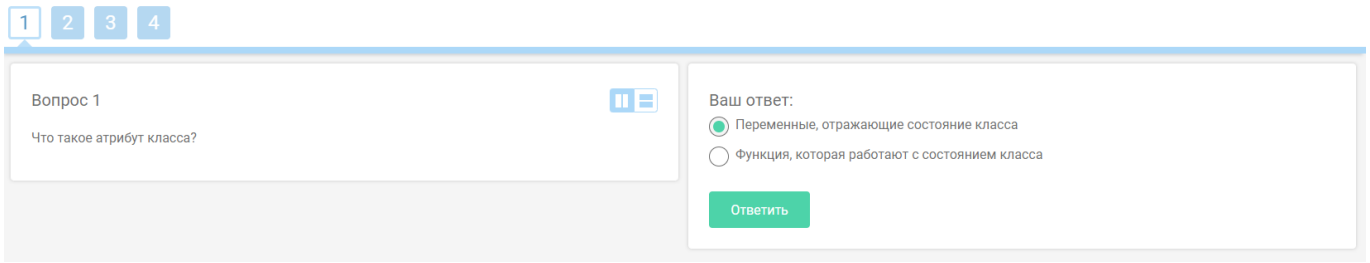
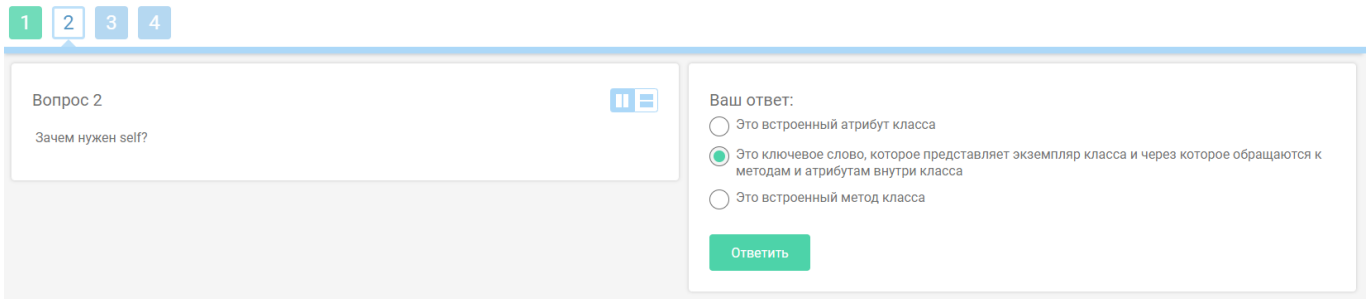
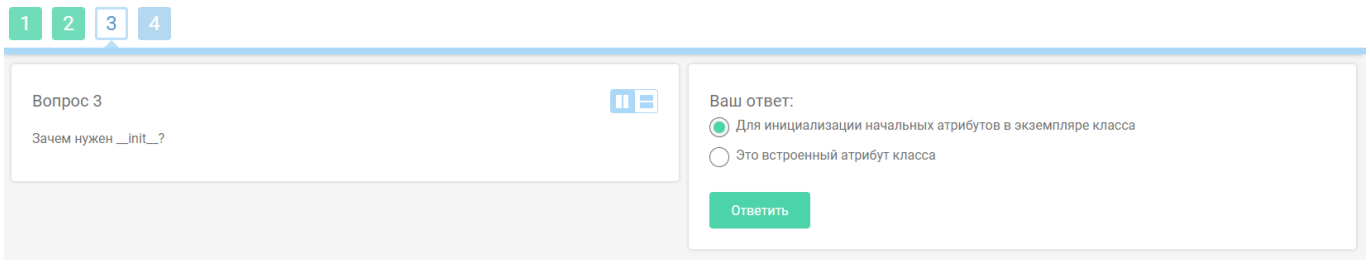
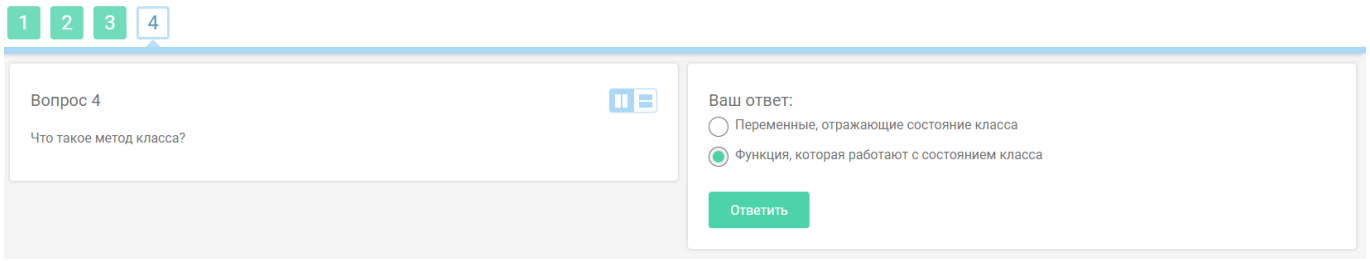
Тест по лекции 1

