#1

Створіть клас Point, який відповідатиме за відображення геометричної точки на площині.

Реалізуйте через конструктор \_\_init\_\_ ініціалізацію двох атрибутів: координати x та координати y.

Приклад:

point = Point(5, 10)

print(point.x) # 5

print(point.y) # 10

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

#2

У класу Point через конструктор \_\_init\_\_ оголошено два атрибути: координати x та y. Приховати доступ до них з допомогою подвійного підкреслення: \_\_x та \_\_y

Реалізуйте для класу Point механізми setter та getter до атрибутів \_\_x та \_\_y за допомогою декораторів property та setter.

Приклад:

point = Point(5, 10)

print(point.x) # 5

print(point.y) # 10

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = x

self.\_\_y = y

@property

def x(self):

return self.\_\_x

@property

def y(self):

return self.\_\_y

@x.setter

def x(self, x):

self.\_\_x = x

@y.setter

def y(self, y):

self.\_\_y = y

#3

У класу Point до механізму setter властивостей x і y додайте перевірку на значення, що вводиться. Дозвольте встановлювати значення властивостей x та y для екземпляра класу, тільки якщо вони мають числове значення (int або float).

Приклад:

point = Point("a", 10)

print(point.x) # None

print(point.y)

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

#if isinstance(x, (int, float)):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

#if isinstance(y, (int, float)):

self.\_\_y = y

point = Point("nnm", 10)

print(point.x) # None

print(point.y) # 10

#4

Реалізуйте клас Vector. Властивість coordinates визначає координати вектора і є екземпляром класу Point. Нагадаємо, що вектором називають спрямований відрізок з початком та кінцем. Початок у нас буде в точці (0, 0), а кінець вектора ми задаватимемо атрибутом coordinates.

Реалізуйте можливість звертатися до координат екземпляра класу Vector через квадратні дужки:

vector = Vector(Point(1, 10))

print(vector.coordinates.x) # 1

print(vector.coordinates.y) # 10

vector[0] = 10 # Встановлюємо координату x вектора 10

print(vector[0]) # 10

print(vector[1]) # 10

Щоб отримати значення, використовуючи квадратні дужки об'єкта print(vector[0]), реалізуйте метод \_\_getitem\_\_ у класу Vector.

Для запису значення координат вектора через індекс, як vector[0] = 10, реалізуйте метод \_\_setitem\_\_ у класу Vector.

Звернення до координати x проводиться за індексом 0, а звернення до координати y - за індексом 1.

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

elif index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

elif index == 1:

return self.coordinates.y

vector = Vector(Point(1, 10))

print(vector.coordinates.x) # 1

print(vector.coordinates.y) # 10

vector[0] = 10 # Встановлюємо координату x вектора 10

print(vector[0]) # 10

print(vector[1]) # 10

#5

Реалізуйте для класу Point та Vector магічний метод \_\_str\_\_. Для класу Point метод повинен повертати рядок виду Point(x,y), а для класу Vector - рядок Vector(x,y), як у прикладі нижче (замість x,y необхідно підставити значення координат екземпляра класу):

point = Point(1, 10)

vector = Vector(point)

print(point) # Point(1,10)

print(vector) # Vector(1,10)

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

def \_\_str\_\_(self):

return f'Point({self.x}, {self.y})'

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

if index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

if index == 1:

return self.coordinates.y

def \_\_str\_\_(self):

return f'Vector({self.coordinates.x}, {self.coordinates.y})'

#6

Для екземпляра класу Vector реалізуйте функтор. Створіть для класу Vector метод \_\_call\_\_. Він має реалізувати наступну поведінку:

vector = Vector(Point(1, 10))

print(vector()) # (1, 10)

При виклику екземпляра класу як функції він повертає кортеж з координатами вектора.

