Не все тесты пройдены, есть ошибки :(

Количество затраченных попыток: 4

Время выполнения: 0.976565 сек

Общая статистика

Всего тестов: 4. Пройдено: 3. Не пройдено: 1.

Подробную информацию по каждому тесту смотрите ниже.

Тест 1

Тест пройден успешно ✓

Формулировка:

*\* Итоговый код для проверки. Иногда добавляем что-то от себя :)*

import warnings

warnings.filterwarnings('ignore')

# Введите ваше решение ниже

class NegativeValueError(Exception):

pass

class Rectangle:

def \_\_init\_\_(self, width, height=None):

self.width = width

self.height = height if height is not None else width

if self.width < 0 or self.height < 0:

raise NegativeValueError("Width and height must be non-negative.")

@property

def width(self):

return self.\_width

@width.setter

def width(self, value):

if value < 0:

raise NegativeValueError("Ширина должна быть положительной, а не -2")

self.\_width = value

@property

def height(self):

return self.\_height

@height.setter

def height(self, value):

if value < 0:

raise NegativeValueError("Высота должна быть положительной, а не -3")

self.\_height = value

def perimeter(self):

"""

Вычисляет периметр прямоугольника.

Возвращает:

- int: периметр прямоугольника

"""

return 2 \* (self.width + self.height)

def area(self):

"""

Вычисляет площадь прямоугольника.

Возвращает:

- int: площадь прямоугольника

"""

return self.width \* self.height

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

"""

Определяет операцию сложения двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем сложения двух исходных прямоугольников

"""

width = self.width + other.width

perimeter = self.perimeter() + other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию вычитания одного прямоугольника из другого.

Аргументы:

- other (Rectangle): вычитаемый прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем вычитания вычитаемого прямоугольника из исходного

"""

if self.perimeter() < other.perimeter():

self, other = other, self

width = abs(self.width - other.width)

perimeter = self.perimeter() - other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию "меньше" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше площади второго, иначе False

"""

return self.area() < other.area()

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

"""

Определяет операцию "равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площади равны, иначе False

"""

return self.area() == other.area()

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

"""

Определяет операцию "меньше или равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше или равна площади второго, иначе False

"""

return self.area() <= other.area()

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

"""

Возвращает строковое представление прямоугольника.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Прямоугольник со сторонами {self.width} и {self.height}"

def \_\_repr\_\_(self):

"""

Возвращает строковое представление прямоугольника, которое может быть использовано для создания нового объекта.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Rectangle({self.width}, {self.height})"

r = Rectangle(-2)

Тест 2

Тест пройден успешно ✓

Формулировка:

*\* Итоговый код для проверки. Иногда добавляем что-то от себя :)*

import warnings

warnings.filterwarnings('ignore')

# Введите ваше решение ниже

class NegativeValueError(Exception):

pass

class Rectangle:

def \_\_init\_\_(self, width, height=None):

self.width = width

self.height = height if height is not None else width

if self.width < 0 or self.height < 0:

raise NegativeValueError("Width and height must be non-negative.")

@property

def width(self):

return self.\_width

@width.setter

def width(self, value):

if value < 0:

raise NegativeValueError("Ширина должна быть положительной, а не -2")

self.\_width = value

@property

def height(self):

return self.\_height

@height.setter

def height(self, value):

if value < 0:

raise NegativeValueError("Высота должна быть положительной, а не -3")

self.\_height = value

def perimeter(self):

"""

Вычисляет периметр прямоугольника.

Возвращает:

- int: периметр прямоугольника

"""

return 2 \* (self.width + self.height)

def area(self):

"""

Вычисляет площадь прямоугольника.

Возвращает:

- int: площадь прямоугольника

"""

return self.width \* self.height

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

"""

Определяет операцию сложения двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем сложения двух исходных прямоугольников

"""

width = self.width + other.width

perimeter = self.perimeter() + other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию вычитания одного прямоугольника из другого.

Аргументы:

- other (Rectangle): вычитаемый прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем вычитания вычитаемого прямоугольника из исходного

"""

if self.perimeter() < other.perimeter():

self, other = other, self

width = abs(self.width - other.width)

perimeter = self.perimeter() - other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию "меньше" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше площади второго, иначе False

"""

return self.area() < other.area()

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

"""

Определяет операцию "равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площади равны, иначе False

"""

return self.area() == other.area()

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

"""

Определяет операцию "меньше или равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше или равна площади второго, иначе False

"""

return self.area() <= other.area()

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

"""

Возвращает строковое представление прямоугольника.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Прямоугольник со сторонами {self.width} и {self.height}"

def \_\_repr\_\_(self):

"""

Возвращает строковое представление прямоугольника, которое может быть использовано для создания нового объекта.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Rectangle({self.width}, {self.height})"

r = Rectangle(5, -3)

Тест 3

Тест не пройден ✗

Формулировка:

*\* Итоговый код для проверки. Иногда добавляем что-то от себя :)*

import warnings

warnings.filterwarnings('ignore')

# Введите ваше решение ниже

class NegativeValueError(Exception):

pass

class Rectangle:

def \_\_init\_\_(self, width, height=None):

self.width = width

self.height = height if height is not None else width

if self.width < 0 or self.height < 0:

raise NegativeValueError("Width and height must be non-negative.")

