

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет по лабораторной работе № 4.3
«Наследование и полиморфизм в языке Python»**

по дисциплине «Основы программной инженерии»

Выполнила:
Образцова Мария Дмитриевна,
2 курс, группа ПИЖ-б-о-21-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2023 г.

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

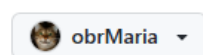
Методика и порядок выполнения работы

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 *



Repository name *

OPI_4.3

✓ OPI_4.3 is available.

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

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

.gitignore template: Python

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

Choose a license

License: MIT License

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

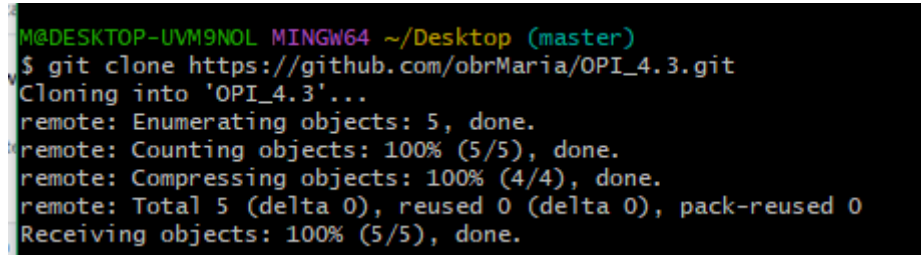
This will set `main` as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

