

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №7
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Евдаков Евгений Владимирович
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Воронкин Р. А., доцент департамента
цифровых, робототехнических систем
и электроники института
перспективной инженерии

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: классы данных в Python.

Цель: приобретение навыков по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Задание 1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами. Клонировал свой репозиторий на свой компьютер. Организовал свой репозиторий в соответствии с моделью ветвления git-flow, появилась новая ветка develop в которой буду выполнять дальнейшие задачи.

```
C:\Users\Gaming-PC\Postman>git clone https://github.com/EvgenyEvdakov/Laba_4.7.git
Cloning into 'Laba_4.7'...
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 (from 0)
Receiving objects: 100% (5/5), done.
```

Рисунок 1. Клонирование репозитория

Задание 2. Создал файлы poetry.lock и pyproject.toml, также установил необходимые пакеты isort, black, flake8, mypy, pre-commit, pytest.

```
[tool.poetry]
name = "laba-4-7"
version = "0.1.0"
description = ""
authors = ["Evgeni <kkrutkov02@gmail.com>"]
readme = "README.md"
packages = [{include = "laba_4"}]

[tool.poetry.dependencies]
python = "^3.10"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"

Do you confirm generation? (yes/no) [yes]
(.venv) PS C:\Users\evdak\Laba_4.7> poetry install
Updating dependencies
Resolving dependencies... (0.1s)

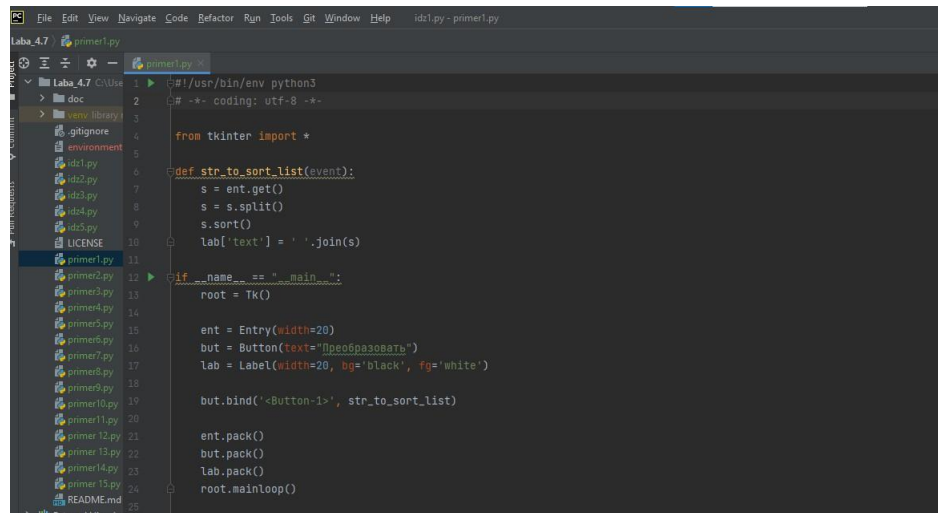
Writing lock file
```

Рисунок 2. Создание виртуального окружения

Задание 3. Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл primer1.py.

Далее добавил новый файл primer1.py.

Условие примера: необходимо написать скрипт, в котором появляется окно, в текстовое поле которого можно ввести список слов, нажать кнопку и получить его отсортированный вариант.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  def str_to_sort_list(event):
7      s = ent.get()
8      s = s.split()
9      s.sort()
10     lab['text'] = ' '.join(s)
11
12 if __name__ == "__main__":
13     root = Tk()
14
15     ent = Entry(width=20)
16     but = Button(text="Преобразовать")
17     lab = Label(width=20, bg='black', fg='white')
18
19     but.bind('<Button-1>', str_to_sort_list)
20
21     ent.pack()
22     but.pack()
23     lab.pack()
24     root.mainloop()
```

Рисунок 3. Код первого примера

Далее запустим программу на выполнение.

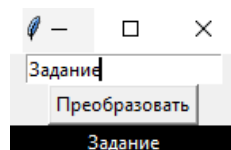
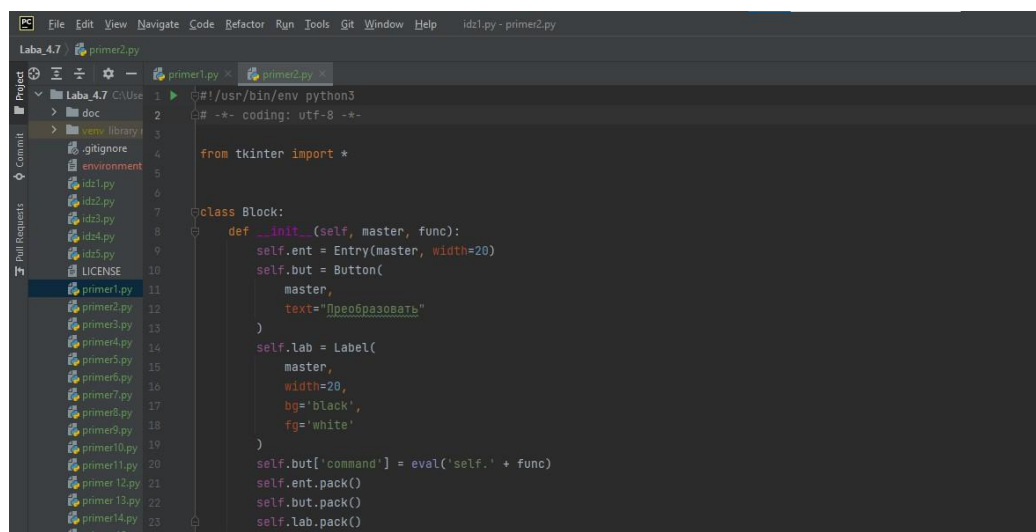


Рисунок 4. Результат выполнения первого примера

Далее добавил новый файл primer2.py.

Условие примера: необходимо написать код, в окне которого будут выведены два однотипных блока, кнопки которых выполняют разные действия.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  class Block:
7      def __init__(self, master, func):
8          self.ent = Entry(master, width=20)
9          self.but = Button(
10              master,
11              text="Преобразовать"
12          )
13          self.lab = Label(
14              master,
15              width=20,
16              bg='black',
17              fg='white'
18          )
19          self.but['command'] = eval('self.' + func)
20          self.ent.pack()
21          self.but.pack()
22          self.lab.pack()
```