Якщо при виклику ми передаємо параметр число, ми виконуємо добуток вектора на число — множимо кожну координату на вказане число та повертаємо кортеж з новими координатами вектора.

vector = Vector(Point(1, 10))

print(vector(5)) # (5, 50)

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

def \_\_str\_\_(self):

return f"Point({self.x},{self.y})"

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

if index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

if index == 1:

return self.coordinates.y

def \_\_call\_\_(self, value=None):

if value:

return self.coordinates.x \* int(value), self.coordinates.y \* int(value)

else:

return self.coordinates.x, self.coordinates.y

def \_\_str\_\_(self):

return f"Vector({self.coordinates.x},{self.coordinates.y})"

#7

Реалізуйте для класу Vector операції додавання та віднімання векторів. Тобто перевизначите для нього математичні оператори \_\_add\_\_ та \_\_sub\_\_

Є два вектори: a з координатами (x1, y1) та b з координатами (x2, y2).

Тоді додавання векторів b + a - це новий вектор з координатами (x2 + x1, y2 + y1). Віднімання – зворотна операція, b - a - це новий вектор з координатами (x2 - x1, y2 - y1)

Приклад коду:

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(10, 10))

vector3 = vector2 + vector1

vector4 = vector2 - vector1

print(vector3) # Vector(11,20)

print(vector4) # Vector(9,0)

*class* Point:

*def* \_\_init\_\_(*self*, *x*, *y*):

*self*.\_\_x = None

*self*.\_\_y = None

*self*.x = *x*

*self*.y = *y*

    @property

*def* x(*self*):

        return *self*.\_\_x

    @x.setter

*def* x(*self*, *x*):

        if (type(*x*) == int) or (type(*x*) == float):

*self*.\_\_x = *x*

    @property

*def* y(*self*):

        return *self*.\_\_y

    @y.setter

*def* y(*self*, *y*):

        if (type(*y*) == int) or (type(*y*) == float):

*self*.\_\_y = *y*

*def* \_\_str\_\_(*self*):

        return *f*"Point({*self*.x},{*self*.y})"

*class* Vector:

*def* \_\_init\_\_(*self*, *coordinates*: Point):

*self*.coordinates = *coordinates*

*def* \_\_setitem\_\_(*self*, *index*, *value*):

        if *index* == 0:

*self*.coordinates.x = *value*

        if *index* == 1:

*self*.coordinates.y = *value*

*def* \_\_getitem\_\_(*self*, *index*):

        if *index* == 0:

            return *self*.coordinates.x

        if *index* == 1:

            return *self*.coordinates.y

*def* \_\_call\_\_(*self*, *value*=None):

        if *value*:

*self*.coordinates.x = *self*.coordinates.x \* *value*

*self*.coordinates.y = *self*.coordinates.y \* *value*

        return *self*.coordinates.x, *self*.coordinates.y

*def* \_\_add\_\_(*self*, *vector*):

        # if vector:

        #self.coordinates.x + vector.coordinates.x

        #self.coordinates.y + vector.coordinates.y

        return Vector(Point(*self*.coordinates.x + *vector*.coordinates.x, *self*.coordinates.y + *vector*.coordinates.y))

*def* \_\_sub\_\_(*self*, *vector*):

        # if vector:

        #self.coordinates.x - vector.coordinates.x

        #self.coordinates.y - vector.coordinates.y

        return Vector(Point(*self*.coordinates.x - *vector*.coordinates.x, *self*.coordinates.y - *vector*.coordinates.y))

*def* \_\_str\_\_(*self*):

        return *f*"Vector({*self*.coordinates.x},{*self*.coordinates.y})"

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(10, 10))

vector3 = vector2 + vector1

vector4 = vector2 - vector1

print(vector1)

print(vector2)

print(vector3)  # Vector(11,20)

print(vector4)  # Vector(9,0)

#8

Реалізуйте для класу Vector операцію скалярного добутку векторів. Тобто перевизначте для нього математичний оператор \_\_mul\_\_

Є два вектори: a з координатами (x1, y1) та вектор b з координатами (x2, y2).

Тоді скалярний добуток векторів b\*a - це таке число x2\*x1+y2\*y1.

