

План модуля:

- ▶ Класс и экземпляр класса
- ▶ Методы и свойства
- ▶ Конструктор класса



Вспомним

Переменные


- ▶ Что такое переменная?



Вспомним

Переменные

- ▶ Что такое переменная?

a
 - именованный контейнер


- ▶ Что может быть внутри переменной?




Вспомним

Переменные

► Что такое переменная?

a
 - именованный контейнер

► Что может быть внутри переменной?

a **b** **c**
 - любые значения. Числа, строки, ...



Вспомним

Списки

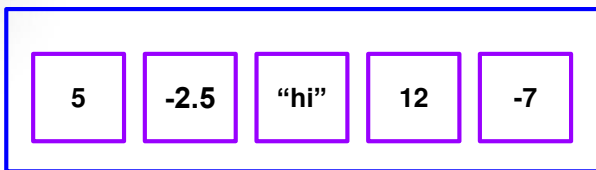
- ▶ Что такое список(list)?



Вспомним

Списки

► Что такое список(list)?



- контейнер с контейнерами

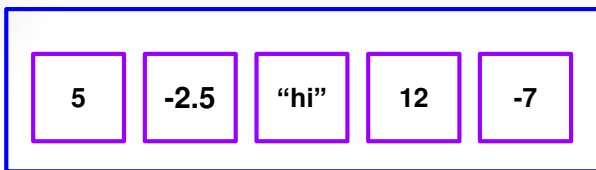
► Как обратиться к элементам списка?



Вспомним

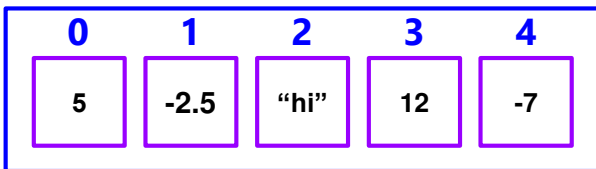
Списки

► Что такое список(list)?



- контейнер с контейнерами

► Как обратиться к элементам списка?



- по индексам. Нумерованные контейнеры



Вспомним

Словари

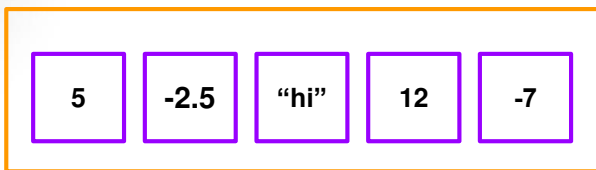
- ▶ Что такое словарь(dict)?



Вспомним

Словари

► Что такое словарь(dict)?



- контейнер с контейнерами

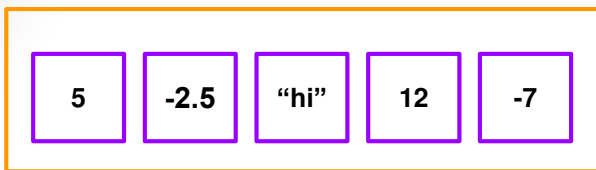
► Как обратиться к элементам словаря?



Вспомним

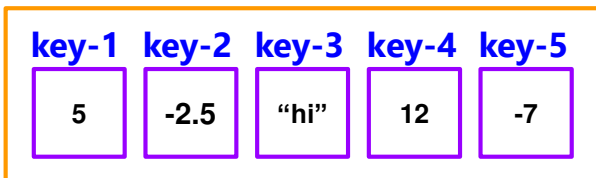
Словари

► Что такое словарь(dict)?



- контейнер с контейнерами

► Как обратиться к элементам словаря?



- по ключам. Именованные контейнеры



Вспомним

Строки

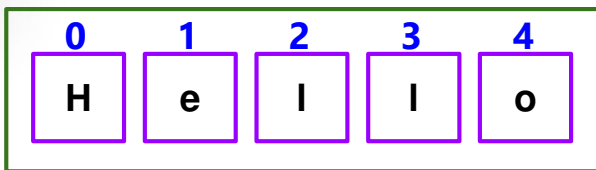
- ▶ Что такое строка(str)?



