Программирование
на Python

Презентация занятия

Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

15 занятие

инжиниум®

МГТУ им. Н.Э. Баумана

2019

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

1.1 Атрибуты объекта класса:

```
class Player:               # определение класса
    name = "Katya"          # атрибуты объекта
    age = 40                # класса
    def show(self):         # метод класса
        print(self.name)    # вывод значений атрибута
        print(self.age)     #
p = Player()
p.show()
```

1.2 Атрибуты экземпляра класса:

```
class Player:               # определение класса
    def __init__(self, name, age):    # Конструктор
        self.name = name    # Атрибут экземпляра класса
        self.age = age      # self - это ссылка на экземпляр класса
    def show(self):         # метод класса
        print(self.name)    # вывод значений атрибута
        print(self.age)     #
p = Player("Name", 20)
p.show()
```

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

1.3 Назначение атрибуту значения по умолчанию

```
class Car():
    def __init__(self, make, model, year):
        """Инициализирует атрибуты описания автомобиля."""
        self.make = make
        self.model = model
        self.year = year
        self.probeg = 0
    def show_probeg(self):
        """Выводит пробег машины в милях."""
        print("This car has " + str(self.probeg) + " miles on it.")

my_new_car = Car('audi', 'a4', 2016)
my_new_car.show_probeg()
```

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

2. ИЗМЕНЕНИЕ ЗНАЧЕНИЙ АТРИБУТОВ

2.1 Прямое изменение значения атрибута

```
my_new_car = Car('audi', 'a4', 2016)
my_new_car.show_probeg()

my_new_car.probeg = 23
my_new_car.show_probeg()
```

2.2 Изменение значения атрибута с использованием метода

```
class Car():
    ...
    def update_probeg(self, values):
        self.probeg = values
```

```
my_new_car.update_probeg(20)
my_new_car.show_probeg()
```

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

2.3 Изменение значения атрибута с приращением

```
class Car():
    ...
    def update_probeg(self, values):
        self.probeg += values
my_new_car.update_probeg(20)
my_new_car.show_probeg()
my_new_car.update_probeg(3)
my_new_car.show_probeg()
```

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3. СТАТИЧЕСКИЕ И КЛАССОВЫЕ МЕТОДЫ

Согласно модели данных Python, язык предлагает три вида методов:

1. Статические
2. Классовые
3. Экземпляра класса

Понимание принципов их работы поможет в создании красивого и эффективного кода.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3. СТАТИЧЕСКИЕ И КЛАССОВЫЕ МЕТОДЫ

```
class ToyClass:
    def instancemethod(self):
        return 'instance method called', self

    @classmethod
    def classmethod(cls):
        return 'class method called', cls

    @staticmethod
    def staticmethod():
        return 'static method called'
```

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.1 Методы экземпляра класса

- Принимают объект класса (**self**) как первый аргумент
- Количество параметров метода не ограничено
- **self** позволяет менять состояние объекта и обращаться к другим его методам и параметрам
- **self.__class__** дает доступ к атрибутам класса и возможность менять состояние самого класса

*То есть **методы экземпляров класса позволяют менять как состояние определённого объекта, так и класса***.

Встроенный пример метода экземпляра—str.upper():
>>> "welcome".upper() # <- вызывается на строковых данных
'WELCOME'

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.2 Методы класса

- Методы класса принимают класс в качестве параметра, который принято обозначать как **cls**
- Для обозначения используется декоратор ***classmethod***
- Методы класса привязаны к самому классу, а не его экземпляру!

То есть ***они могут менять состояние класса, что отразится на всех объектах этого класса, но не могут менять конкретный объект.***

Встроенный пример метода класса—dict.fromkeys()— возвращает новый словарь с переданными элементами в качестве ключей.

>>> dict.fromkeys('AEIOU') # <- вызывается при помощи класса dict {'A': None, 'E': None, 'I': None, 'O': None, 'U': None}

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.3 Статические методы

- Обозначаются при помощи декоратора ***staticmethod***.
- Им не нужен определённый первый аргумент (ни self, ни cls).
- Их можно воспринимать как методы, которые “не знают, к какому классу относятся”.

