Esercitazione 6

12 dicembre 2017

Termine per la consegna dei lavori: **martedì 19 dicembre 2017** ore **23.55**.

Istruzioni

Ogni esercizio dovrà esser salvato in un file con estensione `.py` e consegnato tramite Moodle su sito `https://elearning.unipd.it/math`, sezione "**Matematica: Laurea Triennale**" corso di "**Laboratorio di Programmazione**" cliccando su "**Esecitazione5**".
È obbligatorio che all'interno di ogni file sia riportato il vostro nome, cognome e numero di matricola (potete riportarli all'interno di una riga commento all'inizio del file, es: `#Mario Rossi 1234567`).
Controllate che l'esecuzione del comando:

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

per ognuno degli esercizi produca l'output desiderato.

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: Pari o Dispari

Scrivere la funzione `ric_pd(L)` che accetta come parametro una lista `L` e ritorna `True` se e solo se il numero di elementi di tipo intero divisibili per 5 nella lista è pari. Non è possibile utilizzare la funzione `len`, nè operazioni matematiche di incremento e somma.

Esempio: `[5,3,10,8,'a']` ritornerà `True`, perchè nella lista vi sono due elementi interi divisibili per 5.

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: *Listception bis*

Scrivere la funzione iterativa `depth(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. Tale funzione deve ritornare la profondità massima di annidamento, escludendo le liste vuote.
Esempio: `[]`, `[1,4,5]` e `[1,[]]` avranno come profondità massima 0, 1 ed 1 rispettivamente. `[1,2,[3,4],5,[6,[]]]` avrà come profondità massima 2.

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 $400 = 399 + 1$.