Homework di laboratorio di Analisi dei Sistemi, 2017/2018

***, ***

Sono stati eseguiti i primi tre esercizi dell'homework del laboratorio di Analisi dei Sistemi.
La versione di Python utilizzata è la 2.7.12.
Gli esercizi sono organizzati in una directory per esercizio, nella quale sono presenti i test e una directory `src/` con all'interno il file sorgente.
All'interno di ogni directory, è presente un file bash con la sequenza di istruzioni utilizzate.

Esercizio 1:
L'esercizio prevedeva dei test di unità sulle procedure del calcolo fiscale viste a lezione, pertanto il file sorgente non è stato modificato. L'unica modifica del file è la chiamata alla funzione `main()` commentata.
Sono stati creati 4 file di test.
Il file `test_date.py` contiene due funzioni di test:

- `test_mese(self)` testa la corretta conversione da mese a lettera
- `test_giorno(self)`testa la corretta conversione dal giorno di nascita a cifra

Il file `test_string.py` contiene cinque funzioni di test; ogni funzione testa la conversione del nome (o cognome) nella relativa stringa. Sono state testate diverse varianti di nome, come nomi composti o nomi multipli, così come per i cognomi, ovvero cognomi doppi o con caratteri speciali.
Il file `test_codici.py` contiene due funzioni di test:

- `test_codice_comune(self)` testa la corretta conversione da comune a relativa sequenza alfanumerica
- `test_codice_controllo(self)`testa la corretta generazione del carattere di controllo

Relativamente alla funzione `test_codice_comune(self)`, il codice sorgente prevede solamente tre comuni, pertanto l'inserimento di comuni non appartenenti alla lista genereranno un errore nel codice.
Il file `test_cf.py` contiene una funzione di test che testa il programma nella sua interezza, passando una lista di input e intercettando l'output della funzione.
Sono presenti, seppur commentati, tutti i test che rilevano errori nel codice sorgente.
I test sono stati lanciati tramite comando `python-coverage run -p nome_file.py`, per ogni file di test, ed infine sono stati combinati tramite comando `python-coverage combine`.
Prima di lanciare i vari test, l'ambiente viene pulito tramite comando `python-coverage erase`.
La percentuale di coverage ottenuta è stata del 100%.

```
paolo@paolo-VirtualBox:~/Scrivania/esercizi lab analisi sistemi/cf$ ./cf_test_auto.sh

START
cleaning coverage...
cleaning html...
starting tests...
test_date...
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
test_string...
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
test_codici...
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
test_cf...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
combining...
printing report...
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
src/codice_fiscale      72      0   100%
test_cf                 27      0   100%
test_codici             14      0   100%
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
test_string...
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
test_codici...
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
test_cf...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
combining...
printing report...
Name                   Stmts   Miss  Cover   Missing
----------------------------------------------------
src/codice_fiscale        72      0   100%
test_cf                   27      0   100%
test_codici               14      0   100%
test_date                 18      0   100%
test_string               18      0   100%
----------------------------------------------------
TOTAL                    149      0   100%
creating html...
Done.
paolo@paolo-VirtualBox:~/Scrivania/esercizi lab analisi sistemi/cf$
```

Esercizio 2:

