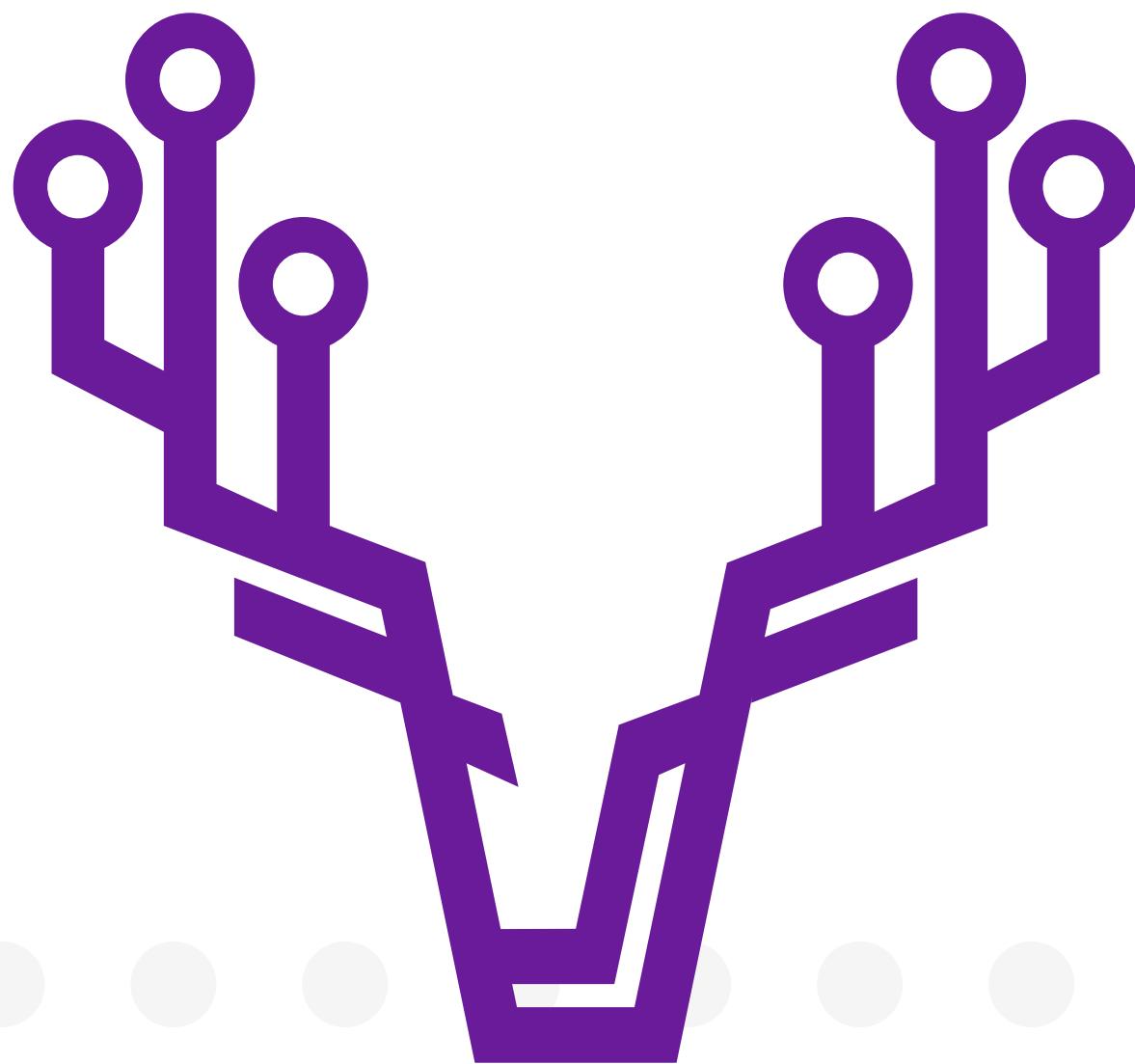


Хорошо слышно и видно?

Ставьте +, если все хорошо
Пишите, если есть проблемы

Вебинар
Объектно-ориентированное
программирование



Y-LAB
UNIVERSITY

Преподаватель



Михаил Вотинов

- Более 8 лет работаю в ИТ
- Занимаюсь развитием отдела Python разработки в качестве тимлида
- Интересуюсь разработкой и реализацией архитектуры высоконагруженных веб-сервисов

Цели вебинара

После занятия вы сможете:

1

Разрабатывать классы



2

Управлять объектами

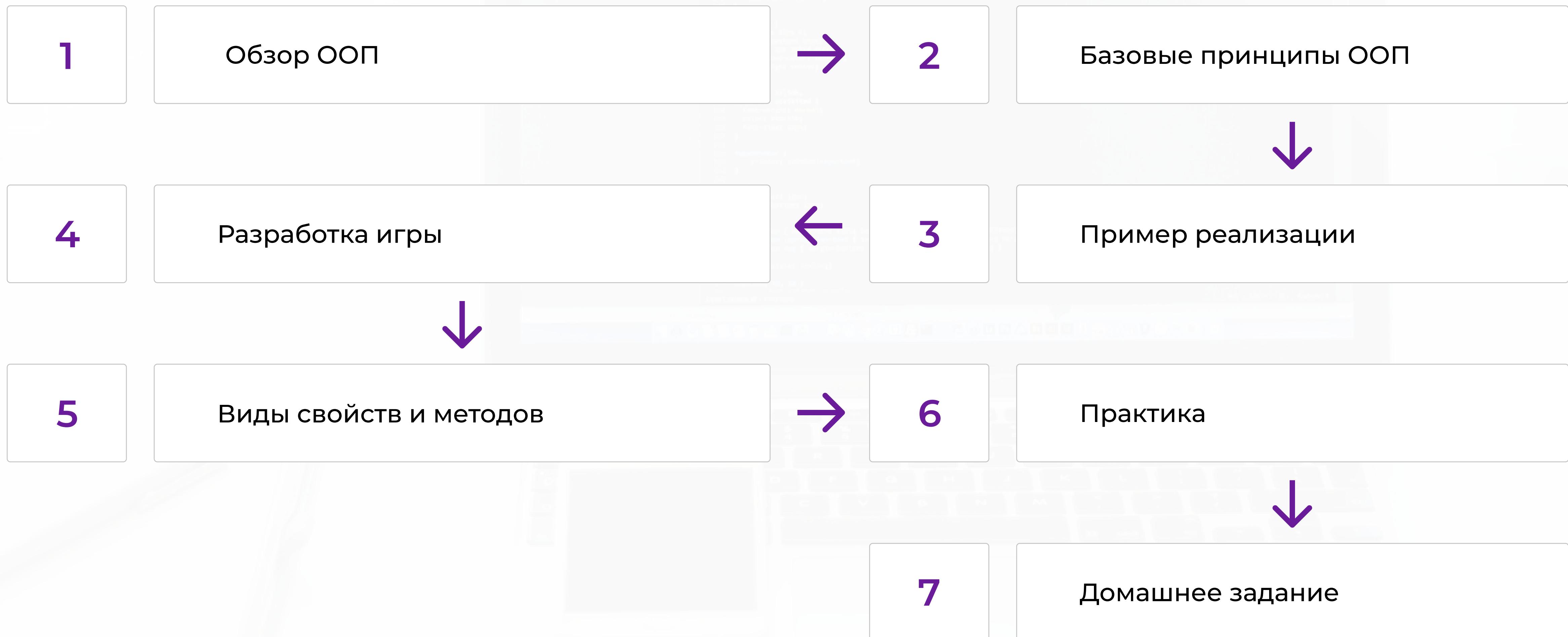


3

Проектировать структуру программы в стиле ООП



Маршрут вебинара



Правила вебинара

- Активно участвуйте
- Задавайте вопросы в чат
- Оффтоп вопросы можем обсуждать после занятия в общем чате

ООП – объектно-ориентированное программирование.

Это парадигма программирования, оперирующая понятиями **класса и объекта**.