Create repository

Рисунок –создание репозитория

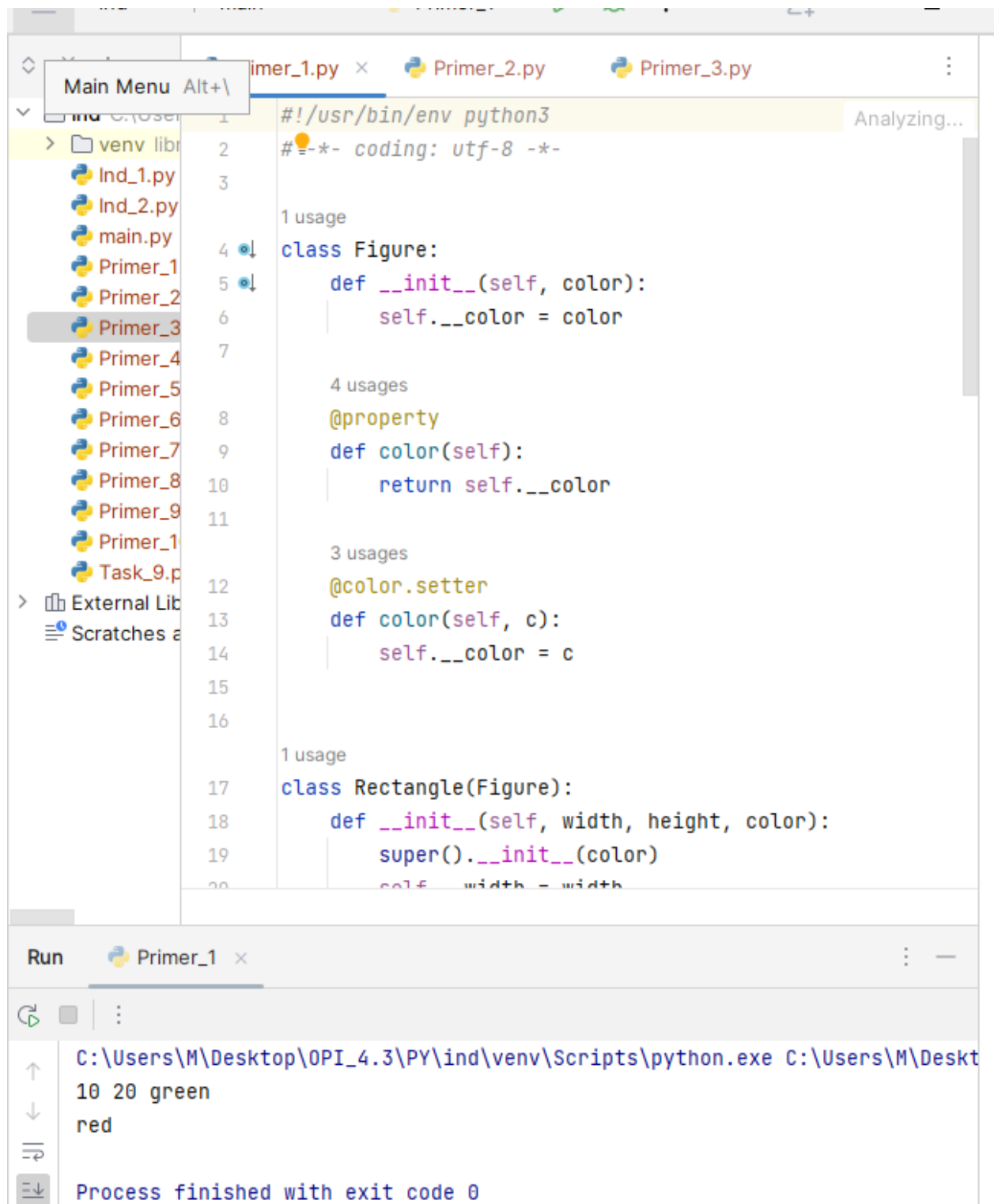
3. Выполните клонирование созданного репозитория. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.



```
M@DESKTOP-UVM9NOL MINGW64 ~/Desktop (master)
$ git clone https://github.com/obrMaria/OPI_4.3.git
Cloning into 'OPI_4.3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок – клонирование созданного репозитория

4. Проработать примеры лабораторной работы.



ind main Primer_2

Primer_2.py x Primer_3.py

ind C:\User

venv libr

Ind_1.py

Ind_2.py

main.py

Primer_1

Primer_2

Primer_3

Primer_4

Primer_5

Primer_6

Primer_7

Primer_8

Primer_9

Primer_1

Task_9.p

External Lib

Scratches a

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

1 usage

@color.setter

def color(self, c):

self.__color = c

1 usage

def info(self):

print("Figure")

print("Color: " + self.__color)

1 usage

class Rectangle(Figure):

def __init__(self, width, height, color):

super().__init__(color)

self.__width = width

self.__height = height

2 usages

@property

def width(self):

Figure > info()

Run Primer_2 x

C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use

Figure

Color: orange

Rectangle

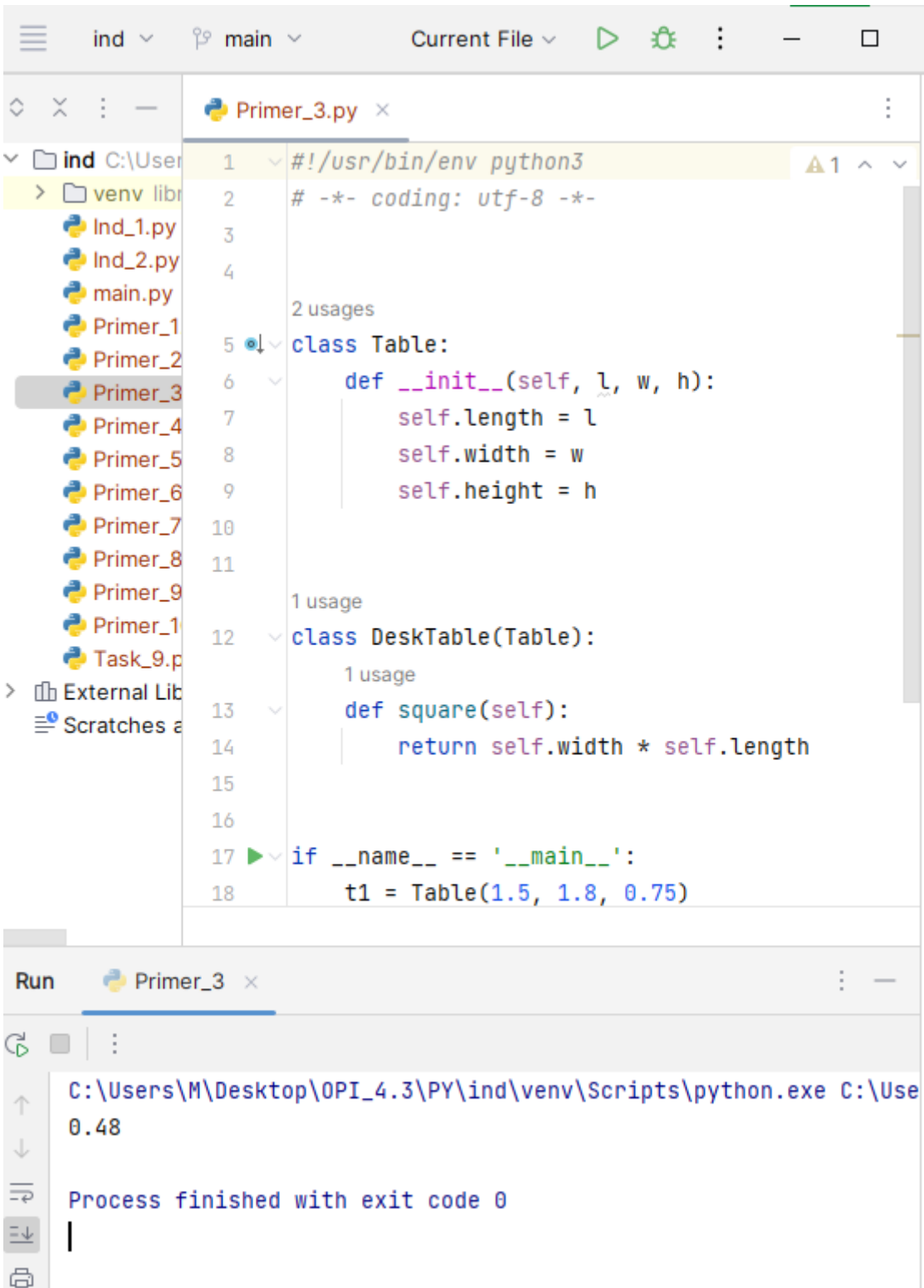
Color: green

Width: 10

Height: 20

Area: 200

Process finished with exit code 0



ind main Current File

Primer_4.py Primer_5.py Primer_6.py

ind C:\User

venv libr

Ind_1.py

Ind_2.py

main.py

Primer_1

Primer_2

Primer_3

Primer_4

Primer_5

Primer_6

Primer_7

Primer_8

Primer_9

Primer_1

Task_9.p

External Lib

Scratches a

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

2 usages
class Table:
    def __init__(self, l, w, h):
        self.length = l
        self.width = w
        self.height = h

1 usage
class KitchenTable(Table):
    def __init__(self, l, w, h, p):
        Table.__init__(self, l, w, h)
        self.places = p

if __name__ == '__main__':
    t4 = KitchenTable(1.5, 2, 0.75, 6)
```

