

Углубленный Python

Лекция 7

ООП

Кандауров Геннадий



образование

Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK

Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3

Введение в Python, основные понятия, тестирование

Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

Подписаться

Создать топик

Поиск по авторам, заголовку и тексту топика...

Найти

Добро пожаловать на курс!

Углубленный Python

Изменить

Удалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад

Углубленный Python → Добро пожаловать на курс! 0

Екатерина Черкасова 7 дней назад

Стажировка → Приглашаем мобильных, фронтенд- и бэкэнд-разработчиков на Weekend Offer! 0

Дарья Вовченко 9 дней назад

Углубленный Python → Добро пожаловать в образовательные проекты VK Образование! 0

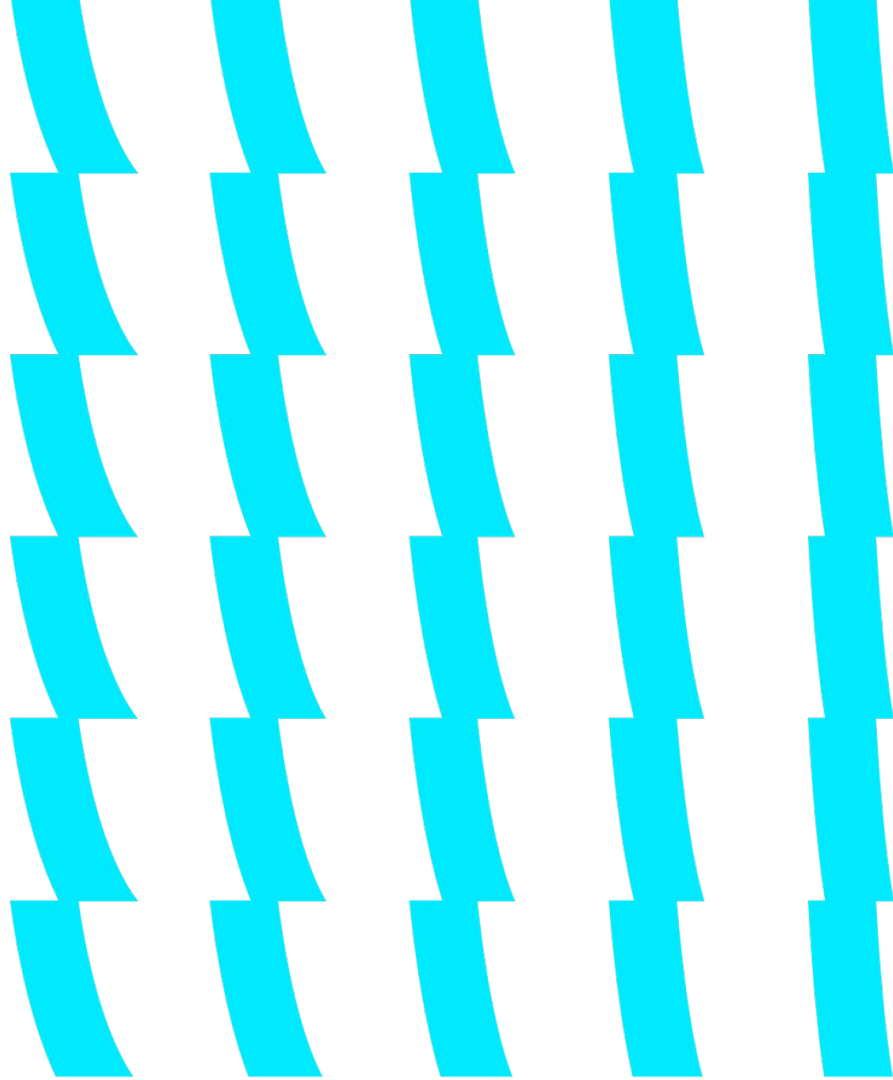
Дарья Вовченко 9 дней назад

Разработка веб-сервисов на

Содержание занятия

1. Классы
2. ООП
3. Генераторы

Классы



Классы

```
class A:
    @staticmethod
    def print_static():
        print("static")

    @classmethod
    def print_cls(cls):
        print(f"class_method for {cls.__name__}")

    def __init__(self, val):
        self.val = val
        self._protected_val = val
        self._private_val = val

    def print_offset(self, offset=10):
        print(self.val + offset)

    def __str__(self):
        return f"{self.__class__.__name__}:val={self.val}"
```

Классы: доступ к атрибутам

Чтобы найти атрибут объекта `obj`, python обыскивает:

1. Сам объект (`obj.__dict__` и его системные атрибуты)
2. Класс объекта (`obj.__class__.__dict__`).
3. Классы, от которых унаследован класс объекта (`obj.__class__.__mro__`)

Классы: магические методы

```
object.__new__(cls[, ...])
```

Статический метод, создает новый экземпляр класса.