Базируется на принципах:

1. Наследование

2. Инкапсуляция

3. Полиморфизм

Чем ООП лучше процедурного подхода? Или не лучше?

Преимущества ООП:

- Подходит для крупных и серьезных программ
- Позволяет создавать расширяемые и тестируемые системы
- Возможность повторного использования реализуемой логики
- Обеспечивает простоту модификации программы
- Естественная и интуитивно понятная декомпозиция программы

Чем ООП лучше процедурного подхода? Или не лучше?

Недостатки ООП:

- Требует опыта и квалификации
- Программы получаются более громоздкие и сложные
- Повышается сложность проектирования программы
- Больше исходного кода – больше мест для возникновения ошибок
- Программа может работать немного медленнее



Форма для выпекания

Выпечка

ООП подходит, когда программа оперирует с некоторыми понятиями из реального мира или имеющими определенную концепцию.

Классы используются, когда можно описать это понятие некоторыми свойствами и действиями, которыми оно обладает.

Если понятие можно охарактеризовать наличием имени или **наименования**, **характеристиками** и возможными **действиями**, то такое понятие можно представить в виде **класса**.

Класс
Наименование
Характеристики
Действия

Python

Классы и объекты



Класс
Наименование
Характеристики
Действия

- **имя** класса
- **свойства** (переменные внутри класса)
- **методы** (функции внутри класса)

Имя класса
Свойства
Методы
Автомобиль
Свойство: Модель
Свойство: Цвет
Свойство: Год выпуска
Свойство: Пробег
Свойство: Фото
Метод: Начать движение вперед
Метод: Тормозить
Метод: Начать движение назад

Класс

Автомобиль
Свойство: Модель
Свойство: Цвет
Свойство: Год выпуска
Свойство: Пробег
Свойство: Фото
Метод: Начать движение вперед
Метод: Тормозить
Метод: Начать движение назад

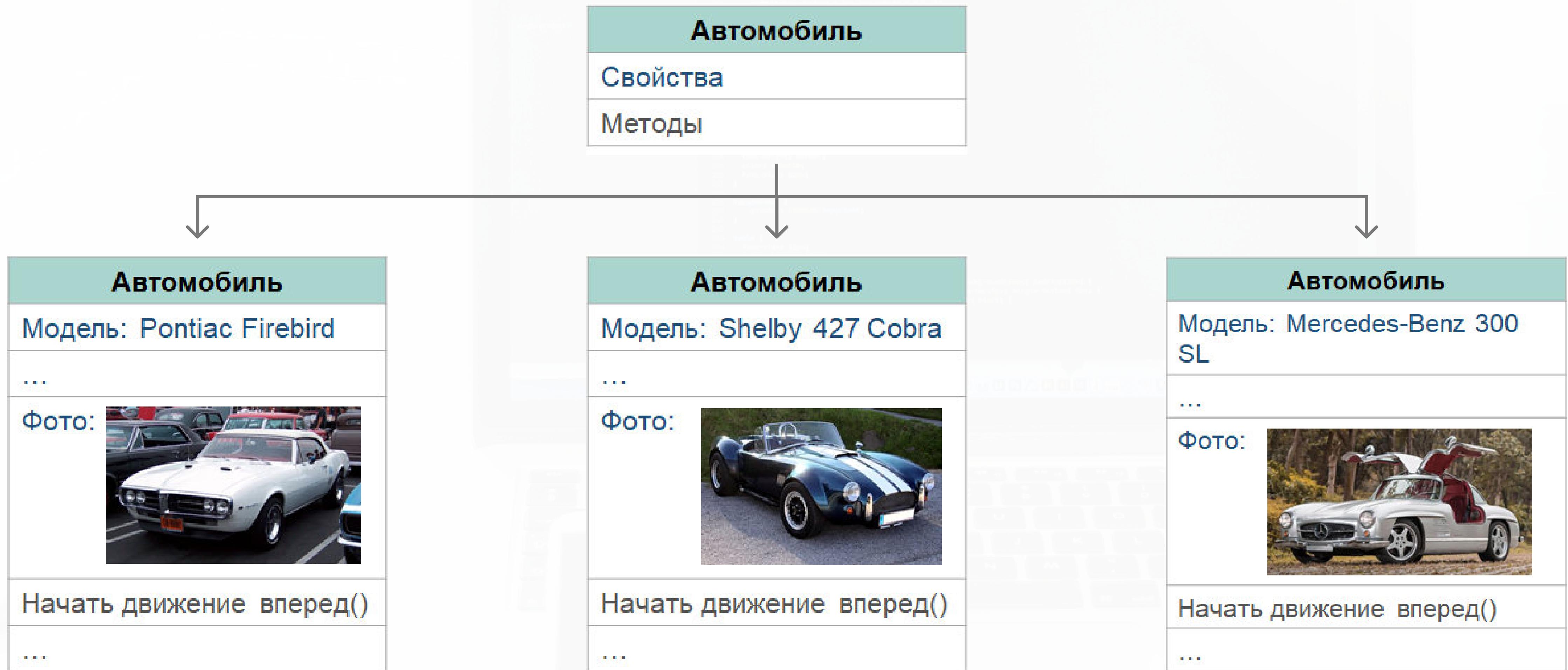


Объект

Автомобиль
Модель: Pontiac Firebird
Цвет: Белый
Год выпуска: 1967
Пробег: 35647 км
Фото: 
Начать движение вперед()
Тормозить()
Начать движение назад()

Python

Классы и объекты



Как спроектировать класс?

Думайте о вещах, которые объект **знает и делает**.

Что объект знает?

Что объект делает?

Корзина покупок

Содержимое корзины

Добавить в корзину()
Удалить из корзины()
Оформить покупку()

Будильник

Время сигнала
Режим сигнала

Назначить время()
Получить время()
Выключить()

Кнопка

Надпись
Цвет

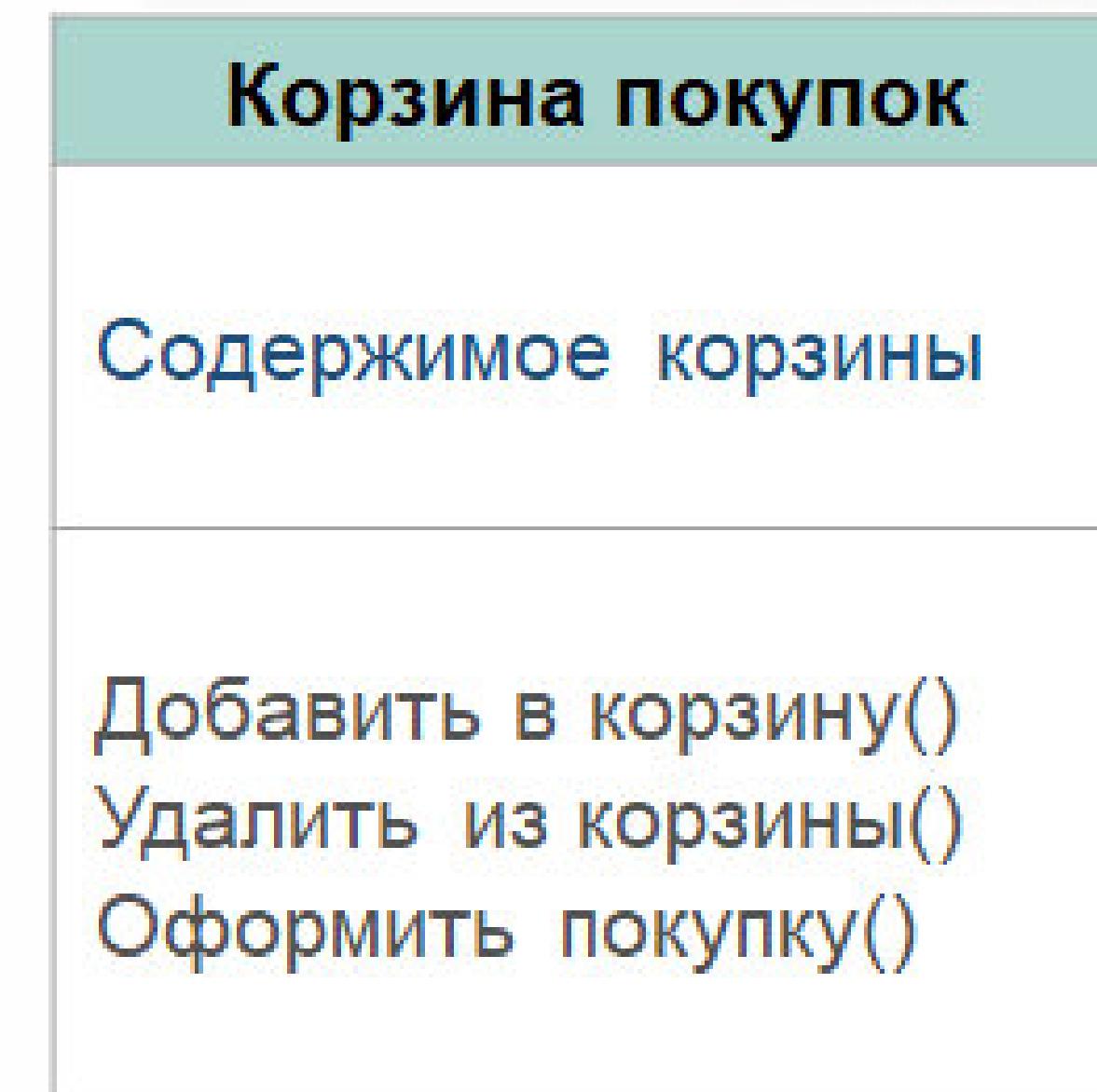
Назначить цвет()
Назначить надпись()
Обработать нажатие()