Вспомним

Строки

► Что такое строка(str)?



- контейнер с символами



Вспомним

Методы

- ▶ Что такое методы? И для чего они нужны?



Вспомним

Методы

- ▶ Что такое методы? И для чего они нужны?

```
my_list = [2, 5, -6]  
my_list.append(8)
```

```
my_dict = {"key1": 12, "key2": 23}  
my_dict.get("key1")
```

Методы - **функции**, выполняющие операции над объектом, у которого вызываются

- ▶ А у **int** и **float** есть методы?



Вспомним

Методы

- ▶ Что такое методы? И для чего они нужны?

```
my_list = [2, 5, -6]  
my_list.append(8)
```

```
my_dict = {"key1": 12, "key2": 23}  
my_dict.get("key1")
```

Методы - **функции**, выполняющие операции над объектом, у которого вызываются

- ▶ А у **int** и **float** есть методы?

```
n = 5  
n.bit_length()    # 3
```

```
f = 2.5  
f.as_integer_ratio()    # (5, 2)
```



В python ВСЕ объекты

Вы уже работали с объектами

```
n = 5
```

```
type(n) # <class 'int'>
```

```
f = 2.5
```

```
type(f) # <class 'float'>
```

```
my_list = []
```

```
type(my_list) # <class 'list'>
```



Цели

- ✓ Понять что такое объект
- ✓ Научиться создавать свои объекты
- ✓ Лучше понять устройство объектов
- ✓ Через объекты глубже погрузиться в язык



Класс

Класс как контейнер

```
class Point:
```

```
    x = 7
```

```
    y = 4
```



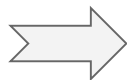
Класс

Класс как контейнер

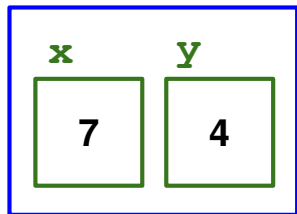
```
class Point:
```

```
    x = 7
```

```
    y = 4
```



Point



Похоже на словарь?

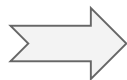
Класс

Класс как контейнер

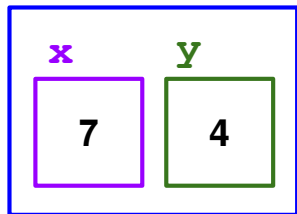
```
class Point:
```

```
    x = 7
```

```
    y = 4
```



Point



```
# Читаем  
print(Point.x)
```



7



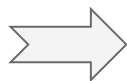
Класс

Класс как контейнер

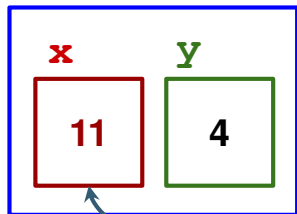
```
class Point:
```

```
    x = 7
```

```
    y = 4
```



Point



```
# Читаем  
print(Point.x)
```

```
# Записываем  
Point.x = 11
```

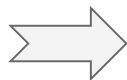
Класс

Класс как контейнер

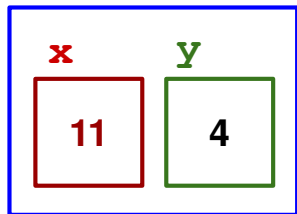
```
class Point:
```

```
    x = 7
```

```
    y = 4
```



Point



```
# Читаем  
print(Point.x)
```

```
# Записываем  
Point.x = 11
```

```
# Снова читаем  
print(Point.x)
```



