

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций  
«Наследование и полиморфизм в языке Python»**

**Отчет по лабораторной работе № 4.3  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. \_\_\_\_\_. « » 2023г.

Подпись студента \_\_\_\_\_

Работа защищена « \_\_\_\_\_ 20 \_\_\_\_ г.

Проверила Воронкин Р.А. \_\_\_\_\_

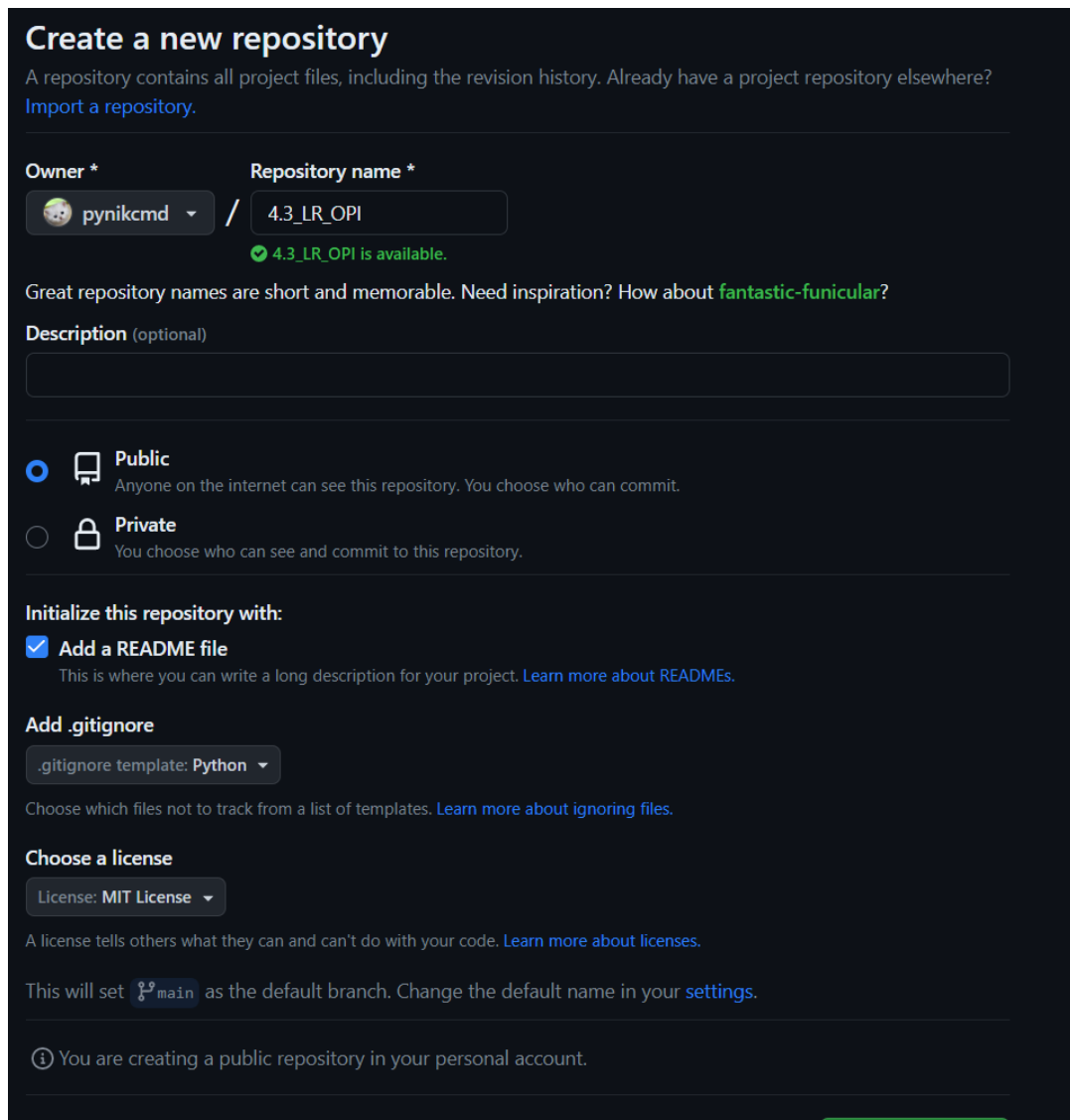
(подпись)

Ставрополь 2023

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

## Методика и порядок выполнения работы


1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет 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.](#)

---


**Owner \***  pynikcmd / **Repository name \***


✔ 4.3\_LR\_OPI is available.

