

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

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

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

**Отчет по лабораторной работе № 4.1(13)  
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)

Ставрополь 2023

Цель работы: приобретение навыков по работе с классами и объектами 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 \*** **Repository name \***

Yana-Kh / Lab-4.1-OPJ

✔ Lab-4.1-OPJ is available.

Great repository names are short and memorable. Need inspiration? How about **literate-winner**?

**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.](#)

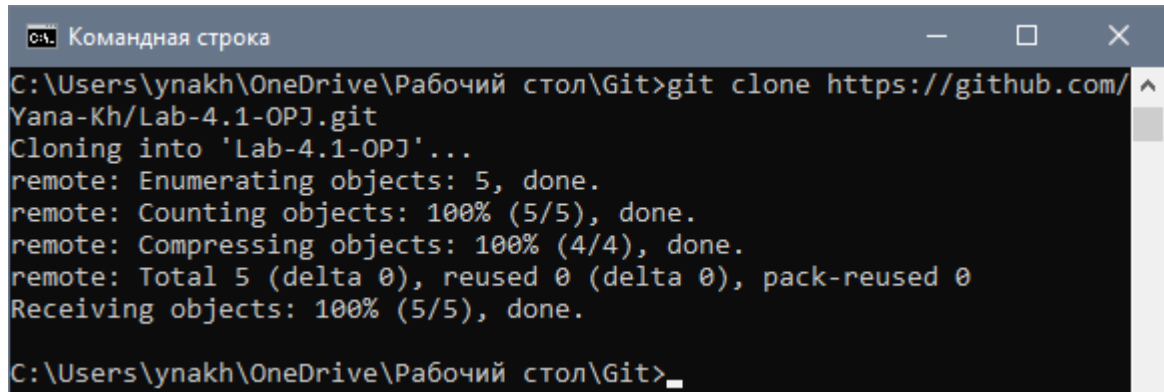
This will set **main** as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

**Create repository**

Рисунок 1 – Создание репозитория

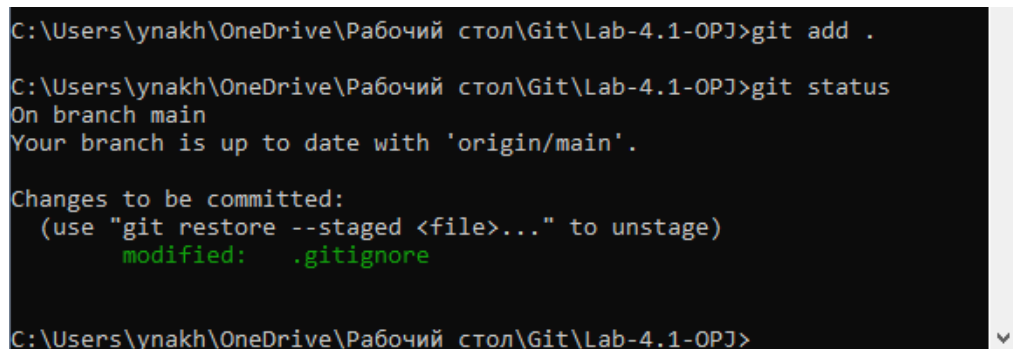
3. Выполните клонирование созданного репозитория.



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/Lab-4.1-OPJ.git
Cloning into 'Lab-4.1-OPJ'...
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.
C:\Users\ynakh\OneDrive\Рабочий стол\Git>
```

Рисунок 2 – Клонирование репозитория

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

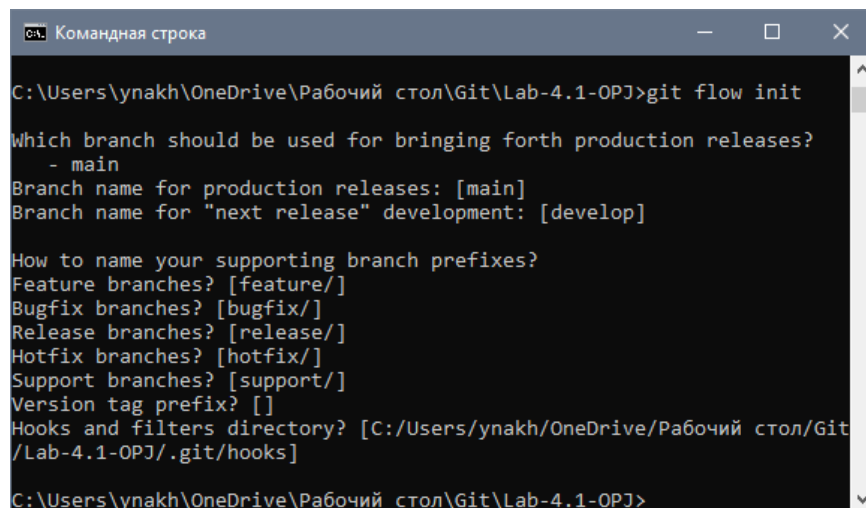


