

**РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**Федеральное государственное автономное образо-**  
**вательное учреждение высшего образования**  
**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**  
**«Элементы объектно-ориентированного программирования в языке**  
**Python»**

**Отчет по лабораторной работе № 4.3**  
**по дисциплине «Объектно-ориентированное программирование»**

Выполнил студент группы ИВТ-б-о-21-1  
Богадуров Василий Игоревич.

« » \_\_\_\_\_ 20\_\_г.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2023

**Цель работы:** приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

**Порядок выполнения работы:**

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*

 ItsMyLife1337 /

Great repository names are short and memorable. Need inspiration? How about [friendly-system](#) ?

Description (optional)



- ☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**  
You choose who can see and commit to this repository.

Рисунок 1 – Создание репозитория

2. Выполнение клонирования созданного репозитория.

```
c:\Users\Admin\Desktop\git>git clone https://github.com/ItsMyLife1337/OOP_4.3.git
Cloning into 'OOP_4.3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
c:\Users\Admin\Desktop\git>
```

Рисунок 2 – Клонирование репозитория

3. Организация репозитория в соответствии с моделью ветвления git-flow.

```

c:\Users\Admin\Desktop\git>cd /d c:\users\admin\desktop\git\OOP_4.3

c:\Users\Admin\Desktop\git\OOP_4.3>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Admin/Desktop/git/OOP_4.3/.git/hooks]

c:\Users\Admin\Desktop\git\OOP_4.3>_

```

Рисунок 3 – Ветвление по модели git-flow

#### 4. Проработка примера лабораторной работы.

```

PS C:\Users\Admin\Desktop\git\OOP_4.2\My_project> & C:/Users/Admin/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Admin/Desktop/git/OOP_4.3/Examples/Example_3_Subclass_realization.py
True
True

```

Рисунок 4 – Результат выполнения примера

**Разработайте программу по следующему описанию:** В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня. В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки. Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, увеличивается уровень. Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

```

PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project> & C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Admin/Desktop/git/OOP_4.3/My_Project/Standart_task.py
Солдат 40 идет за героем 1
Идентификационный номер солдата: 40
Идентификационный номер героя: 1
PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project>

```

Рисунок 5 – Результат выполнения задания

### Выполнение индивидуального задания.

**Задание 1. Вариант – 1.** Составить программу с использованием иерархии классов. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанных классов.

1. Создать базовый класс Car (машина), характеризуемый торговой маркой (строка), числом цилиндров, мощностью. Определить методы переназначения и изменения мощности. Создать производный класс Lorry (грузовик), характеризуемый также грузоподъемностью кузова. Определить функции переназначения марки и изменения грузоподъемности.

```

PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project> & C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Admin/Desktop/git/OOP_4.3/My_Project/ind1.py
Brand: Toyota, Cylinders: 4, Power: 200 hp
Brand: Honda, Cylinders: 4, Power: 250 hp
Brand: Volvo, Cylinders: 6, Power: 350 hp, Capacity: 10 tons
Brand: MAN, Cylinders: 6, Power: 350 hp, Capacity: 15 tons
PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project>

```

Рисунок 6 – Результат выполнения индивидуального задания  
№1

### Задание 2. Вариант 1

Требуется реализовать абстрактный базовый класс, определив в нем абстрактные методы и свойства. Эти методы определяются в производных классах. В базовых классах должны быть объявлены абстрактные методы ввода/вывода, которые реализуются в производных классах.

Вызывающая программа должна продемонстрировать все варианты вызова переопределенных абстрактных методов. Написать функцию вывода, получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов.

1. Создать абстрактный базовый класс Figure с абстрактными методами вычисления площади и периметра. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие – в производных. Площадь трапеции:

$$S = (a + b) \times h / 2.$$

```
PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project> & C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Admin/Desktop/git/OOP_4.3/My_Project/ind2.py
Rectangle:
Area: 50
Perimeter: 30
Circle:
Area: 153.86
Perimeter: 43.96
Trapezium:
Area: 16.0
Perimeter: None
PS C:\Users\Admin\Desktop\git\OOP_4.3\My_Project>
```

Рисунок 7 – Результат выполнения индивидуального задания 2

## Ответы на контрольные вопросы:

### 1. Что такое наследование как оно реализовано в языке Python?

#### Ответ:

Наследование в объектно-ориентированном программировании (ООП) - это способ создания нового класса на основе уже существующего класса. Класс, который наследует свойства и методы от другого класса, называется производным классом или подклассом, а класс, от которого наследуют, называется базовым классом или суперклассом.