Знания объекта о себе называются **переменными** экземпляра класса (объекта).

Действия, которые может совершать объект называются **методами**.

Переменные
(состояние объекта)

Методы
(поведение объекта)



← То, что объект знает

← То, что объект делает

Разница между классом и объектом:

Класс – шаблон для создания объектов

Объект – конкретный экземпляр класса

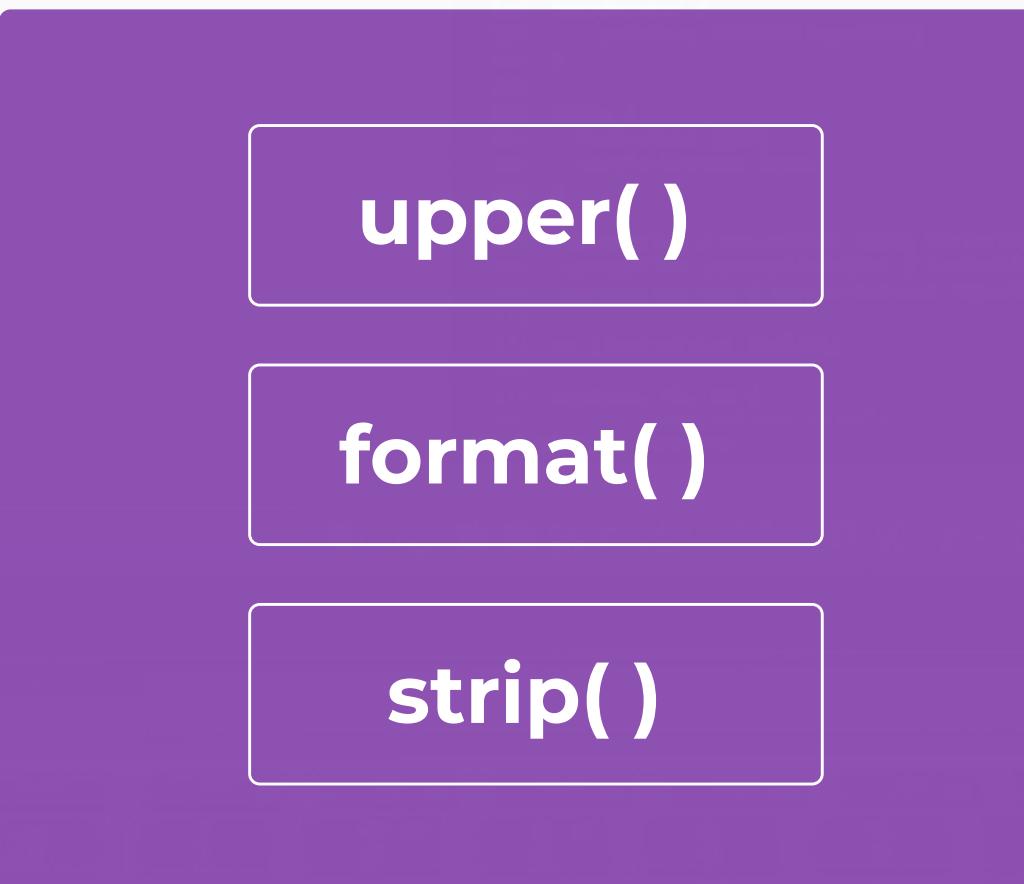
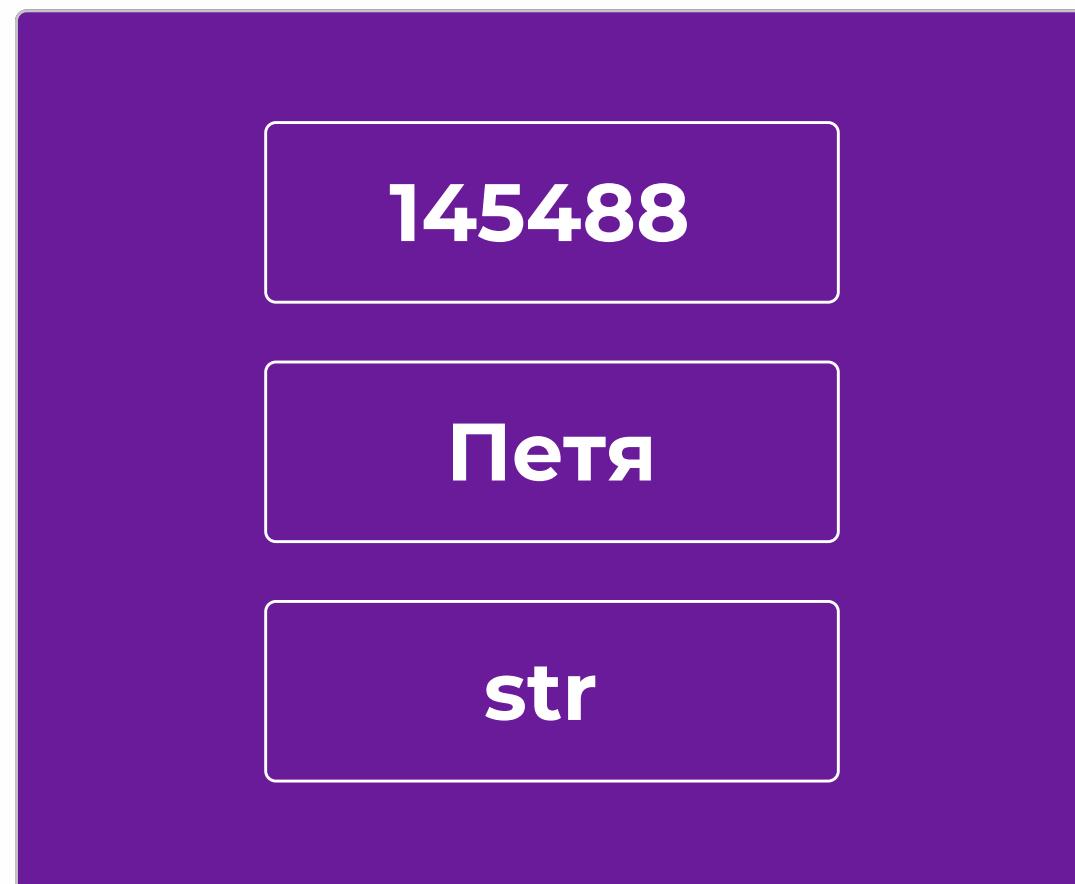
Класс описывает содержимое (состояние) и поведение (методы для работы с содержимым).

