GeekBrains

Погружение в Python

Урок 11

ООП. Особенности Python

Содержание урока

План курса

1. Основы Python
2. Простые типы данных
3. Коллекции
4. Функции
5. Итераторы и генераторы
6. Модули
7. Файлы и файловая система
8. Сериализация
9. Декораторы
10. ООП. Начало
11. ООП. Особенности Python
12. ООП. Финал
13. Исключения
14. Основы тестирования
15. Обзор стандартной библиотеки Python

Что будет на уроке сегодня

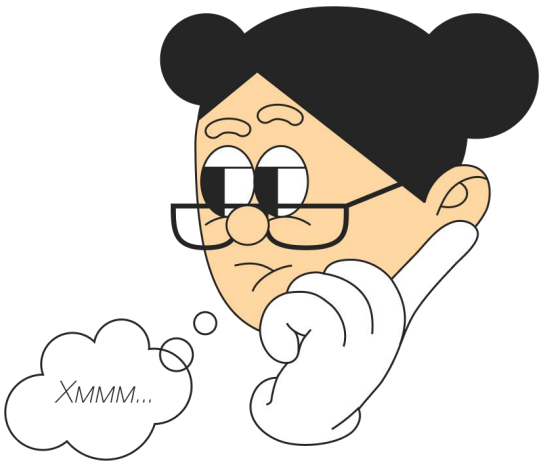
- Разберёмся с созданием и удалением классов
- Узнаем о документировании классов
- Изучим способы представления экземпляров
- Узнаем о возможностях переопределения математических операций
- Разберёмся со сравнением экземпляров
- Узнаем об обработке атрибутов

Создание
и удаление

Создание экземпляра класса, __init__

Объект созданный в результате вызова класса называется его экземпляром

```
def __init__(self, *args, **kwargs):
    self.param = args
    ...
```



Контроль создания класса через __new__

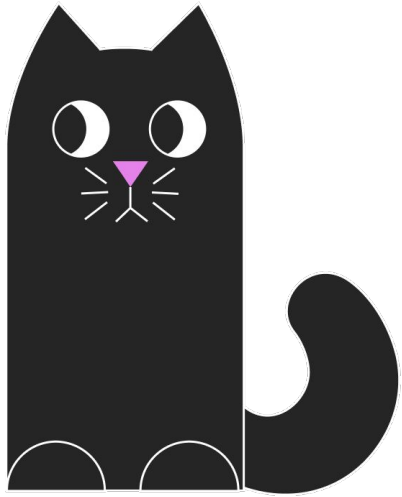
Метод __new__ срабатывает в момент создания класса и может его изменить

- Расширение неизменяемых классов
- Шаблон Одиночка, Singleton
- Другие модификации, метапрограммирование

Удаление экземпляра класса, __del__

Команда del не удаляет объект, а уменьшает счётчик ссылок объекта.

Дандер метод del срабатывает при достижении счётчиком ссылок нуля. Выполняет перед удалением объекта из памяти сборщиком мусора.

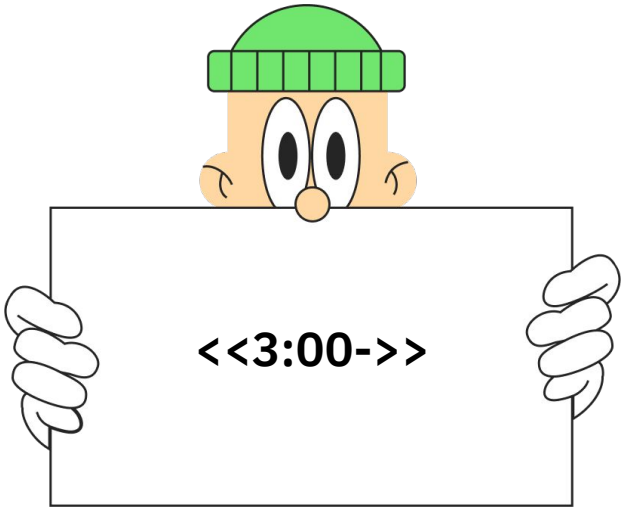


Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Создание и удаление

```
class Count:
    _count = 0
    _last = None

    def __new__(cls, *args, **kwargs):
        if cls._count < 3:
            cls._last = super().__new__(cls)
            cls._count += 1
        return cls._last

    def __init__(self, name: str):
        self.name = name
```