Рисунок 5. Код второго примера

Далее запустим программу на выполнение.

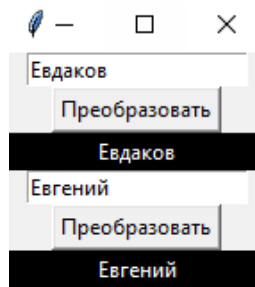


Рисунок 6. Результат выполнения второго примера

Далее добавил новый файл primer3.py.

Условие примера: необходимо изучить метод Button – кнопка, для этого напишем код, задавая ей разные значения.

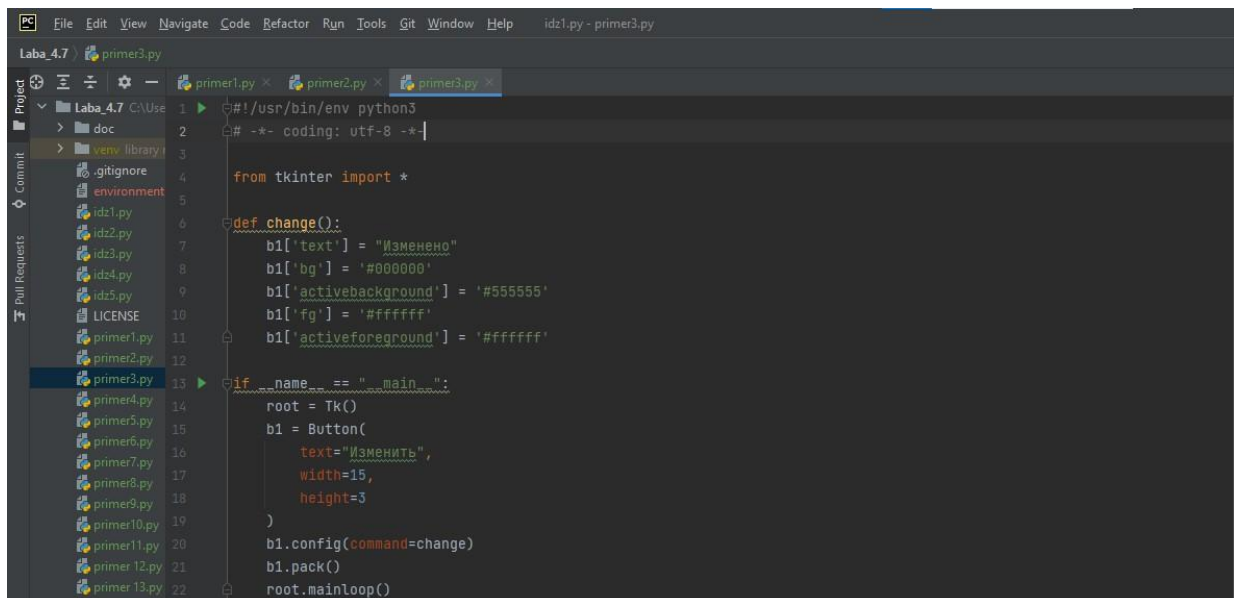


Рисунок 7. Код третьего примера

Далее запустим программу на выполнение.

