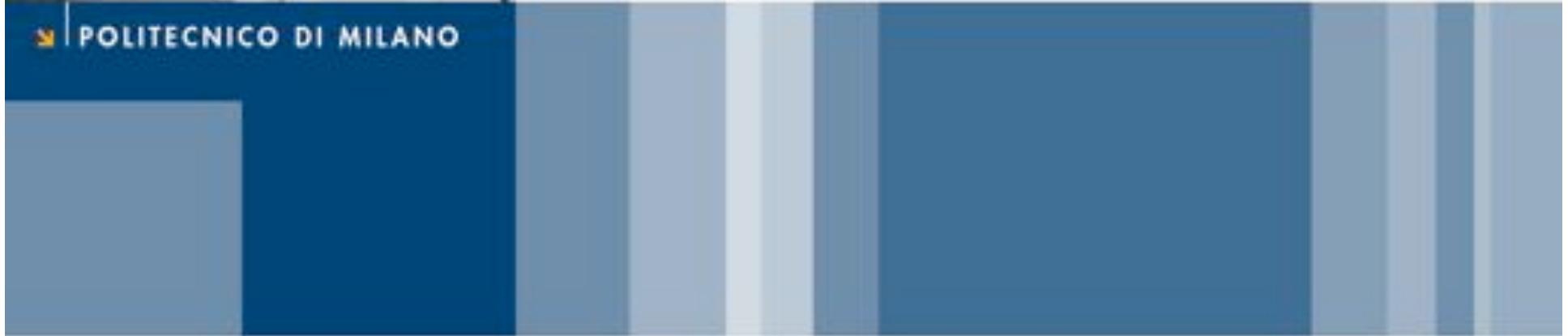




POLITECNICO DI MILANO



2. Ancora Python

Informatica 3

Andrea Mocci, mocci@elet.polimi.it



Una nota sui confronti



- Ogni oggetto in python è confrontabile con un altro
- Sequenze (o tuple):
 - Ordine lessicografico
 - Si comparano gli elementi partendo dalla prima posizione
 - Se sono uguali, si confronta l'elemento successivo
 - Se sono diversi, questo determina il risultato complessivo
 - Se tutti gli elementi sono uguali, le liste sono uguali
 - Se una sequenza è contenuta nell'altra a partire dal primo elemento, allora la sequenza più corta è la più piccola

Confronto tra sequenze



- $[] < [1]$
- $[1, 2, 3] < [1, 2, 4]$
- $(1, 2, 3) != [1, 2, 3]$
- $'ABC' < 'C' < 'Pascal' < 'Python'$
- $(1, 2) < (1, 2, -1)$
- Se l'elemento è una sequenza, vale lo stesso ragionamento
- $[1, 2, [1, 2], 3] > [1, 2, []]$

E se confronto tipi diversi?



- I tipi sono ordinati in base al loro nome!
- Una list è più piccola di una (qualsiasi) string
- Una string è più piccola di una (qualsiasi) tuple
 - Quindi:
 - 1 < 'stringa'
 - [1, 2, 'a'] > [1, 2, 4]
- Eccezione: per i tipi numerici non vale, dipende solo dal valore
 - 0 == 0.0
 - 1 > 0.5

Espressioni Booleane

- I valori numerici 0, 0.0 e qualunque elemento nullo di altri tipi, quali la lista nulla [] e la tupla nulla () sono interpretati come il valore booleano False.
- La semantica esatta delle espressioni booleane è un po' complessa in realtà, ma non occorre saperne i dettagli

Esercizio 1 (testo)



- Determinare l'output del seguente programma python:

```
#  
#  Esercizio 1  
  
def f(x, y):  
    return x + y * 2  
  
def g(x, y = [1]):  
    return x > y  
  
a = [1, 2]  
b = ['a', 'b']  
  
print g(f(a, b))  
print f(b, a)  
print g(f(a, b), f(b, a))
```

Esercizio 1 (1)



All'inizio:

```
f(a, b) == [1, 2] + ['a', 'b'] * 2  
== [1, 2, 'a', 'b', 'a', 'b']
```

$g(f(a, b)) ==$

```
[1, 2, 'a', 'b', 'a', 'b'] > [1] ==
```

True

Esercizio 1 (2)



Quindi:

```
f(b, a) == ['a', 'b'] + [1, 2] * 2
          == ['a', 'b', 'a', 'b', 1, 2]
```

E infine:

```
g(f(a, b), f(b, a)) ==
g([1,2,'a','b','a','b'], ['a','b',1,2,1,2])
          == ([1,2,'a','b','a','b'] > ['a','b',1,2,1,2])
          == False
```

Esercizio 1 (bonus)



- Bonus:

```
#  
#  Esercizio 1  
#  
...  
def h(y):  
    def f(x):  
        return x + y  
    return f  
  
f = h(a)  
print f(b)
```

Attenzione!

- La funzione h restituisce una funzione, la funzione f, dichiarata nello scope di h
- La funzione restituita da h dipende dall'argomento y
- f è tale che, date le seguenti dichiarazioni:

```
f = h(a)
```

```
def i(z):  
    return z + a
```

```
print f(b) == i(b)
```

- La print finale stampa True

Attenzione!



