

**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(15)
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

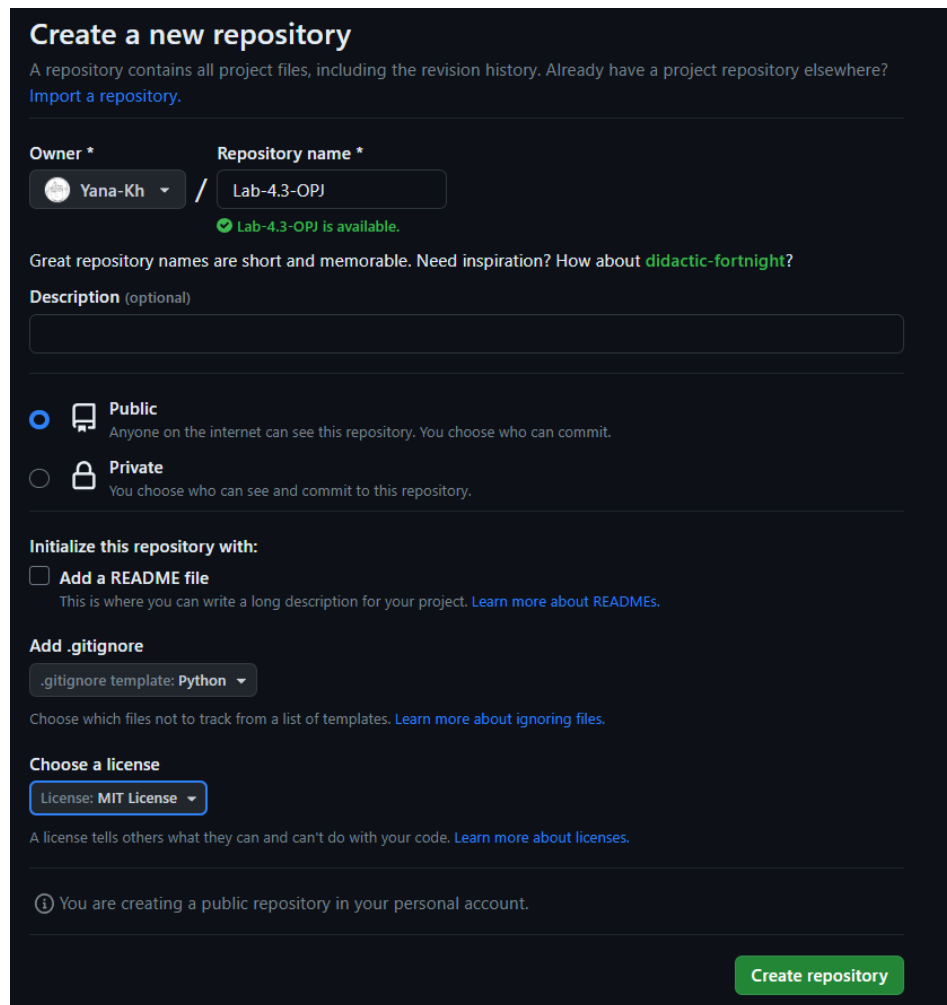
Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT 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 * Yana-Kh / Repository name * Lab-4.3-OPJ

✓ Lab-4.3-OPJ is available.

Great repository names are short and memorable. Need inspiration? How about **didactic-fortnight**?

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.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

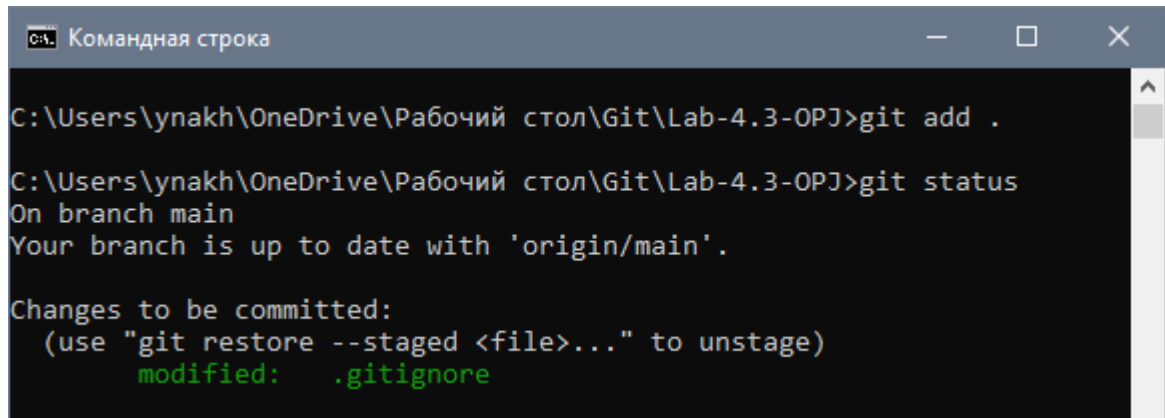
Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/Lab-4.3-OPJ.git
Cloning into 'Lab-4.3-OPJ'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2 – Клонирование репозитория