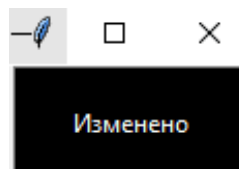
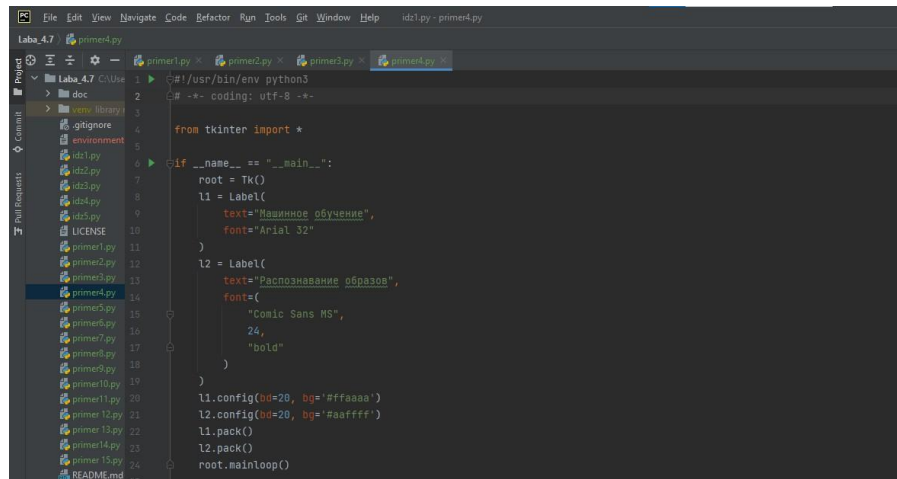


Рисунок 8. Результат выполнения третьего примера

Далее добавил новый файл primer4.py.

Условие примера: необходимо изучить метод Label – метка.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  if __name__ == "__main__":
7      root = Tk()
8      l1 = Label(
9          text="Машинное обучение",
10         font="Arial 32"
11     )
12     l2 = Label(
13         text="Распознавание образов",
14         font=(
15             "Comic Sans MS",
16             24,
17             "bold"
18         )
19     )
20     l1.config(bg="#ffaaaa')
21     l2.config(bg="#aaffff')
22     l1.pack()
23     l2.pack()
24     root.mainloop()
```

Рисунок 9. Код четвертого примера

Далее запустим программу на выполнение.

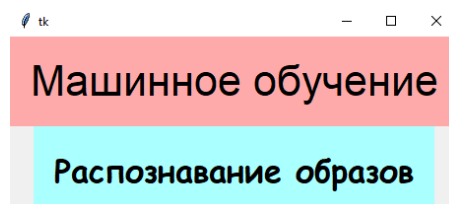
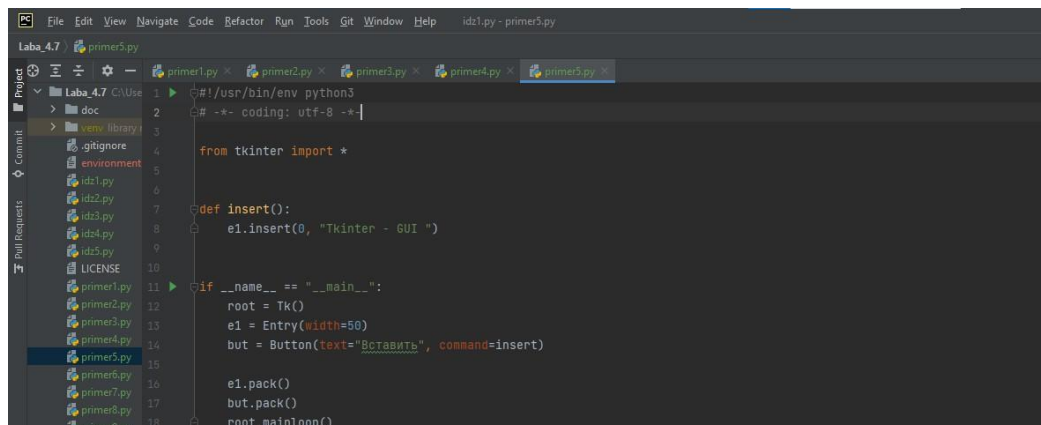


Рисунок 10. Результат выполнения четвертого примера

Далее добавил новый файл primer5.py.

Условие примера: необходимо изучить метод Entry – однострочное текстовое поле.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  def insert():
7      e1.insert(0, "Tkinter - GUI Я сделал это")
8
9
10 if __name__ == "__main__":
11     root = Tk()
12     e1 = Entry(width=50)
13     but = Button(text="Вставить", command=insert)
14
15     e1.pack()
16     but.pack()
17     root.mainloop()
```

Рисунок 11. Код пятого примера

Далее запустим программу на выполнение.

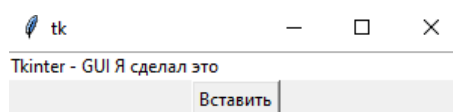
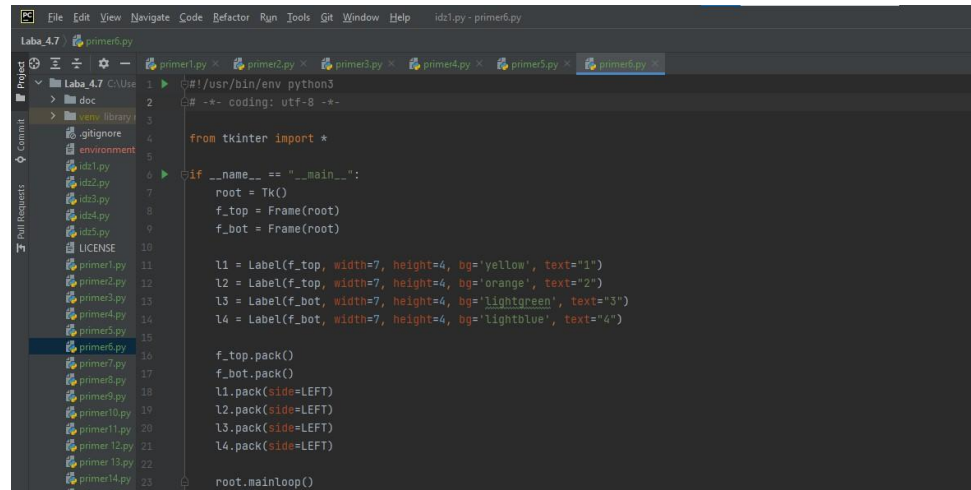


Рисунок 12. Результат выполнения пятого примера

Далее добавил новый файл primer6.py.