Приклад коду:

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(10, 10))

scalar = vector2 \* vector1

print(scalar) # 110

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

def \_\_str\_\_(self):

return f"Point({self.x},{self.y})"

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

if index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

if index == 1:

return self.coordinates.y

def \_\_call\_\_(self, value=None):

if value:

self.coordinates.x = self.coordinates.x \* value

self.coordinates.y = self.coordinates.y \* value

return self.coordinates.x, self.coordinates.y

def \_\_add\_\_(self, vector):

x = self.coordinates.x + vector.coordinates.x

y = self.coordinates.y + vector.coordinates.y

return Vector(Point(x, y))

def \_\_sub\_\_(self, vector):

x = self.coordinates.x - vector.coordinates.x

y = self.coordinates.y - vector.coordinates.y

return Vector(Point(x, y))

def \_\_mul\_\_(self, vector):

return vector.coordinates.x\*self.coordinates.x+vector.coordinates.y\*self.coordinates.y

def \_\_str\_\_(self):

return f"Vector({self.coordinates.x},{self.coordinates.y})"

#9

Перш ніж ми приступимо до операцій порівняння векторів, реалізуйте метод визначення довжини вектора - len для класу Vector

Для вектора a з координатами (x1, y1) його довжина визначається за такою формулою:

(x1 \*\* 2 + y1 \*\* 2) \*\* 0.5.

Приклад коду:

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(10, 10))

print(vector1.len()) # 10.04987562112089

print(vector2.len()) # 14.142135623730951

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

def \_\_str\_\_(self):

return f"Point({self.x},{self.y})"

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

if index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

if index == 1:

return self.coordinates.y

def \_\_call\_\_(self, value=None):

if value:

self.coordinates.x = self.coordinates.x \* value

self.coordinates.y = self.coordinates.y \* value

return self.coordinates.x, self.coordinates.y

def \_\_add\_\_(self, vector):

x = self.coordinates.x + vector.coordinates.x

y = self.coordinates.y + vector.coordinates.y

return Vector(Point(x, y))

def \_\_sub\_\_(self, vector):

x = self.coordinates.x - vector.coordinates.x

y = self.coordinates.y - vector.coordinates.y

return Vector(Point(x, y))

def \_\_mul\_\_(self, vector):

return (

self.coordinates.x \* vector.coordinates.x

+ self.coordinates.y \* vector.coordinates.y

)

def len(self):

return (self.coordinates.x \*\* 2 + self.coordinates.y \*\* 2) \*\* 0.5

def \_\_str\_\_(self):

return f"Vector({self.coordinates.x},{self.coordinates.y})"

#10

Реалізуйте всі методи порівняння для класу Vector. З метою спрощення порівнювати екземпляри класу Vector будемо тільки за їх довжиною, використовуючи метод len, не враховуючи напрямок векторів.

Приклад коду:

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(3, 10))

print(vector1 == vector2) # False

print(vector1 != vector2) # True

print(vector1 > vector2) # False

print(vector1 < vector2) # True

print(vector1 >= vector2) # False

print(vector1 <= vector2) # True

class Point:

def \_\_init\_\_(self, x, y):

self.\_\_x = None

self.\_\_y = None

self.x = x

self.y = y

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, x):

if (type(x) == int) or (type(x) == float):

self.\_\_x = x

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, y):

if (type(y) == int) or (type(y) == float):

self.\_\_y = y

def \_\_str\_\_(self):

return f"Point({self.x},{self.y})"

class Vector:

def \_\_init\_\_(self, coordinates: Point):

self.coordinates = coordinates

def \_\_setitem\_\_(self, index, value):

if index == 0:

self.coordinates.x = value

if index == 1:

self.coordinates.y = value

def \_\_getitem\_\_(self, index):

if index == 0:

return self.coordinates.x

if index == 1:

return self.coordinates.y

def \_\_call\_\_(self, value=None):

if value:

self.coordinates.x = self.coordinates.x \* value

self.coordinates.y = self.coordinates.y \* value

return self.coordinates.x, self.coordinates.y

def \_\_add\_\_(self, vector):