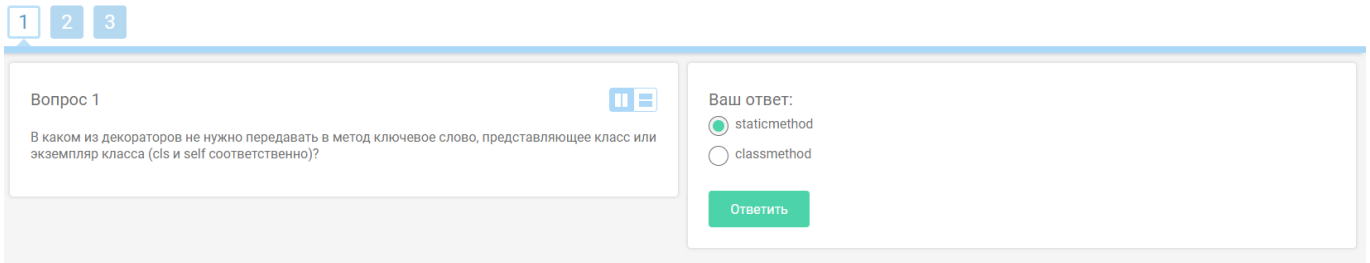


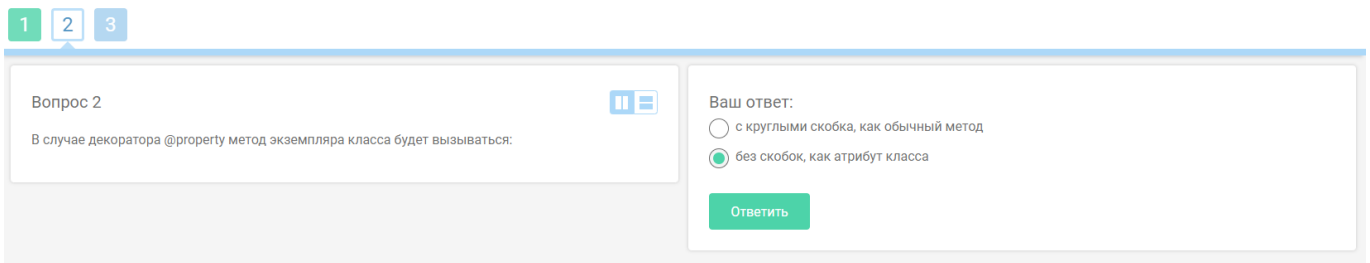






Тест по лекции 2





1 2 3

Вопрос 3

Как можно поменять логику изменения атрибута?

Ваш ответ:

- Через декоратор classmethod
- Через декоратор property
- Через декоратор property и setter property метода

Ответить

Тест по лекции 3

1 2 3

Вопрос 1

Что такое super?

Ваш ответ:

- Ключевое слово, представляющее родительский класс
- Метод родительского класса
- Атрибут родительского класса

Ответить

1 2 3

Вопрос 2

Вы отнаследовались от класса с реализованным методом __init__. Но вам нужно добавить свои атрибуты в дочерний класс помимо тех, что есть в родительском. Что нужно сделать с дочерним методом __init__?

