Python
2-месяц 2-урок

Тема: Основные принципы ООП ,
Наследование , Инкапсуляция, Полиморфизм, Абстрация

Абстракция

Абстракция — это выделение основных, наиболее значимых характеристик объекта и игнорирование второстепенных.

Любой составной объект реального мира — это абстракция. Говоря "ноутбук", вам не требуется дальнейших пояснений, вроде того, что это организованный набор пластика, металла, жидкокристаллического дисплея и микросхем. Абстракция позволяет игнорировать нерелевантные детали, поэтому для нашего сознания это один из главных способов справляться со сложностью реального мира. Если б, подходя к холодильнику, вы должны были иметь дело с отдельно металлом корпуса, пластиковыми фрагментами, лакокрасочным слоем и мотором, вы вряд ли смогли бы достать из морозилки замороженную клубнику.

Пример Абстракции

```
class Laptop:
    def __init__(self, size, color):
        self.size = size
        self.color = color
```

Наследование

Это способность одного класса расширять понятие другого, и главный механизм повторного использования кода в ООП. Вернёмся к нашему автосимулятору. На уровне абстракции "Автотранспорт" мы не учитываем особенности каждого конкретного вида транспортного средства, а рассматриваем их "в целом". Если же более детализировано приглядеться, например, к грузовикам, то окажется, что у них есть такие свойства и возможности, которых нет ни у легковых, ни у пассажирских машин. Но, при этом, они всё ещё обладают всеми другими характеристиками, присущими автотранспорту.

Мы могли бы сделать отдельный класс "Грузовик", который является наследником "Автотранспорта". Объекты этого класса могли бы определять все прошлые атрибуты (цвет, год выпуска), но и получить новые. Для грузовиков это могли быть грузоподъёмность, снаряженная масса и наличие жилого отсека в кабине. А методом, который есть только у грузовиков, могла быть функция сцепления и отцепления прицепа.

Пример Наследование

```
class Laptop:
    def __init__(self, size, color):
        self.size = size
        self.color = color


class GameLaptop(Laptop):
    def __init__(self, size, color, powerful_graphics_card, colorful_keyboard):
        super().__init__(size, color)
        self.graphic = powerful_graphics_card
        self.colorful_keyboard = colorful_keyboard
```

Инкапсуляция

Инкапсуляция — это ещё один принцип, который нужен для безопасности и управления сложностью кода. Инкапсуляция блокирует доступ к деталям сложной концепции. Абстракция подразумевает возможность рассмотреть объект с общей точки зрения, а инкапсуляция не позволяет рассматривать этот объект с какой-либо другой.

Вы разработали для муниципальных служб класс "Квартира". У неё есть свойства вроде адреса, метража и высоты потолков. И методы, такие как получение информации о каждом из этих свойств и, главное, метод, реализующий постановку на учёт в Росреестре. Это готовая концепция, и вам не нужно чтобы кто-то мог добавлять методы "открыть дверь" и "получить место хранения денег". Это А) Небезопасно и Б) Избыточно, а также, в рамках выбранной реализации, не нужно. Работникам Росреестра не требуется заходить к вам домой, чтобы узнать высоту потолков — они пользуются только теми документами, которые вы сами им предоставили.

Пример инкапсуляции

```
class SomeClass:
    def _private(self):
        print("Это внутренний метод объекта")


obj = SomeClass()
obj._private() # это внутренний метод объекта
```

Пример инкапсуляции

```
class SomeClass():
    def __init__(self):
        self.__param = 42 # защищенный атрибут


obj = SomeClass()
obj.__param # AttributeError: 'SomeClass' object has no attribute '__param'
obj._SomeClass__param # 42
```

Полиморфизм

Полиморфизм подразумевает возможность нескольких реализаций одной идеи. Простой пример: у вас есть класс "Персонаж", а у него есть метод "Атаковать". Для воина это будет означать удар мечом, для рейнджера — выстрел из лука, а для волшебника — чтение заклинания "Огненный Шар". В сущности, все эти три действия — атака, но в программном коде они будут реализованы совершенно по-разному.

Примеры Полиморфизма

```
class Mammal:
    def move(self):
        print('Двигается')


class Kangaroo(Mammal):
    def move(self):
        print('Прыгает')
```

Выводы :

```
animal = Mammal()
animal.move() # Двигается
kangaroo = Kangaroo()
Kangaroo.move() # Прыгает
```

Примеры Полиморфизма

```
class English:
  def greeting(self):
    print ("Hello")

class French:
  def greeting(self):
    print ("Bonjour")

def intro(language):
  language.greeting()
```

Выводы :

```
john = English()
gerard = French()
intro(john) # Hello
intro(gerard) # Bonjour
```

Домашняя Работа

Задача №1 Наследование (+ 3 класса объекта)

1. Создать трехступенчатую концепцию (дед-отец-ребенок) любого примера который вам ближе
2. Все три класса должны иметь свои особенные методы и атрибуты как минимум два доп метода у каждого класса
3. Также создать хотя бы по одному объекту к каждому классу

Задача №2 Инкапсуляция (+ 1 класс объекта)

1. Создать один класс в котором вы пропишете по одному методу (внутреннего и защищенного)
2. В этом классе должно быть также по одному атрибуту (внутреннего и защищенного)

Задача № 3 Полиморфизм без наследования(+ 3 класса объекта)

1. Создать три разных класса в котором будут одинаковые методы по названию например (attack)
2. Но логика этого самого метода будут разные как в случае примера с мечником , лучником и волшебником

Задача № 4 Полиморфизм с наследованием (+ 3 класса объекта)

1. Все тоже самое как в случае с Полиморфизмом без наследование , единственное различие здесь присутствует наследование трехступенчатой концепций (дед-отец-ребенок)