11

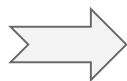
Класс

Класс как контейнер

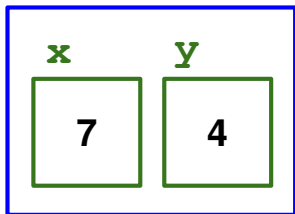
```
class Point:
```

```
    x = 7
```

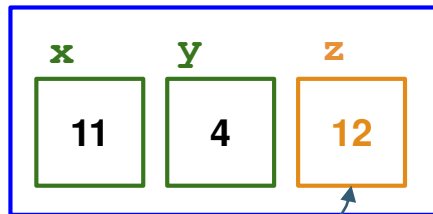
```
    y = 4
```



Point



Point



```
# Создаем новую  
Point.z = 12
```

```
# Читаем  
print(Point.z)
```



12

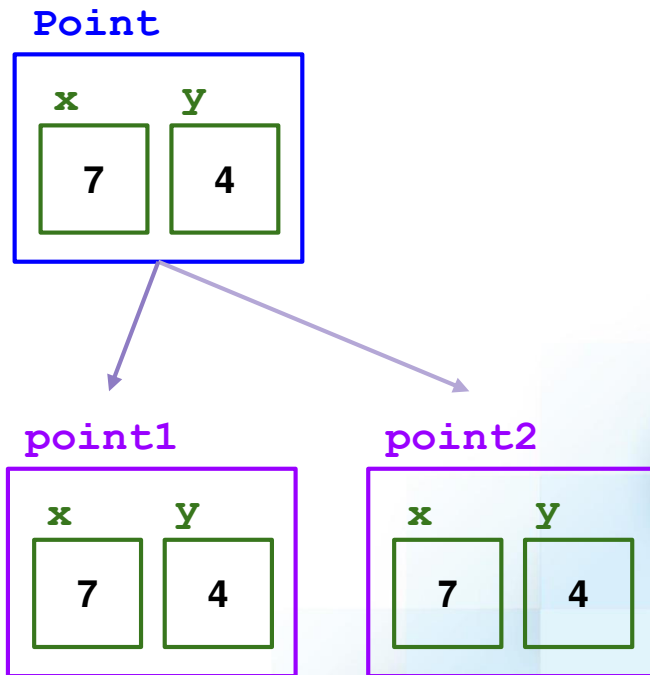


Класс

Объекты из класса

```
class Point:
    x = 7
    y = 4

# Создадим несколько объектов-точек
point1 = Point()
point2 = Point()
print(point1.x) # 7
print(point2.x) # 7
```



Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

```
    y = 4
```

```
# Создадим несколько объектов-точек
```

```
point1 = Point()
```

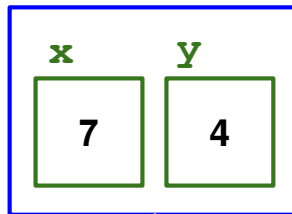
```
point2 = Point()
```

```
point1.x = 10
```

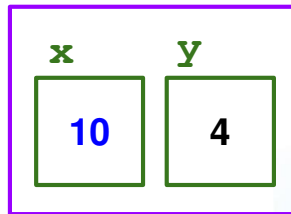
```
print(point1.x)    # ???
```

```
print(point2.x)    # ???
```

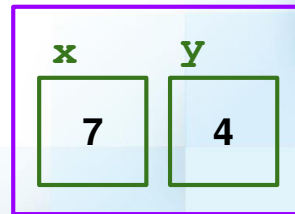
Point



point1



point2



?

Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

```
    y = 4
```

```
# Создадим несколько объектов-точек
```

```
point1 = Point()
```

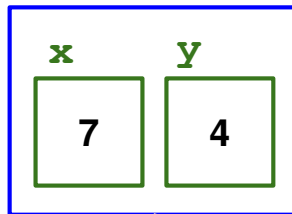
```
point2 = Point()
```

```
point1.x = 10
```

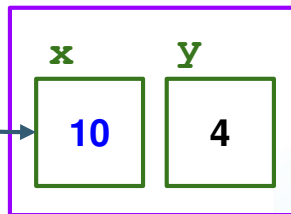
```
print(point1.x)    # 10
```

```
print(point2.x)    # 7
```

