МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 4.1

«Элементы объектно-ориентированного программирования в языке Python»

по дисциплине «Основы программной инженерии»

Выполнил студент группы
ПИЖ-б-о-21-1
Зиберов Александр
« » мая 2023 г.
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.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: https://github.com/afk552/lab4.1

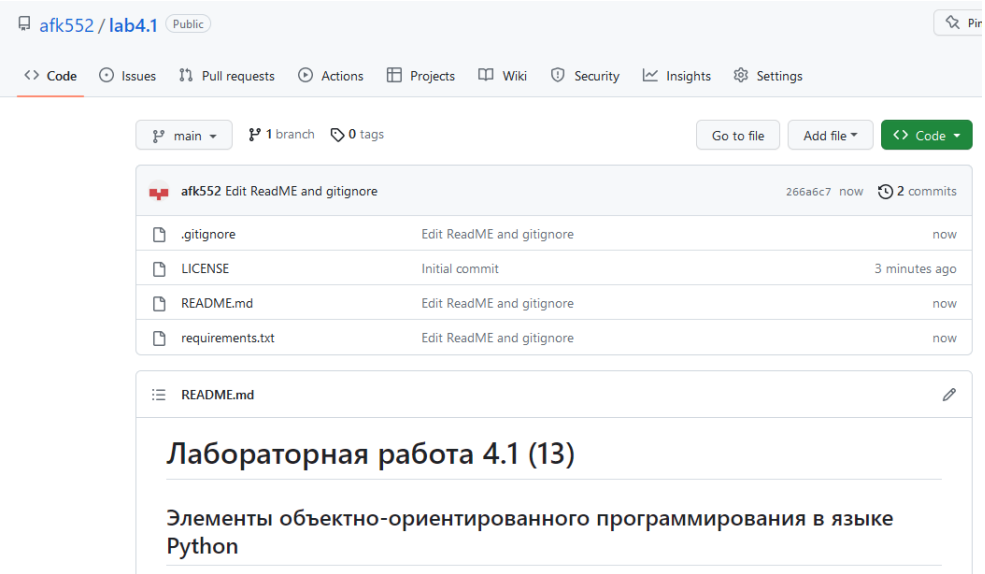


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

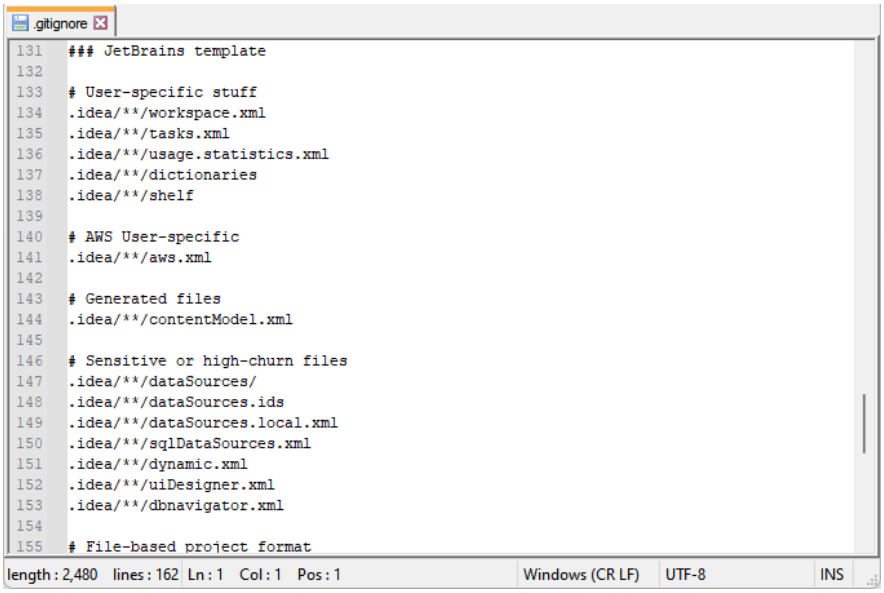


Рисунок 2 – Окно блокнота

Организуйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab4.1>git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 3 – Окно командной строки