Каждый из создаваемых объектов может иметь собственные значения состояния (переменных), с которыми он работает в методах.

Например, корзина для покупок, может содержать разный набор продуктов в ней для разных покупателей.

Python оперирует с объектами. Они хранятся в памяти. Как они устроены?

```
my_name = "Петя"
```



```
my_name.upper()
```

```
my_name.format()
```

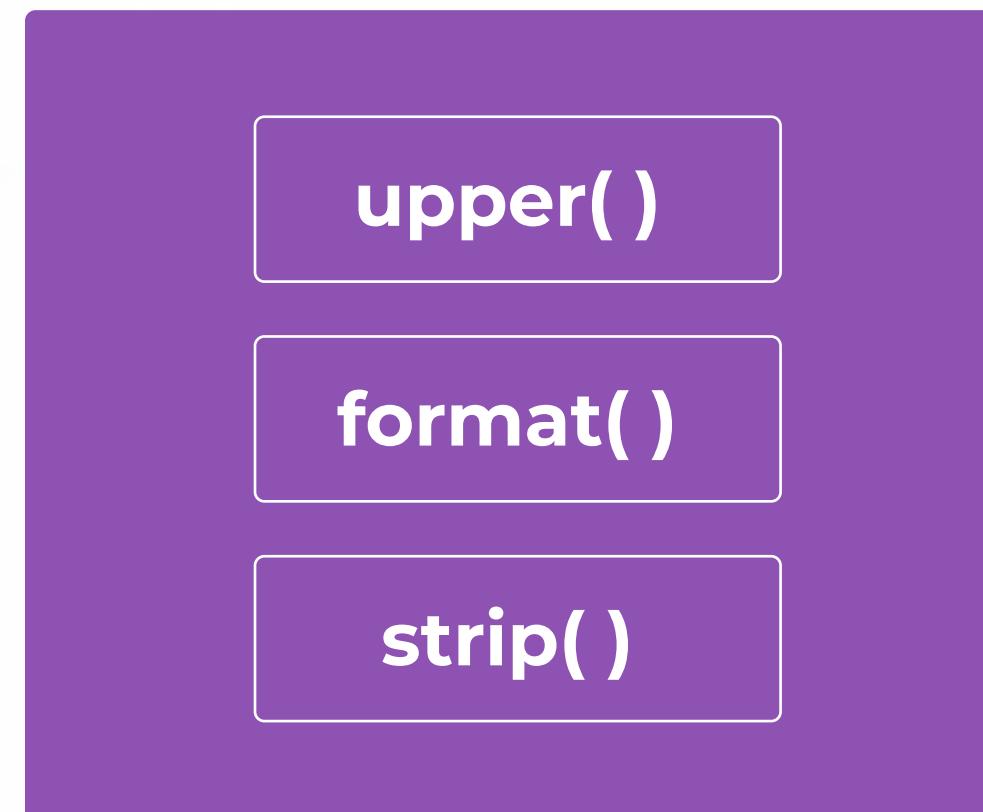
```
my_name.strip()
```

Шаблон
(тип)

Конкретный экземпляр
(объект)

Python

Классы и объекты



Объект наследует свойства и методы от класса и оперирует ими, сохраняя состояние внутри себя

Шаблон – **класс**

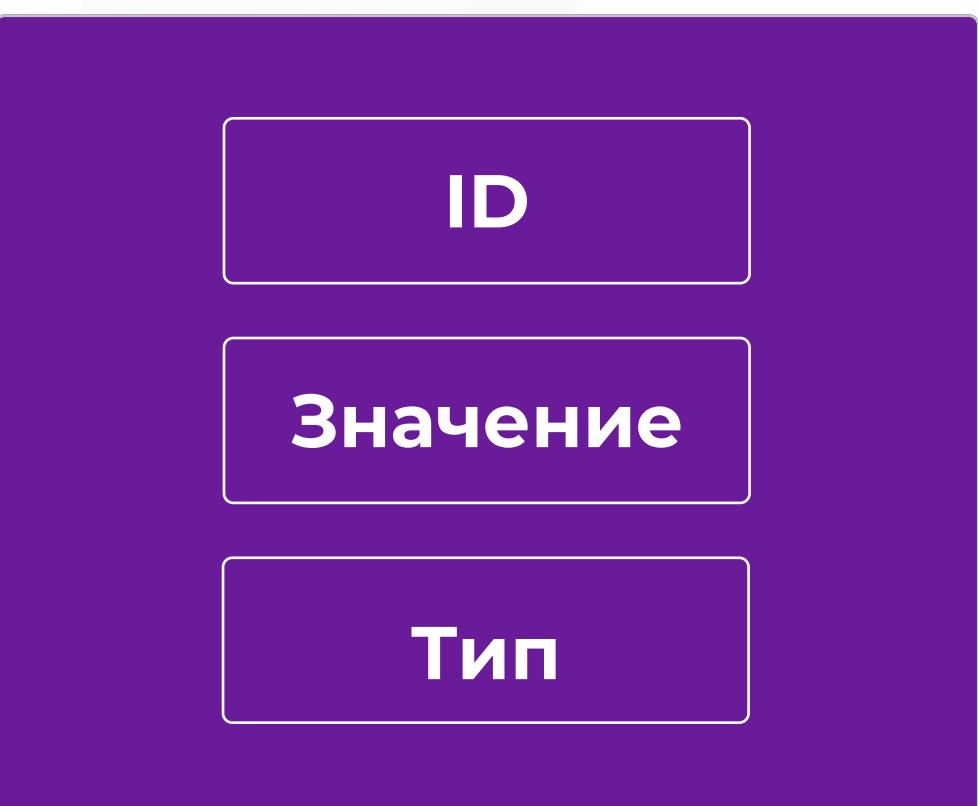
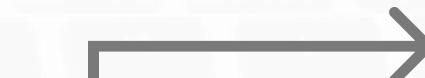
Класс определяет свойства и методы

Свойства – какие-либо параметры (переменные внутри класса)

Методы – набор функций, выполняющих заданные алгоритмы работы



Экземпляр класса – **объект**



Экземпляр класса – **объект**

class – ключевое слово для объявления класса

Структура класса:

- название
- свойства (переменные)
- методы (функции)

class название: } обязательно – двоеточие

свойства

методы () }

4 пробела

отсутствие символов
окончания строки

```
class ShoppingCart:  
    contents = []  
  
    def add(self, item):  
        self.contents.append(item)  
  
    def remove(self, item):  
        self.contents.remove(item)  
  
    def checkout(self):  
        print('Оформление покупок: ')  
        for index, item in enumerate(  
            self.contents, start=1  
        ):  
            print(f'{index}. {item}.')
```