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



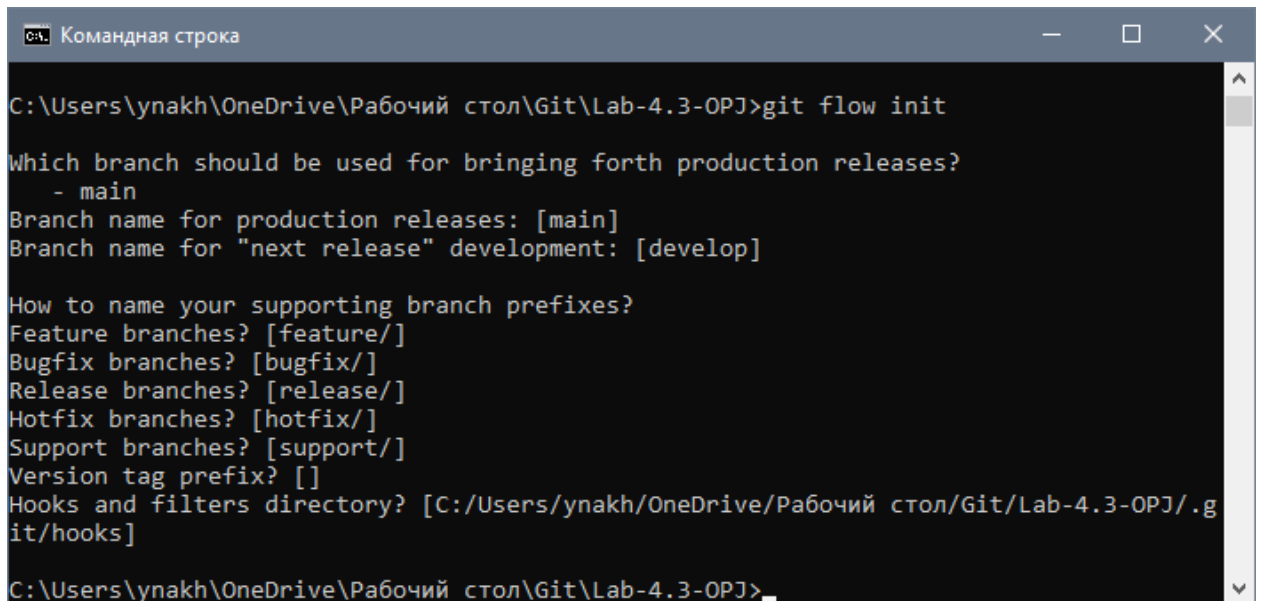
```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.3-OPJ>git add .

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.3-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
```

Рисунок 3 – Дополнение файла .gitignore

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



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.3-OPJ>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/ynakh/OneDrive/Рабочий стол/Git/Lab-4.3-OPJ/.git/hooks]
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

