Python Intermedio

Giugno 2006

Funzioni

- Se non si sono mai viste si possono considerare il modo piu' semplice per poter creare del codice riutilizzabile.

Classi

- Senza voler entrare in termini tecnici si possono considerare come dei blocchi di codice indipendente.
  In altre parole contengono sia le variabili (attributi) sia il codice per elaborarle (metodi)

Eccezioni

- Gestione degli errori

Funzioni

Codice normale:

- Immaginiamo di dover calcolare i numeri di Fibonacci minori di 10

```
a, b = 0, 1
while b < 10:
      print b
      a, b = b, a+b
```

Programmazione funzionale

- se volessimo adesso ricalcolare i numeri di Fibonacci minori di 20
  dovremmo riscrivere lo stesso codice.

- allora riscriviamo il tutto per poter utilizzare lo stesso codice piu' volte

```
def Fib(n):
      """ stampa la serie di Fibonacci"""     # questa e' una docString
      a, b = 0, 1                             #  (help Fib) ritorna la docString
      while b < n:
            print b
            a, b = b, a+b
      # se omettiamo l'istruzione di return verra' ritornato il valore di default none
      # (si usa definire procedura una funzione che non torna valore !)
```

- mettiamo in uso la funzione

```
Fib(20)
```

Funzioni

Funzionale con ritorno di valori

- se volessimo ritornare una lista basta cambiare il codice in:

```
def Fib(n):
    """ stampa la serie di Fibonacci
    ma ritorna la lista dei valori"""
    lis = []                    # questa e' la lista vuota
    a, b = 0, 1
    while b < n:
        lis.append(b)           # qui aggiungiamo i valori nella lista
        # print b               # commentato
        a, b = b, a+b
    return lis
```

- mettiamo in uso la funzione

```
f = Fib(50)
```

- si possono definire funzioni con un numero variabile di parametri
  e ci sono tre modi di farlo:

Funzioni con piu' parametri in ingresso

- una cosa molto utile e' definire dei parametri di default:

```
def richiesta_conferma(prompt, num=4, ris='Si o No, prego!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('s', 'si', 'S', 'Si?): return True
        if ok in ('n', 'no', 'N', 'No'): return False
        num -= 1
        if num < 0:
            raise IOError('errore Utente')
        print ris
```

- mettiamola in uso

```
richiesta_conferma('Vuoi realmente uscire?')
```

- attenzione si puo' operare sui parametri della funzione:
  o per posizione o per nome!!
- attenzione che i valori di default vengono valutati solo una volta quando la funzione e' creata!!!

Funzioni con numero di parametri arbitrari in ingresso

-questo e' l'ultimo modo ma non per questo il meno usato

```
def fprintf(file, format, *args):          # *args non e' altro che una tupla
    file.write(format % args)
```

- se si vuol passare una lista di default:che si accumula:

```
def f(a, L=[]):
    L.append(a)                            # print f(1) risponde con [1]
    return L                               # print f(2) risponde con [1, 2]
```

- se invece si vuol passare una lista di default:che non venga accumulata:

```
def f(a, L=none):
    if L is None:                          # print f(1) risponde con [1]
        L = []                             # print f(2) risponde con [2]
    L.append(a)
    return L
```

- se si vuol passare un dizionario:

```
def f(**b):
    print(b)                               # la mettiamo in uso  f(pippo=10)
```

Classi

Fondamenti:

- un oggetto ha due parti fondamentali
  Attributi
- contengono i dati dell'oggetto (variabili)
  Metodi
- sono le funzioni con accesso implicito

- una Classe definisce una struttura (“Attributi,Metodi”) e permette di creare oggetti
- un oggetto della stessa classe condivide i Metodi e la stessa Lista di Attributi
  (ma non gli stessi valori)