L'esercizio prevedeva la creazione di una procedura non ricorsiva per il calcolo dell'IRPEF. La procedura implementata calcola l'imposta lorda.
Sono stati creati 4 file di test.
Il file `test_correttezza_insert.py` contiene una funzione, `test_reddito_insert(self)`, che testa la funzione che restituisce un booleano che indica se l'input inserito è corretto (ovvero un valore convertibile in float non negativo) o meno.
Il file `test_calcolo_scaglione.py` contiene una funzione che testa il corretto riconoscimento del metodo testato dello scaglione utilizzato per il calcolo dell'aliquota; in particolare, inserendo un reddito, la funzione dovrebbe restituire una lista contenente il valore massimo dello scaglione precedente (minimo 0) e il valore dell'aliquota.
Il file `test_calcola_irpef.py` contiene una funzione che controlla l'esatto calcolo dell'IRPEF; siccome il calcolo restituisce un valore `float`, l'istruzione di controllo impiegata utilizza come tolleranza il valore *0.01*.
Il file `test_irpef.py` contiene una funzione di test che testa il programma nella sua interezza, passando una lista di input e intercettando l'output della funzione.
I test sono stati lanciati tramite comando `python-coverage run -p nome_file.py`, per ogni file di test, ed infine sono stati combinati tramite comando `python-coverage combine`.
Prima di lanciare i vari test, l'ambiente viene pulito tramite comando `python-coverage erase`.
La percentuale di coverage ottenuta è stata del 100%.

```
paolo@paolo-VirtualBox:~/Scrivania/esercizi lab analisi sistemi/irpef$ ./irpef_test_auto.sh

START
cleaning coverage...
cleaning html...
starting tests...
test_correttezza_insert...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_calcolo_scaglione...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_calcola_irpef...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_irpef...
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
combining...
printing report...
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
src/irpef                    40      0   100%
test_calcola_irpef           13      0   100%
test_calcolo_scaglione       12      0   100%
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_calcolo_scaglione...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_calcola_irpef...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_irpef...
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
combining...
printing report...
Name                       Stmts   Miss  Cover   Missing
--------------------------------------------------------
src/irpef                     40      0   100%
test_calcola_irpef            13      0   100%
test_calcolo_scaglione        12      0   100%
test_correttezza_insert       14      0   100%
test_irpef                    36      0   100%
--------------------------------------------------------
TOTAL                        115      0   100%
creating html...
Done.
paolo@paolo-VirtualBox:~/Scrivania/esercizi lab analisi sistemi/irpef$
```

Esercizio 3:
L'esercizio prevedeva la creazione di una procedura per il calcolo della Pasqua. L'algoritmo utilizzato è quello di Gauss (implementazione dell'algoritmo di Gauss).
Sono stati creati 3 file di test.
Il file `test_anno.py` contiene una funzione, `test_year_inserted(self)`, che testa la funzione che restituisce un booleano che indica se l'input inserito è corretto (ovvero intero maggiore o uguale a 1583 e minore o uguale a 2499) o meno.
Il file `test_calcolo.py` contiene 3 funzioni:

- `test_easter_in_march(self)` testa la corretta individuazione della Pasqua nel caso in cui la somma di due variabili utilizzate nell'algoritmo sia minore di 10 (`d+e<10`).
- `test_easter_in_april(self)` testa la corretta individuazione della Pasqua nel caso in cui la somma di due variabili utilizzate nell'algoritmo sia maggiore o uguale a 10 (`d+e>=10`).
- `test_easter_exception(self)` testa la corretta individuazione della Pasqua nel caso limite in cui il calcolo finora effettuato indichi come risultato il 26 aprile oppure il 25 aprile (con altre condizioni specifiche).

Il file `test_pasqua.py` contiene una funzione di test che testa il programma nella sua interezza, passando una lista di input e intercettando l'output della funzione.
I test sono stati lanciati tramite comando `python-coverage run -p nome_file.py`, per ogni file di test, ed infine sono stati combinati tramite comando `python-coverage combine`.
Prima di lanciare i vari test, l'ambiente viene pulito tramite comando `python-coverage erase`.
La percentuale di coverage ottenuta è stata del 100%.

```
paolo@paolo-VirtualBox:~/Scrivania/esercizi lab analisi sistemi/pasqua$ ./pasqua_test_auto.sh


START
cleaning coverage...
cleaning html...
starting tests...
test_anno...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
test_calcolo...
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
test_pasqua...
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
combining...
printing report...
Name           Stmts   Miss  Cover   Missing
------------------------------------------
src/pasqua        34      0   100%
test_anno         14      0   100%
test_calcolo      16      0   100%
test_pasqua       41      0   100%
------------------------------------------
TOTAL            105      0   100%
creating html...
Done.
```