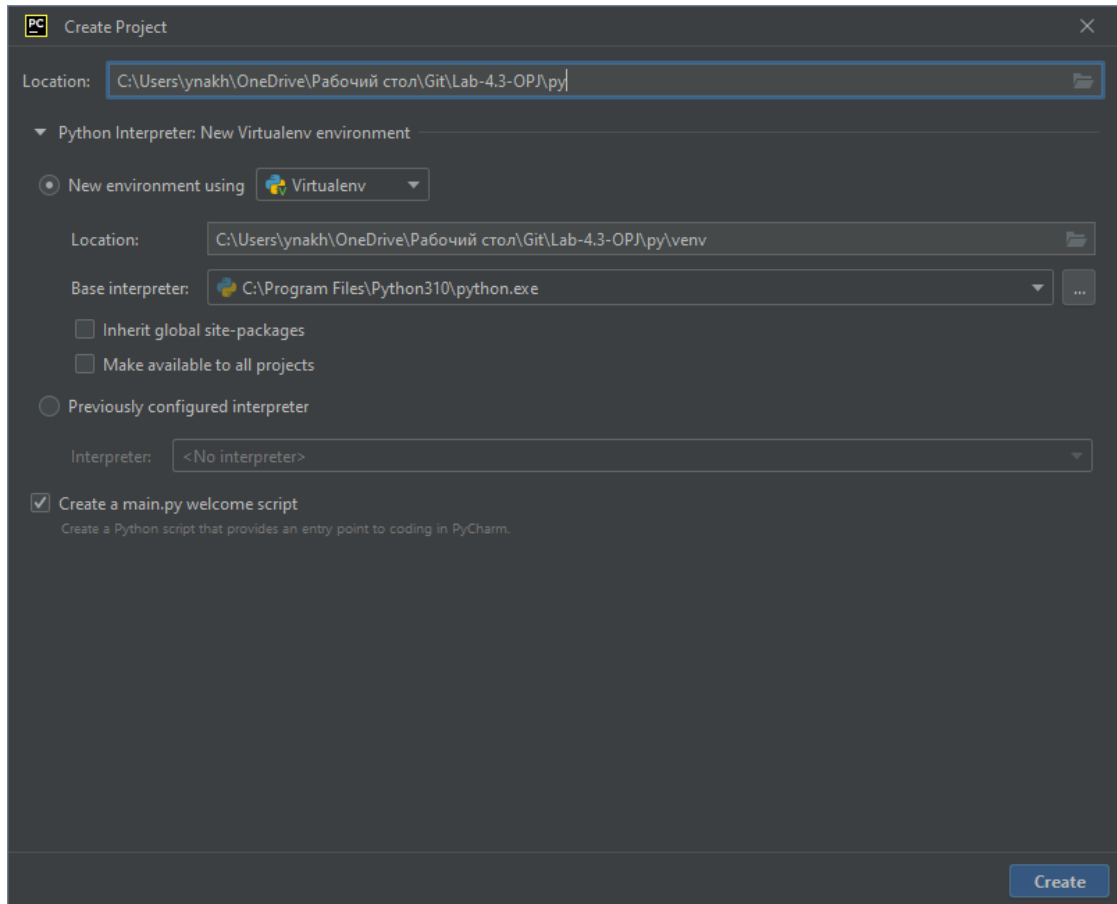


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1.

Код:

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

class Figure:
    def __init__(self, color):
        self.__color = color

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, c):
        self.__color = c
```

```

class Rectangle(Figure):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

if __name__ == '__main__':
    rect = Rectangle(10, 20, "green")
    print(rect.width,
          rect.height,
          rect.color
        )
    rect.color = "red"
    print(rect.color)

```

```

10 20 green
red

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

Пример 2.

Код:

```

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

class Figure:
    def __init__(self, color):

```

```

        self.__color = color

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, c):
        self.__color = c

    def info(self):
        print("Figure")
        print("Color: " + self.__color)

class Rectangle(Figure):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

    def info(self):
        print("Rectangle")
        print("Color: " + self.color)
        print("Width: " + str(self.width))
        print("Height: " + str(self.height))
        print("Area: " + str(self.area()))

if __name__ == '__main__':
    fig = Figure("orange")
    fig.info()
    rect = Rectangle(10, 20, "green")
    rect.info()

```

```
Figure
Color: orange
Rectangle
Color: green
Width: 10
Height: 20
Area: 200

Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

Пример 3.

Код:

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

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

class DeskTable(Table):
    def square(self):
        return self.width * self.length

if __name__ == '__main__':
    t1 = Table(1.5, 1.8, 0.75)
    t2 = DeskTable(0.8, 0.6, 0.7)
    print(t2.square())
```

```
0.48

Process finished with exit code 0
```

Рисунок 8 – Результат работы программы

Пример 4.