Primer_4

C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use

Process finished with exit code 0

ind main Current File

Primer_4.py Primer_5.py Primer_6.py

ind C:\User
venv libr
Ind_1.py
Ind_2.py
main.py
Primer_1
Primer_2
Primer_3
Primer_4
Primer_5
Primer_6
Primer_7
Primer_8
Primer_9
Primer_1
Task_9.p
External Lib
Scratches a

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Python program showing
# implementation of abstract
# class through subclassing

3 usages
class parent:
    def geeks(self):
        pass

2 usages
class child(parent):
    def geeks(self):
        print("child class")
```

2

Run Primer_5

C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use
True
True
Process finished with exit code 0

ind main Current File

Primer_4.py Primer_5.py Primer_6.py

ind C:\User

venv libr

Ind_1.py Ind_2.py main.py Primer_1 Primer_2 Primer_3 Primer_4 Primer_5 Primer_6 Primer_7 Primer_8 Primer_9 Primer_1 Task_9.p External Lib Scratches a

1

#!/usr/bin/env python3

2

-*- coding: utf-8 -*-

3

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()

Run Primer_6

C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use

Abstract Base Class

subclass

Process finished with exit code 0

ind C:\User

venv libr

Ind_1.py

Ind_2.py

main.py

Primer_1

Primer_2

Primer_3

Primer_4

Primer_5

Primer_6

Primer_7

Primer_8

Primer_9

Primer_1

Task_9.p

External Lib

Scratches a

