Esercitazione 8

1 gennaio 2017

Termine per la consegna dei lavori: **martedì 10 gennaio** 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. Solo per questa esercitazione, la consegna può essere effettuata su un unico file `.py`. 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 consegna8
```

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

Creare la classe `Number` che rappresenta un qualsiasi numero reale. In particolare la classe deve contenere i seguenti metodi:

1. costruttore (`__init__`) con un parametro `n` che come valore di default vale 42;
2. `is_decimal(self)` che ritorna vero se il numero non è intero (notare che con intero si intende se rappresenta un numero senza decimali);
3. `is_integer(self)` che ritorna vero se il numero è intero (senza decimali);
4. `is_even(self)` che ritorna vero se il numero è (intero) pari;
5. `is_odd(self)` che ritorna vero se il numero è (intero) dispari;
6. `gcd(self, m)` che calcola **ricorsivamente** il massimo comun divisore tra il numero e `m`, se entrambi interi, altrimenti ritorna 1;
7. `lcm(self, m)` che calcola il minimo comun multiplo tra il numero e `m`, se entrambi interi, altrimenti ritorna 0;
8. i metodi speciali `__eq__(self, segment)`, `__lt__(self, segment)` e `__gt__(self, segment)`, i quali dato `m` restituiscono un booleano che rappresenta rispettivamente se, i due numeri sono uguali, se il numero è minore o maggiore di `m`. **Attenzione**: `m` può essere sia un valore numerico python (e.g., `float` o `int`) ma anche un oggetto di tipo `Number`!!
9. il metodo speciale `__repr__(self)` che ritorna una stringa di rappresentazione per il numero;
10. `is_prime(self)` che ritorna vero se il numero è primo;
11. `is_mersenne(self)` che ritorna vero se il numero è un primo di Mersenne. Un numero primo p è detto di Mersenne se è definito come $p = 2^q - 1$ dove q è primo;
12. `is_square(self)` che ritorna vero se il numero è un quadrato perfetto;
13. `is_twin(self, m)` che ritorna vero se il numero è un primo gemello di m (che quindi dovrà essere primo). Due numeri primi p e q sono detti gemelli se sono entrambi primi e $|p - q| = 2$;
14. `is_perfect(self)` che ritorna vero se il numero è perfetto. Un numero n è detto perfetto se è un intero positivo pari ed è uguale alla somma di tutti i suoi divisori escluso se stesso. Esempio: 6 è perfetto perché $6 = 3 + 2 + 1$;
15. `is_fibonacci(self)` che ritorna vero se il numero è contenuto nella sequenza di Fibonacci. **Attenzione**: non si deve generare l'intera sequenza di Fibonacci per eseguire questo test;
16. `is_triangular(self)` che ritorna vero se il numero è triangolare. Un numero n è detto triangolare se si può esprimere come $m(m + 1)/2$ con m intero ≥ 1;

17. `to_rational(self)` che ritorna una coppia `(n,d)` che rappresentano numeratore e denominatore della rappresentazione frazionaria, ridotta ai minimi termini, del numero. **Attenzione**: per motivi dovuti alla precisione di macchina questo metodo non portà essere particolarmente preciso, l'importante è che riesca a gestire numeri con almeno 6 decimali dopo la virgola.

Creare una funzione `apply_all(n)` che ha come parametro un numero n che applica, nell'**ordine** di cui sopra, e ritorna sotto forma di lista, **tutte le funzioni unarie** escludendo i metodi speciali. Applicare la funzione ai numeri 28, 32.25, 144, 144.0, 6765 e 8191.

Infine, dare un esempio per ognuno dei metodi restanti (non unari).