Esercitazione 7

19 dicembre 2017

Termine per la consegna dei lavori: **venerdì 29 dicembre** ore **23.55**.

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 consegna7
```

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 `Point3D` che rappresenta un punto in uno spazio euclideo tridimensionale. In particolare la classe deve contenere:

1. un costruttore (`__init__`) con tre parametri `(x, y, z)` che come valori di default valgono 0;
2. la definizione del metodo `distance(self, point)` che restituisce la distanza del punto da `point`;
3. la definizione del metodo speciale `__repr__(self)` che ritorna una stringa di rappresentazione per il punto, ad esempio `"Point3D(x,y,z)"` dove `x`, `y`, `z` sono le coordinate del punto.

Per ognuno di questi metodi dare almeno un esempio di invocazione.
Nota: l'operatore `__repr__` è invocato quando un oggetto di tipo `Point3D` viene stampato con `print`.

Esercizio 2

Creare la classe `Segment3D` che rappresenta un segmento in uno spazio euclideo tridimensionale. In particolare la classe deve contenere:

1. un costruttore (`__init__`) con due parametri `(pointA, pointB)` dove `pointA` ha valore di default l'origine e `pointB` il punto di coordinate (1,1,1);
2. la definizione del metodo `length(self)` che restituisce la lunghezza del segmento;
3. la definizione dei metodi speciali `__eq__(self, segment)`, `__lt__(self, segment)` e `__gt__(self, segment)`, i quali dato il segmento `segment` restituiscono un booleano che rappresenta rispettivamente se, i due segmenti sono uguali, se il segmento ha una lunghezza minore o maggiore di `segment`;
4. la definizione del metodo speciale `__repr__(self)` che ritorna una stringa di rappresentazione per il segmento.

Per ognuno di questi metodi dare almeno un esempio di invocazione. **Nota**: i metodi `__eq__`, `__lt__` e `__gt__` vengono invocati quando si effettuano confronti, rispettivamente, con gli operatori ==, < e >.

Esercizio 3

Creare una classe `Cube3D` che rappresenta un cubo in uno spazio euclideo tridimensionale. Si assume che il cubo non abbia alcun tipo di rotazione, ovvero le facce sono parallele agli assi. In particolare la classe deve contenere:

1. un costruttore (`__init__`) con due parametri `(origin, edge)` dove `origin` è un `Point3D` che rappresenta il vertice con le coordinate `x`,`y` e `z` minime (ovvero il vertice in basso a sinistra della faccia "frontale" del cubo), con valore di default l'origine degli assi, e `edge` è la lunghezza del lato, con valore di default 1;

2. la definizione del metodo `volume(self)` che restituisce il valore del volume del cubo;

3. la definizione del metodo speciale `__repr__(self)`, che funziona analogamente al corrispondente metodo della classe `Point3D`;

4. la definizione dei metodi speciali `__eq__(self, cube)`, `__lt__(self, cube)` e `__gt__(self, cube)`, i quali dato il cubo `cube` restituiscono un booleano che rappres enta rispettivamente se, i due cubi sono uguali, se il cubo ha un volume minore o maggiore di `cube`;

5. la definizione del metodo `contains_point(self, point)` il quale ritorna un booleano che indica se il punto `point` è contenuto nel cubo;

6. la definizione del metodo `contains_segment(self, segment)` il quale ritorna un booleano che indica se il cubo contiene **completamente** il segmento `segment`;

7. la definizione del metodo `intersect_segment(self, segment)` il quale ritorna un booleano che indica se il cubo interseca il segmento `segment`, ma **non** lo contiene completamente;

8. la definizione del metodo `intersect_cube(self, cube)` il quale ritorna un booleano che indica se il cubo interseca `cube`.

Per ognuno di questi metodi dare almeno un esempio di invocazione.

Esercizio 4

Definire un *main* (metodo principale), che sfrutta le classi `Point3D`, `Segment3D` e `Cube3D`, il quale deve generare casualmente 20 cubi, con lato compreso nell'intervallo reale $[1, 20]$ e con origine di coordinate reali $x, y, z \in [0, 10]$, e 40 segmenti con estremi di coordinate reali $x, y, z \in [0, 10]$.
Una volta generate le suddette entità, ricercare:

1. il cubo che contiene completamente più segmenti: in caso di *ex aequo* restituire il cubo con volume minore;

2. il cubo che interseca più cubi: in caso di *ex aequo* seguire le stesse istruzioni del punto precedente;

3. il segmento che interseca ma non è contenuto in più cubi: in caso di *ex aequo* restituire il segmento più corto;

4. il segmento contenuto in più cubi: in caso di *ex aequo* restituire il segmento più lungo.