- In pratica, f è la funzione ad un argomento che concatena questo argomento alla variabile a (in questo caso)
- Quindi:

`f(b) == b + a == ['a', 'b', 1, 2]`

Esercizio 2

- Dato il seguente codice Python, calcolare l'output del programma, per ogni istruzione di output giustificare il valore delle variabili stampate

```
#  
# Esercizio 2  
#  
def a(b, c):  
    if b:  
        c += b * 2  
    else:  
        c = c * 2  
    b = []  
    c = [b, 1]  
    a(b, c)  
    print c  
    print a(0, c) + b
```

Esercizio 2

- Attenzione a... parametri formali, parametri attuali, operatori di assegnamento
- Cosa vuol dire $c += b * 2$?
- È equivalente a $c = c + b * 2$?
 - NO! In python, se c è una reference la prima espressione MODIFICA l'oggetto puntato da c concatenandovi $b * 2$ (comportamento *in-place*)
 - Nel secondo caso, c NON è modificata!

Augmented Assignments



- Sono gli operatori di assegnamento `+=`, `-=`, `/=`, `*=` ...
- L'operazione corrispondente, quando possibile, è effettuata in-place, ossia l'oggetto referenziato viene modificato anzichè creare un nuovo oggetto
- Ad esempio:

```
>>> c = [1, 2]
```
- `>>> id(c)`
- 329528
- `>>> c+= [3]`
- `>>> id(c)`
- 329528
- `>>> c = c + [3]`
- `>>> id(c)`
- 358680

Esercizio 2



Quindi:

- `b == []` # È come se fosse True!
- `c == [[[]], 1]` # dopo `c = [b, 1]`
- # dopo `a(b, c)`:
- `c == [[[]], 1] + [[]] * 2`
`== [[[], 1, [], []]]`

Esercizio 2



- `print a(0, c) + b`
- E' corretta ?
- NO: `a(0, c)` non restituisce nulla (in python, ha come tipo a runtime `NoneType`)
- `b` è una sequenza
- Quindi `a(0, c) + b` genera il seguente errore:
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'NoneType' and 'list'

Esercizio 3

- Scrivere una funzione in Python che, data in ingresso una lista di interi contenente numeri interi oppure sottoliste piatte (non contenenti altre liste ma solo numeri interi), restituisca una nuova lista in cui ciascuna sottolista viene sostituita con il valore del prodotto dei suoi elementi.
- Esempio: data la lista [0, [1 2 3], 4, [5 6], 7, [8 9]] la funzione deve restituire [0, 6, 4, 30, 7, 72]

Esercizio 3

- Iniziamo dal sottoproblema più semplice
- Scriviamo la funzione che, data una lista, calcoli il prodotto dei suoi elementi
- L'abbiamo già vista nella scorsa esercitazione...

```
def accumulate(f, lis, zero):  
    res = zero  
    for i in lis :  
        res = f(res, i)  
    return res
```

```
def mul(a, b):  
    return a * b
```

Esercizio 3



- Usando mul, accumulate e 1 possiamo, data una lista, calcolare il prodotto dei suoi elementi
- Torniamo al problema principale:
Come faccio a sapere se un elemento di una lista è un intero o una lista?

```
>>> a = 0
>>> isinstance(a, int) # True
>>> a = [0]
>>> isinstance(a, list) # True
```

Esercizio 3



```
def fun(l):
    res = []
    for x in l:
        if isinstance(x, int):
            res.append(x)
        elif isinstance(x, list):
            res.append(accumulate(mul, x, 1))
    return res
```

Esercizio 4

- Dato il seguente codice python:

```
1: a = 0
2: def f():
3:     print a
4: def h(x):
5:     a = 3
6:     def k():
7:         print a
8:         x()
9:     x = k
10:    x()
11: g = f
12: h(g)
13: g()
14: print a
```

- Simulare l'esecuzione del codice calcolandone l'output.

Esercizio 5

- Si consideri il seguente codice python:

```
def f(n, k):
    if (n < 2):
        return k(1)
    else:
        def L(res):
            return k(n*res)
    return f(n-1,L)

def identity(t):
    return t

print f(5, identity)
```

- Cosa stampa l'ultima print?
- Cosa calcola la funzione f quando il secondo argomento è identity? E per una k qualunque?

Esercizio 6



Dato il seguente codice Python: calcolare l'output del programma, per ogni istruzione di output giustificare il valore delle variabili stampate (output non giustificati non verranno presi in considerazione).

```
def f(a,b):          a=0
    a=a+1
    if (a==1):        b=[1]
        b.append(2)
    elif (a==2 or a==3): a=a+1;
        b()
    else:             f(a,b)
        b=b+1
        a=a+2
def g():             f(a,b)
    print a           print b
```

Esercizio 7



```
def f(a,b):          a=0
    a=a+1
    if (a==1):
        b.append(2)
    elif (a==2 or a==3):
        b()
    else:
        b=b+1
def g():
    print a
    b=[1]
    f(a,b)
    print a,b
    a=a+1;
    b=g
    f(a,b)
    a=a+2
    f(a,b)
    print b
```

Esercizio 8



```
class A(object):  
    def f(self):  
        print "Padre"  
  
class B(A):  
    def g(self):  
        print "Figlio"  
  
def g():  
    print "Funzione G"
```

```
a = A()  
b = B()  
  
a.f() # @@1  
b.f() # @@2  
  
a.f=g  
a.f() # @@3  
b.f() # @@4  
  
a1 = A()  
b1 = B()  
a1.f() # @@5  
b1.f() # @@6
```