УРОК 29. ВВЕДЕНИЕ В ООП

ООП

Объектно-ориентированное программирование (ООП) - это парадигма программирования, в которой программа организована вокруг объектов, которые представляют сущности из реального мира или абстрактные концепции.

Объекты включают данные и методы, которые работают с этими данными. Можно представить объект как "контейнер" данных и функциональности, связанных с этими данными.
ООП обычно используется, когда требуется моделировать сложные системы с большим количеством взаимодействующих объектов или когда требуется повторное использование кода. Функциональный подход, с другой стороны, обычно используется для написания более простых, линейных программ, которые основываются на функциях и передаче данных между ними.

Базовые принципы (концепции) ООП:

- Наследование: возможность создания новых классов на основе существующих, позволяет наследовать данные и методы от родительского класса.
- Инкапсуляция: сокрытие деталей реализации и предоставление публичного интерфейса для взаимодействия с объектом.
- Полиморфизм: возможность объектов одного класса проявлять разные формы поведения в зависимости от контекста.
- Абстракция: представление объектов в программе в виде абстрактных сущностей с общими характеристиками и поведением.

КЛАСС И ЭКЗЕМПЛЯР КЛАССА

Класс - это шаблон или описание для создания объектов.

Он определяет состояние и поведение объектов, которые будут созданы на его основе.

Экземпляр класса, или объект, является конкретной реализацией класса, имеющей свои собственные значения полей и состояние.

Ключевое слово class используется для определения нового класса в Python. Определение класса включает имя класса, которое следует соглашениям о именовании переменных, и тело класса, содержащее поля и методы.

```python
class MyClass:
    pass
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Создать класс и определить объекты этого класса: 2 человека - Том и Боб.

```python
class Person:
    pass

tom = Person()      # определение объекта tom
bob = Person()      # определение объекта bob
```

ФУНКЦИЯ DIR

Функция dir возвращает список имен, определенных в пространстве имен, переданном в качестве аргумента. Для класса она возвращает список всех имен, определенных в классе, включая поля и методы.

```python
class MyClass:
    name = "John"
    age = 30

print(dir(MyClass))
# Результат: ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'age', 'name']
```

Поля в классе - это переменные, определенные внутри класса.

Они могут быть доступны как через имя класса, так и через имя экземпляра класса. Если поле определено в экземпляре класса, то оно будет иметь уникальное значение для каждого экземпляра.

```python
class MyClass:
    class_field = "Class Field"
```


```
my_object = MyClass()
my_object.instance_field = "Instance Field"

print(MyClass.class_field)
# Результат: "Class Field"

print(my_object.class_field)
# Результат: "Class Field"

print(my_object.instance_field)
# Результат: "Instance Field"
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

1. Какие вы запомнили концепции ООП? Что они означают?

2. Чем отличаются поля класса и поля экземпляра класса?

3. Сопоставьте применение с концепциями ООП:

1. В умном доме для всех электроприборов имеется команда "Включить". Такая же операция есть у умной настольной лампы.	A. Инкапсуляция
2. Умный чайник при получении команды "Включить" проверяет, есть ли в нем вода, и только при ее наличии начинает кипячение.	B. Наследование
3. В умном доме команду "Включить" можно дать любому электроприбору, и можно также дать команду "Включи мою любимую песню" колонке с голосовым помощником.	C. Полиморфизм
	D. Абстракция

ИНКАПСУЛЯЦИЯ

Инкапсуляция - это механизм, который позволяет скрыть детали реализации класса и предоставить доступ к полям и методам только через публичный интерфейс.

Создание экземпляра класса включает вызов конструктора класса с помощью оператора ().

```python
class MyClass:
    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print(f"Hello, {self.name}!")

my_object = MyClass("John")
my_object.say_hello()
# Результат: "Hello, John!"
```

Это лишь некоторые основные пункты введения в ООП на Python. ООП предоставляет более мощные возможности, такие как наследование, полиморфизм, абстракция и многое другое, которые могут быть использованы для создания сложных и структурированных программных решений.

ПРАКТИЧЕСКАЯ РАБОТА

1. У созданного класса Person посмотреть на уже определенные атрибуты.
   Добавить какие-то поля.
   Создать новый экземпляр класса.
   Обратиться к именам полей через имя экземпляра класса, имя метода.
   Рассмотреть отличия.
   Попробовать изменять различными способами поля.
   Удалить поля разными способами.

2. Создать класс Dog для представления собаки. Класс должен иметь атрибуты name (имя) и breed (порода), а также метод bark(), который выводит сообщение о том, что собака лает. Затем создать экземпляр класса Dog с заданным именем и породой и вызовите метод bark().

   Ожидаемый вывод:

   Шарик породы Дворняга лаял.

ПОЛЕЗНЫЕ МАТЕРИАЛЫ

1. Классы в Python
2. Python class: как работать с классами, разбор на примерах