```
19 print("I can walk and run")
20
21
22 class Snake(Animal):
23     def move(self):
24         print("I can crawl")
25
26
27 class Dog(Animal):
28     def move(self):
29         print("I can bark")
30
31
32 class Lion(Animal):
33     def move(self):
34         print("I can roar")
35
36
37 if __name__ == '__main__':
38     c = Animal()
```

Run Primer_7 x

C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use

Traceback (most recent call last):

File "C:\Users\M\Desktop\OPI_4.3\PY\ind\Primer_7.py", line 38,

c = Animal()

^^^^^^^^

TypeError: Can't instantiate abstract class Animal with abstract

Process finished with exit code 1

ind main Current File

Primer_8.py Primer_9.py Primer_10.py

ind C:\User
venv libr
Ind_1.py
Ind_2.py
main.py
Primer_1
Primer_2
Primer_3
Primer_4
Primer_5
Primer_6
Primer_7
Primer_8
Primer_9
Primer_1
Task_9.p
External Lib
Scratches a

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

13 usages
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError()
        self.__numerator = abs(a)
        self.__denominator = abs(b)
        self.__reduce()
        # Сокращение дроби

2 usages
def __reduce(self):
    # Функция для нахождения наибольшего о
    def gcd(a, b):
        if a == 0:
```

34 ^ v

Run Primer_8

3/4

Введите обыкновенную дробь:

```
Primer_9.py x Primer_10.py
ind C:\User
> folder venv libr
  Ind_1.py
  Ind_2.py
  main.py
  Primer_1
  Primer_2
  Primer_3
  Primer_4
  Primer_5
  Primer_6
  Primer_7
  Primer_8
  Primer_9
  Primer_1
  Task_9.p
External Lib
Scratches a

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Python program showing
5  # abstract base class work
6  from abc import ABC, abstractmethod
7
8
9  class Polygon(ABC):
10     @abstractmethod
11     def noofsides(self):
12         pass
13
14
15  class Triangle(Polygon):
16     # overriding abstract method
17     def noofsides(self):
18         print("I have 3 sides")
```

```
Primer_8 x Primer_9 x
C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Us
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0
```

The image shows a screenshot of a Python IDE. The top pane displays a file named `Primer_10.py` with the following code:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  # Python program showing
6  # abstract base class work
7  from abc import ABC
8
9
10 class Animal(ABC):
11     def move(self):
12         pass
13
14
15 class Human(Animal):
16     def move(self):
17         print("I can walk and run")
18
```

The bottom pane shows the execution output for `Primer_10`. The command executed is `C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\Use`. The output is:

```
I can walk and run
I can crawl
I can bark
I can roar
Process finished with exit code 0
```

5. Выполнить индивидуальные задания.

Задание

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.

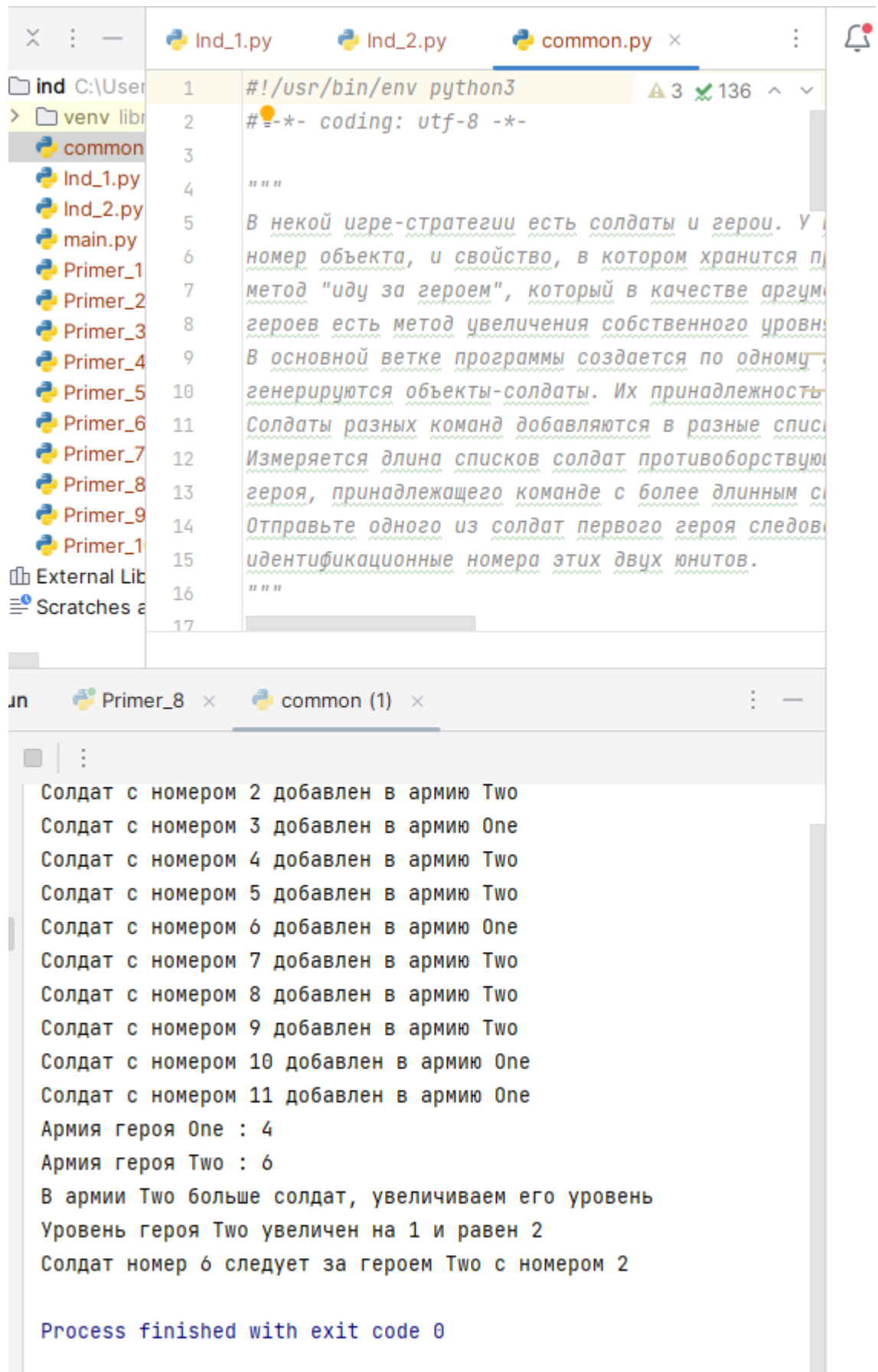


Рисунок – общее задание

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.

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.

The image shows a Python IDE with two windows. The top window, titled 'Ind_1.py' and 'Ind_2.py', displays the following code:

```
18 self.brand = brand
19 self.year = year
20
21 def start(self):
22     print(f"The {self.brand} is starti
23
24 def stop(self):
25     print(f"The {self.brand} is stoppi
26
27
28 # Определение производного класса Car (авт
29 class Car(Transport):
30     def __init__(self, brand, year, fuel_t
31         super().__init__(brand, year)
32         self.fuel_type = fuel_type
33
34 def refuel(self):
35     print(f"Refueling the {self.brand}
36
37 if __name__ == "__main__":
```

The bottom window, titled 'Primer_8' and 'Ind_1', shows the execution output:

```
C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\
The Toyota is starting.
Refueling the Toyota with gasoline fuel.
The Toyota is stopping.
The Honda is starting.
The Honda is doing a wheelie!
The Honda is stopping.

Process finished with exit code 0
```

Рисунок – Индивидуальное задание №1

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.

Создать абстрактный класс CURRENCY (валюта) для работы с денежными суммами. Определить виртуальные функции перевода в рубли и вывода на экран. Реализовать производные классы Dollar (доллар) и Euro (евро) со своими функциями перевода и вывода на экран.

The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Ind_2.py', contains the following Python code:

```
1 usage
50 def to_rubles(self):
51     return self.amount * 90.0 # Пример конвертации:
52
53 1 usage
54 def display(self):
55     print(f"€{self.amount}")
56
57 if __name__ == "__main__":
58     dollar = Dollar(10)
59     euro = Euro(5)
60
61     # Вызов методов класса Dollar
62     dollar.display()
63     rubles = dollar.to_rubles()
64     print(f"In rubles: {rubles}")
65
66     # Вызов методов класса Euro
67     euro.display()
68     rubles = euro.to_rubles()
69     print(f"In rubles: {rubles}")
70
```

The bottom window, titled 'Primer_8', shows the execution output:

```
C:\Users\M\Desktop\OPI_4.3\PY\ind\venv\Scripts\python.exe C:\U
$10
In rubles: 750.0
€5
In rubles: 450.0

Process finished with exit code 0
```

Рисунок – Индивидуальное задание №2

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`.