Esercitazione 6

13 dicembre 2016

Termine per la consegna dei lavori: **martedì 20 dicembre** ore **23.59**.

Istruzioni

I lavori dovranno essere salvati in una cartella che deve contenere tutto e solo ciò che volete venga consegnato e valutato, d'ora in poi sarà **obbligatoriamente** un file **eseguibile** con estensione `.py` per ognuno degli esercizi. Controllate che l'esecuzione del comando:

```
python <nome_file>.py
```

per ognuno degli esercizi produca l'output desiderato.
Per consegnare gli elaborati dovete raggiungere la cartella contenente i file da inviare in modalità terminale (`cd path_della_cartella`) e quindi eseguire il comando:

```
consegna consegna6
```

verrà visualizzata la lista di tutto ciò che è stato inviato.
Consegne successive (entro il termine per la consegna) sovrascriveranno le precedenti, verrà valutata solo l'ultima consegna sottomessa.
È obbligatorio che all'interno di ogni file sia riportato il vostro nome, cognome e numero di matricola (riportati all'interno di una riga commento all'inizio del file, es: `#Mario Rossi 1234567`).

ATTENZIONE!

Gli unici moduli importabili ammessi in questa esercitazione sono i moduli `math` e `random`. Esercizi risolti utilizzando altri moduli importati riceveranno il punteggio minimo. Python contiene delle **built-in functions**, che potete utilizzare e la cui lista si può trovare a questo indirizzo: `https://docs.python.org/2/library/functions.html`.

Esercizio 1: Fattorizzazione

Scrivere la funzione `factorize(n)` che accetta come parametro un numero naturale `n` ≥ 1 e in modo ricorsivo ritorna la lista di **tutti** i suoi fattori primi. Potete avvalervi di funzioni di supporto, ma la soluzione **deve** essere ricorsiva.

Esempio: $144 = 2^4 \cdot 3^2$ avrà come fattori la lista `[2,2,2,2,3,3]`.

Esercizio 2: Permutazioni

Scrivere la funzione `permutations(s)` che accetta come parametro una stringa `s` e ritorna la lista di tutte le permutazioni senza ripetizione delle sue lettere, escludendo i "doppioni", ovvero, lettere uguali sono considerate uguali anche nelle permutazioni.

Esempio: `'bob'` avrà *solo* le permutazioni `'bob'`, `'bbo'` e `'obb'`.

Esercizio 3: *Listception*

Creare la funzione iterativa `nested_sum(l)` che accetta come parametro una lista `l` la quale può contenere come suoi elementi o un valore intero o una lista che a sua volta è formata allo stesso modo. Quindi `l` può contenere un numero indefinito di liste annidate i quali elementi "finali" sono interi. *Attenzione* che possono esserci liste vuote. Tale funzione deve ritornare la somma di **tutti** i valori interi contenuti in `l`.
Crearne una versione ricorsiva `nested_sum_ric(l)`.

Esempio: `[[1,2,[]],[3,4,[5,6,7]],[8,[9,[10]]]]` ritornerà come somma 55.

Esercizio 4: Numeri vampiri

Un numero n composto da m cifre è detto *Vampiro* se:

1. m è pari;
2. può essere scritto come prodotto di due numeri composti da $m/2$ cifre i quali contengono tutte e sole le cifre contenute in n e al più uno di questi numeri termina con lo 0.

Scrivere un programma che ritorni la lista di tutti i numeri vampiri < 200000.

Esempio: 1260 che può essere scritto come $21 \cdot 60$ è valido, $143500 = 350 \cdot 410$ **non** è valido perché 350 e 410 finiscono entrambi con 0.

Esercizio 5: Numeri di Lucas-Carmichael

Un numero n è detto di *Lucas-Carmichael* se:

1. è composto;
2. non è divisibile per un quadrato;

3. per ogni primo p che divide n vale che $(p+1)$ divide $(n+1)$.

Scrivere una funzione che ritorna la lista di tutti i numeri di Lucas-Carmichael < 100000. Se vi è comodo potete usare la funzione `factorize(n)` dell'Esercizio 1.

Esempio: 399 è un numero di Lucas-Carmichael perché non è divisibile per un quadrato ed essendo $399 = 3 \cdot 7 \cdot 19$ si ha che $4 = 3 + 1, 8 = 7 + 1$ e $20 = 19 + 1$ dividono tutti $400 = 399 + 1$.