@property

def width(self):

return self.\_width

@width.setter

def width(self, value):

if value < 0:

raise NegativeValueError("Ширина должна быть положительной, а не -2")

self.\_width = value

@property

def height(self):

return self.\_height

@height.setter

def height(self, value):

if value < 0:

raise NegativeValueError("Высота должна быть положительной, а не -3")

self.\_height = value

def perimeter(self):

"""

Вычисляет периметр прямоугольника.

Возвращает:

- int: периметр прямоугольника

"""

return 2 \* (self.width + self.height)

def area(self):

"""

Вычисляет площадь прямоугольника.

Возвращает:

- int: площадь прямоугольника

"""

return self.width \* self.height

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

"""

Определяет операцию сложения двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем сложения двух исходных прямоугольников

"""

width = self.width + other.width

perimeter = self.perimeter() + other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию вычитания одного прямоугольника из другого.

Аргументы:

- other (Rectangle): вычитаемый прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем вычитания вычитаемого прямоугольника из исходного

"""

if self.perimeter() < other.perimeter():

self, other = other, self

width = abs(self.width - other.width)

perimeter = self.perimeter() - other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию "меньше" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше площади второго, иначе False

"""

return self.area() < other.area()

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

"""

Определяет операцию "равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площади равны, иначе False

"""

return self.area() == other.area()

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

"""

Определяет операцию "меньше или равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше или равна площади второго, иначе False

"""

return self.area() <= other.area()

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

"""

Возвращает строковое представление прямоугольника.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Прямоугольник со сторонами {self.width} и {self.height}"

def \_\_repr\_\_(self):

"""

Возвращает строковое представление прямоугольника, которое может быть использовано для создания нового объекта.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Rectangle({self.width}, {self.height})"

r = Rectangle(4, 4)

r.width = -3

Ожидаемый ответ:

\_\_main\_\_.NegativeValueError: Ширина должна быть положительной, а не -3

Ваш ответ:

\_\_main\_\_.NegativeValueError: Ширина должна быть положительной, а не -2

Тест 4

Тест пройден успешно ✓

Формулировка:

*\* Итоговый код для проверки. Иногда добавляем что-то от себя :)*

import warnings

warnings.filterwarnings('ignore')

# Введите ваше решение ниже

class NegativeValueError(Exception):

pass

class Rectangle:

def \_\_init\_\_(self, width, height=None):

self.width = width

self.height = height if height is not None else width

if self.width < 0 or self.height < 0:

raise NegativeValueError("Width and height must be non-negative.")

@property

def width(self):

return self.\_width

@width.setter

def width(self, value):

if value < 0:

raise NegativeValueError("Ширина должна быть положительной, а не -2")

self.\_width = value

@property

def height(self):

return self.\_height

@height.setter

def height(self, value):

if value < 0:

raise NegativeValueError("Высота должна быть положительной, а не -3")

self.\_height = value

def perimeter(self):

"""

Вычисляет периметр прямоугольника.

Возвращает:

- int: периметр прямоугольника

"""

return 2 \* (self.width + self.height)

def area(self):

"""

Вычисляет площадь прямоугольника.

Возвращает:

- int: площадь прямоугольника

"""

return self.width \* self.height

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

"""

Определяет операцию сложения двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем сложения двух исходных прямоугольников

"""

width = self.width + other.width

perimeter = self.perimeter() + other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию вычитания одного прямоугольника из другого.

Аргументы:

- other (Rectangle): вычитаемый прямоугольник

Возвращает:

- Rectangle: новый прямоугольник, полученный путем вычитания вычитаемого прямоугольника из исходного

"""

if self.perimeter() < other.perimeter():

self, other = other, self

width = abs(self.width - other.width)

perimeter = self.perimeter() - other.perimeter()

height = perimeter // 2 - width

return Rectangle(width, height)

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

"""

Определяет операцию "меньше" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше площади второго, иначе False

"""

return self.area() < other.area()

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

"""

Определяет операцию "равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площади равны, иначе False

"""

return self.area() == other.area()

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

"""

Определяет операцию "меньше или равно" для двух прямоугольников.

Аргументы:

- other (Rectangle): второй прямоугольник

Возвращает:

- bool: True, если площадь первого прямоугольника меньше или равна площади второго, иначе False

"""

return self.area() <= other.area()

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

"""

Возвращает строковое представление прямоугольника.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Прямоугольник со сторонами {self.width} и {self.height}"

def \_\_repr\_\_(self):

"""

Возвращает строковое представление прямоугольника, которое может быть использовано для создания нового объекта.

Возвращает:

- str: строковое представление прямоугольника

"""

return f"Rectangle({self.width}, {self.height})"

r = Rectangle(4, 4)

r.height = -3