Python

Классы и объекты. Пример реализации



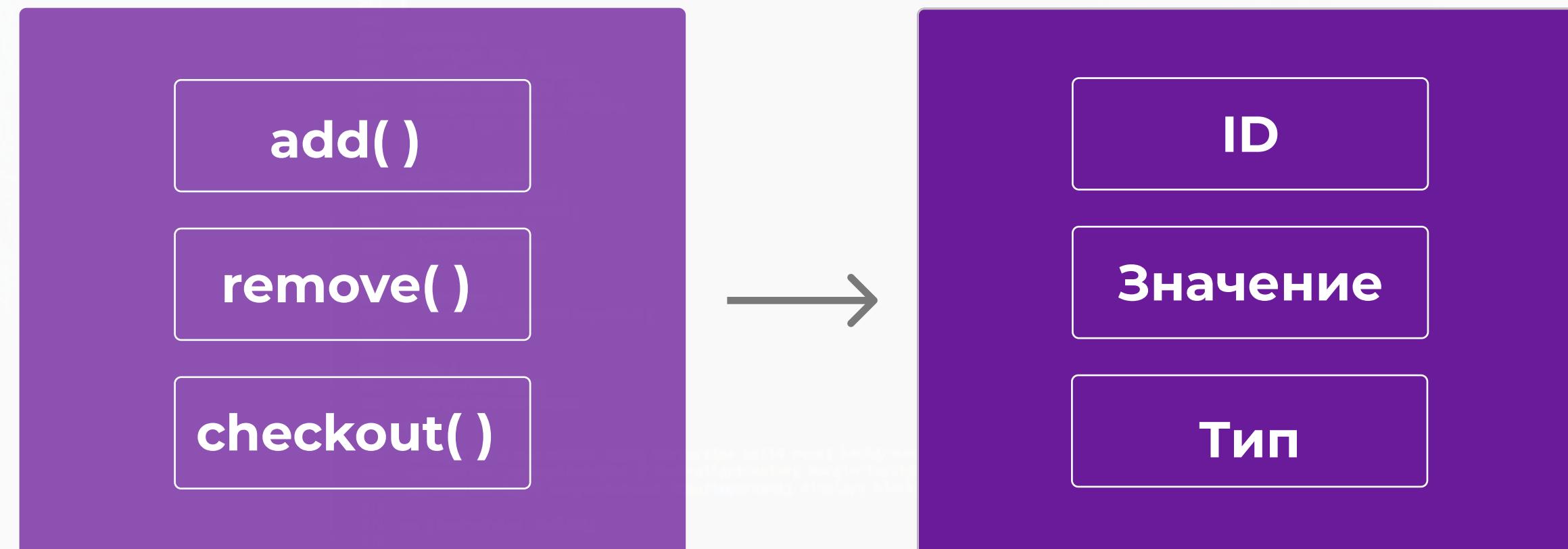
```
class ShoppingCart:  
    contents = []  
  
    def add(self, item):  
        self.contents.append(item)  
  
    def remove(self, item):  
        self.contents.remove(item)  
  
    def checkout(self):  
        print('Оформление покупок: ')  
        for index, item in enumerate(  
            self.contents, start=1  
        ):  
            print(f'{index}. {item}.')
```

```
cart = ShoppingCart()  
cart.add('Хлеб')  
cart.add('Молоко')  
cart.add('Колбаса')  
cart.add('Каша')  
  
cart.remove('Колбаса')  
  
cart.checkout()
```

Python

Классы и объекты. Пример реализации

```
cart = ShoppingCart()  
cart.add('Хлеб')  
cart.add('Молоко')  
cart.add('Колбаса')  
cart.add('Каша')  
  
cart.remove('Колбаса')  
  
cart.checkout()
```



```
# Оформление покупок:  
# 1. Хлеб.  
# 2. Молоко.  
# 3. Каша.
```

Класс «ShoppingCart» является типом для создаваемых объектов.
Объект «cart» содержит методы и свойства класса и оперирует ими.

```
cart = ShoppingCart()
```

Переменная **cart** содержит ссылку на объект. Свойства класса также являются переменными и они могут ссылаться на другие объекты.

Это значит, что переменные одного объекта могут ссылаться на другие объекты и работать с его методами и свойствами.



Объект пользователя

Игра – «Угадай число»

Необходимо угадать число. Число генерируется случайным образом от 1 до 10.



Python

Разработка игры



Игрок
Имя Предполагаемое число
Назначить имя() Угадать()

```
from random import randint

class Player:
    name = None
    number = None

    def set_name(self, name):
        self.name = name

    def guess(self):
        self.number = randint(1, 10)
        print(f'Возможно это число - {self.number}.')
```

Игра
Игрок
Начать игру()

```
from random import randint

class Game:
    player = None
    number = None

    def start(self):
        self.player = Player()
        self.player.set_name('Петя')
        self.number = randint(1, 10)
        print(f'Загадано число - {self.number}.')
        self.player.guess()
        print(f'Игрок {self.player.name} предполагает, что это число - {self.player.number}.')
        if self.number == self.player.number:
            print('Поздравляем! Число угадано.')
        else:
            print('Число не угадано. Попробуйте еще раз.'))
```

Запуск игры:

```
game = Game()  
game.start()
```

```
# Загадано число – 2.  
# Возможно это число – 2.  
# Игрок Петя предполагает, что это число – 2.  
# Поздравляем! Число угадано.
```

```
# Загадано число – 10.  
# Возможно это число – 5.  
# Игрок Петя предполагает, что это число – 5.  
# Число не угадано. Попробуйте еще раз.
```

Конструирование объекта.

`__init__` – метод конструктора объекта.

Конструктор задает значения свойствам объекта при инициализации объекта.

```
def __init__(self, argument, ...):  
    ссылка на инициализируемый объект  
    аргументы являются опциональными  
тело метода
```

```
class MyClass:  
    def __init__(self, argument):  
        self.value = argument  
        self.custom = 'Значение'  
        self.from_method = self.get_value()  
  
    def get_value(self):  
        return 'Значение из метода'  
  
my_object = MyClass('Мое значение')  
print(my_object.value)  
# Мое значение  
print(my_object.custom)  
# Значение  
print(my_object.from_method)  
# Значение из метода
```

Обновление класса игрока

Добавлен конструктор

```
from random import randint

class Player:

    def __init__(self, name):
        self.name = name
        self.number = None

    def guess(self):
        self.number = randint(1, 10)
        print(f'Возможно это число - {self.number}.')
```

Обновление класса игры

добавлен конструктор

объект игрока формируется
вне класса

```
from random import randint

class Game:
    def __init__(self, player):
        self.player = player
        self.number = None

    def start(self):
        self.number = randint(1, 10)
        print(f'Загадано число - {self.number}.')
        self.player.guess()
        print(f'Игрок {self.player.name} предполагает, что это число - {self.player.number}.')
        if self.number == self.player.number:
            print('Поздравляем! Число угадано.')
        else:
            print('Число не угадано. Попробуйте еще раз.')

    def check(self):
        if self.number == self.player.number:
            return True
        else:
            return False
```

Обновление клиентского кода

создание объекта игрока
создание объекта игры
с передачей игрока

```
{     player = Player('Петя')
      game = Game(player)
      game.start()

      # Загадано число - 2.
      # Возможно это число - 4.
      # Игрок Петя предполагает, что это число - 4.
      # Число не угадано. Попробуйте еще раз.
```

Можно обращаться к объекту игрока через объект игры

объект игры имеет переменную *player*,
которая ссылается на объект игрока

```
{     print(game.player.name)    # Петя
      print(game.player.number)   # 4
```

Виды свойств (атрибутов, переменных):

- **Свойства класса** определяются для класса и действуют на уровне класса.
Размещаются в определении класса. Общие для экземпляров (объектов) данного класса.
- **Свойства объекта** определяются для объекта и действуют на уровне объекта.
Размещаются в конструкторе объекта. Такие свойства уникальны для экземпляра (объекта) и их состояние хранится в нем.

При обращении ко свойству объекта поиск свойства происходит, начиная с самого объекта, а затем переходит к его классу.

Свойства класса определяются для класса и действуют на уровне класса.

Размещаются в определении класса. Общие для экземпляров (объектов) данного класса.

```
from random import randint

class Game:
    добавлено свойство
    класса
    {
        players_count = 0

        def __init__(self, player):
            self.player = player
            self.number = None

        обращение к нему по
        имени класса
        {
            Game.players_count += 1
            def start(self):
                ...
    }
```

Свойства класса определяются для класса и действуют на уровне класса.

Размещаются в определении класса. Общие для экземпляров (объектов) данного класса.