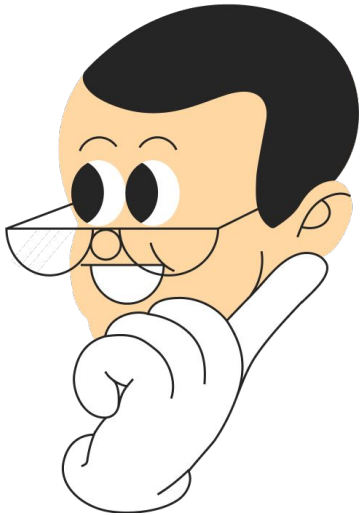


Строка документации

Строка документации

Наличие строк документации у классов, методов и функций — хороший тон

- help(instance) — справка на основе структуры класса включает документацию
- instance.__doc__ — обращение к документации объекта напрямую

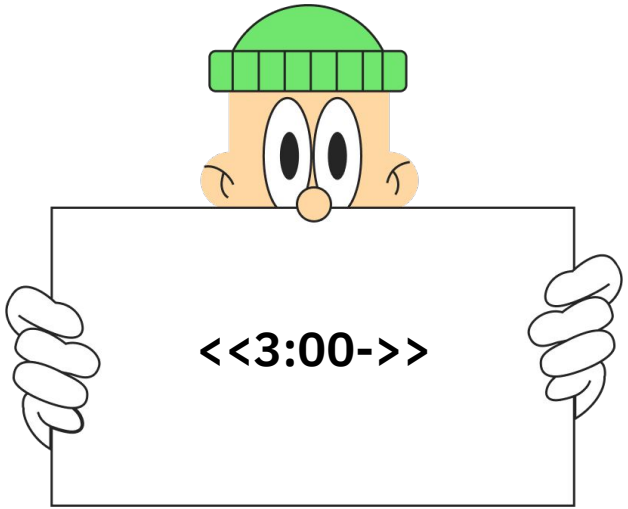


Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Строка документации

```
class MyClass:
    A = 42
    """About class"""

    def __init__(self, a, b):
        """self.__doc__ = None"""
        self.a = a
        self.b = b

    def method(self):
        """Documentation"""
        self.__doc__ = None

help(MyClass)
```

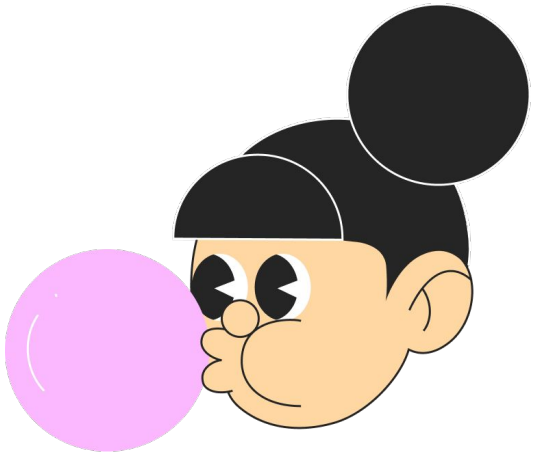


Представления экземпляра

Представление для пользователя, __str__

Функция print ищет __str__ для вывода информации в консоль

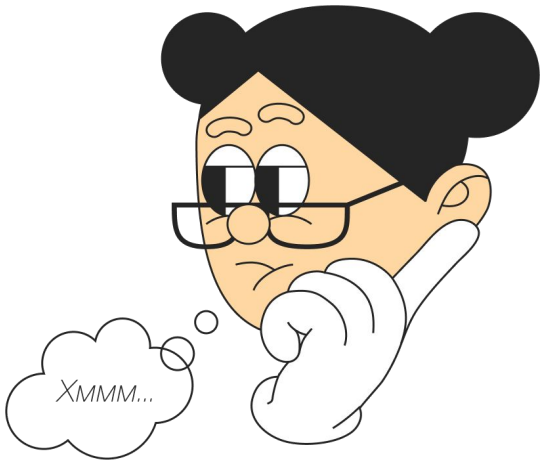
```
class Person:
    ...

        def __str__(self):
            ...
            return 'Текст для пользователя'
```