Примеры

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


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


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

```

```
Run  ex1
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex1.py
paper
paperback
[]

Process finished with exit code 0
```

Рисунок 4 – Пример 1

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


5 usages
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)
```

if __name__ == '__main__' › for river in River.all_rivers

Run ex2

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex2.py
Волга
Сена
Нил

Process finished with exit code 0
```

Рисунок 5 – Пример 2

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


4 usages
class River:
    all_rivers = []

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

    3 usages
    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()
```

Run ex3

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex3.py
Длина Волга равна 3530 км
Длина Сена равна 776 км
Длина Нил равна 6852 км

Process finished with exit code 0
```

Рисунок 6 – Пример 3

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


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

    2 usages
    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")

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

    1 usage
    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)
```

Ship › name_captain()

Run ex4

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex4.py
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
```

Рисунок 7 – Пример 4

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


1 usage
class Rectangle:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height

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

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

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

    1 usage
    @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)
```

Rectangle

Run ex5

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\examples\ex5.py
10
20
50
70
```

Рисунок 8 – Пример 5

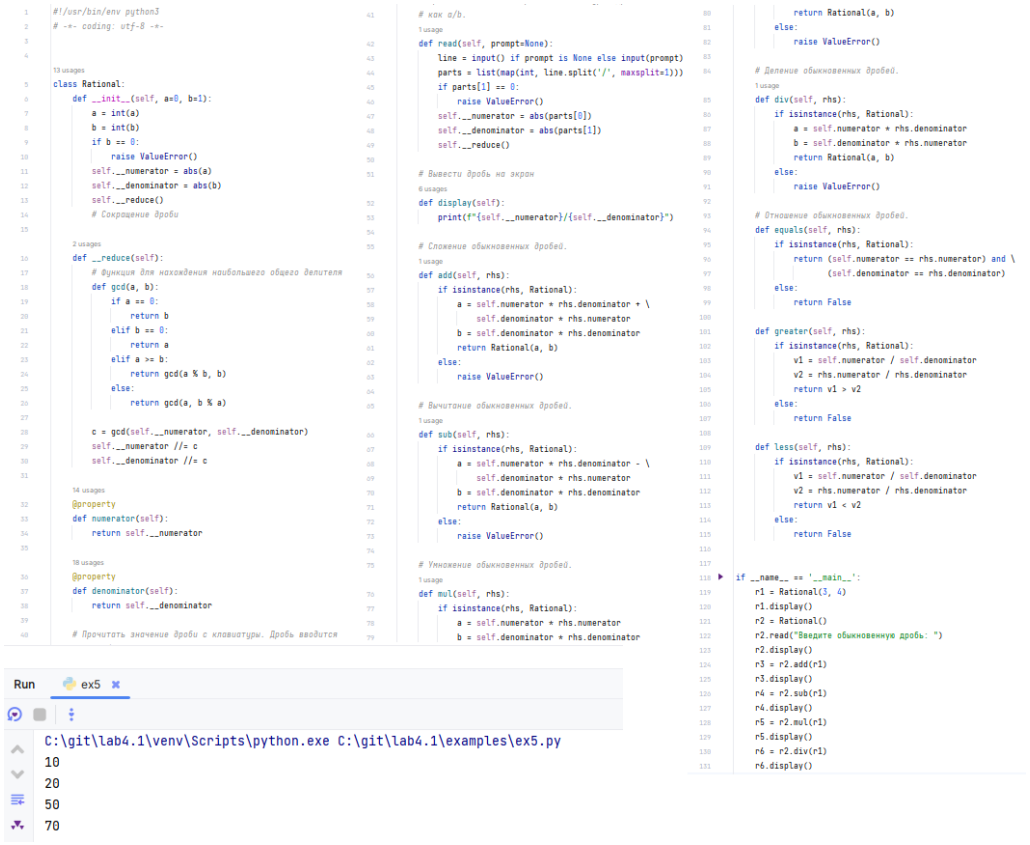


Рисунок 9 – Пример 6

Индивидуальное задание.

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.

7. Поле first — дробное число, левая граница диапазона; поле second — дробное число, правая граница диапазона. Реализовать метод rangecheck() — проверку заданного числа на принадлежность диапазону.

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


1 usage
class Number:

    def __init__(self, first=0.0, second=0.0):
        try:
            self.__first = float(first)
            self.__second = float(second)
        except ValueError:
            ValueError("Проверьте правильность ввода!")

    6 usages
    @property
    def first(self):
        return self.__first

    6 usages
    @property
    def second(self):
        return self.__second

    1 usage
    def read(self):
        lower_bound = float(input("Введите левую границу диапазона: "))
        upper_bound = float(input("Введите правую границу диапазона: "))
        if isinstance(lower_bound, float) and (isinstance(upper_bound, float)):
            self.__first = float(lower_bound)
            self.__second = float(upper_bound)
        else:
            raise ValueError("Проверьте введенные границы диапазона!")

    1 usage
    def display(self):
        print(f"1: {self.first}")
        print(f"2: {self.second}")

    def __contains__(self, number):
        return self.first <= number <= self.second

    1 usage
    def rangecheck(self, number=None):
        if self.first > self.second:
            raise ValueError("Введенная левая граница больше правой!")
        if isinstance(number, float):
            pass
        else:
            try:
                number = float(input("Задайте число: "))
            except ValueError:
                ValueError("Введено не число!")
        if self.first <= float(number) <= self.second:
            print(f"Число {number} в диапазоне [{self.first}, {self.second}]")
        else:
            print(f"Число {number} не в [{self.first}, {self.second}]")


if __name__ == "__main__":
    test = Number()
    test.read()
    test.display()
    test.rangecheck()
    test.rangecheck(23.2)
```