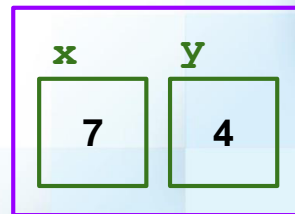
Point



point1



point2



Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

```
    y = 4
```

```
# Создадим несколько объектов-точек
```

```
point1 = Point()
```

```
point2 = Point()
```

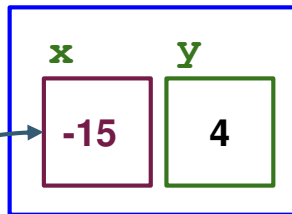
```
point1.x = 10
```

```
Point.x = -15
```

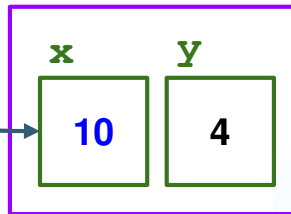
```
print(point1.x)    # ???
```

```
print(point2.x)    # ???
```

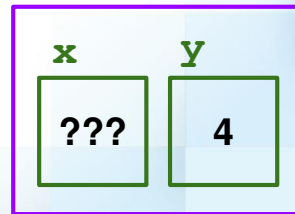
Point



point1



point2



?



Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

```
    y = 4
```

```
# Создадим несколько объектов-точек
```

```
point1 = Point()
```

```
point2 = Point()
```

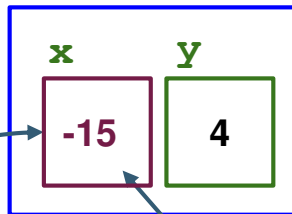
```
point1.x = 10
```

```
Point.x = -15
```

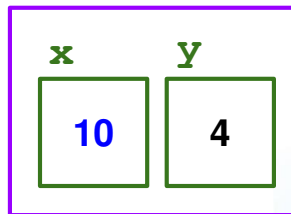
```
print(point1.x) # 10
```

```
print(point2.x) # -15
```

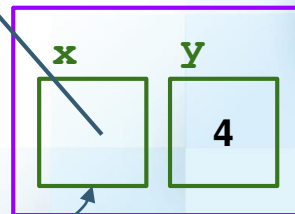
Point



point1



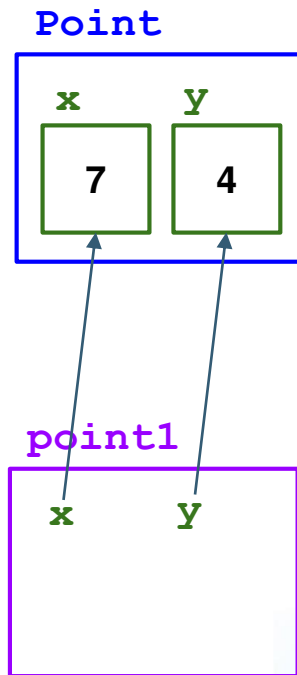
point2



Класс

Объекты из класса

```
class Point:  
    x = 7  
    y = 4  
  
point1 = Point()  
print(point1.x)    # 7
```



Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

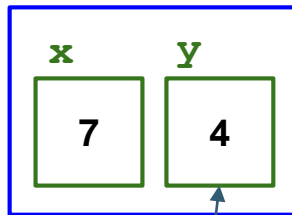
```
    y = 4
```

```
point1 = Point()
```

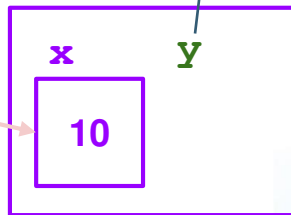
```
print(point1.x)    # 7
```

```
point1.x = 10
```

Point



point1



Класс

Объекты из класса

```
class Point:
```

```
    x = 7
```

```
    y = 4
```

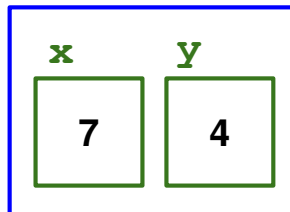
```
point1 = Point()
```

```
print(point1.x)    # 7
```

