    else:
        s = rmax(l[1:])
        ma = l[0] if l[0] > s['max'] else s['max']
        mi = l[0] if l[0] < s['min'] else s['min']
        return {'max':ma, 'min':mi}

print rmax([50,70,2,3,9,4,30])    # {'max':70, 'min':2}
```

Esercizio

Esercizio

Modificare la funzione `crypt` in maniera da renderla ricorsiva

Esercizio

Esercizio

Modificare la funzione `crypt` in maniera da renderla ricorsiva

```
def crypt (s,f):
    return f(s[0]) + crypt(s[1:],f)
```

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.
- Occorre pepe per Rocco.
- I tropici, mamma. Mi ci porti?
- Ettore evitava le madame lavative e rotte.
- Eran i mesi di seminare.
- Etna gigante.
- Alla bisogna tango si balla.
- Alle carte t'alleni nella tetra cella.
- Was it a car or a cat i saw?
- Eva, can I stab bats in a cave?
- Madam in Eden, I'm Adam.

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
```

Esercizio

La funzione di Ackermann è una funzione ricorsiva definita come:

$$
\begin{aligned}
&f(0,0,z) = z \\
&f(0,y+1,z) = f(0,y,z) + 1 \\
&f(1,0,z) = 0 \\
&f(x+2,0,z) = 1 \\
&f(x+1,y+1,z) = f(x,[f(x+1,y,z)],z)
\end{aligned}
$$

con $x, y, z \in \mathbb{N}$.

Esercizio

Per ogni valore di x si ha una funzione ottenuta iterando una funzione più semplice per y volte su z. Le prime funzioni sono:

- $f(0, y, z) = y + z$
- $f(1, y, z) = y \times z$
- $f(2, y, z) = z^y$
- $f(3, y, z) = z^{z^{z^{\cdot}}}$
- ...

Esercizio

Definire in python la funzione di Ackermann

Soluzione

```
def A(x,y,z):
    if (x==0 and y==0):          # f(0,0,z) = z
        return z
    elif (x==0 and y>0):         # f(0,y+1,z) = f(0,y,z)+1
        return A(0,y-1,z) + 1
    elif (x==1 and y==0):        # f(1,0,z) = 0
        return 0
    elif (x>1 and y==0):         # f(x+2,0,z) = 1
        return 1
    elif (x>0 and y>0):          # f(x+1,y+1,z) = ...
        return A(x-1,A(x,y-1,z),z)
```