```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.1-OPJ>git add .
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.1-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.1-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

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



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.1-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-4.1-OPJ/.git/hooks]
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-4.1-OPJ>
```

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

## 6. Создайте проект PyCharm в папке репозитория.

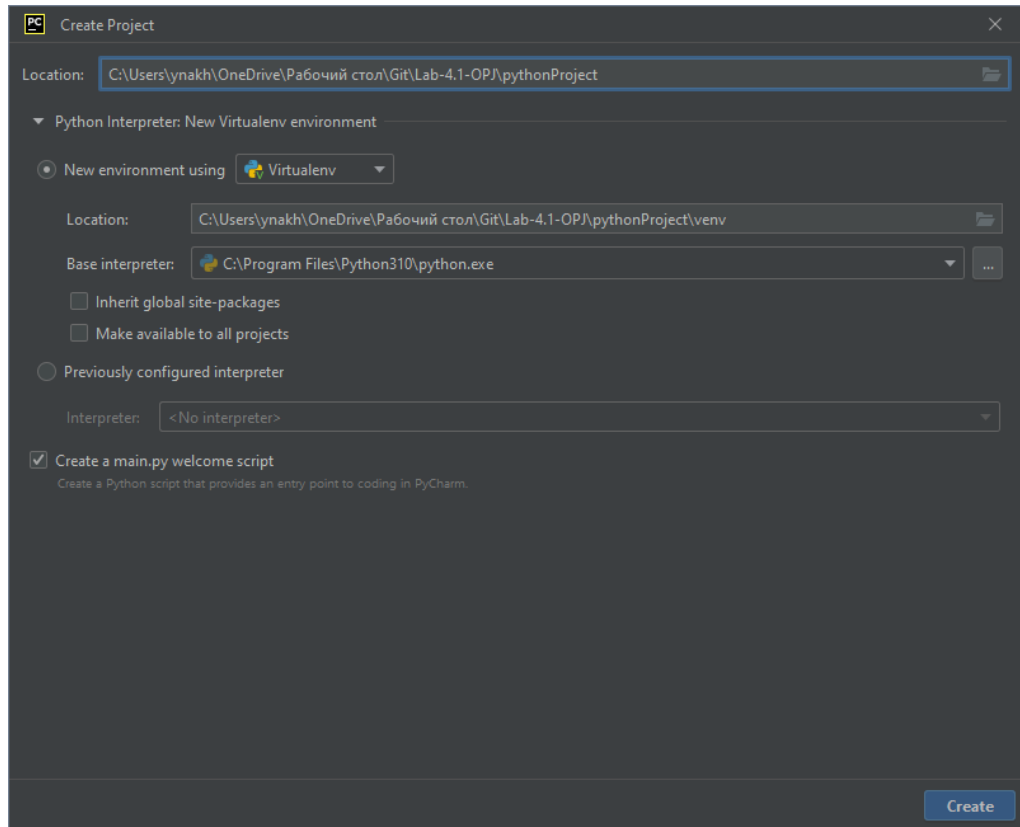


Рисунок 5 – Создание проекта PyCharm

## 7. Проработать примеры лабораторной работы.

### Пример 1.

Код:

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

class Book:
    material = "paper"
    cover = "paperback"
    all_books = []

if __name__ == '__main__':
    print(Book.material)
    print(Book.cover)
    print(Book.all_books)
```

```
paper
paperback
[]

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

## Пример 2.

Код:

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

class River:
    # список всех рек
    all_rivers = []

    def __init__(self, name, length):
        self.name = name
        self.length = length
        # добавляем текущую реку в список всех рек
        River.all_rivers.append(self)

if __name__ == '__main__':
    volga = River("Волга", 3530)
    seine = River("Сена", 776)
    nile = River("Нил", 6852)
    # далее печатаем все названия рек
    for river in River.all_rivers:
        print(river.name)
```

```
Волга
Сена
Нил

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

## Пример 3.

Код:

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

class River:
    all_rivers = []
```

```

def __init__(self, name, length):
    self.name = name
    self.length = length
    River.all_rivers.append(self)

def get_info(self):
    print("Длина {0} равна {1} км".format(self.name, self.length))

if __name__ == '__main__':
    volga = River("Волга", 3530)
    seine = River("Сена", 776)
    nile = River("Нил", 6852)
    volga.get_info()
    seine.get_info()
    nile.get_info()