объекты передаются
напрямую

```
{ Game(Player('Вася')).start()  
  Game(Player('Петя')).start()  
  Game(Player('Федя')).start()
```

вывод значения
свойства класса

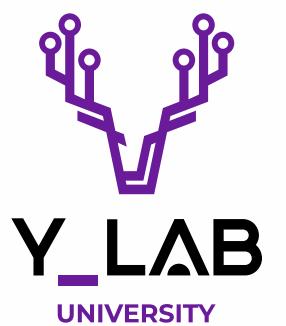
```
{ print(Game.players_count)  
  # 3
```

Виды методов (функций):

- **Методы экземпляра** (объекта) действуют на уровне объекта.
Имеют ссылку на свой объект в качестве первого аргумента метода – `self`. Это позволяет получать доступ к свойствам и другим методам этого объекта, изменять его состояние.
- **Методы класса** действуют на уровне класса.
Имеют ссылку на класс в качестве первого аргумента метода – `cls`. Это позволяет модифицировать состояние класса, общее для всех экземпляров (объектов) этого класса.
- **Статический метод** не имеет ссылок ни на класс ни на объект.
Не может модифицировать состояние ни класса ни объекта. Используются, как вспомогательный метод без ссылок на объект и класс.

Python

Классы и объекты. Методы



Методы экземпляра

Первый аргумент – ссылка на экземпляр этого класса (объект, созданный от этого класса).

Обычно этот аргумент называется как **self**.

```
def method_name(self,  
                argument, ...):
```

ссылка на
инициализируемый
объект

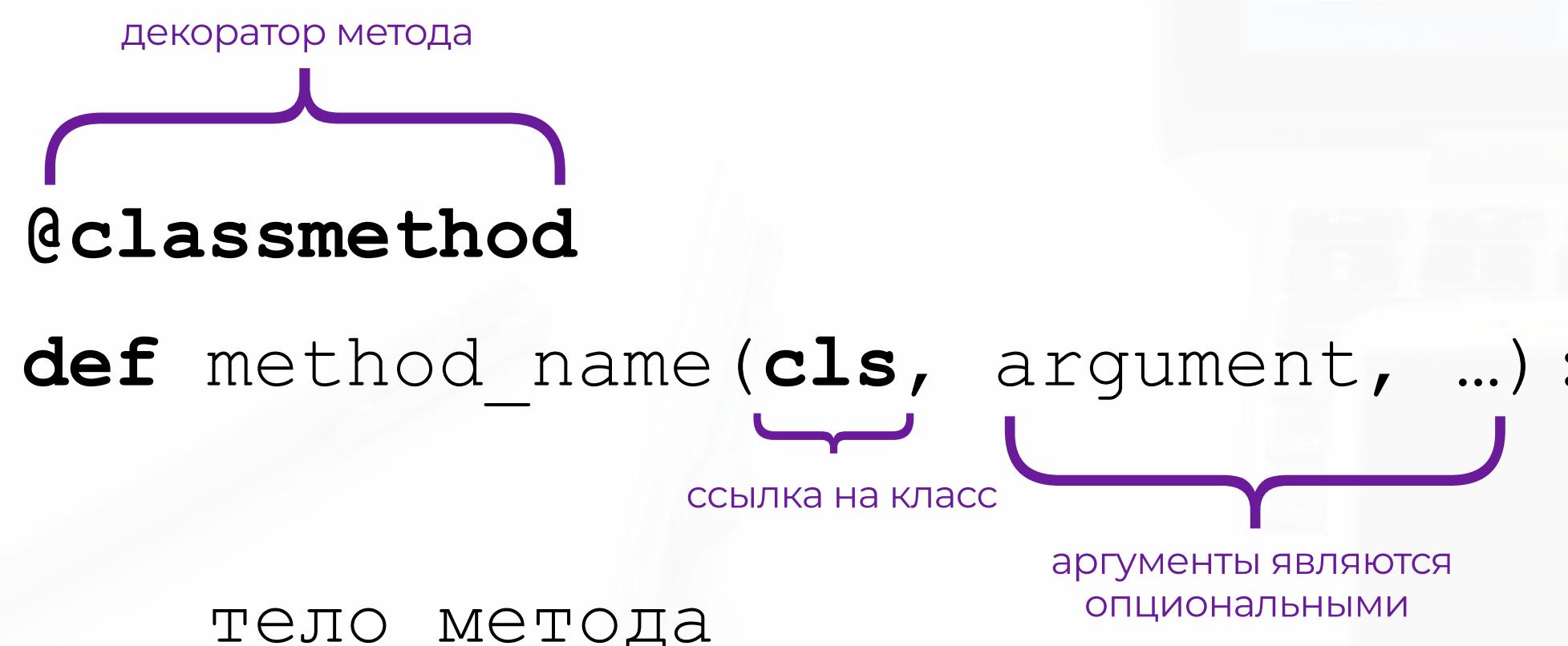
аргументы являются
опциональными

```
class MyClass:  
    def my_method(self):  
        return 'Метод экземпляра', self  
  
my_object = MyClass()  
print(my_object.my_method())  
# ('Метод экземпляра',  
# <MyClass object at 0x10f255dc0>  
# )
```

Методы класса

Первый аргумент – ссылка на класс, от которого создается экземпляр (объект).

Обычно этот аргумент именуется как **cls**.



```
decorator method
@classmethod
def method_name(cls, argument, ...):
    # ссылка на класс
    # аргументы являются
    # опциональными
    # тело метода
```

декоратор метода

@classmethod

def method_name(**cls**, argument, ...):

ссылка на класс

аргументы являются
опциональными

тело метода

```
class MyClass:
    @classmethod
    def my_method(cls):
        return 'Метод класса', cls

my_object = MyClass()
print(my_object.my_method())
# (
#     'Метод класса',
#     <class MyClass>
# )
```

Статические методы

Не принимают специальных аргументов для ссылки на класс или объект.

Могут принимать опциональные параметры.

декоратор метода
@staticmethod

def method_name(argument, ...):

аргументы являются
опциональными

тело метода

```
class MyClass:  
  
    @staticmethod  
    def my_method():  
        return 'Статический метод'
```

```
my_object = MyClass()  
print(my_object.my_method())  
# Статический метод
```

Методы класса и статические методы можно вызвать от самого класса без создания его экземпляра. **Методы объекта** при этом недоступны для вызова, так как они принадлежат экземпляру.

При этом возможно вызывать **метод класса** или **статический метод** от экземпляра.

Python

Классы и объекты. Методы. Пример реализации



```
class MyClass:

    def method(self):
        return 'Метод экземпляра', self

    @classmethod
    def classmethod(cls):
        return 'Метод класса', cls

    @staticmethod
    def staticmethod():
        return 'Статический метод'
```

```
print(MyClass.classmethod())
# (
#     'Метод класса',
#     <class MyClass>
# )

print(MyClass.staticmethod())
# Статический метод

print(MyClass.method())
# TypeError: method() missing 1 required
# positional argument: 'self'
```

Python

Классы и объекты. Методы. Пример реализации



```
class MyClass:

    def method(self):
        return 'Метод экземпляра', self

    @classmethod
    def classmethod(cls):
        return 'Метод класса', cls

    @staticmethod
    def staticmethod():
        return 'Статический метод'
```

```
my_object = MyClass()

print(my_object.method())
# (
#     'Метод экземпляра',
#     <MyClass object at 0x10f255dc0>
# )

print(my_object.classmethod())
# (
#     'Метод класса',
#     <class MyClass>
# )

print(my_object.staticmethod())
# Статический метод
```

ООП базируется на принципах:

1. Наследование

2. Инкапсуляция

3. Полиморфизм

Наследование позволяет создавать родительский класс для других, более специфичных классов. Когда один класс наследует другой, это значит, что дочерний класс наследует родительский, расширяя его функциональность. Таким образом создается иерархия классов.

Инкапсуляция защищает переменные экземпляра от прямого доступа для их модификации.