Условие примера: необходимо изучить метод pack.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from tkinter import *
5
6 if __name__ == "__main__":
7     root = Tk()
8     f_top = Frame(root)
9     f_bot = Frame(root)
10
11     l1 = Label(f_top, width=7, height=4, bg='yellow', text="1")
12     l2 = Label(f_top, width=7, height=4, bg='orange', text="2")
13     l3 = Label(f_bot, width=7, height=4, bg='lightgreen', text="3")
14     l4 = Label(f_bot, width=7, height=4, bg='lightblue', text="4")
15
16     f_top.pack()
17     f_bot.pack()
18     l1.pack(side=LEFT)
19     l2.pack(side=LEFT)
20     l3.pack(side=LEFT)
21     l4.pack(side=LEFT)
22
23     root.mainloop()
```

Рисунок 13. Код шестого примера

Далее запустим программу на выполнение.

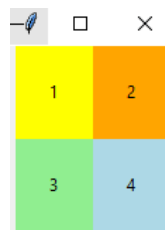
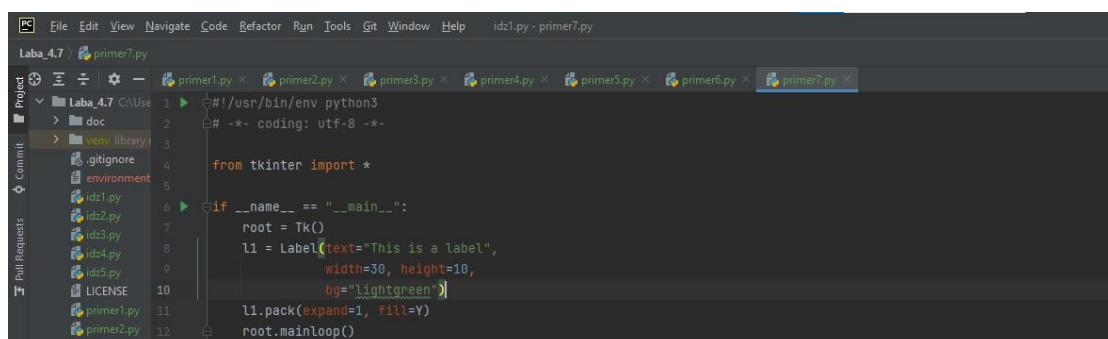


Рисунок 14. Результат выполнения шестого примера

Далее добавил новый файл primer7.py.

Условие примера: необходимо изучить метод pack. Используем следующие два свойства – fill (заполнение) и expand (расширение). По умолчанию expand равен нулю (другое значение – единица), а fill – NONE (другие значения BOTH , X , Y). Создадим окно с одной меткой:



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from tkinter import *
5
6 if __name__ == "__main__":
7     root = Tk()
8     l1 = Label(text="This is a label",
9               width=30, height=10,
10               bg="lightgreen")
11     l1.pack(expand=1, fill=Y)
12
13     root.mainloop()
```

Рисунок 15. Код седьмого примера

Далее запустим программу на выполнение.



Рисунок 16. Результат выполнения седьмого примера

Далее добавил новый файл primer8.py.

Условие примера: необходимо изучить метод Text – многострочное текстовое поле.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  if __name__ == "__main__":
7      root = Tk()
8
9      text = Text(width=25, height=5, bg="darkgreen",
10                 fg="white", wrap=WORD)
11
12      text.pack()
13      root.mainloop()

```

Рисунок 17. Код восьмого примера

Далее запустим программу на выполнение.

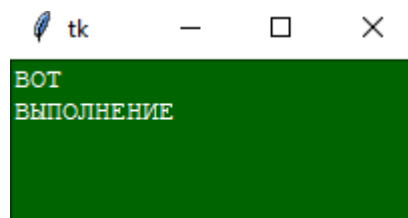


Рисунок 18. Результат выполнения восьмого примера

Далее добавил новый файл primer9.py.

Условие примера: необходимо изучить метод Text и Scrollbar.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from tkinter import *
5
6  if __name__ == "__main__":
7      root = Tk()
8
9      text = Text(width=20, height=7)
10     text.pack(side=LEFT)
11
12     scroll = Scrollbar(command=text.yview)
13     scroll.pack(side=LEFT, fill=Y)
14
15     text.config(yscrollcommand=scroll.set)
16
17     root.mainloop()

```

Рисунок 19. Код девятого примера

Далее запустим программу на выполнение.