После создание экземпляра вызывается (уже у экземпляра) метод `__init__`.

`__init__` ничего не должен возвращать (кроме `None`), иначе - `TypeError`

```
class Singleton:
```

```
    _instance = None
```

```
    def __new__(cls, *args, **kwargs):
```

```
        if cls._instance is None:
```

```
            cls._instance = super().__new__(cls, *args, **kwargs)
```

```
        return cls._instance
```

Классы: магические методы

`object.__del__(self)`

Финализатор

```
class Connector:
    def __init__(self, db_name):
        self.conn = DbDriver(db_name)
    def __del__(self):
        self.conn.close()
        print("DEL")
```

```
db = Connector("users")
```

```
del db  # ???
```


Классы: магические методы

```
object.__set_name__(self, owner, name)
```

Автоматически вызывается при создании класса-владельца `owner`.

Хук вызывается, когда объекту было присвоено имя `name` в этом классе

```
class Attr:
    def __set_name__(self, owner, name):
        print(f"{locals()}=")
        self.name = name

class A:
    x = Attr()  # Automatically calls: x.__set_name__(A, "x")

a = A()
```

Классы: магические методы

```
object.__call__(self[, args...])
```

```
class Adder:  
    def __init__(self, val):  
        self.val = val  
  
    def __call__(self, value):  
        return self.val + value
```

```
a = Adder(10)
```

```
a(5)  # 15
```

Классы: магические методы

`object.__slots__`

Позволяет явно указать поля, которые будут в классе.

В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

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

Классы: магические методы

Доступ к атрибутам

- `__getattr__(self, name)`
- `__getattribute__(self, name)`
- `__setattr__(self, name, val)`
- `__delattr__(self, name)`
- `__dir__(self)`

Классы: наследование

```
class Timing:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    def duration(self):
        print("Timing.duration")
        return self.end - self.start
```

```
class MinuteTiming(Timing):
    def duration(self):
        print("MinuteTiming.duration")
        seconds = super().duration()
        return seconds / 60
```

```
>>> m = MinuteTiming(1000, 7000)
```

```
>>> m.duration()
```

```
MinuteTiming.duration
```

```
Timing.duration
```

```
100.0
```

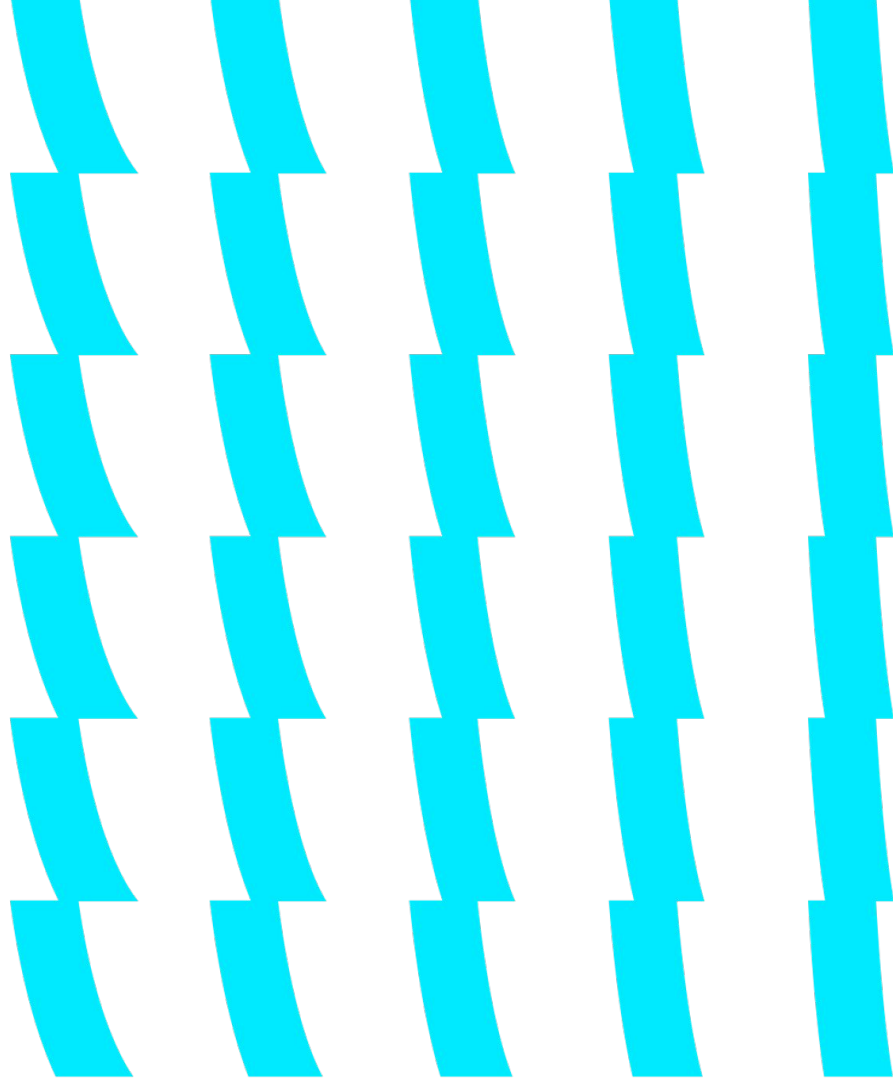
Классы: `__init_subclass__`

```
class Timing:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    @classmethod
    def __init_subclass__(cls, **kwargs):
        print("INIT subclass", cls, kwargs)
        super().__init_subclass__(**kwargs)
```

```
class MinuteTiming(Timing):
    def duration(self):
        print("MinuteTiming.duration")
        seconds = super().duration()
        return seconds / 60
```