Полиморфизм позволяет для единого интерфейса иметь различную реализацию. Это значит, что можно переопределять унаследованные методы в дочернем классе и задавать им другую реализацию.



```
class Shape:  
    title = 'Фигура'  
  
    def area(self):  
        pass  
  
    def perimeter(self):  
        pass
```

```
class Square(Shape):  
    title = 'Квадрат'  
  
    def __init__(self, x):  
        super().__init__()  
        self.x = x  
  
    def area(self):  
        return self.x ** 2  
  
    def perimeter(self):  
        return 4 * self.x
```

```
class Cube(Square):  
    title = 'Куб'  
  
    def area(self):  
        return 6 * self.x ** 2  
  
    def perimeter(self):  
        return 12 * self.x
```

```
class Rectangle(Square):  
    title = 'Прямоугольник'  
  
    def __init__(self, x, y):  
        super().__init__(x)  
        self.y = y  
  
    def area(self):  
        return self.x * self.y  
  
    def perimeter(self):  
        return 2 * (self.x + self.y)
```

Для защиты доступа к свойствам и методам используется соглашение:

- Без подчеркивания – публичный доступ (public)
- Одно подчеркивание – тип доступа на уровне класса (protected)
- Двойное подчеркивание – тип доступа на уровне объекта (private)

Python не ограничивает доступ. Это соглашение.

Синтаксис Python не предусматривает ключевые слова public, protected и private.

```
class Shape:  
    _title = 'Фигура'  
  
    def area(self):  
        pass  
  
    def perimeter(self):  
        pass
```

```
class Square(Shape):  
    _title = 'Квадрат'  
  
    def __init__(self, x):  
        super().__init__()  
        self.__x = x  
  
    def area(self):  
        return self.__x ** 2  
  
    def perimeter(self):  
        return 4 * self.__x
```

```
square = Square(10)  
print(square._title)  
# Квадрат  
  
square = Square(10)  
print(square.__x)  
# AttributeError: 'Square' object has no attribute '__x'  
  
square = Square(10)  
print(square._Square__x)  
# 10
```

```
class Shape:  
    title = 'Фигура'  
  
    def area(self):  
        pass  
  
    def perimeter(self):  
        pass
```

```
class Square(Shape):  
    title = 'Квадрат'  
  
    def __init__(self, x):  
        super().__init__()  
        self.x = x  
  
    def area(self):  
        return self.x ** 2  
  
    def perimeter(self):  
        return 4 * self.x
```

```
class Cube(Square):  
    title = 'Куб'  
  
    def area(self):  
        return 6 * self.x ** 2  
  
    def perimeter(self):  
        return 12 * self.x
```

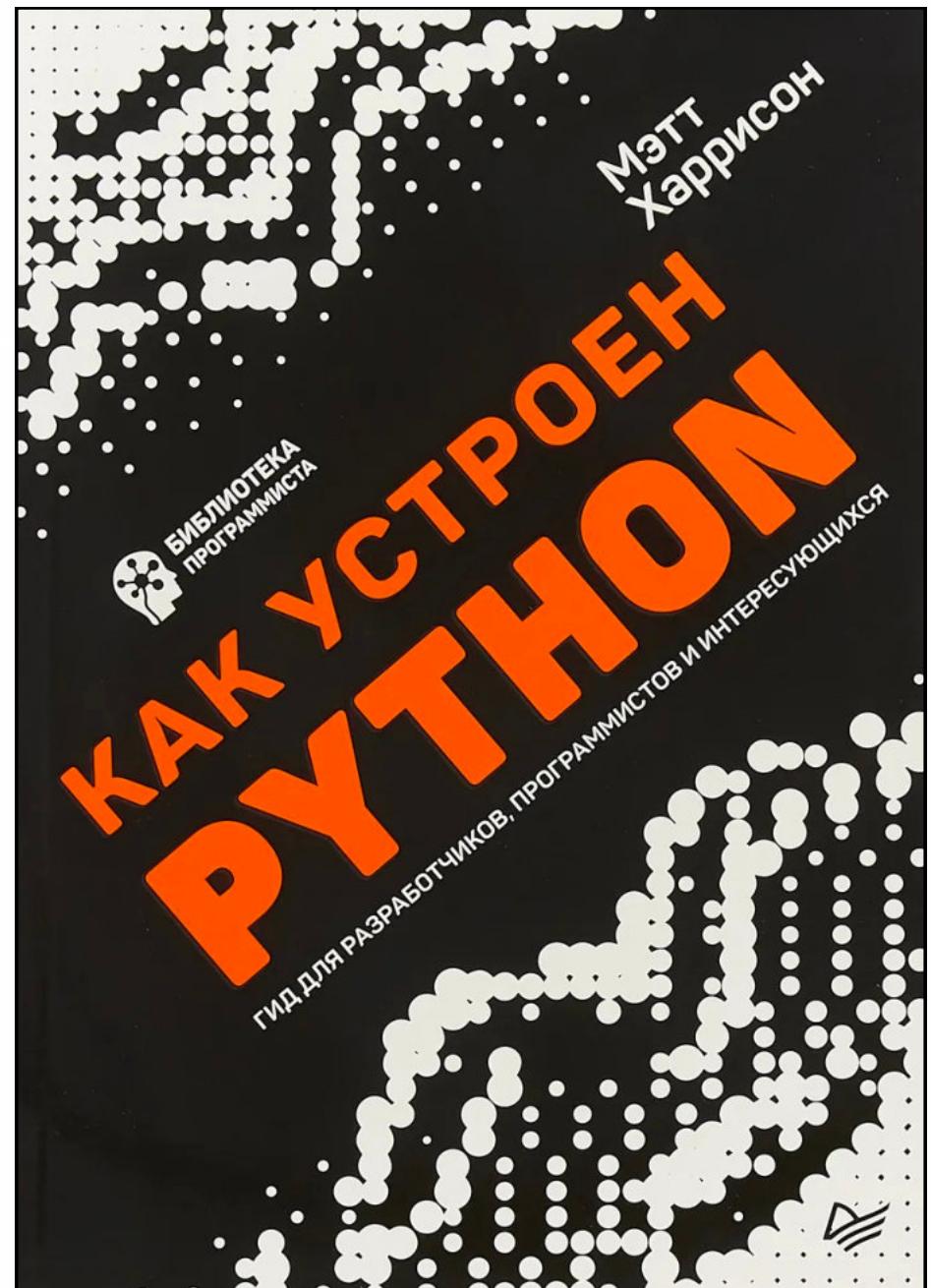
```
class Rectangle(Square):  
    title = 'Прямоугольник'  
  
    def __init__(self, x, y):  
        super().__init__(x)  
        self.y = y  
  
    def area(self):  
        return self.x * self.y  
  
    def perimeter(self):  
        return 2 * (self.x + self.y)
```

- Создание классов
- Определение свойств класса
- Определение свойств экземпляра
- Определение методов экземпляра
- Определение методов класса
- Определение статических методов
- Инициализация экземпляра
- Работа с объектом

[Ссылка на практику](#)