x = self.coordinates.x + vector.coordinates.x

y = self.coordinates.y + vector.coordinates.y

return Vector(Point(x, y))

def \_\_sub\_\_(self, vector):

x = self.coordinates.x - vector.coordinates.x

y = self.coordinates.y - vector.coordinates.y

return Vector(Point(x, y))

def \_\_mul\_\_(self, vector):

return (

self.coordinates.x \* vector.coordinates.x

+ self.coordinates.y \* vector.coordinates.y

)

def len(self):

return (self.coordinates.x \*\* 2 + self.coordinates.y \*\* 2) \*\* 0.5

def \_\_str\_\_(self):

return f"Vector({self.coordinates.x},{self.coordinates.y})"

def \_\_eq\_\_(self, vector):

return self.len() == vector.len()

def \_\_ne\_\_(self, vector):

return self.len() != vector.len()

def \_\_lt\_\_(self, vector):

return self.len() < vector.len()

def \_\_gt\_\_(self, vector):

return self.len() > vector.len()

def \_\_le\_\_(self, vector):

return self.len() <= vector.len()

def \_\_ge\_\_(self, vector):

return self.len() >= vector.len()

vector1 = Vector(Point(1, 10))

vector2 = Vector(Point(3, 10))

print(vector1 == vector2) # False

print(vector1 != vector2) # True

print(vector1 > vector2) # False

print(vector1 < vector2) # True

print(vector1 >= vector2) # False

print(vector1 <= vector2) # True

#11

Необхідно реалізувати клас RandomVectors, який зможе створювати об'єкт, що ітерується, і дозволяти ітеруватися по випадковим векторам.

Формат класу:

RandomVectors(max\_vectors: int, max\_points: int) -> Iterable(max\_vectors, max\_points)

де:

* max\_vectors — визначає максимальну кількість елементів (примірників класу Vector) в ітерованій послідовності
* max\_points — визначає максимальне значення для координат x та y (в діапазоні 0...max\_points)

Щоб екземпляри класу RandomVectors були об'єктами, що ітеруються, в класі повинен бути реалізований метод \_\_iter\_\_, який повертає ітератор. Ітератор – це будь-який об'єкт, який на кожному кроці ітерації (крок ітерації – це виклик методу next() для цього ітератора) повертає таке значення - і так до вичерпання кількості ітерацій (визначається параметром max\_vectors).

У нашому випадку ітератором буде клас Iterable, у якому необхідно реалізувати метод \_\_next\_\_. Він у конструкторі отримує ті ж параметри max\_vectors та max\_points, що і клас RandomVectors.

Метод \_\_next\_\_ повинен видавати кожне наступне значення зі списку self.vectors. Створіть у конструкторі набір випадкових векторів self.vectors завдовжки max\_vectors за допомогою randrange. Атрибут current\_index вказівник-індекс на поточний вектор зі списку vectors, необхідний для ітерування.

Приклад роботи класу `RandomVectors:

vectors = RandomVectors(5, 10)

for vector in vectors:

print(vector)

Вивід має бути схожим на цей:

Vector(7,7)

Vector(0,0)

Vector(8,9)

Vector(1,9)

Vector(6,6)

Деталізуємо наше завдання:

1. Клас RandomVectors повинен мати метод \_\_iter\_\_, який має повернути об'єкт ітератора (клас Iterable)
2. Об'єкт ітератора (примірник класу Iterable) повинен мати метод \_\_next\_\_
3. Метод \_\_next\_\_ стежить за кількістю можливих кроків ітерації, вони визначаються параметром max\_vectors
4. Якщо ми вичерпали можливі кроки, то метод \_\_next\_\_ генерує виняток StopIteration
5. В іншому випадку метод \_\_next\_\_ повертає вектор з випадковими координатами (примірник класу Vector), розмір координат вектора визначається параметром max\_points.

from random import randrange

*class* Point:

*def* \_\_init\_\_(*self*, *x*, *y*):

*self*.\_\_x = None

*self*.\_\_y = None

*self*.x = *x*

*self*.y = *y*

    @property

*def* x(*self*):

        return *self*.\_\_x

    @x.setter

*def* x(*self*, *x*):

        if (type(*x*) == int) or (type(*x*) == float):

*self*.\_\_x = *x*

    @property

*def* y(*self*):

        return *self*.\_\_y

    @y.setter

*def* y(*self*, *y*):

        if (type(*y*) == int) or (type(*y*) == float):

*self*.\_\_y = *y*

*def* \_\_str\_\_(*self*):

        return *f*"Point({*self*.x},{*self*.y})"

*class* Vector:

*def* \_\_init\_\_(*self*, *coordinates*: Point):

*self*.coordinates = *coordinates*

*def* \_\_setitem\_\_(*self*, *index*, *value*):

        if *index* == 0:

*self*.coordinates.x = *value*

        if *index* == 1:

*self*.coordinates.y = *value*

*def* \_\_getitem\_\_(*self*, *index*):

        if *index* == 0:

            return *self*.coordinates.x

        if *index* == 1:

            return *self*.coordinates.y

*def* \_\_call\_\_(*self*, *value*=None):

        if *value*:

*self*.coordinates.x = *self*.coordinates.x \* *value*

*self*.coordinates.y = *self*.coordinates.y \* *value*

        return *self*.coordinates.x, *self*.coordinates.y

*def* \_\_add\_\_(*self*, *vector*):

        x = *self*.coordinates.x + *vector*.coordinates.x

        y = *self*.coordinates.y + *vector*.coordinates.y

        return Vector(Point(x, y))

*def* \_\_sub\_\_(*self*, *vector*):

        x = *self*.coordinates.x - *vector*.coordinates.x

        y = *self*.coordinates.y - *vector*.coordinates.y

        return Vector(Point(x, y))

*def* \_\_mul\_\_(*self*, *vector*):

        return (

*self*.coordinates.x \* *vector*.coordinates.x

            + *self*.coordinates.y \* *vector*.coordinates.y

        )

*def* len(*self*):

        return (*self*.coordinates.x \*\* 2 + *self*.coordinates.y \*\* 2) \*\* 0.5

*def* \_\_str\_\_(*self*):

        return *f*"Vector({*self*.coordinates.x},{*self*.coordinates.y})"

*def* \_\_eq\_\_(*self*, *vector*):

        return *self*.len() == *vector*.len()

*def* \_\_ne\_\_(*self*, *vector*):

        return *self*.len() != *vector*.len()

*def* \_\_lt\_\_(*self*, *vector*):

        return *self*.len() < *vector*.len()

*def* \_\_gt\_\_(*self*, *vector*):

        return *self*.len() > *vector*.len()

*def* \_\_le\_\_(*self*, *vector*):

        return *self*.len() <= *vector*.len()

*def* \_\_ge\_\_(*self*, *vector*):

        return *self*.len() >= *vector*.len()

*class* Iterable:

*def* \_\_init\_\_(*self*, *max\_vectors*, *max\_points*):

*self*.current\_index = 0

*self*.vectors = []

*self*.max\_points = *max\_points*

*self*.max\_vectors = *max\_vectors*

*def* \_\_next\_\_(*self*):

        if *self*.current\_index < *self*.max\_vectors:

*self*.current\_index += 1

*self*.vectors = Vector(

                Point(randrange(0, *self*.max\_points), randrange(0, *self*.max\_points)))

            return *self*.vectors

        raise StopIteration

*class* RandomVectors:

*def* \_\_init\_\_(*self*, *max\_vectors*=10, *max\_points*=50):

*self*.max\_vectors = *max\_vectors*

*self*.max\_points = *max\_points*

*def* \_\_iter\_\_(*self*):

        return Iterable(*self*.max\_vectors, *self*.max\_points)

vectors = RandomVectors(5, 10)

for vector in vectors:

    print(vector)