Представление для создания экземпляра, __repr__

Строка должна создать новый экземпляр, если скопировать её в код

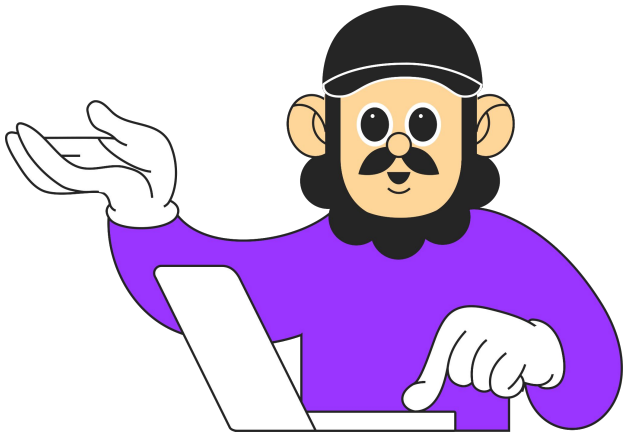
```
class Person:
    ...

        def __repr__(self):
            ...
            return f'Person({self.param})'
```



Приоритет методов

Варианты срабатывания __str__ и __repr__

- print(user)
  __str__
- print(f'{user}')
  __str__
- print(repr(user))
  __repr__
- print(f'{user = }')
  __repr__
- print(collections)
  __repr__

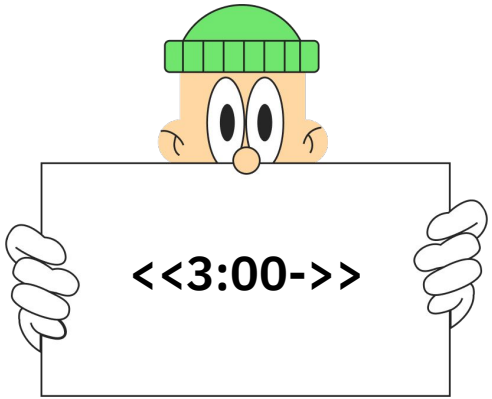


Перед вами несколько строк кода.
Что в нём неверно?

У вас 3 минуты.

Представление экземпляра

```
class MyClass:

    def __init__(self, a, b):
        self.a = a
        self.b = b
        self.c = a + b

    def __str__(self):
        return f'MyClass(a={self.a}, b={self.b}, c={self.c})'

    def __repr__(self):
        return str(self.a) + str(self.b) + str(self.c)
```



Математика и логика

Переопределение

Операция в Python	Основной метод	Right метод	In place метод
+	__add__(self, other)	__radd__(self, other)	__iadd__(self, other)
-	__sub__(self, other)	__rsub__(self, other)	__isub__(self, other)
*	__mul__(self, other)	__rmul__(self, other)	__imul__(self, other)
@	__matmul__(self, other)	__rmatmul__(self, other)	__imatmul__(self, other)
/	__truediv__(self, other)	__rtruediv__(self, other)	__itruediv__(self, other)
//	__floordiv__(self, other)	__rfloordiv__(self, other)	__ifloordiv__(self, other)
%	__mod__(self, other)	__rmod__(self, other)	__imod__(self, other)

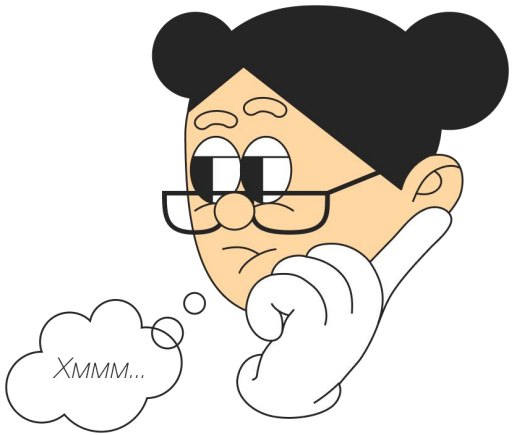
Переопределение, продолжение

Операция в Python	Основной метод	Right метод	In place метод
divmod()	__divmod__(self, other)	__rdivmod__(self, other)	__idivmod__(self, other)
**, pow()	__pow__(self, other[, modulo])	__rpow__(self, other[, modulo])	__ipow__(self, other[, modulo])
<<	__lshift__(self, other)	__rlshift__(self, other)	__ilshift__(self, other)
>>	__rshift__(self, other)	__rrshift__(self, other)	__irshift__(self, other)
&	__and__(self, other)	__rand__(self, other)	__iand__(self, other)
^	__xor__(self, other)	__rxor__(self, other)	__ixor__(self, other)
|	__or__(self, other)	__ror__(self, other)	__ior__(self, other)