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

**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.

---

**Initialize this repository with:**


☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).


---

 You are creating a public repository in your personal account.

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/4.3_LR_OPI.git
Cloning into '4.3_LR_OPI'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 5.12 KiB | 1.71 MiB/s, done.
```

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

 .gitignore [Update .gitignore](#)

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
D:\fgit\4.3_LR_OPI>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? [D:/fgit/4.3_LR_OPI/.git/hooks]
```

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  class Figure:
5      def __init__(self, color):
6          self.__color = color
7
8          @property
9          def color(self):
10             return self.__color
11
12         @color.setter
13         def color(self, c):
14             self.__color = c
15
16
17  class Rectangle(Figure):
18      def __init__(self, width, height, color):
19          super().__init__(color)
```

Figure > color()

Run: Primer\_1 x

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python.exe D  
10 20 green  
red

Пример 1

```
4
5 class Figure:
6     def __init__(self, color):
7         self.__color = color
8
9     @property
10    def color(self):
11        return self.__color
12
13    @color.setter
14    def color(self, c):
15        self.__color = c
16
17    def info(self):
18        print("Figure")
19        print("Color: " + self.__color)
20
21
22 class Rectangle(Figure):
    Rectangle > height() > else
```

Run: Primer\_2 x

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python.exe

Figure

Color: orange

Rectangle

Color: green

Width: 10

Height: 20

Area: 200

Пример 2

```
5 class Table:
6     def __init__(self, l, w, h):
7         self.length = l
8         self.width = w
9         self.height = h
10
11
12 class DeskTable(Table):
13     def square(self):
14         return self.width * self.length
15
16
17 if __name__ == '__main__':
18     t1 = Table(1.5, 1.8, 0.75)
19     t2 = DeskTable(0.8, 0.6, 0.7)
20     print(t2.square())
21
```

Run: Primer\_3 x

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python  
0.48

Пример 3

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  class Table:
5      def __init__(self, l, w, h):
6          self.length = l
7          self.width = w
8          self.height = h
9
10
11  class KitchenTable(Table):
12      def __init__(self, l, w, h, p):
13          Table.__init__(self, l, w, h)
14          self.places = p
15
16
17  if __name__ == '__main__':
18      t4 = KitchenTable(1.5, 2, 0.75, 6)
19
```

Table > \_\_init\_\_

Run: Primer\_4 ×

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python.

Process finished with exit code 0

Пример 4

```
11 class parent:
12     def geeks(self):
13         pass
14
15
16 class child(parent):
17     def geeks(self):
18         print("child class")
19
20
21 if __name__ == '__main__':
22     print(issubclass(child, parent))
23     print(isinstance(child(), parent))
24
```

Run: Primer\_5 ×


D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\pyth







True

True

Пример 5

```
4
5 # Python program invoking a
6 # method using super()
7 from abc import ABC
8
9
10 class R(ABC):
11     def rk(self):
12         print("Abstract Base Class")
13
14
15 class K(R):
16     def rk(self):
17         super().rk()
18         print("subclass")
19
20
```

Run:  Primer\_6 x

  D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\pyt  
 Abstract Base Class  
  subclass  


Пример 6



```
1
2 class Snake(Animal):
3     def move(self):
4         print("I can crawl")
5
6
7 class Dog(Animal):
8     def move(self):
9         print("I can bark")
10
11
12 class Lion(Animal):
13     def move(self):
14         print("I can roar")
15
16
17 if __name__ == '__main__':
18     c = Animal()
```

Traceback (most recent call last):  
File "D:\fgit\4.3\_LR\_OPI\Tasks\Primer 7.py", line 38, in <module>  
 c = Animal()  
 ^^^^^^^^^  
TypeError: Can't instantiate abstract class Animal with abstract method move

Process finished with exit code 1

Пример 7

```
117
118 ► if __name__ == '__main__':
119     r1 = Rational(3, 4)
120     r1.display()
121     r2 = Rational()
122     r2.read("Введите обыкновенную дробь: ")
123     r2.display()
124     r3 = r2.add(r1)
125     r3.display()
126     r4 = r2.sub(r1)
127     r4.display()
128     r5 = r2.mul(r1)
129     r5.display()
130     r6 = r2.div(r1)
131     r6.display()
132
```

if \_\_name\_\_ == '\_\_main\_\_'