```

```

Длина Волга равна 3530 км
Длина Сена равна 776 км
Длина Нил равна 6852 км

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

#### Пример 4.

Код:

```

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

class Ship:
    def __init__(self, name, capacity):
        self.name = name
        self.capacity = capacity
        self.cargo = 0

    def load_cargo(self, weight):
        if self.cargo + weight <= self.capacity:
            self.cargo += weight
            print("Loaded {} tons".format(weight))
        else:
            print("Cannot load that much")

    def unload_cargo(self, weight):
        if self.cargo - weight >= 0:
            self.cargo -= weight
            print("Unloaded {} tons".format(weight))
        else:
            print("Cannot unload that much")

    def name_captain(self, cap):
        self.captain = cap
        print("{} is the captain of the {}".format(self.captain, self.name))

```

```

if __name__ == '__main__':
    black_pearl = Ship("Black Pearl", 800)
    black_pearl.name_captain("Jack Sparrow")
    print(black_pearl.captain)
    black_pearl.load_cargo(600)
    black_pearl.unload_cargo(400)
    black_pearl.load_cargo(700)
    black_pearl.unload_cargo(300)

```

```

Jack Sparrow is the captain of the Black Pearl
Jack Sparrow
Loaded 600 tons
Unloaded 400 tons
Cannot load that much
Cannot unload that much

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

Пример 5.

Код:

```

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

class Rectangle:
    def __init__(self, width, height):
        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)
    print(rect.width)
    print(rect.height)
    rect.width = 50
    print(rect.width)
    rect.height = 70
    print(rect.height)

```

```

10
20
50
70

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

## Пример 6.

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: 5/6
5/6
19/12
1/12
5/8
10/9

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

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

#### Задание 1.

Парой называется класс с двумя полями, которые обычно имеют имена `first` и `second`. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- метод инициализации `__init__` ; метод должен контролировать значения аргументов на корректность;
- ввод с клавиатуры `read` ;
- вывод на экран `display` .

Реализовать внешнюю функцию с именем `make_тип()` , где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

#### Вариант 29(9)

Поле `first` — целое положительное число, часы; поле `second` — целое положительное число, минуты. Реализовать метод `minutes()` — приведение времени в минуты.

Код:

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

"""
Парой называется класс с двумя полями, которые обычно имеют имена first и
second.
Требуется реализовать тип данных с помощью такого класса. Во всех заданиях
обязательно должны
присутствовать:
метод инициализации __init__ ; (метод должен контролировать значения
аргументов на корректность)
ввод с клавиатуры read ;
вывод на экран display .
Реализовать внешнюю функцию с именем make_тип() , где тип – тип реализуемой
структуры.
```

Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.  
Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

Вариант 29(9)

9. Поле `first` – целое положительное число, часы; поле `second` – целое положительное число, минуты. Реализовать метод `minutes()` – приведение времени в минуты.  
"""

```
class myTime:
    def __init__(self, first=0, second=0):
        if isinstance(first, int) and isinstance(second, int) and first >= 0
and second >= 0:
            self.first = first
            self.second = second
        else:
            raise ValueError

    def read(self):
        try:
            self.first = int(input("Enter hours: "))
            self.second = int(input("Enter minutes: "))
        except:
            print("Error")

    def display(self):
        print(f"First (hours): {self.first}")
        print(f"Second (minutes): {self.second}")

    def minutes(self):
        print(f"Time in minutes: {self.first * 60 + self.second}")

def make_myTime(first, second):
    return myTime(first, second)

if __name__ == '__main__':
    mew_time = make_myTime(4, 23)
    mew_time.display()
    mew_time.minutes()
    mew_time.read()
    mew_time.display()
    mew_time.minutes()
```

```
First (hours): 4
Second (minutes): 23
Time in minutes: 263
Enter hours: 3
Enter minutes: 19
First (hours): 3
Second (minutes): 19
Time in minutes: 199

Process finished with exit code 0
```

Рисунок 8 – Результат работы программы

## Задание 2.

Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

- метод инициализации `__init__` ;
- ввод с клавиатуры `read` ;
- вывод на экран `display` .