Код:

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

class Table:
```

```

def __init__(self, l, w, h):
    self.length = l
    self.width = w
    self.height = h

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

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

```

```
Process finished with exit code 0
```

Рисунок 9 – Результат работы программы

Пример 5.

Код:

```

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

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

class parent:
    def geeks(self):
        pass

class child(parent):
    def geeks(self):
        print("child class")

if __name__ == '__main__':
    print(issubclass(child, parent))
    print(isinstance(child(), parent))

```

```
True
True
```

```
Process finished with exit code 0
```

Рисунок 10 – Результат работы программы

Пример 6.

Код:

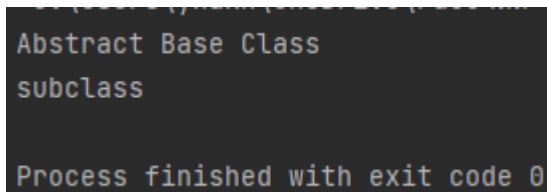
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Python program invoking a
# method using super()
from abc import ABC

class R(ABC):
    def rk(self):
        print("Abstract Base Class")

class K(R):
    def rk(self):
        super().rk()
        print("subclass")

if __name__ == '__main__':
    r = K()
    r.rk()
```



```
Abstract Base Class
subclass

Process finished with exit code 0
```

Рисунок 11 – Результат работы программы

Пример 7.

Код:

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

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

    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            if a == 0:
```

```

        return b
    elif b == 0:
        return a
    elif a >= b:
        return gcd(a % b, b)
    else:
        return gcd(a, b % a)

c = gcd(self.__numerator, self.__denominator)
self.__numerator //= c
self.__denominator //= c

@property
def numerator(self):
    return self.__numerator

@property
def denominator(self):
    return self.__denominator

# Прочитать значение дроби с клавиатуры. Дробь вводится
# как a/b.
def read(self, prompt=None):
    line = input() if prompt is None else input(prompt)
    parts = list(map(int, line.split('/', maxsplit=1)))
    if parts[1] == 0:
        raise ValueError()
    self.__numerator = abs(parts[0])
    self.__denominator = abs(parts[1])
    self.__reduce()

# Вывести дробь на экран
def display(self):
    print(f"{self.__numerator}/{self.__denominator}")

# Сложение обыкновенных дробей.
def add(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError()

# Вычитание обыкновенных дробей.
def sub(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError()

# Умножение обыкновенных дробей.
def mul(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError()

```

```

# Деление обыкновенных дробей.
def div(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator
        return Rational(a, b)
    else:
        raise ValueError()

# Отношение обыкновенных дробей.
def equals(self, rhs):
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
            (self.denominator == rhs.denominator)
    else:
        return False

def greater(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator
        return v1 > v2
    else:
        return False

def less(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator
        return v1 < v2
    else:
        return False

if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()
    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()
    r3 = r2.add(r1)
    r3.display()
    r4 = r2.sub(r1)
    r4.display()
    r5 = r2.mul(r1)
    r5.display()
    r6 = r2.div(r1)
    r6.display()

```

```
3/4
Введите обыкновенную дробь: 1/2
1/2
5/4
1/4
3/8
2/3

Process finished with exit code 0
```

Рисунок 12 – Результат работы программы

Пример 8.

Код:

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

# Python program showing
# abstract base class work
from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")

# Driver code
if __name__ == '__main__':
    R = Triangle()
```

```
R.noofsides()  
K = Quadrilateral()  
K.noofsides()  
R = Pentagon()  
R.noofsides()  
K = Hexagon()  
K.noofsides()
```

```
I have 3 sides  
I have 4 sides  
I have 5 sides  
I have 6 sides  
  
Process finished with exit code 0
```

Рисунок 13 – Результат работы программы

Пример 9.

Код:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
# Python program showing  
# abstract base class work  
from abc import ABC  
  
class Animal(ABC):  
    def move(self):  
        pass  
  
class Human(Animal):  
    def move(self):  
        print("I can walk and run")  
  
class Snake(Animal):  
    def move(self):  
        print("I can crawl")  
  
class Dog(Animal):  
    def move(self):  
        print("I can bark")  
  
class Lion(Animal):  
    def move(self):  
        print("I can roar")
```

```

if __name__ == '__main__':
    # Driver code
    R = Human()
    R.move()
    K = Snake()
    K.move()
    R = Dog()
    R.move()
    K = Lion()
    K.move()

```

```

I can walk and run
I can crawl
I can bark
I can roar

Process finished with exit code 0

```

Рисунок 14 – Результат работы программы

8. Для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.

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 29(9) 9. Создать класс `Pair` (пара чисел); определить методы изменения полей и вычисления произведения чисел. Определить производный класс `RightAngled` с полями-катетами. Определить методы вычисления гипотенузы и площади треугольника.

Код:

```

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

"""
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 29(9) 9. Создать класс Pair (пара чисел); определить методы изменения
полей

```

и вычисления произведения чисел. Определить производный класс `RightAngled` с полями-катетами.

Определить методы вычисления гипотенузы и площади треугольника.