Run: Primer\_8 ×

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python.exe

3/4

Введите обыкновенную дробь: 4/5

4/5

31/20

1/20

3/5

16/15

Пример 8

```
38
39 # Driver code
40 if __name__ == '__main__':
41     R = Triangle()
42     R.noofsides()
43     K = Quadrilateral()
44     K.noofsides()
45     R = Pentagon()
46     R.noofsides()
47     K = Hexagon()
48     K.noofsides()
49
```

if \_\_name\_\_ == '\_\_main\_\_'

Run: Primer\_9 X

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\py  
I have 3 sides  
I have 4 sides  
I have 5 sides  
I have 6 sides

Пример 9

```
34
35 if __name__ == '__main__':
36     # Driver code
37     R = Human()
38     R.move()
39     K = Snake()
40     K.move()
41     R = Dog()
42     R.move()
43     K = Lion()
44     K.move()
45
```

Run: Primer\_10 X

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\py  
I can walk and run  
I can crawl  
I can bark  
I can roar

Пример 10

8. Решите задачу:

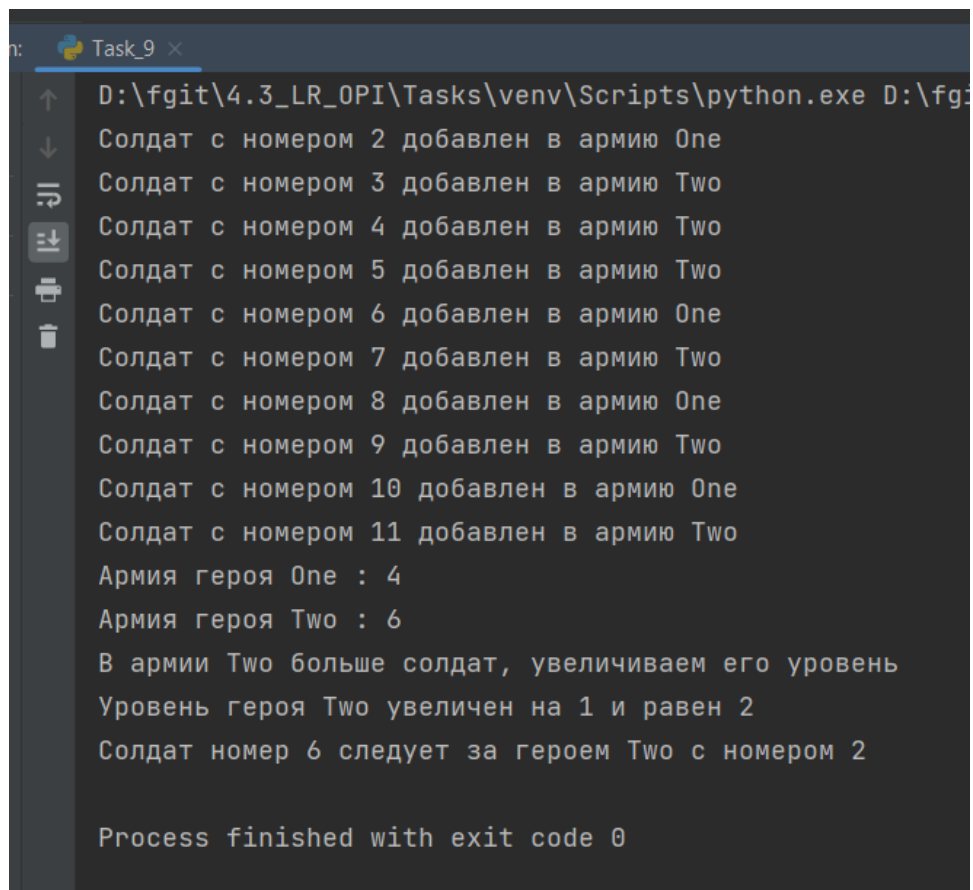
9. Разработайте программу по следующему описанию.

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.



```
n: Task_9 x
D:\fgit\4.3_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgi
Солдат с номером 2 добавлен в армию One
Солдат с номером 3 добавлен в армию Two
Солдат с номером 4 добавлен в армию Two
Солдат с номером 5 добавлен в армию Two
Солдат с номером 6 добавлен в армию One
Солдат с номером 7 добавлен в армию Two
Солдат с номером 8 добавлен в армию One
Солдат с номером 9 добавлен в армию Two
Солдат с номером 10 добавлен в армию One
Солдат с номером 11 добавлен в армию Two
Армия героя One : 4
Армия героя Two : 6
В армии Two больше солдат, увеличиваем его уровень
Уровень героя Two увеличен на 1 и равен 2
Солдат номер 6 следует за героем Two с номером 2

Process finished with exit code 0
```