Дескрипторы





Дескриптор это атрибут объекта со “связанным поведением”, то есть такой атрибут, при доступе к которому его поведение переопределяется методом протокола дескриптора. Эти методы `__get__`, `__set__` и `__delete__`. Если хотя бы один из этих методов определен в объекте , то можно сказать что этот объект дескриптор.

Раймонд Хеттингер

Дескрипторы

- Если определен один из методов `__get__`, `__set__` и `__delete__`, объект считается дескриптором.
- Если объект дескриптора определяет `__get__`, `__set__`, то он считается data дескриптором.
- Если объект дескриптора определяет `__get__`, то является non-data дескриптором.

Дескрипторы

```
>>> class A:
    def foo(self):
        pass
    a = A()
```

```
>>> a.foo.__class__.__get__
<slot wrapper '__get__' of 'method' objects>
```

```
>>> A.__dict__['foo'] # Внутренне хранится как функция
<function foo at 0x00C45070>
```

```
>>> A.foo # Доступ через класс возвращает несвязанный метод
<unbound method A.foo>
```

```
>>> a.foo # Доступ через экземпляр объекта возвращает связанный метод
<bound method A.foo of <__main__.A object at 0x00B18C90>>
```

Дескрипторы

```
class MyDescriptor:
    def __get__(self, obj, objtype):
        print(f"get {obj} cls={objtype}")

    def __set__(self, obj, val):
        print(f"set {val} for {obj}")

    def __delete__(self, obj):
        print(f"delete from {obj}")
```

```
class MyClass:
    field = MyDescriptor()
```

```
>>> inst = MyClass()
>>> MyClass.field
get None cls=<class '__main__.MyClass'>
>>> inst.field
get <__main__.MyClass object ...> cls=<class
'__main__.MyClass'>
>>> inst.field = 1
set 1 for <__main__.MyClass object ...>
>>> del inst.field
delete from <__main__.MyClass object ...>
```

Домашнее задание #7

- Reader/Writer разных форматов данных с единым интерфейсом
- Генератор для чтения и фильтрации файла
- Тесты
- flake8 и pylint перед сдачей

Напоминание отметиться на портале Vol 2

+ ОСТАВИТЬ ОТЗЫВ ПОСЛЕ ЛЕКЦИИ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK

Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3
Введение в Python, основные
понятия, тестирование
Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

ПодписатьсяСоздать топик

Поиск по авторам, заголовку и тексту топика...Найти

Добро пожаловать на курс!

Углубленный Python

ИзменитьУдалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад
Углубленный Python → Добро пожаловать
на курс! 0

Екатерина Черкасова 7 дней назад
Стажировка → Приглашаем мобильных,
фронтенд- и бэкэнд-разработчиков на
Weekend Offer! 0

Дарья Вовченко 9 дней назад
Углубленный Python → Добро пожаловать
в образовательные проекты VK
Образование! 0

Дарья Вовченко 9 дней назад
Разработка веб-сервисов на

Спасибо за
внимание



образование