Python

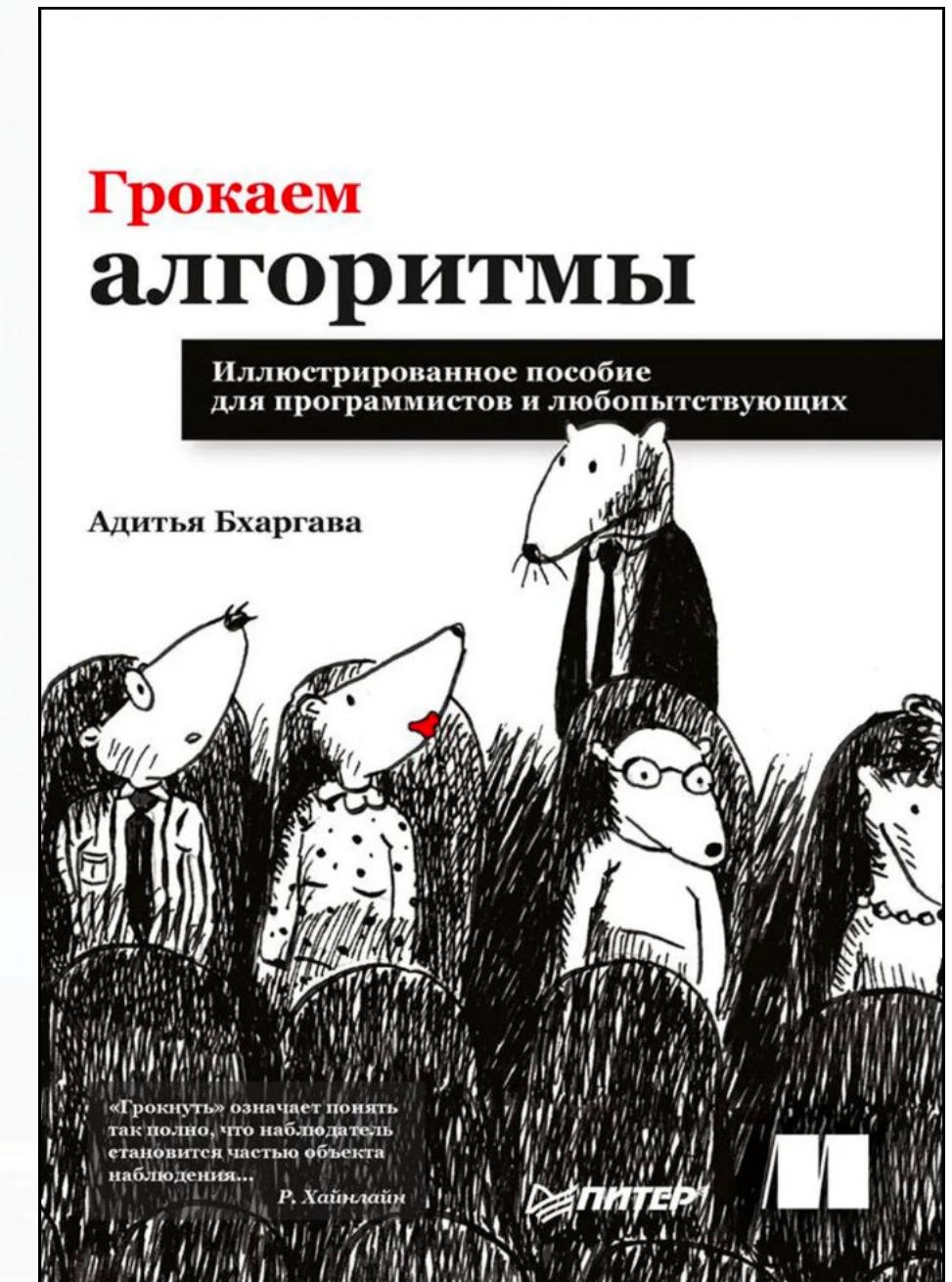
Список материалов для изучения



Мэтт Харрисон: Как устроен Python.
Гид для разработчиков,
программистов и интересующихся



Дэн Бейдер: Чистый Python.
Тонкости программирования
для профи



Бхаргава Адитья: Грекаем
алгоритмы. Иллюстрированное
пособие для программистов
и любопытствующих

При помощи ООП спроектировать и реализовать **геометрический калькулятор** для вычислений, производимых над фигурами. Калькулятор должен поддерживать вычисления для плоских и объемных фигур.

Плоские фигуры: круг, квадрат, прямоугольник, треугольник, трапеция, ромб.

Объемные фигуры: сфера, куб, параллелепипед, пирамида, цилиндр, конус.

Реализовать как минимум один общий метод вычисления для всех фигур и как минимум один специфичный для определенных фигур. Например, площадь – общий метод для всех фигур, медиана – специфичный метод для ряда фигур.

Необходимо: реализовать графический интерфейс для возможностей взаимодействия пользователя с программой и визуализации фигур (с учетом введенных параметров фигуры).

При реализации использовать все виды методов: статический, метод класса и экземпляра.

- Достигли ли целей вебинара?
- Что запомнилось / понравилось?

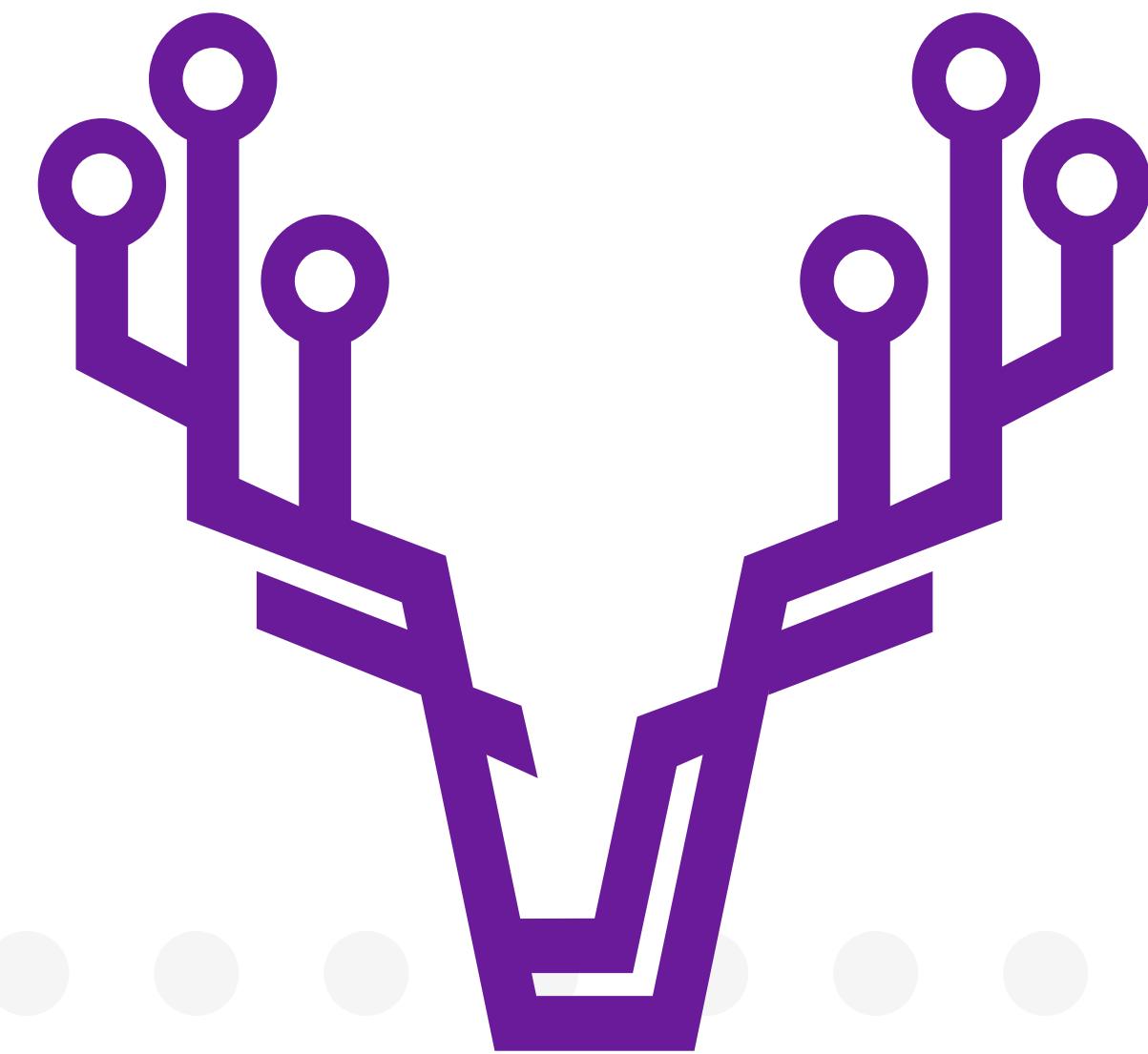
Обратная связь

Пожалуйста, пройдите опрос о занятии.
Нам очень важно ваше мнение!

Опрос о занятии



Спасибо за
внимание!



Y-LAB
UNIVERSITY