1 Esercitazione 1: Sequenze pseudo-casuali

Una sequenza pseudo-casuale è una sequenza

$$x_0 x_1 x_2 \dots x_i \dots$$

nella quale ogni elemento x_i viene generato a partire dal precedente. Il primo elemento x_0 viene detto *seme* e di fatto determina tutti gli elementi successivi.

1.1 Esercizio 1: il *seed*

Generare i primi 20 numeri di una sequenza pseudocasuale nell'intervallo $[1, 100)$ e stamparli sullo schermo. Verificare su internet il funzionamento della `random.seed()` e usarla per generare due volte la stessa identica sequenza di 20 numeri.

1.2 Esercizio 2: Costruiamo un generatore

Un semplice (ma efficace) generatore di sequenze di numeri pseudo-casuali utilizza il metodo della congruenza lineare. In generale, il calcolo del prossimo elemento segue

$$x_{i+1} = (a \times x_i + b) \ mod \ M$$

i parametri a (motiplicatore), b (incremento) e M (modulo) sono parametri interi e sono definiti a priori. *mod* è l'operatore di modulo. Questo metodo genera una sequenza pseudo-casuale di numeri interi compresi tra 0 e M. La bontá del generatore dipende da questi parametri. Di seguito alcuni parametri studiati in letteratura:

- $M = 2^{31}$, $a = \pi * 10^8$, $b = 453806245$ (D. Knuth)
- $M = 2^{31} - 1$, $a = 7^5$, $b = 0$ (Goodman e Miller)
- $M = 2^{31}$, $a = 5^{13}$, $b = 0$ (Gordon)
- $M = 2^{31}$, $a = 2^{16} + 3$, $b = 0$ (Leormont e Lewis)

Scrivere un programma Python che utilizza una terna di parametri a, b, M (a scelta dello studente), legge in input il seme x_0 e stampa i primi 100 numeri pseudo casuali della sequenza.

1.3 Esercizio 3: Spostamenti casuali

Utilizzare il modulo `Turtle` e `Random` per effettuare uno spostamento casuale. La tartaruga si muove di 5 pixel, ruota casualmente verso sinistra o verso destra di un angolo compreso fra (-30,30) e ripete l'operazione per un numero di volte fornito in input dall'utente.

Guardate su internet la documentazione di `turtle.color` e usatela per scegliere il colore della penna della tartaruga.

1.4 Realizziamo un modulo per i test

Come avete sperimentato la `testEqual` usata nel testo interattivo non puó essere importata in PyCharm. Il motivo è che fa parte di una libreria (`test`) il cui uso è riservato agli sviluppatori Python. Vi riporto il testo che si trova all'inizio della documentazione:

> Note: The test package is meant for internal use by Python only. It is documented for the benefit of the core developers of Python. Any use of this package outside of Python's standard library is discouraged as code mentioned here can change or be removed without notice between releases of Python.

Quindi giustamente PyCharm non la importa. Tuttavia a lezione abbiamo visto come implementare una funzione che fa la stessa cosa – realizzarla

```
def testEqual(x,y):
    """Testa l'uguaglianza fra due valori (int, str)"""
    #qua inserire il codice

testEqual(2,2)        #Pass
testEqual(2,3)        #Expected 2 got 3
testEqual("ss","ss")  #Pass
testEqual("h","g")    #Expected h got g
testEqual(2.0,2)      #Pass
```

Scrivete la funzione, testatela e (quando funziona bene) salvatela in un file chiamato `testMy.py` Se a questo punto volete usarla in un altro file (nella stessa cartella) basta importarla come nel seguente esempio:

```
import testMy
testMy.testEqual(4,4)
```

Verificate che sia davvero cosí.

1.5 Aggiungiamo a `testMy.py` la funzione di test per i reali

A lezione abbiamo visto come l'operatore `==` non funzioni bene con i numeri reali e come realizzare una funzione di test più corretta. Realizzate questa funzione

```
    def testEqualF(x, y, e):
    """Testa l'uguaglianza fra due valori (float)"""
    #qua inserire il codice

    testEqualF(math.hypot(1, 1.73205),1.999999,0.001) #pass
```

ed aggiungetela al file `testMy.py` Provate ad usarla con

```
from testMy import*
testEqualF(4,3.999999,0.1)
```

Verificate che sia davvero cosí.