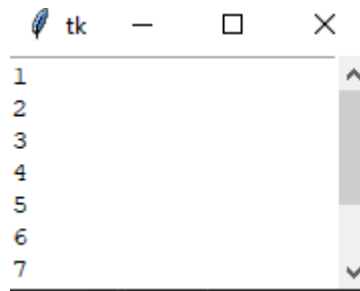


Рисунок 20. Результат выполнения девятого примера

Далее добавил новый файл primer10.py.

Условие примера: необходимо изучить метод Text. Методы get и delete могут принимать не два, а один аргумент. В таком случае будет обрабатываться только один символ в указанной позиции.

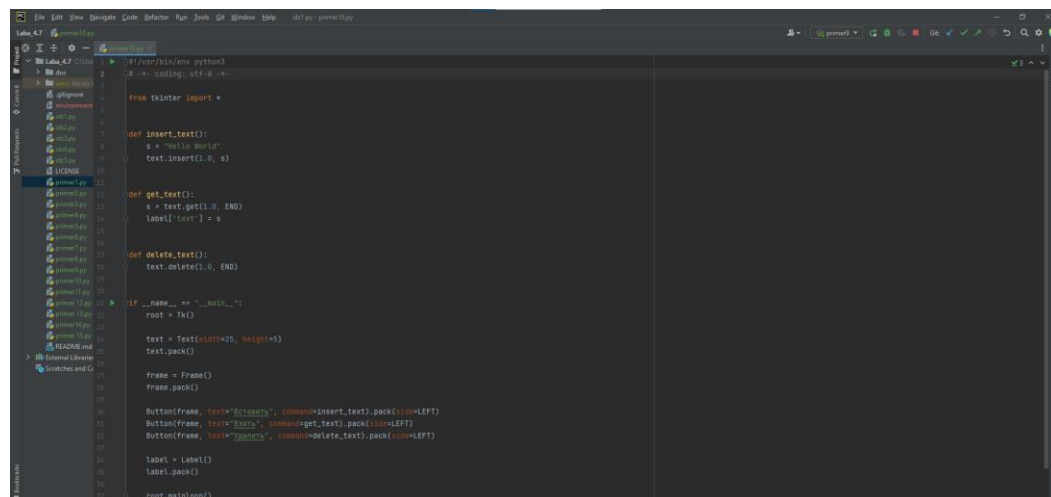


Рисунок 21. Код десятого примера

Далее запустим программу на выполнение.

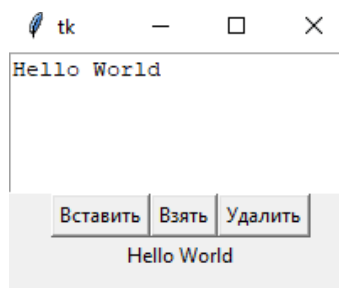
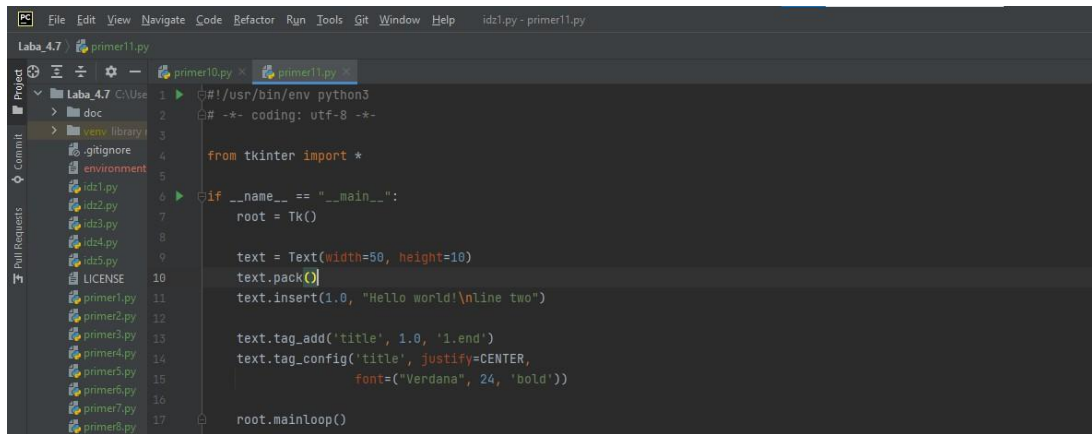


Рисунок 22. Результат выполнения десятого примера

Далее добавил новый файл primer11.py.