Основные методы

Левый объект вызывает свой метод и возвращает новый экземпляр класса

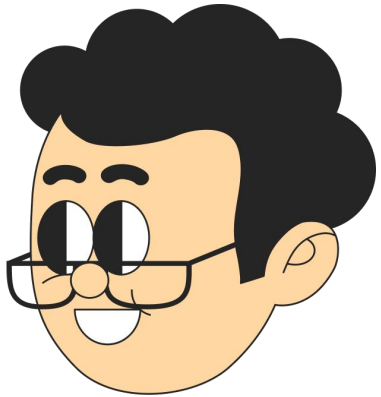
```
class Name:
    ...
    def __add__(self, other):
        ...
        return Name(param)


c = a + b
```



Right методы

Левый объект не находит нужный метод, поэтому правый объект вызывает свой метод и возвращает новый экземпляр класса

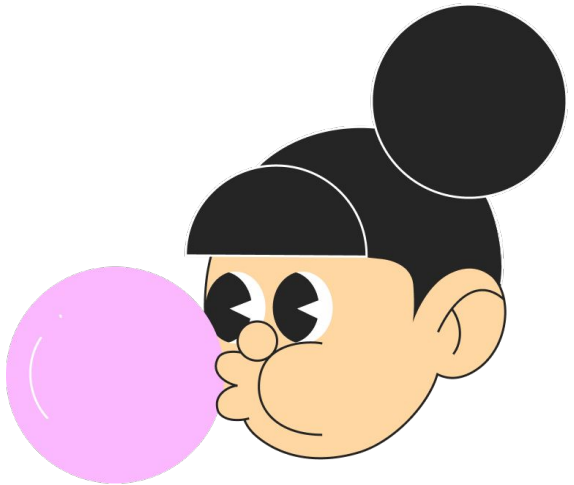
```
class Name:
    ...
    def __radd__(self, other):
        ...
        return Name(param)

c = b + a
```



In place методы

Объект вызывает свой метод и изменяет своё значение

```
class Name:
    ...
    def __iadd__(self, other):
        ...
        return self


a += b
```

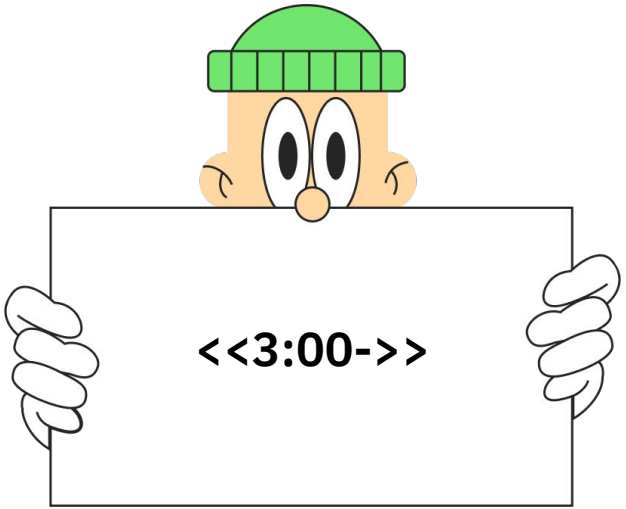


Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Переопределения

```
class MyClass:

    def __init__(self, data):
        self.data = data

    def __and__(self, other):
        return MyClass(self.data + other.data)

    def __str__(self):
        return str(self.data)


a = MyClass((1, 2, 3, 4, 5))
b = MyClass((2, 4, 6, 8, 10))
print(a & b)
```

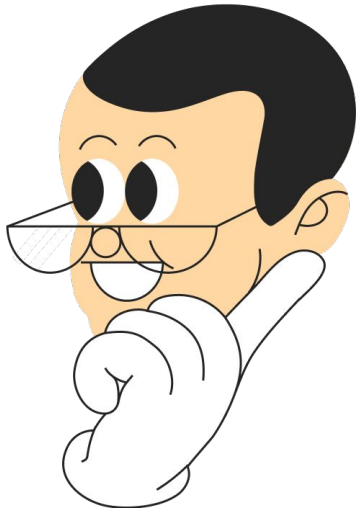


Сравнение экземпляров класса

Сравнение экземпляров класса

Python поддерживает определение шести основных операций сравнения экземпляров

- __eq__ - равно, ==
- __ne__ - не равно, !=
- __gt__ - больше, >
- __ge__ - не больше, меньше или равно, <=
- __lt__ - меньше, <
- __le__ - не меньше, больше или равно, >=



