Lista 9

Przypomnienie: instrukcja „`assert condition`" (gdzie `condition` to warunek logiczny) rzuca wyjątek `AssertionError` dokładnie wtedy, gdy `condition` jest fałszywy. Na przykład „`assert x > 4`" rzuci wyjątek dokładnie wtedy, gdy `x <= 4`. Możesz użyć tej instrukcji do napisania prostych testów napisanych klas, np.:

```
assert Complex(1, 2) * Complex(1, -2) == Complex(5, 0)
```

Zadanie 1 (1 punkt). Pociąg składa się z lokomotywy oraz nieujemnej liczby nierozróżnialnych wagonów. Lokomotywa ma pewną moc (dodatnia liczba rzeczywista, w watach). Napisz klasę `Train` reprezentującą pociąg. Klasa powinna zawierać metody:

- `__init__(self, power, n)` – inicjalizator, którego parametry `power` i `n` oznaczają odpowiednio moc lokomotywy i liczbę wagonów w składzie pociągu.
- `add_car(self)` – dodaje do pociągu jeden wagon.
- `remove_car(self)` – usuwa z pociągu jeden wagon lub rzuca wyjątek, jeśli pociąg nie ma wagonów.
- `get_resistance(self)` — zwraca *współczynnik tarcia* C_r pociągu dany równaniem:

$$C_r = 50n + 100,$$

  gdzie n to liczba wagonów w pociągu (lokomotywa nie jest wagonem).
- `get_max_speed(self)` — zwraca maksymalną prędkość pociągu. Prędkość ta jest wiekszym z pierwiastków równania

$$C_r v^2 + 10 C_r v = P,$$

  gdzie C_r to współczynnik tarcia pociągu, a P to moc lokomotywy.

Zadanie 2 (1,5 punktu). Napisz **własną**[1] klasę `Complex` służącą do reprezentacji liczb zespolonych. Zaimplementuj następujące metody specjalne[2]:

- `__init__(self, real=0, imag=0)` – inicjalizator tworzący liczbę zespoloną postaci `real` + `imag`$\cdot i$.
- `__abs__(self)` – zwraca liczbę odpowiadającą |`self`|.
- `__neg__(self)` – zwraca liczbę odpowiadającą −`self`.
- `__str__(self)` – zwraca napis reprezentujący `self`.
- `__add__(self, other_num)` – zwraca liczbę odpowiadającą `self` + `other_num`.
- `__sub__(self, other_num)` – zwraca liczbę odpowiadającą `self` − `other_num`.
- `__mul__(self, other_num)` – zwraca liczbę odpowiadającą `self` · `other_num`.
- `__truediv__(self, other_num)` – zwraca liczbę odpowiadającą `self` / `other_num`.
- `__eq__(self, other_num)` – sprawdza, czy `self` = `other_num`.

Zaimplementuj też metodę `conjugate(self)`, zwracającą liczbę odpowiadającą sprzężeniu `self`.

Zadanie 3 (1,5 punktu). Napisz klasę `Triangle` służącą do reprezentowania domkniętych trójkątów na płaszczyźnie, zawierającą następujące metody:

- `__init__(self, x1, y1, x2, y2, x3, y3)` – inicjalizator tworzący trójkąt, którego wierzchołki (w kolejności przeciwnej do ruchu wskazówek zegara) to kolejno (`x1`, `y1`), (`x2`, `y2`), (`x3`, `y3`).
- `circumference(self)` – zwraca obwód trójkąta `self`.
- `area(self)` – zwraca pole trójkąta `self`.
- `contains_point(self, x, y)` – zwraca `True`, jeśli trójkąt `self` zawiera punkt o współrzędnych (`x1`, `y1`) i `False` w przeciwnym wypadku.
- `contains(self, other_triangle)` – zwraca `True`, jeśli trójkąt `other_triangle` zawiera się w trójkącie `self`, `False` w przeciwnym wypadku.
- `contained_in(self, other_triangle)` – zwraca `True`, jeśli trójkąt `self` zawiera się w trójkącie `other_triangle`, `False` w przeciwnym wypadku.
- (za 0,25 punktu) `draw(self)` – rysuje i wyświetla trójkąt `self` z użyciem biblioteki `matplotlib`.

Możesz założyć, że reprezentujemy jedynie trójkąty niezdegenerowane.

[1] W zadaniu nie można używać analogicznej, wbudowanej klasy `complex`.
[2] Metody w tym zadaniu przeciążają odpowiednie operatory.