Условие примера: необходимо изучить метод Теги.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from tkinter import *
5
6 if __name__ == "__main__":
7     root = Tk()
8
9     text = Text(width=50, height=10)
10    text.pack()
11    text.insert(1.0, "Hello world!\nline two")
12
13    text.tag_add('title', 1.0, '1.end')
14    text.tag_config('title', justify=CENTER,
15                    font=("Verdana", 24, 'bold'))
16
17    root.mainloop()
```

Рисунок 23. Код одиннадцатого примера

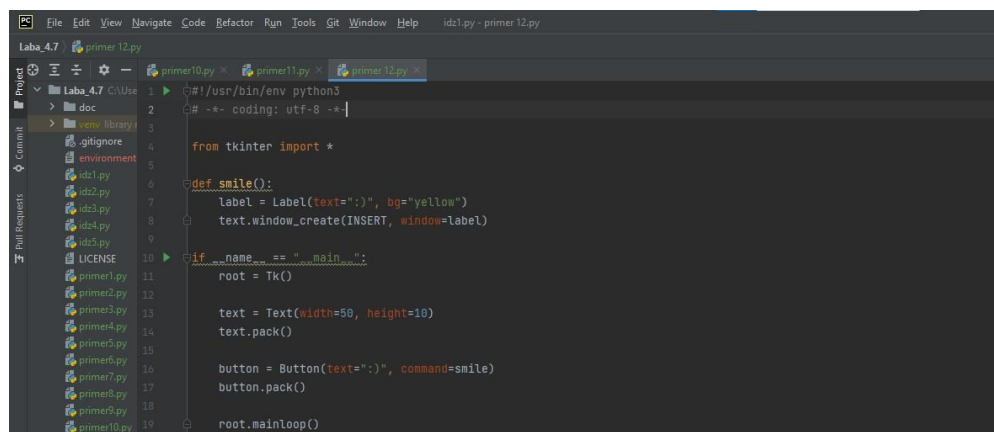
Далее запустим программу на выполнение.



Рисунок 24. Результат выполнения одиннадцатого примера

Далее добавил новый файл primer12.py.

Условие примера: необходимо изучить способ вставки виджетов в текстовое поле.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from tkinter import *
5
6 def smile():
7     label = Label(text=":) ", bg="yellow")
8     text.window_create(INSERT, window=label)
9
10 if __name__ == "__main__":
11     root = Tk()
12
13     text = Text(width=50, height=10)
14     text.pack()
15
16     button = Button(text=":", command=smile)
17     button.pack()
18
19     root.mainloop()
```

Рисунок 25. Код двенадцатого примера

Далее запустим программу на выполнение.

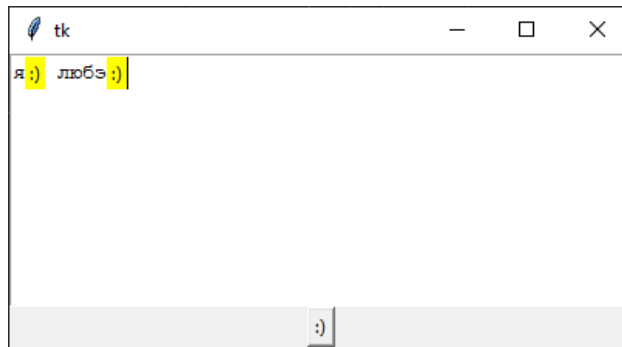


Рисунок 26. Результат выполнения двенадцатого примера

Далее добавил новый файл primer13.py.

Условие примера: необходимо изучить метод Radiobutton и Checkbutton. Переменные Tkinter.

```

File Edit View Navigate Code Refactor Run Tools Git Window Help id1.py - primer 13.py
Laba_4.7 primer13.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 from tkinter import *
5
6 def change():
7     if var.get() == 0:
8         label['bg'] = 'red'
9     elif var.get() == 1:
10        label['bg'] = 'green'
11     elif var.get() == 2:
12        label['bg'] = 'blue'
13
14 if __name__ == "__main__":
15     root = Tk()
16
17     var = IntVar()
18     var.set(0)
19
20     red = Radiobutton(text="Red", variable=var, value=0)
21     green = Radiobutton(text="Green", variable=var, value=1)
22     blue = Radiobutton(text="Blue", variable=var, value=2)
23
24     button = Button(text="Изменить", command=change)
25     label = Label(width=20, height=10)
26
27     red.pack()
28     green.pack()
29     blue.pack()
30     button.pack()
31     label.pack()
32
33     root.mainloop()

```

Рисунок 27. Код тринадцатого примера

Далее запустим программу на выполнение.

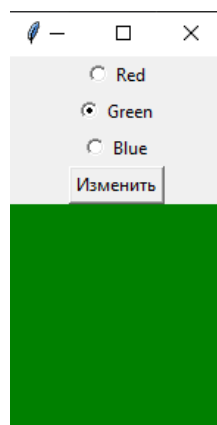


Рисунок 28. Результат выполнения тринадцатого примера

Далее добавил новый файл primer14.py.

Условие примера: необходимо изучить метод Checkbutton – флажок.