Рисунок 10 – Код программы индивидуального задания 1

```
Введите левую границу диапазона: 12.5
Введите правую границу диапазона: 34.8
1: 12.5
2: 34.8
Задайте число: 13.7
Число 13.7 в диапазоне [12.5, 34.8]
Число 23.2 в диапазоне [12.5, 34.8]

Process finished with exit code 0
```

Рисунок 11 – Результат выполнения программы индивидуального задания 1

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 ;

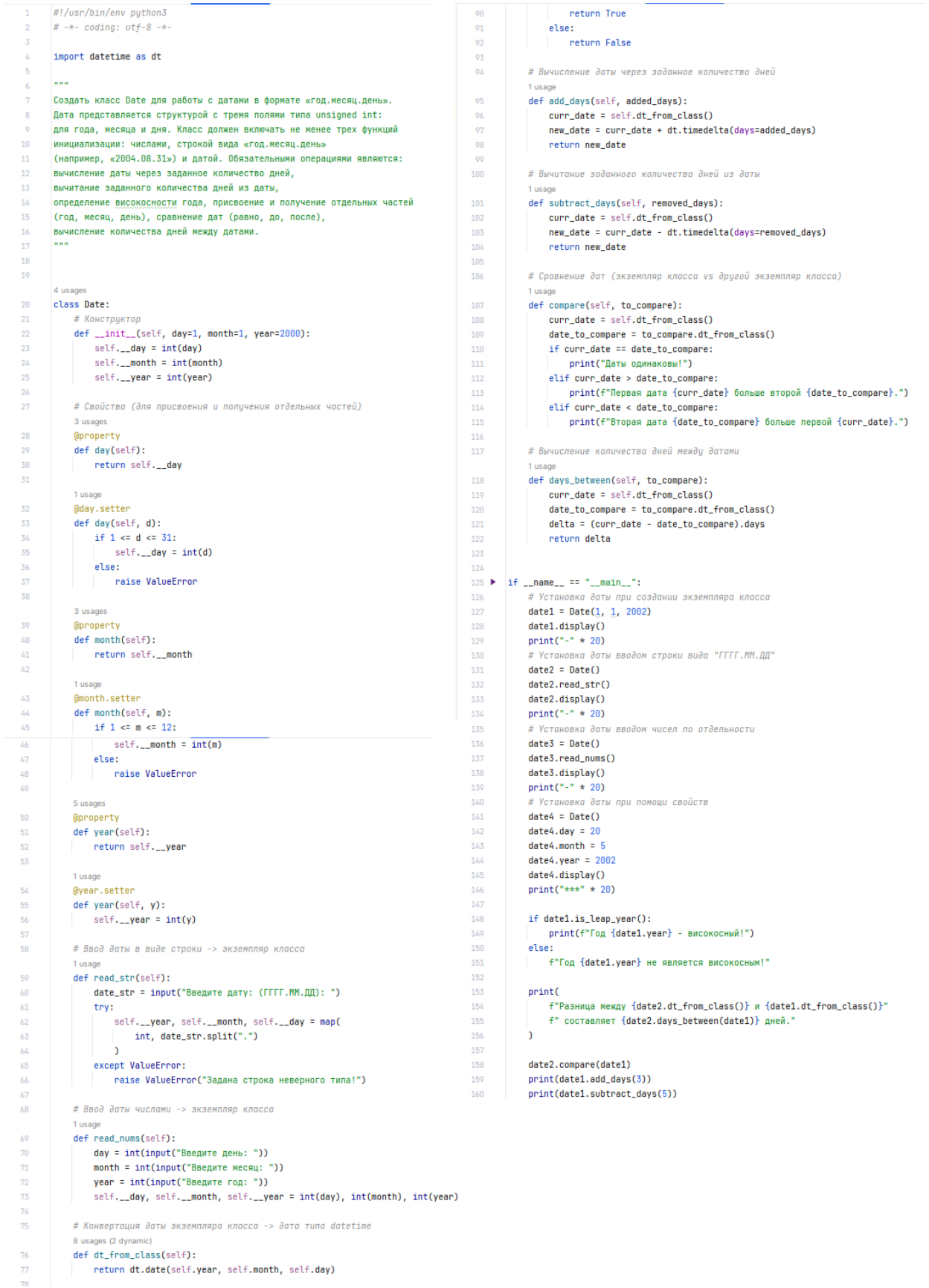
вывод на экран display .

7. Создать класс Date для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями типа unsigned int: для года, месяца и дня. Класс должен включать не менее трех функций инициализации: числами, строкой вида «год.месяц.день» (например, «2004.08.31») и датой. Обязательными операциями являются: вычисление даты через заданное количество дней, вычитание заданного количества дней из даты, определение високосности года, присвоение и получение отдельных частей (год, месяц, день), сравнение дат (равно, до, после), вычисление количества дней между датами.

```
C:\git\lab4.1\venv\Scripts\python.exe C:\git\lab4.1\indiv_task\task2.py
1.1.2002
--------------------
Введите дату: (ГГГГ.ММ.ДД): 2002.12.03
3.12.2002
--------------------
Введите день: 13
Введите месяц: 04
Введите год: 2005
13.4.2005
--------------------
20.5.2002
************************************************************
Разница между 2002-12-03 и 2002-01-01 составляет 336 дней.
Первая дата 2002-12-03 больше второй 2002-01-01.
2002-01-04
2001-12-27

Process finished with exit code 0
```

Рисунок 12 – Результат выполнения индивидуального задания 2



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

import datetime as dt

"""
Создать класс Date для работы с датами в формате «год.месяц.день».
Дата представляется структурой с тремя полями типа unsigned int:
для года, месяца и дня. Класс должен включать не менее трех функций
инициализации: числами, строкой вида «год.месяц.день»
(например, «2004.08.31») и датой. Обязательными операциями являются:
вычисление даты через заданное количество дней,
вычитание заданного количества дней из даты,
определение високосности года, присвоение и получение отдельных частей
(год, месяц, день), сравнение дат (равно, до, после),
вычисление количества дней между датами.
"""