Неизменяемые экземпляры, хеширование, дандер __hash__

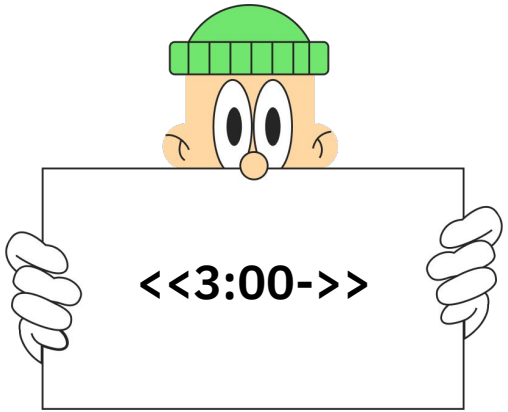
	__eq__ есть	__eq__ нет
__hash__ есть	Неизменяемый объект реализованный разработчиком	✘ Запрещённая комбинация! Разработчик допустил ошибку
__hash__ нет	Изменяемый объект. Python устанавливает __hash__ = None	Неизменяемый объект. Python сам реализует оба дандера

Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Сравнение экземпляров

```
class MyClass:

    def __init__(self, a, b):
        self.a = a
        self.b = b
        self.c = a + b

    def __str__(self):
        return f'MyClass(a={self.a}, b={self.b}, c={self.c})'

    def __eq__(self, other):
        return (sum((self.a, self.b)) - self.c) == (sum((other.a, other.b)) - other.c)


x = MyClass(42, 2)
y = MyClass(73, 3)
print(x == y)
```

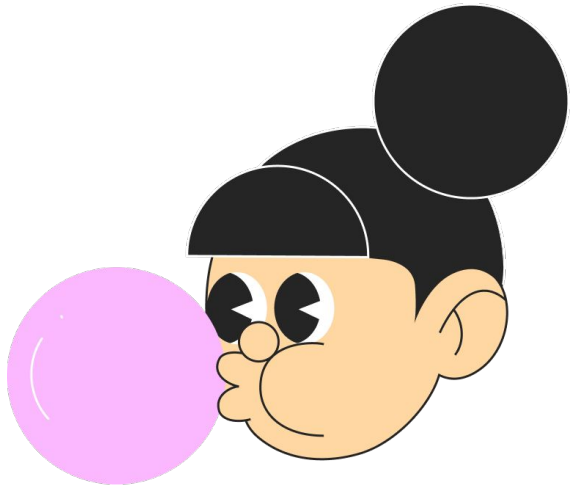


Обработка атрибутов

Получение значения атрибута, __getattribute__

Дандер __getattribute__ вызывается при любой попытке обращения к атрибутам экземпляра

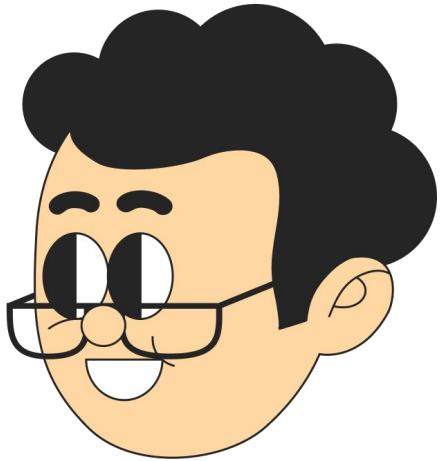
```
class Name:
    ...
    def __getattribute__(self, item):
        ...
        return object.__getattribute__(self, item)
```



Присвоение атрибуту значения, __setattr__

Дандер __setattr__ срабатывает каждый раз,
когда в коде есть операция присвоения

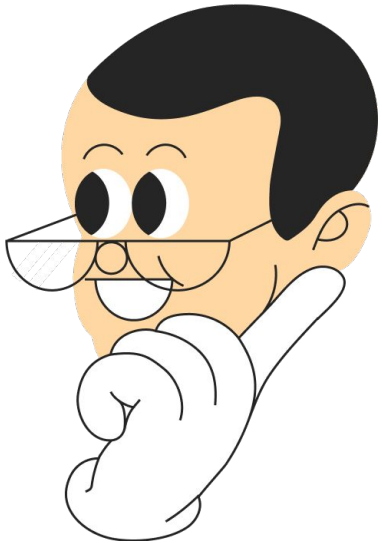
```
class Name:
    ...
    def __setattr__(self, key, value):
        ...
        return object.__setattr__(self, key, value)
```