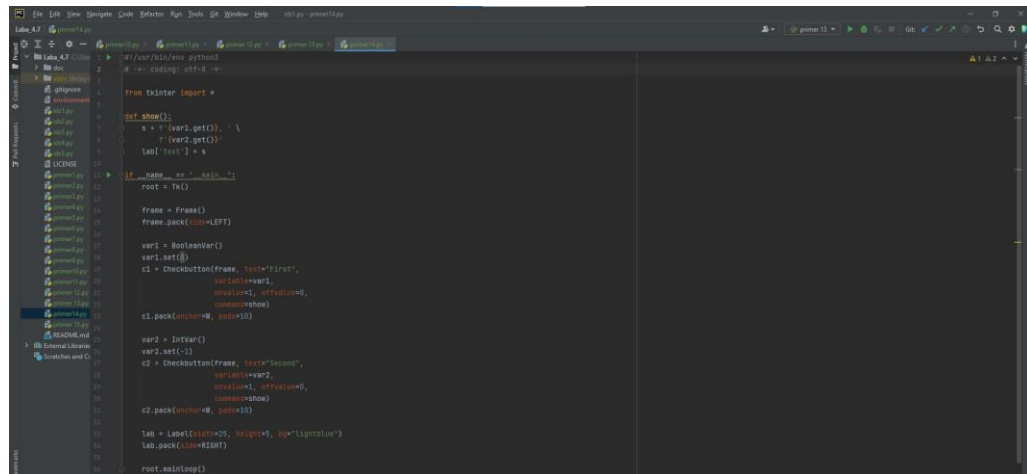


Рисунок 29. Код четырнадцатого примера

Далее запустим программу на выполнение.

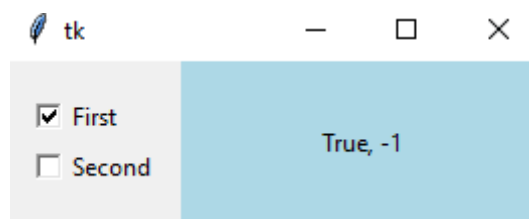


Рисунок 30. Результат выполнения четырнадцатого примера

Далее добавил новый файл primer15.py.

Условие примера: необходимо изучить метод Checkbutton – флажок. С помощью методов select и deselect флажков можно их программно включать и выключать, это и необходимо проверить.

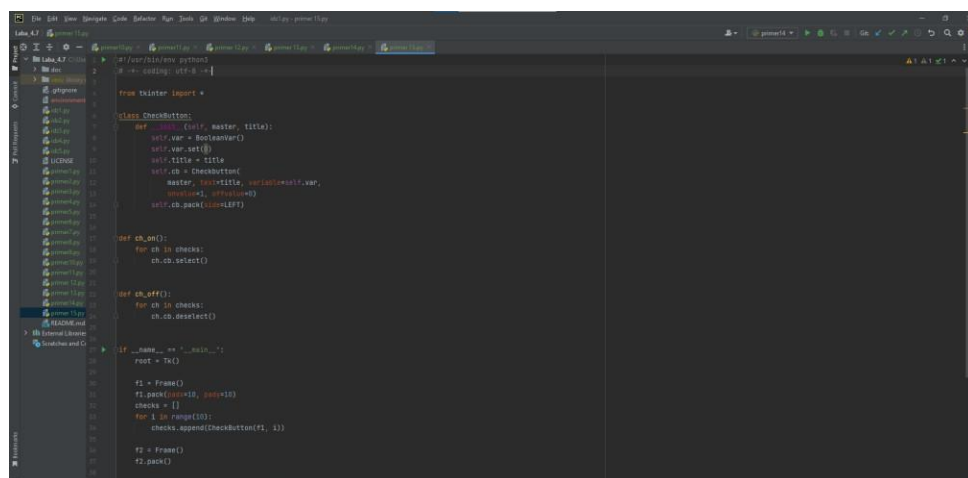


Рисунок 31. Код пятнадцатого примера

Далее запустим программу на выполнение.

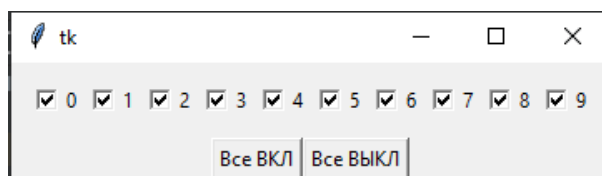


Рисунок 32. Результат выполнения пятнадцатого примера

Задание 4.

Индивидуальное задание

Вариант 6

Создал новый файл под названием idz1.py.

Условие задания: напишите простейший калькулятор, состоящий из двух текстовых полей, куда пользователь вводит числа, и четырех кнопок "+", "-", "*", "/". Результат вычисления должен отображаться в метке. Если арифметическое действие выполнить невозможно (например, если были введены буквы, а не числа), то в метке должно появляться слово "ошибка".