```
"""
import math

class Pair:
    def __init__(self, a, b):
        self._a = a
        self._b = b

    @property
    def a(self):
        return self._a

    @a.setter
    def a(self, value):
        self._a = value

    @property
    def b(self):
        return self._b

    @b.setter
    def b(self, value):
        self._b = value

    def mul(self):
        return self._a * self._b

class RightAngled(Pair):
    def __init__(self, a, b):
        super().__init__(a, b)

    def hypotenuse(self):
        return math.sqrt(math.pow(self._a, 2) + math.pow(self._b, 2))

    def square(self):
        return self._a * self._b / 2

if __name__ == '__main__':
    # Родительский класс
    print("Родительский класс:")
    p1 = Pair(2, 10)
    print(f"a = {p1.a}, b = {p1.b}")
    print(f"Произведение = {p1.mul()}")
    p1.a = 12
    p1.b = 3
    print(f"a = {p1.a}, b = {p1.b}")
    print(f"Произведение = {p1.mul()}")

    # Дочерний класс
    print("\nДочерний класс:")
    p2 = RightAngled(3, 4)
    print(f"Катет 1 = {p2.a}, катет 2 = {p2.b}")
    print(f"Гипотенуза: {p2.hypotenuse():.2f}")
    print(f"Площадь: {p2.square():.2f}")
```

```
Родительский класс:  
a = 2, b = 10  
Произведение = 20  
a = 12, b = 3  
Произведение = 36  
  
Дочерний класс:  
Катет 1 = 3, катет 2 = 4  
Гипотенуза: 5.00  
Площадь: 6.00
```

Рисунок 15 – Результат работы программы

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.

Номер варианта необходимо получить у преподавателя.

Вариант 29(12) 12. Создать абстрактный базовый класс Integer (целое) с виртуальными арифметическими операциями и функцией вывода на экран. Определить производные классы Decimal (десятичное) и Binary (двоичное), реализующие собственные арифметические операции и функцию вывода на экран. Число представляется массивом, каждый элемент которого — цифра.

Код:

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

Номер варианта необходимо получить у преподавателя.

Вариант 29(12) 12. Создать абстрактный базовый класс Integer (целое) с виртуальными арифметическими операциями и функцией вывода на экран. Определить производные классы Decimal (десятичное) и Binary (двоичное), реализующие собственные арифметические операции и функцию вывода на экран. Число представляется массивом, каждый элемент которого — цифра.

```
"""
import decimal
from abc import ABC, abstractmethod

class Integer(ABC):
    @property
    @abstractmethod
    def number(self):
        return self._number

    @number.setter
    @abstractmethod
    def number(self, value):
        self._number = value

    @abstractmethod
    def add(self, other):
        pass

    @abstractmethod
    def sub(self, other):
        pass

    @abstractmethod
    def mul(self, other):
        pass

    @abstractmethod
    def display(self):
        print("\nБазовый абстрактный класс Integer")

    @abstractmethod
    def input(self):
        pass

class Decimal(Integer):
    def __init__(self, value):
        self._number = decimal.Decimal(value)

    @property
    def number(self):
        return self._number

    @number.setter
    def number(self, value):
        self._number = value
```

```

def add(self, other):
    # Реализация сложения для Decimal
    if isinstance(other, Decimal):
        return self.number + other.number
    else:
        raise ValueError()

def sub(self, other):
    # Реализация вычитания для Decimal
    if isinstance(other, Decimal):
        return self.number - other.number
    else:
        raise ValueError()

def mul(self, other):
    if isinstance(other, Decimal):
        return self.number * other.number
    else:
        raise ValueError()

def display(self):
    # Реализация вывода на экран для Decimal
    print(f"Decimal: {self.number}")

def input(self):
    self.number = decimal.Decimal(input("Введите десятичное (Decimal)
число: "))

class Binary(Integer):
    def __init__(self, value):
        self._number = bin(value)[2:]