Обращение к несуществующему атрибуту, __getattr__

Если свойство отсутствует, в первую очередь вызывается дандер __getattribute__. В случае возврата им ошибки AttributeError вызывается метод __getattr__

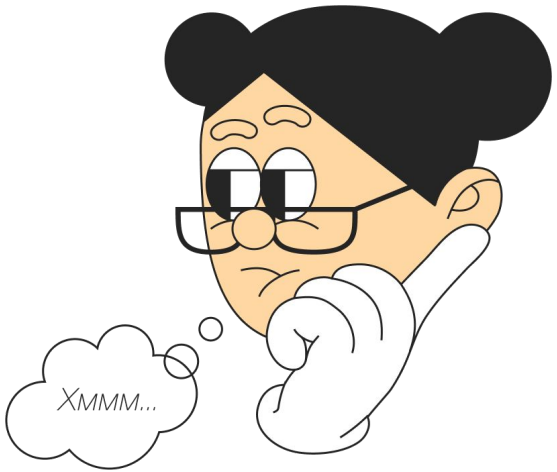
```
class Name:
    ...
    def __getattr__(self, item):
        ...
        return ...
```



Удаление атрибута, __delattr__

Дандер __delattr__ вызывается при попытке удалить атрибут командой del

```
class Name:
    ...
    def __delattr__(self, item):
        ...
        object.__delattr__(self, item)
```

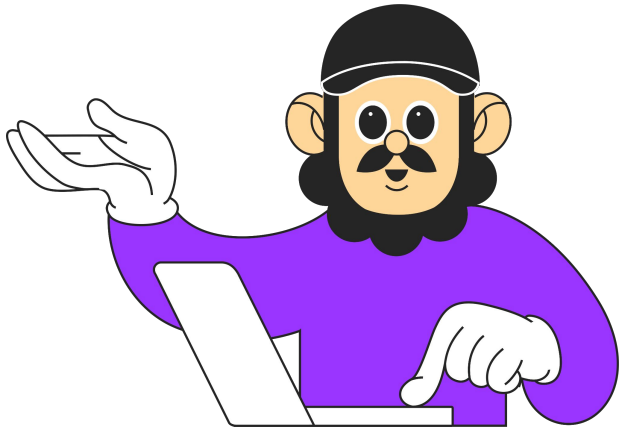


Функции setattr(), getattr() и delattr()

- **setattr(object, name, value)**
  аналог object.name = value

- **getattr(object, name[, default])**
  аналог object.name or default

- **delattr(object, name)**
  аналог del object.name



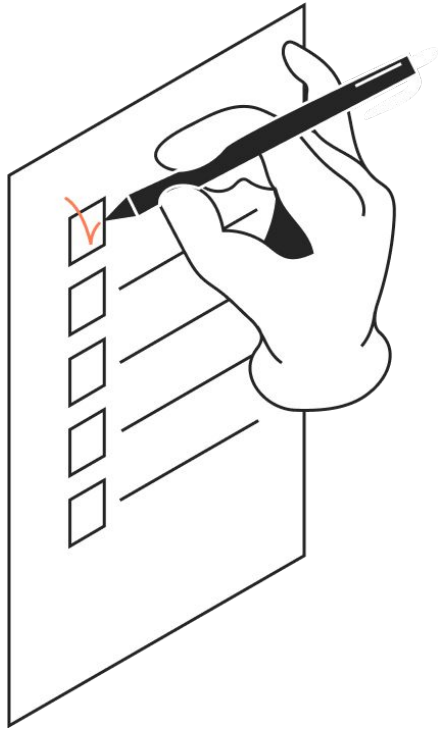
Итоги занятия

На этой лекции мы

- Разобрались с созданием и удалением классов
- Узнали о документировании классов
- Изучили способы представления экземпляров
- Узнали о возможностях переопределения математических операций
- Разобрались со сравнением экземпляров
- Узнали об обработке атрибутов

Задание

Возьмите 1-3 задачи из прошлых занятий
и попробуйте перенести переменные и функции
в класс. Добавьте к ним дандер методы из лекции
для решения изначальной задачи..



Спасибо за внимание