В Python наследование реализуется очень просто. Вот пример:

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

    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

# Пример использования наследования
dog = Dog("Buddy")
print(dog.name) # Выведет "Buddy"
print(dog.speak()) # Выведет "Woof!"

cat = Cat("Whiskers")
```

```
print(cat.name) # Выведет "Whiskers"  
print(cat.speak()) # Выведет "Meow!"
```

В этом примере классы Dog и Cat наследуют от базового класса Animal. Это означает, что классы Dog и Cat наследуют атрибуты и методы, определенные в классе Animal, такие как конструктор init и метод speak. При этом, классы-наследники могут переопределить методы базового класса, чтобы адаптировать их под свои нужды.

## 2. Что такое полиморфизм и как он реализован в языке Python?

**Ответ:**

**Полиморфизм — это принцип объектно-ориентированного программирования, который позволяет объектам различных типов обрабатываться с использованием общего интерфейса. Это означает, что объекты могут проявлять различное поведение в зависимости от их типа или класса.**

В Python полиморфизм может быть достигнут несколькими способами:

**Полиморфизм параметров функций:** Функции могут принимать аргументы различных типов, и их поведение может зависеть от типа переданных объектов.

```
def print_type(obj):  
    print(type(obj))  
  
print_type(5)    # <class 'int'>  
print_type("Hello") # <class 'str'>
```

**Полиморфизм методов:** Различные классы могут предоставлять свои собственные реализации методов с одинаковыми именами, что позволяет использовать их полиморфически.

```
class Dog:  
    def speak(self):  
        return "Woof!"  
  
class Cat:  
    def speak(self):
```

```
    return "Meow!"
```

```
def animal_sound(animal):  
    return animal.speak()
```

```
dog = Dog()  
cat = Cat()
```

```
print(animal_sound(dog)) # Woof!  
print(animal_sound(cat)) # Meow!
```

Полиморфизм через магические методы (dunder-методы): Магические методы, такие как `len`, `add`, `eq`, и другие, позволяют объектам поддерживать стандартные операции. Это делает их полиморфными по отношению к встроенным функциям и операторам.

```
class Circle:  
    def init(self, radius):  
        self.radius = radius  
  
    def add(self, other):  
        return Circle(self.radius + other.radius)
```

```
circle1 = Circle(3)  
circle2 = Circle(5)  
result_circle = circle1 + circle2  
print(result_circle.radius) # 8
```

Все эти методы реализуют идею полиморфизма в Python, позволяя объектам различных типов взаимодействовать с использованием общих интерфейсов.

### **3. Что такое "утиная" типизация в языке программирования Python?**

**Ответ:**

**"Утиная" типизация (duck typing) в языке программирования**



**Python относится к философии, согласно которой тип или класс объекта не важен, пока объект поддерживает необходимые методы или свойства. Принцип "утиной" типизации формулируется как "если это выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, и есть утка".**

**Это означает, что в Python важнее не явное указание типов данных, а возможность объекта выполнять определенные действия.** Например, если объект имеет методы, необходимые для выполнения определенной операции, то этот объект может использоваться в контексте этой операции, независимо от его фактического типа или класса.

Вот пример "утиной" типизации в Python:

```
class Duck:
    def quack(self):
        print("Quack!")

class Person:
    def quack(self):
        print("I'm quacking like a duck!")

def in_the_forest(entity):
    entity.quack()

duck = Duck()
person = Person()

in_the_forest(duck) # Выведет "Quack!"
in_the_forest(person) # Выведет "I'm quacking like a duck!"
```

В этом примере функция `inthe_forest` принимает объект и вызывает метод

quack у этого объекта. Объекты Duck и Person имеют разные типы, но оба поддерживают метод quack, поэтому они могут быть использованы в контексте этой функции.

#### **4. Каково назначение модуля abc языка программирования Python?**

**Ответ:**

**Модуль abc (Abstract Base Classes) в языке программирования Python предназначен для создания абстрактных базовых классов (ABC).** Абстрактный базовый класс в Python - это класс, который может содержать абстрактные методы, то есть методы, которые не имеют реализации, но должны быть переопределены в производных классах.

Назначение модуля abc заключается в том, чтобы обеспечить механизм определения интерфейсов и структур данных, которые должны быть реализованы в производных классах. Это помогает в организации кода, улучшает его читаемость и обеспечивает единообразие интерфейсов.

Пример использования модуля abc для создания абстрактного базового класса:

```
from abc import ABC, abstractmethod
```

```
class Shape(ABC):
```

```
    @abstractmethod
```

```
    def area(self):
```

```
        pass
```

```
    @abstractmethod
```

```
    def perimeter(self):
```

```
        pass
```

```
class Circle(Shape):
```

```
    def __init__(self, radius):
```

```
        self.radius = radius
```

```
def area(self):  
    return 3.14 * self.radius * self.radius
```

```
def perimeter(self):  
    return 2 * 3.14 * self.radius
```

```
class Rectangle(Shape):  
    def __init__(self, width, height):  
        self.width = width  
        self.height = height
```

```
def area(self):  
    return self.width * self.height
```

```
def perimeter(self):  
    return 2 * (self.width + self.height)
```

```
# Пример использования абстрактного базового класса
```

```
circle = Circle(5)
```

```
print(circle.area()) # Выведет площадь круга
```

```
print(circle.perimeter()) # Выведет периметр круга
```

```
rectangle = Rectangle(4, 6)
```

```
print(rectangle.area()) # Выведет площадь прямоугольника
```

```
print(rectangle.perimeter()) # Выведет периметр прямоугольника
```

В этом примере класс Shape является абстрактным базовым классом, который содержит абстрактные методы `area()` и `perimeter()`. Классы Circle и Rectangle являются производными классами, которые переопределяют эти методы. Таким образом, модуль abc позволяет создавать структуры, которые обеспечивают единообразие интерфейсов для классов, реализующих определенную функциональность.

## 5. Как сделать некоторый метод класса абстрактным?

**Ответ:**

В Python абстрактный метод класса может быть создан с использованием декоратора `@abstractmethod` из модуля `abc` (Abstract Base Classes). Этот декоратор позволяет определить метод как абстрактный, то есть метод, который должен быть переопределен в производных классах.

Вот пример того, как сделать метод класса абстрактным:

```
from abc import ABC, abstractmethod
```

```
class AbstractClass(ABC):
```

```
    @abstractmethod
```

```
    def abstract_method(self):
```

```
        pass
```

```
class ConcreteClass(AbstractClass):
```

```
    def abstract_method(self):
```

```
        print("Implementing the abstract method")
```

```
# Этот код будет работать
```

```
instance = ConcreteClass()
```

```
instance.abstract_method() # Выведет "Implementing the abstract method"
```

```
# Этот код вызовет ошибку, так как мы пытаемся создать экземпляр абстрактного класса
```

```
# abstract_instance = AbstractClass()
```

В этом примере метод `abstract_method` в классе `AbstractClass` определен как абстрактный с помощью декоратора `@abstractmethod`. Затем в производном классе `ConcreteClass` этот метод переопределяется с конкретной реализацией.

## 6. Как сделать некоторое свойство класса абстрактным?

## Ответ:

В Python абстрактное свойство класса можно создать с использованием модуля abc (Abstract Base Classes) и декоратора @property в сочетании с абстрактным методом. Это позволяет определить свойство как абстрактное, то есть свойство, которое должно быть переопределено в производных классах.

Вот пример того, как сделать свойство класса абстрактным:

```
from abc import ABC, abstractmethod
```

```
class AbstractClass(ABC):
```

```
    @property
```

```
    @abstractmethod
```

```
    def abstract_property(self):
```

```
        pass
```

```
class ConcreteClass(AbstractClass):
```

```
    @property
```

```
    def abstract_property(self):
```

```
        return "Implementing the abstract property"
```

```
# Этот код будет работать
```

```
instance = ConcreteClass()
```

```
print(instance.abstract_property) # Выведет "Implementing the abstract  
property"
```

```
# Этот код вызовет ошибку, так как мы пытаемся создать экземпляр аб-  
страктного класса
```

```
# abstract_instance = AbstractClass()
```

В этом примере свойство abstract\_property в классе AbstractClass определено как абстрактное с помощью декоратора @property в сочетании с

@abstractmethod. Затем в производном классе ConcreteClass это свойство переопределяется с конкретной реализацией.

## **7. Каково назначение функции isinstance?**

**Ответ:** Функция isinstance() в Python используется для проверки принадлежности объекта к определенному типу или классу. Она возвращает True, если объект является экземпляром указанного класса или его подкласса, и False в противном случае.

Назначение функции isinstance() заключается в том, чтобы предоставить способ динамической проверки типа объекта во время выполнения программы. Это может быть полезно для выполнения различных действий в зависимости от типа объекта или для обеспечения корректной обработки различных типов данных.

**Вывод:** в результате выполнения лабораторной работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.