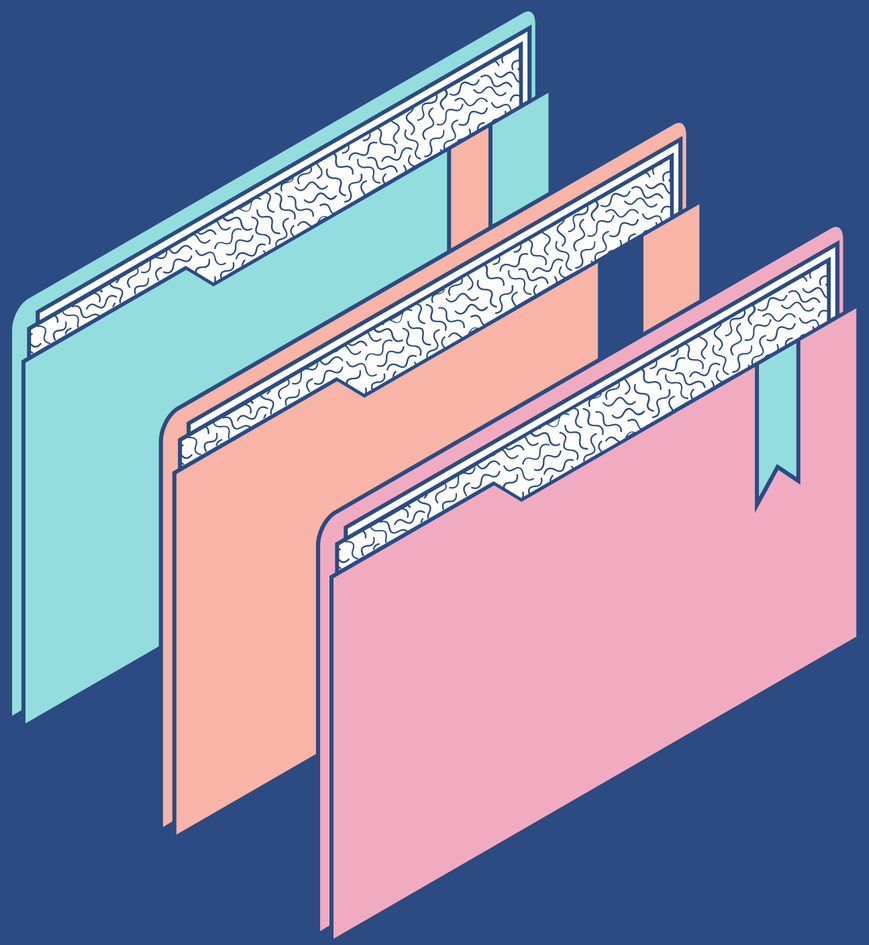
КАФЕДРА №13

Понятие инкапсуляции. Виды атрибутов и методов

ЛИТВИНОВ ВЛАДИСЛАВ

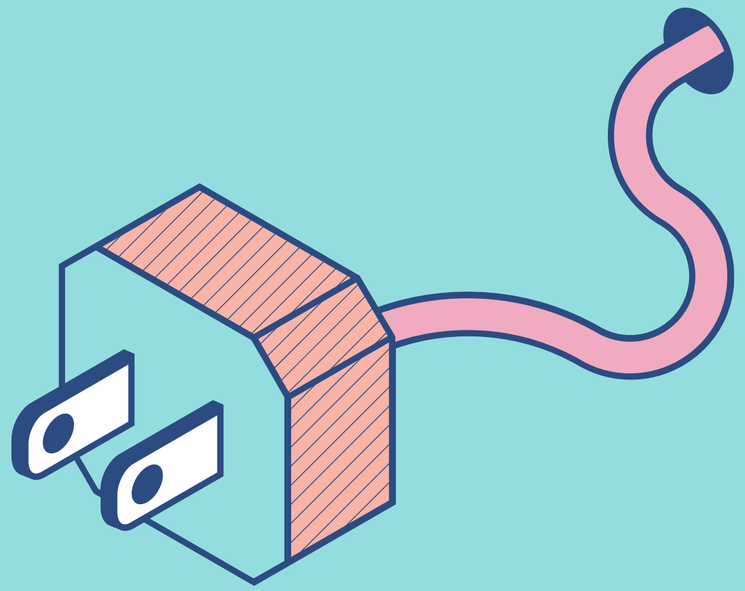


ВОПРОСЫ

ЯКОРЯ В ЭТОЙ ПРЕЗЕНТАЦИИ

- Декораторы в Python
- Понятие инкапсуляции. Применение инкапсуляции
- Методы и атрибуты. Геттеры и сеттеры. Модификаторы доступа