Таким образом, статические методы прикреплены к классу лишь для удобства и не могут менять состояние ни класса, ни его экземпляра.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.4 Конкретные примеры

```
obj = ToyClass()
print(obj.instancemethod())
print(ToyClass.instancemethod(obj))
```

метод instancemethod имеет доступ к объекту класса ToyClass через аргумент self

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.4 Конкретные примеры

Теперь давайте вызовем метод класса:
print(obj.classmethod())

Вывод:
('class method called', <class '__main__.ToyClass'>)

Мы видим, что метод класса *classmethod()* имеет доступ к самому классу *ToyClass*, но не к его конкретному экземпляру объекта.

Помните, в Python всё является объектом. Класс тоже объект, который мы можем передать функции в качестве аргумента.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.4 Конкретные примеры

Вызовем статический метод:
print(obj.staticmethod())

Вывод:
static method called

То есть статические методы не могут получить доступ к параметрам класса или объекта. Они работают только с теми данными, которые им передаются в качестве аргументов.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.4 Конкретные примеры

Теперь давайте вызовем те же самые методы, но на самом классе.

```
>>> ToyClass.classmethod()
('class method called', <class ToyClass at 0x10f453a10>)
>>> ToyClass.staticmethod()
'static method called'
>>> ToyClass.instancemethod()
TypeError: unbound method instancemethod()
must be called with ToyClass instance as
first argument (got nothing instead)
```

Метод класса и статический метод работают, как нужно. Однако вызов метода экземпляра класса выдаёт TypeError, так как метод не может получить на вход экземпляр класса.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

3.5 Запомнить!

- Методы экземпляра класса получают доступ к объекту класса через параметр self и к классу через self.__class__.
- Методы класса не могут получить доступ к определённому объекту класса, но имеют доступ к самому классу через cls.
- Статические методы работают как обычные функции, но принадлежат области имён класса. Они не имеют доступа ни к самому классу, ни к его экземплярам.
- Даже если вы программируете только ради интереса, изучение ООП в Python поможет писать код так, чтобы в дальнейшем было легче искать ошибки и использовать его повторно.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

4. ДЕКОМПОЗИЦИЯ КОДА

4.1 Что такое декомпозиция?

Декомпозиция кода - процесс разбиения кода на независимые блоки - функции и классы.

4.2 Для чего нам нужно декомпозировать?

Для тестирования программного решения.
Протестировать программу по блокам (функциям, классам) проще и быстрее, нежели чем тестировать все программное решение целиком.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

Упражнения

4. Посетители: начните с программы из упражнения 1. Добавьте атрибут number_served со значением по умолчанию 0; он представляет количество обслуженных посетителей . Создайте экземпляр с именем restaurant. Выведите значение number_served, потом измените и выведите снова.
Добавьте метод с именем set_number_served(), позволяющий задать количество обслуженных посетителей. Вызовите метод с новым числом, снова выведите значение.
Добавьте метод с именем increment_number_served(), который увеличивает количество обслуженных посетителей на заданную величину. Вызовите этот метод с любым числом, которое могло бы представлять количество обслуженных клиентов — скажем, за один день.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

5. Попытки входа: добавьте атрибут login_attempts в класс User из упражнения 3. Напишите метод increment_login_attempts(), увеличивающий значение login_ attempts на 1. Напишите другой метод с именем reset_login_attempts(), обнуляющий значение login_attempts . Создайте экземпляр класса User и вызовите increment_login_attempts() несколько раз. Выведите значение login_attempts, чтобы убедиться в том, что значение было изменено правильно, а затем вызовите reset_login_attempts(). Снова выведите login_attempts и убедитесь в том, что значение обнулилось.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

6. Киоск с мороженым: киоск с мороженым — особая разновидность ресторана . Напишите класс IceCreamStand, наследующий от класса Restaurant из упражнения 1 или упражнения 4. Подойдет любая версия класса; просто выберите ту, которая вам больше нравится. Добавьте атрибут с именем flavors для хранения списка сортов мороженого. Напишите метод, который выводит этот список. Создайте экземпляр IceCreamStand и вызовите этот метод.

Тема: Изучение возможностей и синтаксиса Python: Классы и ООП. Часть 2.

Рефлексия

1. Чем отличаются атрибуты объекта класса от атрибутов экземпляра класса?
2. Какими способами можно изменять значения атрибутов?
3. У класса есть _____ и _____. _____ задают значения, а ____ задают функционал.
4. Что такое декомпозиция кода?
5. В каких случаях это может быть полезно?
6. Чем отличается статический метод от классового?