Definizione:

```
class Rettangolo:
""" Questa e' la solita DocString"""
    def __init__(self, base, altezza):      # questo e' un metodo speciale
    """ questo e' il costruttore"""         # viene chiamato ogni qualvolta si istanzia
        self.base = base                    #  un oggetto, serve ad inizializzare i
        self.altezza = altezza              #  i parametri
    def area(self):                         # questo e' un metodo della classe
    """calcola l'area"
        return self.base * self.altezza
```

Classi

Ereditarieta':

- le Classi costituiscono una gerarchia e possono derivare da altre classi ereditando metodi e attrributi. (La classe madare e' detta anche superclasse)
- si puo' ereditare anche da piu' classi contemporaneamente (eredita' multipla)

Classe base:

```
class Geom:
""" classe base di geometria"""
    def desc(self):
    """ descrivo i miei attributi"""
    print "la mia Area e' %s" % (self.area())
```

Classe derivata:

```
class Rett(Geom):
""" Questa classe e' derivata dalla superclasse Geometria"""
    def __init__(self, base, altezza):      # questo e' un metodo speciale
    """ questo e' il costruttore"""         # viene chiamato ogni qualvolta si istanzia
        self.base = base                    #  un oggetto, serve ad inizializzare i
        self.altezza = altezza              #  i parametri
    def area(self):                         # questo e' un metodo della classe
    """calcola l'area"
        return self.base * self.altezza
```

altra Classe derivata:

```
class Cerchio(Geom):
""" Questa classe e' derivata dalla superclasse Geometria"""
    def __init__(self, raggio):
        self.raggio = raggio
    def area(self):
    """calcola l'area"
        import math
        return math.pi * self.raggio * self.raggio
```

Usiamo le classi derivate:

```
>>  import fGeom                    # richiamo il file dove abbiamo definito le classi
>>  r = fGeom.Rett(2,3)             # istanzio l'oggetto rettangolo
>>  c = fGeom.Cerchio(10)           # istanzio l'oggetto cerchio
>>  r.desc()                        # richiamo il metodo derivato !!!
la mia Area e' 6
>>  c.desc()                        # richiamo il metodo derivato !!!
la mia Area e' 314.15.....
```

Polimorfismo:

- la possibilita' che l'esecuzione di uno stesso metodo abbia effetti diversi in base all'oggetto che deve trattare.
- abbiamo visto che l'oggetto rettangolo richiama il calcolo dell'area nello stesso modo dell'oggetto cerchio!!!
  pero' ognuno opera in modo diverso

Override (sovrapposizione):

- una Classe derivata (sottoclasse) puo' implementare il proprio metodo ed avere accesso anche al metodo originale

Metodi Virtuali:

- quando si dichiara un metodo senza fornire l'implementazione

Metodi e Variabili Privatei:

- in python si definiscono privati tutti gli oggetti che iniziano con 2 undescore e finiscono senza (es: __mioParametro)
  se finiscono con 2 underscore sono considerati speciali (es: __init__)
  ma sono raggiungibili anche al di fuori della classe!!!

Eccezioni

Eccezioni :

- tutte le condizioni in cui non ha senso proseguire percio' si deve interrompere e informare l'utente.
- esempi:
  si tenta di leggere un file inesistente
  si cerca di divdere un numero per zero

Costrutti:

try:
- si tenta di eseguire il blocco di codice

except:
- il blocco di codice che gestisce l'eccezzione

raise:
- si solleva una eccezione o si riattivare quella intercettata

Esempio

```
import sys

try:
        f = open(“mioFile.txt”)                                  # apro il file
        s = readline()                                           # leggo una linea
        i = int(s.strip())                                       # converto in intero il dato
except IOError(errno, strerror):                                 # gestione di una particolare
        print “Errore I/O (%s): %s” %(errno, strerror)           #  eccezione con parametri
except ValueError:                                               # gestione di una particolare
        print “Non si puo' convertire il dato in un intero       #  eccezione senza parametri
except:                                                          # gestione generica
        print “Errore inatteso:” sys.exc_info()[0]
        raise                                                    # passo la gestione al sistema
```