4 usages
class Date:
    # Конструктор
    def __init__(self, day=1, month=1, year=2000):
        self.__day = int(day)
        self.__month = int(month)
        self.__year = int(year)

    # Свойства (для присвоения и получения отдельных частей)
    3 usages
    @property
    def day(self):
        return self.__day

    1 usage
    @day.setter
    def day(self, d):
        if 1 <= d <= 31:
            self.__day = int(d)
        else:
            raise ValueError

    3 usages
    @property
    def month(self):
        return self.__month

    1 usage
    @month.setter
    def month(self, m):
        if 1 <= m <= 12:
            self.__month = int(m)
        else:
            raise ValueError

    5 usages
    @property
    def year(self):
        return self.__year

    1 usage
    @year.setter
    def year(self, y):
        self.__year = int(y)

    # Ввод даты в виде строки -> экземпляр класса
    1 usage
    def read_str(self):
        date_str = input("Введите дату: (ГГГГ.ММ.ДД): ")
        try:
            self.__year, self.__month, self.__day = map(
                int, date_str.split(".")
            )
        except ValueError:
            raise ValueError("Задана строка неверного типа!")

    # Ввод даты числами -> экземпляр класса
    1 usage
    def read_nums(self):
        day = int(input("Введите день: "))
        month = int(input("Введите месяц: "))
        year = int(input("Введите год: "))
        self.__day, self.__month, self.__year = int(day), int(month), int(year)

    # Конвертация даты экземпляра класса -> дата типа datetime
    8 usages (2 dynamic)
    def dt_from_class(self):
        return dt.date(self.year, self.month, self.day)

        return True
    else:
        return False

# Вычисление даты через заданное количество дней
1 usage
def add_days(self, added_days):
    curr_date = self.dt_from_class()
    new_date = curr_date + dt.timedelta(days=added_days)
    return new_date

# Вычитание заданного количества дней из даты
1 usage
def subtract_days(self, removed_days):
    curr_date = self.dt_from_class()
    new_date = curr_date - dt.timedelta(days=removed_days)
    return new_date

# Сравнение дат (экземпляр класса vs другой экземпляр класса)
1 usage
def compare(self, to_compare):
    curr_date = self.dt_from_class()
    date_to_compare = to_compare.dt_from_class()
    if curr_date == date_to_compare:
        print("Даты одинаковы!")
    elif curr_date > date_to_compare:
        print(f"Первая дата {curr_date} больше второй {date_to_compare}.")
    elif curr_date < date_to_compare:
        print(f"Вторая дата {date_to_compare} больше первой {curr_date}.")

# Вычисление количества дней между датами
1 usage
def days_between(self, to_compare):
    curr_date = self.dt_from_class()
    date_to_compare = to_compare.dt_from_class()
    delta = (curr_date - date_to_compare).days
    return delta


if __name__ == "__main__":
    # Установка даты при создании экземпляра класса
    date1 = Date(1, 1, 2002)
    date1.display()
    print("-" * 20)
    # Установка даты вводом строки вида "ГГГГ.ММ.ДД"
    date2 = Date()
    date2.read_str()
    date2.display()
    print("-" * 20)
    # Установка даты вводом чисел по отдельности
    date3 = Date()
    date3.read_nums()
    date3.display()
    print("-" * 20)
    # Установка даты при помощи свойств
    date4 = Date()
    date4.day = 20
    date4.month = 5
    date4.year = 2002
    date4.display()
    print("***" * 20)

    if date1.is_leap_year():
        print(f"Год {date1.year} - високосный!")
    else:
        f"Год {date1.year} не является високосным!"

    print(
        f"Разница между {date2.dt_from_class()} и {date1.dt_from_class()}"
        f" составляет {date2.days_between(date1)} дней."
    )

    date2.compare(date1)
    print(date1.add_days(3))
    print(date1.subtract_days(5))
```

Рисунок 13 – Код программы индивидуального задания 2

Вывод: Были приобретены и применены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python.

Контрольные вопросы:

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 Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово.

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?

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 помешает 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.