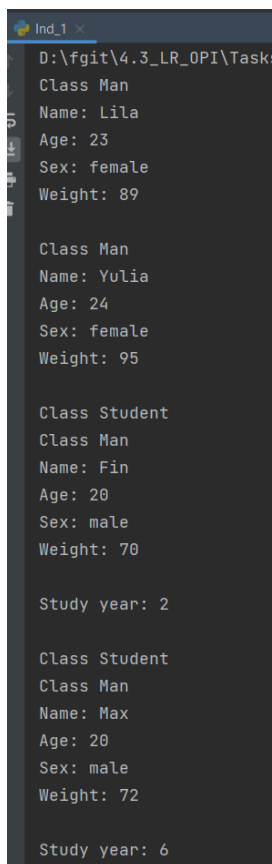
10. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

#### Задание 1

Составить программу с использованием иерархии классов. Номер варианта необходимо получить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанных классов.

#### Вариант – 5

Условие. Создать класс Man (человек), с полями: имя, возраст, пол и вес. Определить методы переназначения имени, изменения возраста и изменения веса. Создать производный класс Student, имеющий поле года обучения. Определить методы переназначения и увеличения года обучения.



```
Ind_1 x
D:\fgit\4.3_LR_OPI\Task
Class Man
Name: Lila
Age: 23
Sex: female
Weight: 89

Class Man
Name: Yulia
Age: 24
Sex: female
Weight: 95

Class Student
Class Man
Name: Fin
Age: 20
Sex: male
Weight: 70

Study year: 2

Class Student
Class Man
Name: Max
Age: 20
Sex: male
Weight: 72

Study year: 6
```

## Задание 2

В следующих заданиях 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.

## Вариант – 10

Условие. Создать абстрактный базовый класс Triad с виртуальными методами увеличения на 1. Создать производные классы Date (дата) и Time (время).

```
1  from abc import ABC, abstractmethod
2
3
4  class Triad(ABC):
5      def __init__(self, a, b, c):
6          self.a = a
7          self.b = b
8          self.c = c
9
10     @abstractmethod
11     def increase(self):
12         pass
13
14     @abstractmethod
15     def output(self):
16         pass
17
18
19  class Date(Triad):
20     def __init__(self, day, month, year):
```

Run: Ind\_2 x

D:\fgit\4.3\_LR\_OPI\Tasks\venv\Scripts\python.exe D:\fgit\4.3\_

Дата: 13.6.2023

Время: 11:31:46

Process finished with exit code 0

11. Зафиксируйте сделанные изменения в репозитории.

12. Выполните слияние ветки для разработки с веткой main / master.

13. Отправьте сделанные изменения на сервер GitHub.

### Контрольные вопросы

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

Синтаксически создание класса с указанием его родителя выглядит

так:

```
class      имя_класса(имя_родителя1,      [имя_родителя2,...,  
имя_родителя_n])
```

`super` – это ключевое слово, которое используется для обращения к родительскому классу.

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

Полиморфизм, как правило, используется с позиции переопределения методов базового класса в классе наследнике. Переопределение прописывается в классе-наследнике.

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

Утиная типизация – это концепция, характерная для языков программирования с динамической типизацией, согласно которой конкретный тип или класс объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает. Другими словами, при работе с объектом его тип не проверяется, вместо этого проверяются свойства и методы этого объекта. Такой подход добавляет гибкости коду, позволяет полиморфно работать с объектами, которые никак не связаны друг с другом и могут быть объектами разных классов. Единственное условие, чтобы все эти объекты поддерживали необходимый набор свойств и методов.

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

По умолчанию Python не предоставляет абстрактных классов. Python поставляется с модулем, который обеспечивает основу для определения абстрактных базовых классов (ABC), и имя этого модуля - `ABC`. `ABC` работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы.



5. Как сделать некоторый метод класса абстрактным?

Метод становится абстрактным, если он украшен ключевым словом `@abstractmethod`.

6. Как сделать некоторое свойство класса абстрактным?

Абстрактные классы включают в себя атрибуты в дополнение к методам, вы можете потребовать атрибуты в конкретных классах, определив их с помощью `@abstractproperty`.

7. Каково назначение функции `isinstance` ?

Встроенная функция `isinstance(obj, Cls)` , используемая при реализации методов арифметических операций и операций отношения, позволяет узнать что некоторый объект `obj` является либо экземпляром класса `Cls` либо экземпляром одного из потомков класса `Cls`.