Esercitazione di Programmazione

Michele Beretta

michele.beretta@unibg.it

Gruppo A	Gruppo B/C	Argomenti
12/03	13/03	Variabili, tipi base, operazioni e funzioni standard di I/O
17/03	20/03	Uso di funzioni e metodi, uso stringhe, conversioni di tipo ed espressioni booleane
16/04	17/04	Costrutto condizionale: `if-elif-else`, strutture iterative `while` e `for`
23/04	24/04	Esercizi su `if-elif-else`, uso dei cicli `while` e `for` (`break`, `continue` ed `else`)
07/05	08/05	Esercizi sui cicli `while` e `for`
14/05	15/05	Liste (introduzione ed esercizi)
21/05	22/05	Moduli e funzioni (introduzione ed esercizi)
27/05	29/05	File, complementi sulle stringhe (introduzione ed esercizi)
04/06	05/06	Dizionari ed esercizi di ripasso

Struttura generale di un modulo Python

Un modulo è un file Python (`.py`).

L'interprete inizia l'esecuzione dalla *prima istruzione*. Con `def` si definisce una funzione, per eseguirla bisogna chiamarla.

```
def func(arg0, arg1):
    # function code

def main():
    # main code
    result = func(arg0, arg1)
    # other code

if __name__ == "__main__":
    main()
```

Struttura generale di un modulo Python

`if __name__ == "__main__"` consente di importare le funzioni del modulo senza eseguire codice, e di eseguire codice quando eseguiamo il file.

Vedere `file1.py`, `file2.py`.

```
# Ignora il codice dentro if __name__ == "__main__"
from module import func
```

Codice di `module.py`

```
def func():
    # ...

if __name__ == "__main__":
    print(123)
```

Visibilità (*Scope*)

File `scope.py`.

```
a = 1       # Variabile globale
def foo():
    a = 2   # Crea una variabile locale a, non modifica quella globale

def bar():
    # Nel corpo della funzione `a` fa riferimento alla variabile globale
    global a
    a = 3   # Viene modificata la variabile globale

if __name__ == "__main__":
    print("a = ", a)
    foo()
    print("a = ", a)
    bar()
    print("a = ", a)
```

Esercizio 1

Si scriva un programma che chiede in input all'utente una sequenza di stringhe da memorizzare in una lista. La sequenza acquisita da tastiera si intende terminata quando si acquisisce la stringa vuota (`""`). Dopodiché si scriva la funzione `stampe_ripetute` che, data questa lista, stampi una volta la prima stringa, due volte la seconda ecc.

Esercizio 2

Si scriva un programma che chieda all'utente di inserire un intero positivo, e una funzione `divisori` che restituisca una lista contenente i divisori di quell'intero.

Esercizio 3

Si scriva un programma contenente la funzione `perfetto` che verifica se un numero positivo inserito dall'utente è un numero perfetto.

Importare la funzione dell'esercizio precedente per risolvere questo esercizio.

NB: un numero è perfetto quando è uguale alla somma dei suoi divisori (escluso se stesso), ad esempio 6, 28, 496, …

Esercizio 4 (part 1)

Si scriva un programma Python che data in input una lista di numeri interi inseriti dall'utente e compresi tra 0 e 20 (qualsiasi altro numero per terminare), stampi a video l'istogramma corrispondente.

Ad es. data in input `[1, 7, 5]`

Output atteso:

```
*
*******
*****
```

utilizzando il carattere `*` per ogni unità.

Si definisca poi una funzione che dato l'istogramma ne costruisca il suo complementare. Si stampi infine a video anche quest'ultimo.

Esercizio 4 (part 2)

Il complementare dell'istogramma è un altro istogramma i cui valori sono quelli mancanti per raggiungere il massimo di 20.

Ad es. il complementare di quello mostrato come esempio è: `[19, 13, 15]`.

Output atteso:

```
********************
************
***************
```

Scrivere una funzione per stampare il complementare dell'istogramma.

Esercizio 4 (part 3)

Definire infine una funzione `somma_istogrammi` che dati in input due istogrammi ne restituisca un terzo pari alla somma dei precedenti. La somma è data dalla somma degli elementi nelle posizioni corrispondenti. Si stampi a video il risultato della somma dei due istogrammi creati in precedenza.

Output atteso:

```
********************
********************
********************
```