Что такое декоратор в Python



```
1  def null_decorator(func):
2      return func
✓ 0.0s
```

Декоратор — это вызываемый объект, который принимает на вход вызываемый объект и возвращает другой вызываемый объект. Декораторы в Python позволяют расширять и изменять поведение вызываемых объектов (функций, методов и классов) без постоянного изменения самого вызываемого объекта.

```
1  @null_decorator
2  def greet():
3      return 'Hello!'
✓ 0.0s
```

```
1  greet()
✓ 0.0s
'Hello!'
```

Инкапсуляция

- Один из основополагающих принципов ООП
- Связь данных с методами которые этими данными управляют
- Набор инструментов для управления доступом к данным или методам которые управляют этими данными
- Ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса
- Инкапсуляция в Python работает лишь на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними.

Подчеркивания. Зачем?

- Одиночное подчеркивание в начале имени атрибута говорит о том, что переменная или метод не предназначен для использования вне методов класса, однако атрибут доступен по этому имени
- Двойное подчеркивание в начале имени атрибута даёт большую защиту: атрибут становится недоступным по этому имени. Однако полностью это не защищает, так как атрибут всё равно остаётся доступным под именем _ИмяКласса__ИмяАтрибута

```python
class A:
    def _private(self):
        print("Это приватный метод!")
```

```python
a = A()
a._private()
```

```
Это приватный метод!
```

```python
class B:
    def __private(self):
        print("Это приватный метод!")
```

```python
b = B()
b.__private()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[26], line 2
      1 b = B()
----> 2 b.__private()

AttributeError: 'B' object has no attribute '__private'
```

```python
b._B__private()
```

```
Это приватный метод!
```

Поговорим о синтаксисе

КОГДА __ПРИВАТНЫЙ МЕТОД НЕ СОВСЕМ ПРИВАТНЫЙ, А _ПРИВАТНЫЙ СОВСЕМ НЕ ПРИВАТНЫЙ

Сплошной договорнячок…

Геттеры и сеттеры

Геттер позволяет получить значение атрибута, а сеттер установить его.

Большая приватность

Используя __приватные атрибуты и сеттеры\геттеры при написании классов, мы как-то можем защитить атрибуты от прямого влияния из глобальной области

Аннотации свойств

- Для создания свойства-геттера над свойством ставится аннотация **@property**.
- Для создания свойства-сеттера над свойством устанавливается аннотация **имя_свойства_геттера.setter**

Обязательно?

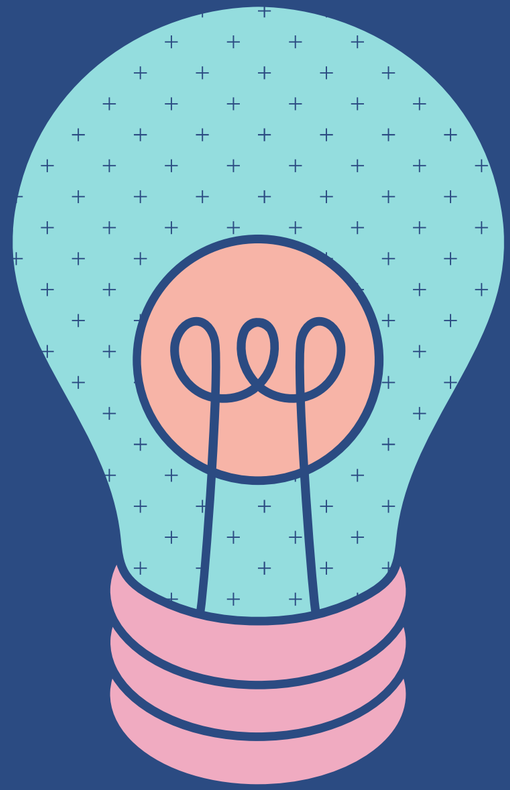
Ни геттеры, ни сеттеры, ни свойства совсем не обязательно использовать. Это всего лишь способы сделать код более изящным

Следующие шаги нашего обучения

- Полиморфизм
- Исключения
- Параллельное программирования

«Определив точно значения слов, вы избавите человечество от половины заблуждений»

РЕНЕ ДЕКАРТ



У вас есть какие-то вопросы?

Оставьте их себе! Надеюсь, что вы узнали что-то новое.

Позвоните мне

+375 (33) 398-54-37

Электронная почта

kordebalet67@gmail.com

Telegram

@Kordebalet

КАФЕДРА №13

Понятие инкапсуляции. Виды атрибутов и методов

ЛИТВИНОВ ВЛАДИСЛАВ