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

### Вариант 29(14)

14. Создать класс `Payment` (зарплата). В классе должны быть представлены поля: фамилия-имя-отчество, оклад, год поступления на работу, процент надбавки, подоходный налог, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная суммы.

Реализовать методы:

- вычисления начисленной суммы,
- вычисления удержанной суммы,
- вычисления суммы, выдаваемой на руки,
- вычисления стажа.

Стаж вычисляется как полное количество лет, прошедших от года поступления на работу, до текущего года.

Начисления представляют собой сумму, начисленную за отработанные дни, и надбавки, то есть доли от первой суммы.

Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной суммы) и подоходный налог.

Подоходный налог составляет 13% от начисленной суммы без отчислений в пенсионный фонд.

Код:

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

from datetime import datetime

"""
Составить программу с использованием классов и объектов для решения задачи.
Во всех заданиях, помимо указанных в задании операций, обязательно должны
быть реализованы следующие методы:
- метод инициализации __init__ ;
- ввод с клавиатуры read ;
- вывод на экран display .
Номер варианта необходимо уточнить у преподавателя. В раздел программы,
начинающийся после инструкции
if __name__ == '__main__':
добавить код, демонстрирующий возможности
разработанного класса.
"""

Вариант 29(14)
14. Создать класс Payment (зарплата). В классе должны быть представлены поля:
фамилия-имя-отчество,
оклад,
год поступления на работу,
процент надбавки,
подоходный налог,
количество отработанных дней в месяце,
количество рабочих дней в месяце,
начисленная и удержанная суммы.

Реализовать методы:
вычисления начисленной суммы,
вычисления удержанной суммы,
вычисления суммы, выдаваемой на руки,
вычисления стажа.

Стаж вычисляется как полное количество лет, прошедших от года поступления на
работу, до текущего года.
Начисления представляют собой сумму, начисленную за отработанные дни, и
надбавки, то есть доли от первой суммы.
Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной
суммы) и подоходный налог.
Подоходный налог составляет 13% от начисленной суммы без отчислений в
пенсионный фонд.
```

```

"""

class Payment:
    def __init__(self, name, salary, year, perc_allow, worked_days,
work_days):
        if isinstance(name, str):
            self.name = name
        else:
            self.name = "not name"
        if isinstance(salary, float) or isinstance(salary, int):
            self.salary = salary
        else:
            self.salary = 0
        if isinstance(year, int) and 2023 >= year >= 0:
            self.year = year
        else:
            self.year = 1700
        if isinstance(perc_allow, float) or isinstance(perc_allow, int):
            self.perc_allow = perc_allow
        else:
            self.perc_allow = 0
        if isinstance(worked_days, int) and 31 >= worked_days >= 1:
            self.worked_days = worked_days
        else:
            self.worked_days = 1
        if isinstance(work_days, int) and 31 >= work_days >= 0:
            self.work_days = work_days
        else:
            self.work_days = 0
        try:
            self.accrued_amount = self.accruedAmount()
            self.income_tax = self.accrued_amount * 0.13
            self.amount_withheld = self.amountWithheld()
        except:
            pass

    def read(self):
        try:
            print("-----ВВОД")
            self.name = input("ФИО: ")
            self.salary = float(input("Оклад: "))
            self.year = int(input("Год поступления на работу: "))
            self.perc_allow = float(input("Процент надбавки: "))
            self.worked_days = int(input("Количество отработанных дней в
месяце: "))
            self.work_days = int(input("Количество рабочих дней в месяце: "))
            self.accrued_amount = self.accruedAmount()
            self.income_tax = self.accrued_amount * 0.13
            self.amount_withheld = self.amountWithheld()
            print("-----")
        except:
            print("Error")

    def display(self):
        print("-----ВЫВОД")
        print(f"ФИО: {self.name}")
        print(f"Оклад: {self.salary}")
        print(f"Год поступления на работу: {self.year}")
        print(f"Стаж: {self.experience()}")
        print(f"Процент надбавки: {self.perc_allow}%")
        print(f"Подходный налог: {self.income_tax:.2f}")
        print(f"Количество отработанных дней в месяце: {self.worked_days}")

```