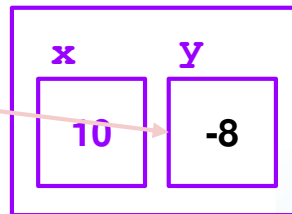
```
point1.x = 10
```

```
point1.y = -8
```

Point



point1



Класс

Конструктор класса

```
class Point:  
    x = 7  
    y = 4
```

```
point1 = Point()  
point1.x = 10  
point1.y = -8
```



```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

```
point1 = Point(10, -8)  
print(point1.x)    # 10  
print(point1.y)    # -8
```

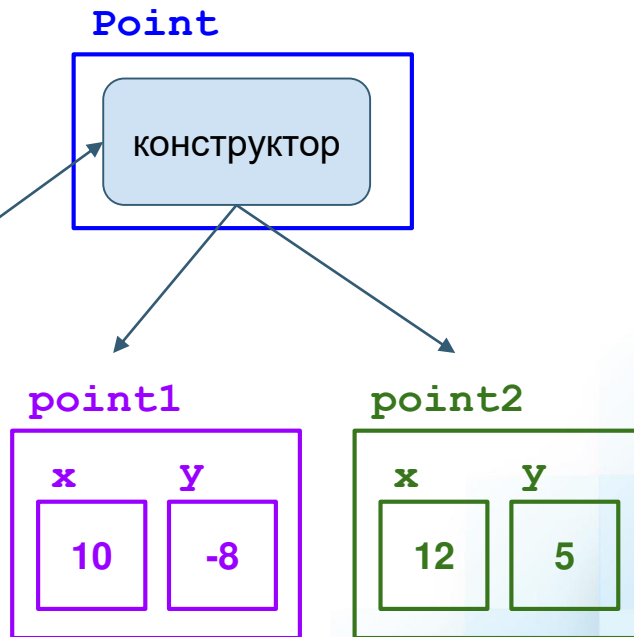


Класс

Конструктор класса

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

```
point1 = Point(10, -8)  
point2 = Point(12, 5)
```



Класс

Пользовательский тип данных

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

point1 = Point(10, -8)
point2 = Point(12, 5)

print(type(point1))  # <class 'Point'>
print(type(point2))  # <class 'Point'>
```



Класс

Пользовательский тип данных

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

point1 = Point(10, -8)
point2 = Point(12, 5)

print(type(point1))  # <class 'Point'>
print(type(point2))  # <class 'Point'>
```

```
class dict:
    ...

my_dict = {}  # my_dict = dict()

class list:
    ...

