Suggerimenti per lo sviluppo Python

Informatica@SEFA 2017/2018 - Laboratorio 10

Massimo Lauria <massimo.lauria@uniroma1.it>
http://massimolauria.net/courses/infosefa2017/

Lunedì, 11 Dicembre 2017

Testare la correttezza del risultato

Testare che una funzione calcoli un determinato valore.

```
from infosefa import testRisultato

def cuboide(x,y,z):
    return x*y*z

testRisultato("A",11,cuboide,1,1,1)
testRisultato("B",0,cuboide,1,1,0)
testRisultato("C",0,cuboide,1,0,1)
testRisultato("D",3,cuboide,0,1,1)
testRisultato("E",8,cuboide,2,2,2)
```

```
FAIL A. Ottenuto 1 invece di 11
GOOD B. Ottenuto 0
GOOD C. Ottenuto 0
FAIL D. Ottenuto 0 invece di 3
GOOD E. Ottenuto 8
```

testRisultato

```
testRisultato(nome,r_atteso,func, arg1, arg2, ..., argN):
```

- `nome` del test
- `r_atteso` risultato che ci si aspetta dal calcolo
- `func` funzione che effettua il calcolo
- Viene eseguito `func(arg1,arg2,...,argN)`
- Verifica che il risultato sia `r_atteso`

Testare la gestione degli errori

Vediamo una funzione genera errori di vario tipo.

```
from infosefa import testErrore,testRisultato                    1
                                                                 2
def produci_errori(err):                                         3
  if err=='divisione':                                           4
    1/0                                                          5
  elif err=='tipo':                                              6
    3+'s'                                                        7
  elif err=='indice':                                            8
    x=[2]                                                        9
    x[100]                                                       10
  else:                                                          11
    return "stringa"                                             12
```

Testare la gestione degli errori (II)

```
                                                                    1
testErrore("A",ZeroDivisionError,produci_errori,'divisione')        2
testErrore("B",TypeError,produci_errori,'stringa')                  3
testErrore("C",IndexError,produci_errori,'indice')                  4
testErrore("D",TypeError,produci_errori,'stringa')                  5
testErrore("E",TypeError,produci_errori,'indice')                   6
```

```
GOOD A. Errore atteso <class 'ZeroDivisionError'>
True
FAIL B. Manca l'errore atteso <class 'TypeError'>
False
GOOD C. Errore atteso <class 'IndexError'>
True
FAIL D. Manca l'errore atteso <class 'TypeError'>
False
FAIL E. Errore list index out of range invece di <class 'TypeError'>
False
```

testErrore

```
testErrore(nome,e_atteso,func, arg1, arg2, ..., argN):
```

- `nome` del test
- `e_atteso` risultato che ci si aspetta dal calcolo
- `func` funzione che effettua il calcolo
- Viene eseguito `func(arg1,arg2,...,argN)`
- Verifica che la funzione sollevi `e_atteso`

Un metodo per scrivere codice

1. Avere degli esempi di calcolo corretto
2. Scrivere dei ’casi di test’
3. Scrivere un tentativo di soluzione
4. Se i test non passano
   - trovare i bug
   - correggere
5. Se i test passano
   - scrivere altri test

Come si trovano i bug?

Il **debugger** è un software che manipola il vostro programma e permette di trovare bug

- in un ambiente controllato
- esecuzione passo passo
- osservando l'evoluzione delle variabili

Molti ambienti di sviluppo Python hanno anche un **debugger** integrato. Ad esempio Thonny.

Esempio: `minmods.py`

Nel file minmods.py potete trovare lo schema di una funzione che deve essere scritta, con i test già pronti.

Proveremo a completare la funzione insieme e a scovare eventuali errori usando un debugger.