Ваш ответ:

- Полностью переопределить метод __init__ и скопировать все атрибуты из родительского класса
- В переопределенном методе __init__ вызвать родителский метод __init__ и доопределить дочерний метод __init__

Ответить

1 2 3

Вопрос 2

В чем заключается механизм наследования?

Ваш ответ:

- Переиспользование атрибутов родительского класса по умолчанию
- Переиспользование методов родительского класса по умолчанию
- Переиспользование методов и атрибутов родительского класса по умолчанию
- Переопределение методов и атрибутов родительского класса по умолчанию

Ответить

Тест по лекции 4

1 2 3 4 5

Вопрос 1

С помощью какого магического метода можно вызвать экземпляр класса, как функцию (например так: MyClass()())?

Ваш ответ:

- __enter__
- __call__

Ответить

1 2 3 4 5

Вопрос 2

Какую операцию поддерживает магический метод __eq__?

Ваш ответ:

- ○ Сравнение “больше”
- ◉ Сравнение “равно”
- ○ Сравнение “меньше”

Ответить

1 2 3 4 5

Вопрос 4

С помощью какого магического метода можно производить операцию сложения между классами?

Ваш ответ:

- ◉ __add__
- ○ __sub__
- ○ __lt__
- ○ __mul__

Ответить

1 2 3 4 5

Вопрос 2

Если для класса переопределен только метод __lt__, можно ли сравнивать этот класс с другим через знак “>”?

Ваш ответ:

- ○ Да, можно, ведь у класса уже определен метод “меньше, чем”
- ◉ Нет, нельзя

Ответить

1 2 3 4 5

Вопрос 3

Можно ли определять методы __enter__ и __exit__ отдельно друг от друга при исполнении блока with?

Ваш ответ:

- ◉ Можно
- ○ Нельзя

Ответить

Домашнее задание 1

Реализуйте 3 метода в классе “Counter”:

__init__(self, initial_count)

Этот метод нужен для инициализации класса с изначальным параметром “изначальный подсчет”

increment(self)

Этот метод должен делать +1 к нашему счетчику подсчетов

get(self)

Этот метод должен возвращать подсчет

```
class Counter:


    a = 0


    def __init__(self, initial_count):
        self.a = initial_count


    def increment(self):
        self.a += 1


    def get(self):
        return self.a



code = []
while data := input():
    code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 2

Реализуйте метод calculate_area в классе “Circle”, в котором уже есть атрибут класса pi, который понадобится для расчета:

Этот метод будет рассчитывать площадь круга по формуле S=πR^2, где R передается в качестве параметра

```
class Circle:
    pi = 3.14

    def calculate_area(self, radius):
        return self.pi * radius ** 2



code = []
while data := input():
    code.append(data)

code = "\n".join(code)
exec(code)
```

Домашнее задание 3

Реализуйте 3 метода в классе “Person”:

__init__(self, age), в котором будет объявляться атрибут возраста

age с декоратором property, который будет возвращать атрибут возраста

age с декоратором age.setter

Этот метод должен обрабатывать изменение проперти “age”: при передаче значений < 0 атрибут возраста = 0, в противном случае атрибут возраста будет равен тому значению, которое передано

```
class Person:
    def __init__(self, age):
        self._age = age

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0:
            self._age = 0
        else:
            self._age = value



code = []
while data := input():
    code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 4

Реализовать метод calculate_period с декоратором classmethod в классе “Pendulum”, в котором написаны два атрибута класса:

```
g =10
pi = 3.14
Этот метод должен вычислять период математического маятника по формуле:
T=2πvl/g
class Pendulum:
    g = 10
    pi = 3.14

    @classmethod
    def calculate_period(cls, length):
        return round(2 * cls.pi * (length / cls.g) ** 0.5, 2)



code = []
while data := input():
    code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 5

В дочернем классе SemiCircle, который наследуется от родительского класса Circle с методами:

__init__(self, radius)

radius(self) с декоратором property

pi с декоратором property

calculate_area, который считает площадь круга по формуле S=πR^2

реализуйте метод calculate_area, который считает половину площади круга. Желательно, чтобы это было сделано через переиспользование метода calculate_area родительского класса.