my_list = []  # my_list = list()
```

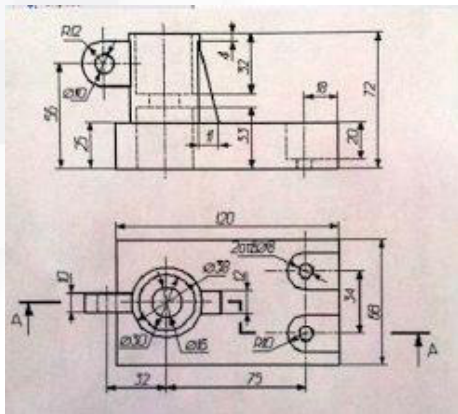


Основы ООП

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса.

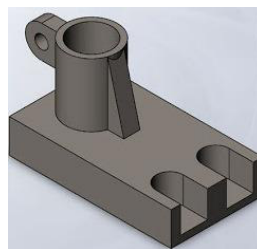
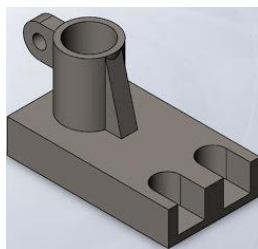
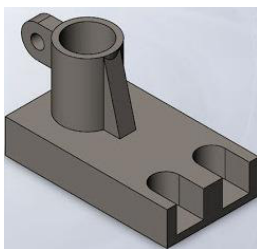
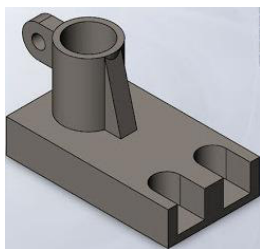


Основы ООП

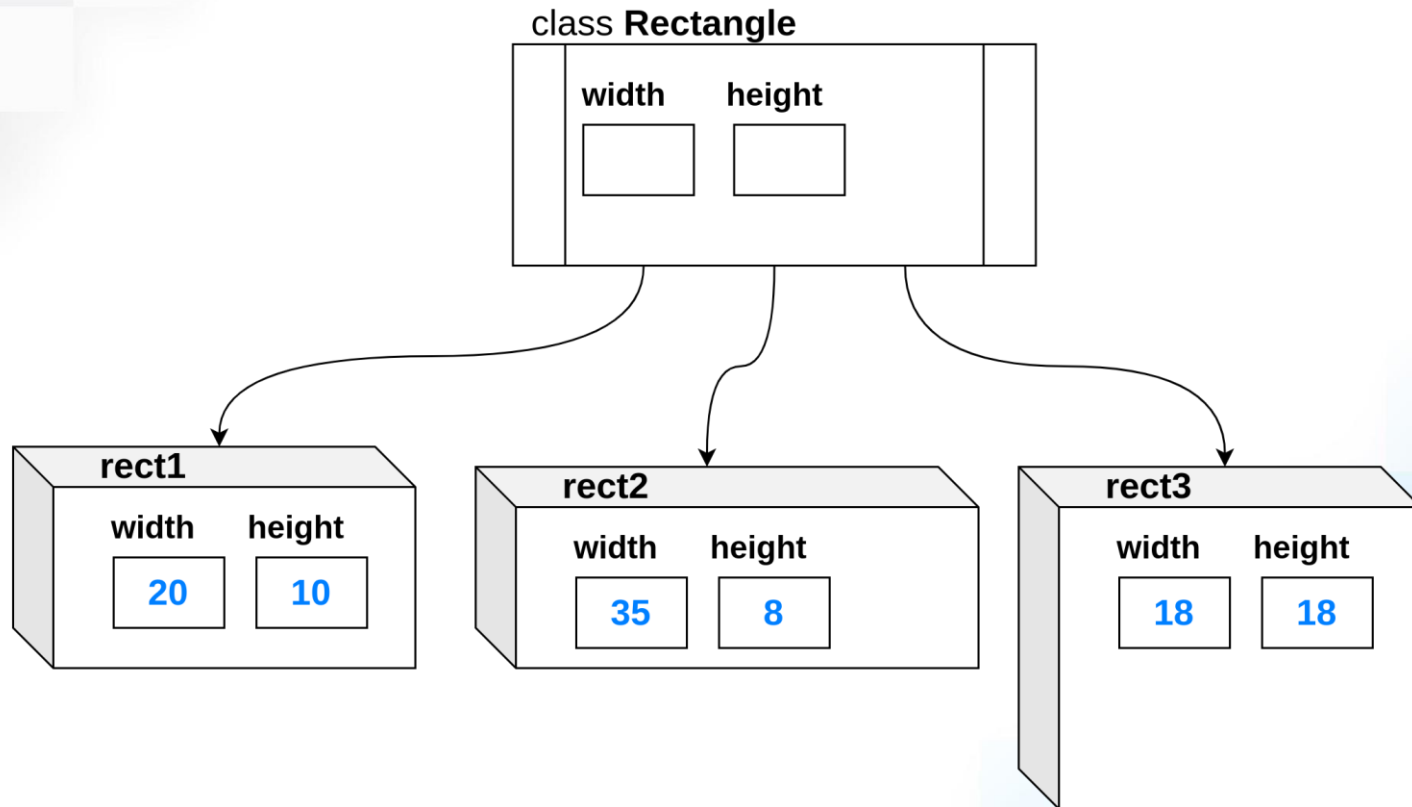


Класс- чертеж для объекта

Объекты - “детали” создаваемые на основании класса (Чертежа)”



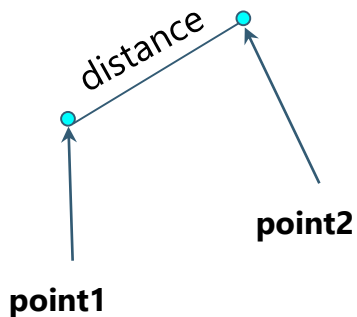
Основы ООП



Мини-практика

План:

- 🕒 Смотрим задание в "**Module-1/practice/points/**"
- 🕒 Выполняем первую задачу **01_distance.py**
- ❓ Отрабатываем новые инструменты
- 👤 Спрашиваем



Класс

Список объектов

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
points = []
```

```
point1 = Point(1, -8)
points.append(point1)
```

```
point2 = Point(6, 5)
points.append(point2)
```

```
print(points)
```



