Тема № 1.5

Основы ООП для разработки web-приложений

Лекция

СОДЕЙСТВИЕ ЗАНЯТОСТИ | Федеральный проект

Что такое ООП

Классы и объекты

```
class Human:
    def__init__(self, name: str, age: int):
        self.name = name
        self.age = age
    def showInfo(self):
        print(self.name, self.age)


petia = Human("Петя", 20)
vasia = Human("Вася", 32)
petia.showInfo()
vasia.showInfo()
```

Объектно-ориентированное программирование - это подход при котором программа воспринимается как набор сущностей, объектов, взаимодействующих друг с другом

Класс это единица ООП описывающая поведение и характеристики некоторой сущности, а объект (экземпляр) это конкретная реализация этой сущности

Инкапсуляция и сокрытие, конструктор класса

```
class Human:
    def __init__(self, name: str, age: int):
        self.__name = name
        self.__age = age
    def showInfo(self):
        print(self.__name, self.__age)
    def setAge(self, age):
        if age > 0:
            self.__age = age
    def getAge(self):
        return self.__age


petia = Human("Петя", 20)
petia.age = 30
petia.setAge(30)
petia.showInfo()
```

Инкапсуляция - объединение данных и методов работы с ними в одной сущности

Сокрытие - все характеристики (поля) должны быть скрыты (private) от изменений извне, работа с этими полями осуществляется посредством специальных методов

Методы get, set методы для получения и изменения скрытых полей

Конструктор - специальный метод, описывающий процесс создания объекта

Свойства

```
class Human:
    def __init__(self, name: str, age: int):
        self.__name = name
        self.__age = age
    def showInfo(self):
        print(self.__name, self.__age)
    @property
    def age(self):
        return self.__age
    @age.setter
    def age(self, age):
        if age > 0:
            self.__age = age

petia = Human("Петя", 20)
petia.age = 30
petia.showInfo()
```

Свойства это способ описать сеттер и геттер, но более удобным способом. При использовании свойств обращение к ним выглядит как обращение к полю.

Для создания свойства используются специальные декораторы

Наследование

```
class Human:
    def __init__(self, name):
        self._name = name


class Worker(Human):
    def __init__(self, name, work):
        super().__init__(name)
        self._work = work
    def info(self):
        print(f"Привет я {self._name},
              я - {self._work}")


vasia = Worker("Вася", "Программист")
#Привет я Вася, я - Программист
vasia.info()
```

Наследование - механизм при котором один класс наследует все поля и методы другого класса (расширяет его)

При вызове конструктора наследника необходимо, так же, вызывать и конструктор родительского класса (суперкласса) путём обращения super()....

При наследовании все приватные поля будут недоступны наследнику, однако есть защищённые поля их отличие только в том, что они будут доступны наследнику, но не доступны извне

Переопределение методов, полиморфизм

```
class Human:
    def __init__(self, name):
        self._name = name
    def info(self):
        print(f"Привет я {self._name}")


class Worker(Human):
    def __init__(self, name, work):
        super().__init__(name)
        self._work = work
    def info(self):
        print(f"Привет я {self._name}, я
- {self._work}")
```

Дочерний класс может переопределить метод базового (родительского) класса. В таком случае при обращении к конкретному классу будет вызван его собственный метод

Полиморфизм, поведение класса при котором он может принимать как форму себя (вызывать свои методы) так и форму родителя (вызывать соответствующие методы базового класса)

Полиморфизм

```
class Human:
    def __init__(self, name):
        self._name = name
    def info(self):
        print(f"Привет, я {self._name}")


class Worker(Human):
    def __init__(self, name):
        super().__init__(name)
    def info(self):
        print(f"Привет, я работник {self._name}")


def function(obj):
    obj.info()
```

В случае использования полиморфизма функция может принять любого наследника класса и вызвать его собственный метод

Строковое представление объекта

```
class Human:
    def___init__(self, name):
        self._name = name
    def info(self):
        print(f"Привет я {self._name}")
    def_str__(self) -> str:
        return self._name


vasia = Human("Вася")
print(vasia)
```

При попытке отправить объект в функцию print() мы получим его адрес в памяти, если мы хотим это изменить, то нужно реализовать специальный метод __str__который позволяет описать конвертацию объекта в строку

Ваши вопросы