    @property
    def number(self):
        return self._number

    @number.setter
    def number(self, value):
        self._number = bin(value)[2:]

    def add(self, other):
        # Реализация сложения для Binary
        if isinstance(other, Binary):
            return bin(int(self.number, 2) + int(other.number, 2))[2:]
        else:
            raise ValueError()

    def sub(self, other):
        # Реализация вычитания для Binary
        if isinstance(other, Binary):
            return bin(int(self.number, 2) - int(other.number, 2))[2:]
        else:
            raise ValueError()

    def mul(self, other):
        # Реализация умножения для Binary
        if isinstance(other, Binary):
            return bin(int(self.number, 2) * int(other.number, 2))[2:]
        else:
            raise ValueError()

    def display(self):

```

```

        # Реализация вывода на экран для Binary
        print(f"Binary: {self.number}")

    def input(self):
        # Реализация ввода для Binary
        self._number = bin(int(input("Введите число для перевода в Binary:
"))) [2:]

if __name__ == '__main__':
    print("Попытка создания экземпляра абстрактного класса:")
    try:
        int1 = Integer(4)
        int1.add(6)
    except:
        print("Err")

    print("\nКласс Decimal")
    dec1 = Decimal(10.23)
    dec2 = Decimal(5.52)
    print(f"Сложение: {dec1.add(dec2):.2f}")
    print(f"Вычитание: {dec1.sub(dec2):.2f}")
    print(f"Умножение: {dec1.mul(dec2):.2f}")
    dec1.display()
    dec2.display()
    print("Ввод")
    dec1.input()
    dec1.display()
    print("\nКласс Binary")
    bin1 = Binary(3)
    bin2 = Binary(2)
    print(f"Сложение: {bin1.add(bin2)}")
    print(f"Вычитание: {bin1.sub(bin2)}")
    print(f"Умножение: {bin1.mul(bin2)}")
    bin1.display()
    bin2.display()
    bin2.input()
    bin2.display()

```

```
Попытка создания экземпляра абстрактного класса:
Err

Класс Decimal
Сложение: 15.75
Вычитание: 4.71
Умножение: 56.47
Decimal: 10.230000000000000426325641456060111522674560546875
Decimal: 5.519999999999999573674358543939888477325439453125
Ввод
Введите десятичное (Decimal) число: 12.32
Decimal: 12.32

Класс Binary
Сложение: 101
Вычитание: 1
Умножение: 110
Binary: 11
Binary: 10
Введите число для перевода в Binary: 5
Binary: 101

Process finished with exit code 0
```

Рисунок 16 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

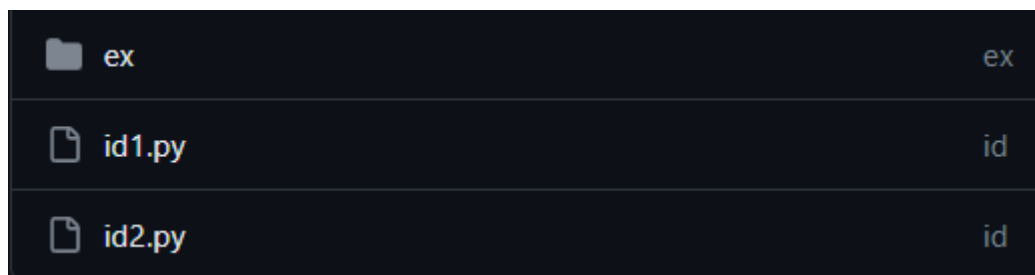


Рисунок 17 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

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

В организации наследования участвуют как минимум два класса: класс родитель и класс потомок. При этом возможно множественное наследование,

в этом случае у класса потомка может быть несколько родителей. Не все языки программирования поддерживают множественное наследование, но в Python можно его использовать. По умолчанию все классы в Python являются наследниками от `object`, явно этот факт указывать не нужно. Синтаксически создание класса с указанием его родителя выглядит так:

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

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

Полиморфизм в Python – это возможность объектов различных классов обрабатываться единообразно, используя общий интерфейс или методы. Это позволяет программисту использовать одну и ту же функцию или метод для разных типов данных.

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

"Утиная" типизация в языке программирования Python - это принцип, при котором тип объекта определяется не на основе его явного объявления, а на основе его возможностей или интерфейса. Это означает, что в Python важно, какие операции или методы может выполнять объект, а не его фактический тип данных.

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

`ABC` работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы. М

5. Как сделать некоторый метод класса абстрактным?

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

6. Как сделать некоторое свойство класса абстрактным?

Для того чтобы сделать свойство класса абстрактным в Python можно использовать модуль `abc` (Abstract Base Classes) из стандартной библиотеки.

Необходимо создать абстрактный базовый класс, унаследовав его от ABC (или ABCMeta) и определить абстрактное свойство с помощью декоратора @abstractmethod перед его объявлением в классе.

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.