Класс

Список объектов

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
points = []
```

```
point1 = Point(1, -8)
points.append(point1)
```

```
point2 = Point(6, 5)
points.append(point2)
```

```
print(points)
```

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

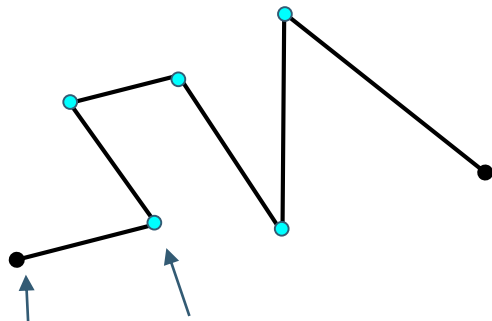
points = [Point(1, -8), Point(6, 5)]
print(points)
```



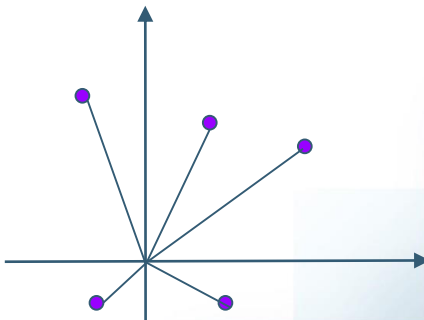
Практика

План:

- 🕒 Смотрим задания в "**Module-1/practice/points/**"
- 🕒 Решаем все оставшиеся задачи...
- 🗋 Отрабатываем новые инструменты
- 👤 Спрашиваем



points = [Point(...), Point(...), ...]



Объект


Методы

Метод - функция объявленная внутри класса и выполняющая операции со свойствами экземпляров данного класса



```
class Rectangle:
    def __init__(self, w, h):
        self.width = w
        self.height = h

    def area(self):
        return self.width * self.height
```



```
rect1 = Rectangle(8, 15)
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()
s2 = rect2.area()
```





```
class Rectangle:
```

```
    def __init__(self, w, h):
```

```
        self.width = w
```

```
        self.height = h
```

```
    def area(self):
```

```
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)
```

```
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()
```

```
s2 = rect2.area()
```



Методы

41



```
class Rectangle:
```

```
    def __init__(self, w, h):
```

```
        self.width = w
```

```
        self.height = h
```

```
    def area(self):
```

```
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)
```

```
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()
```

```
s2 = rect2.area()
```

rect1



Методы

42

class Rectangle:

def __init__(self, w, h):

self.width = w

self.height = h

def area(self):

return self.width*self.height

rect1 = Rectangle(8, 15)

rect2 = Rectangle(12, 14)

s1 = rect1.area()

s2 = rect2.area()

rect1

width

8



Методы

43

class Rectangle:

def **__init__**(self, w, h):

self.width = w

self.height = h

def area(self):

return self.width*self.height

rect1 = Rectangle(8, 15)