```
class Circle:
    _pi = 3.14
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        return self._radius
    @property
    def pi(self):
        return self._pi

    def calculate_area(self):
        return self._pi * self._radius ** 2

class SemiCircle(Circle):
    def calculate_area(self):
        return self._pi * self._radius ** 2 / 2

code = []
while data := input(): code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 6

У нас есть такой же класс Circle из предудущего задания, но теперь появляется класс-примесь CalculateCircleLengthMixin. Затем от двух классов наследуется класс CircleWithMixin, реализуя таким образом множественное наследование. В классе-примеси CalculateCircleLengthMixin реализуйте метод calculate_length, который вычисляет длину окружности по формуле $L=2\pi r$, чтобы использовать его в итоге в классе CircleWithMixin. Желательно, чтобы это было сделано через super и проперти класса Circle.

```
class Circle:
    _pi = 3.14

    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @property
    def pi(self):
        return self._pi

    def calculate_area(self):
        return self._pi * self._radius ** 2



class CalculateCircleLengthMixin:
    def calculate_length(self):
        return 2 * self.pi * self.radius

class CircleWithMixin(CalculateCircleLengthMixin, Circle):
    pass

code = []
while data := input():
   code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 7

В классе Dictionary реализуйте методы __call__ и __init__:

В __init__(self, dictionary) объявите словарь в качестве атрибута

В методе call реализуйте поиск в словаре по ключу

```
class Dictionary:
    a = {}
    def __init__(self, dictionary):
        global a
        a = dictionary

    def __call__(self, key):
        return a[key]

code = []
while data := input():
    code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 8

Перед вам класс ContextDictionary, в котором уже реализованы несколько методов:

__init__: в нем объявлен атрибут self.dictionary, который равен None

put(self, key, value), который кладет значение value по ключу key в self.dictionary, когда этот атрибут является словарем, то есть в контексте with

get(self, key), который забирает значение по ключу key в self.dictionary, когда этот атрибут является словарем, то есть в контексте with

Вам нужно дополнить его, реализовав магические методы контекста:

__enter__: здесь во время контекста наш атрибут self.dictionary превращается в словарь

__exit__: в нем атрибут self.dictionary вновь равен None

Таким образом мы хотим смоделировать поведение класса в контексте with.

```
class ContextDictionary:
    def __init__(self):
        self.dictionary = None

    def __enter__(self):
        self.dictionary = {}
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.dictionary = None

    def put(self, key, value):
        self.dictionary[key] = value

    def get(self, key):
        return self.dictionary[key]

code = []
while data := input():
    code.append(data)
code = "\n".join(code)
exec(code)
```

Домашнее задание 9

Реализуйте следующую иерархию классов:

В классе Rectangle с атрибутами a и b, который являются сторонами прямоугольника реализуйте метод calculate_area, который вычисляет площадь прямоуольника по формуле S =ab:

В классе-наследнике от Rectangle- Square реализуйте метод __init__, в котором принимается только один параметр - сторона. Желательно, чтобы вы это реализовали через super.

В классе-примеси CalculatePerimeterMixin, наследнике Rectangle, реализуйте метод calculate_perimeter, который вычисляет периметр по формуле P =2(a + b)

В классе наследнике SquareWithMixin от двух классов (CalculatePerimeterMixin, Square) реализуйте 3 магических метода:

__eq__, который сравнивает фигуры по сторонам

__gt__, который сравнивает фигуры по площади

__add__, который складывает площади двух фигур

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def calculate_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __init__(self, a):
        super().__init__(a, a)

class CalculatePerimeterMixin(Rectangle):
    def calculate_perimeter(self):
        return 2 * (self.a + self.b)

class SquareWithMixin(CalculatePerimeterMixin, Square):
    def __eq__(self, other):
        return self.a == other.a
```

```
    def __gt__(self, other):
        return self.calculate_area() > other.calculate_area()

    def __add__(self, other):
        return self.calculate_area() + other.calculate_area()


code = []
while data := input():
   code.append(data)
code = "\n".join(code)
exec(code)
```