```

print(f"Количество рабочих дней в месяце: {self.work_days}")
print(f"Начисленная сумма: {self.accrued_amount:.2f}")
print(f"Удержанная сумма: {self.amount_withheld:.2f}")
print(f"Сумма выдаваемая на руки: {self.amountOnHand():.2f}")
print("-----")

def experience(self):
    """
    Стаж
    self.experience =
    """
    return datetime.now().year - self.year

def accruedAmount(self):
    """
    Начисленная сумма
    self.accrued_amount =
    self.income_tax =
    """
    return self.salary * (1 + self.perc_allow / 100) * (self.worked_days
/ self.work_days)
    return self.accrued_amount * 0.13

def amountWithheld(self):
    """
    Удержанная сумма
    self.amount_withheld =
    """
    return self.accrued_amount * 0.01 + self.income_tax

def amountOnHand(self):
    """
    Сумма выдаваемая на руки
    self.amount_on_hand =
    """
    return self.accrued_amount - self.amount_withheld

if __name__ == '__main__':
    person1 = Payment("Петров Игорь Олегович", 32000.0, 2020, 14.0, 19, 28)
    person1.display()
    person2 = Payment(None, None, None, None, None, None)
    person2.read()
    person2.display()

```



```

-----вывод
ФИО: Петров Игорь Олегович
Оклад: 32000.0
Год поступления на работу: 2020
Стаж: 3
Процент надбавки: 14.0%
Подоходный налог: 3218.06
Количество отработанных дней в месяце: 19
Количество рабочих дней в месяце: 28
Начисленная сумма: 24754.29
Удержанная сумма: 3465.60
Сумма выдаваемая на руки: 21288.69
-----

-----ввод
ФИО: Петров Петр Вячеславович
Оклад: 30000
Год поступления на работу: 1992
Процент надбавки: 2
Количество отработанных дней в месяце: 19
Количество рабочих дней в месяце: 25
-----

-----вывод
ФИО: Петров Петр Вячеславович
Оклад: 30000.0
Год поступления на работу: 1992
Стаж: 31
Процент надбавки: 2.0%
Подоходный налог: 3023.28
Количество отработанных дней в месяце: 19
Количество рабочих дней в месяце: 25
Начисленная сумма: 23256.00
Удержанная сумма: 3255.84
Сумма выдаваемая на руки: 20000.16
-----

```

9. Зафиксируйте сделанные изменения в репозитории.

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Как осуществляется объявление класса в языке Python?

Классы объявляются с помощью ключевого слова `class` и имени класса.

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибут класса – это атрибут, общий для всех экземпляров класса. Атрибуты класса определены внутри класса, но вне каких-либо методов. Их значения одинаковы для всех экземпляров этого класса. Так что вы можете рассматривать их как тип значений по умолчанию для всех наших объектов.

Что касается переменных экземпляра, они хранят данные, уникальные для каждого объекта класса

### 3. Каково назначение методов класса?

Методы определяют функциональность объектов, принадлежащих конкретному классу.

### 4. Для чего предназначен метод `__init__()` класса?

Метод `__init__` является конструктором. Конструкторы – это концепция объектноориентированного программирования. Класс может иметь один и только один конструктор. Если `__init__` определен внутри класса, он автоматически вызывается при создании нового экземпляра класса.

### 5. Каково назначение `self`?

Аргумент `self` представляет конкретный экземпляр класса и позволяет нам получить доступ к его атрибутам и методам.

### 6. Как добавить атрибуты в класс?

В дополнение к изменению атрибутов мы также можем создавать атрибуты для класса или конкретного экземпляра. Например, мы хотим видеть информацию о всех видах наших питомцев. Мы могли бы записать ее в самом классе с самого начала или создать переменную следующим образом:

```
Pet.all_specs = [tom.spec, avocado.spec, ben.spec]
tom.all_specs # ["cat", "dog", "goldfish"]
avocado.all_specs # ["cat", "dog", "goldfish"]
ben.all_specs # ["cat", "dog", "goldfish"]
```

Еще мы могли бы создать атрибут для конкретного экземпляра. Например, мы хотим вспомнить породу собаки под именем `Avocado`. Про породы чаще говорят применительно к собакам (у кошек тоже есть породы, но

они не так сильно различаются), поэтому имеет смысл, чтобы только у нее был атрибут с такой информацией:

```
avocado.breed = "corgi"
```

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

В Python любой может обратиться к атрибутам и методам вашего класса, если возникнет такая необходимость. Это существенный недостаток этого языка, т.к. нарушается один из ключевых принципов ООП – инкапсуляция. Хорошим тоном считается, что для чтения/изменения какого-то атрибута должны использоваться специальные методы, которые называются getter/setter, их можно реализовать, но ничего не мешает изменить атрибут напрямую.

```
rect = Rectangle(10, 20)
```

```
rect.get_width() #10
```

```
rect._width      #10
```

8. 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.