rect2 = Rectangle(12, 14)

s1 = rect1.area()

s2 = rect2.area()

rect1

width

8

height

15



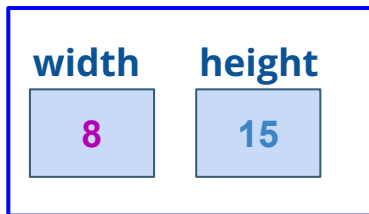
Методы

44

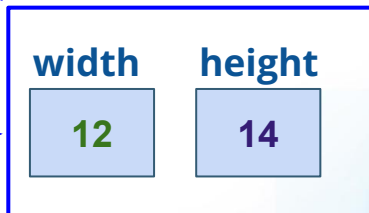
```
class Rectangle:
    def __init__(self, w, h):
        self.width = w
        self.height = h

    def area(self):
        return self.width*self.height
```

rect1



rect2



rect1 = Rectangle(8, 15)
→ rect2 = Rectangle(12, 14)

```
s1 = rect1.area()
s2 = rect2.area()
```



Методы

45

```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```

```
    def area(self):  
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

→ `s1 = rect1.area()`
`s2 = rect2.area()`

rect1

width	height
8	15

rect2

width	height
12	14



Методы

46

```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```



```
    def area(self):  
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()  
s2 = rect2.area()
```

rect1

width	height
8	15

rect2

width	height
12	14



self - ссылка на объект, у которого вызываем метод



Методы

```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```



```
    def area(self):  
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()  
s2 = rect2.area()
```

rect1

width	height
8	15

rect2

width	height
12	14



Методы

48

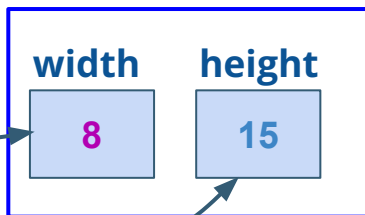
```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```

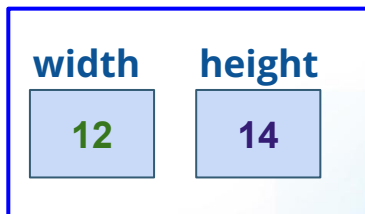


```
    def area(self):  
        return self.width*self.height
```

rect1



rect2



```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()  
s2 = rect2.area()
```



Методы

```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```



```
    def area(self):  
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()  
s2 = rect2.area()
```

8 * 15 → 120

rect1

width	height
8	15

rect2

width	height
12	14



Методы

50

```
class Rectangle:
```

```
    def __init__(self, w, h):  
        self.width = w  
        self.height = h
```

```
    def area(self):  
        return self.width*self.height
```

```
rect1 = Rectangle(8, 15)  
rect2 = Rectangle(12, 14)
```

```
s1 = rect1.area()
```

```
s2 = rect2.area()
```

12 * 14 → 168

rect1

width	height
8	15

rect2

width	height
12	14



```
class Rectangle:
```

```
    def __init__(self, w, h):
```

```
        self.width = w
```

```
        self.height = h
```

```
    def area(self):
```

```
        return self.width*self.height
```

конструктор

свойства
(атрибуты)

метод

```
rect1 = Rectangle(8, 15)
```

объект
(экземпляр класса)

```
s1 = rect1.area()
```



метод - функция внутри класса



ООП

Преимущества и недостатки

- + **Повторное** использование кода;
- + Использование **модульного** подхода в позволяет получить читаемый и гибкий код;
- + Если **ошибка** возникнет в одной части кода, вы можете **исправить** ее **локально**, без необходимости вмешиваться в другие части кода;
- + Инкапсуляция данных вносит **дополнительный уровень безопасности** в разрабатываемую программу;
- Необходимо иметь подробное представление о разрабатываемом программном обеспечении;
- Не каждый аспект программного обеспечения является лучшим решением для реализации в качестве объекта;
- Новые и новые классы, увеличивают размер и сложность программы в геометрической прогрессии;