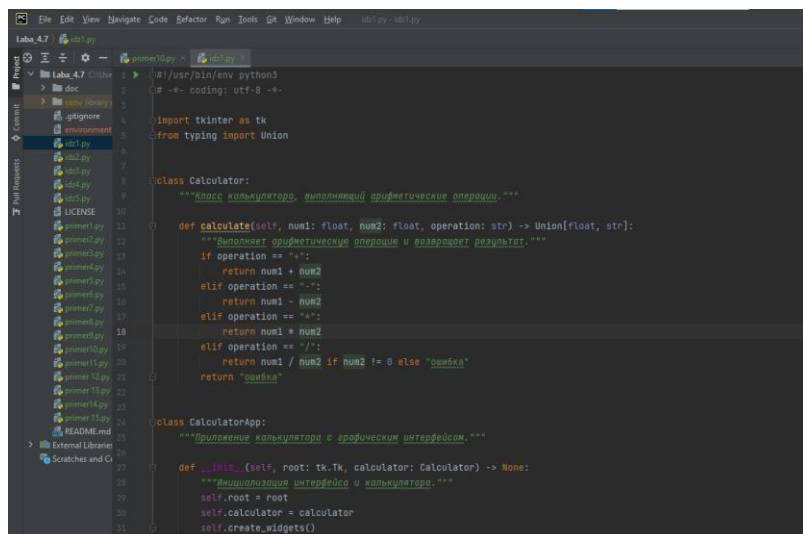


Рисунок 33. Выполнение первого индивидуального задания

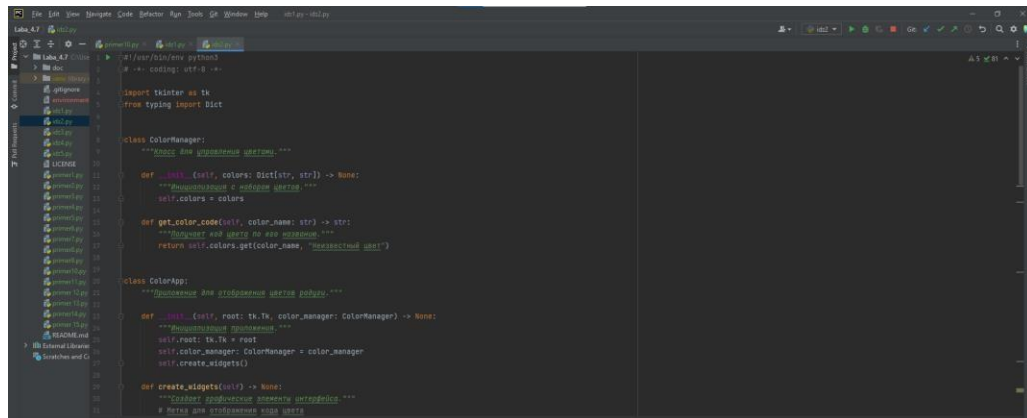
Далее запустим код и проверим его на выполнение.



Рисунок 34. Результат первого индивидуального задания

Создал новый файл под названием idz2.py.

Условие задания: напишите программу, состоящую из семи кнопок, цвета которых соответствуют цветам радуги. При нажатии на ту или иную кнопку в текстовое поле должен вставляться код цвета, а в метку – название цвета. Коды цветов в шестнадцатеричной кодировке: #ff0000 – красный, #ff7d00 – оранжевый, #ffff00 – желтый, #00ff00 – зеленый, #007dff – голубой, #0000ff – синий, #7d00ff – фиолетовый.



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

import tkinter as tk
from typing import Dict

class ColorManager:
    """Класс для управления цветами"""
    def __init__(self, colors: Dict[str, str]) -> None:
        """Инициализация списка цветов"""
        self.colors = colors

    def get_color_code(self, color_name: str) -> str:
        """Получить код цвета по названию"""
        return self.colors.get(color_name, "#ffffff")

class ColorApp:
    """Приложение для управления цветами"""
    def __init__(self, root: tk.Tk, color_manager: ColorManager) -> None:
        """Инициализация приложения"""
        self.root = root
        self.color_manager = color_manager
        self.create_widgets()

    def create_widgets(self) -> None:
        """Создание виджетов"""
        # Метка для отображения кода цвета
        code_label = tk.Label(self.root, text="#ffffff", font=("Arial", 14))
        code_label.pack(pady=10)

        # Текстовое поле для ввода названия цвета
        name_entry = tk.Entry(self.root, font=("Arial", 12))
        name_entry.pack(pady=5)

        # Кнопки для выбора цвета
        colors = {
            "Красный": "#ff0000",
            "Оранжевый": "#ff7d00",
            "Желтый": "#ffff00",
            "Зеленый": "#00ff00",
            "Голубой": "#007dff",
            "Синий": "#0000ff",
            "Фиолетовый": "#7d00ff"
        }

        for name, color in colors.items():
            button = tk.Button(self.root, text=name, background=color, font=("Arial", 12))
            button.pack(pady=5)
```

Рисунок 35. Выполнение второго индивидуального задания

Далее запустим код и проверим его на выполнение.

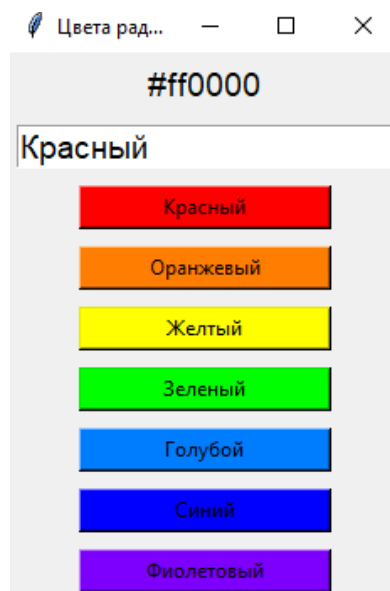


Рисунок 36. Результат второго индивидуального задания

Создал новый файл под названием idz3.py.

Условие задания: перепишите программу из пункта 8 так, чтобы интерфейс выглядел чуть другим образом, а именно, чтобы семь кнопок располагались горизонтально. А именно:



Рисунок 37. Пример выполнения из методички

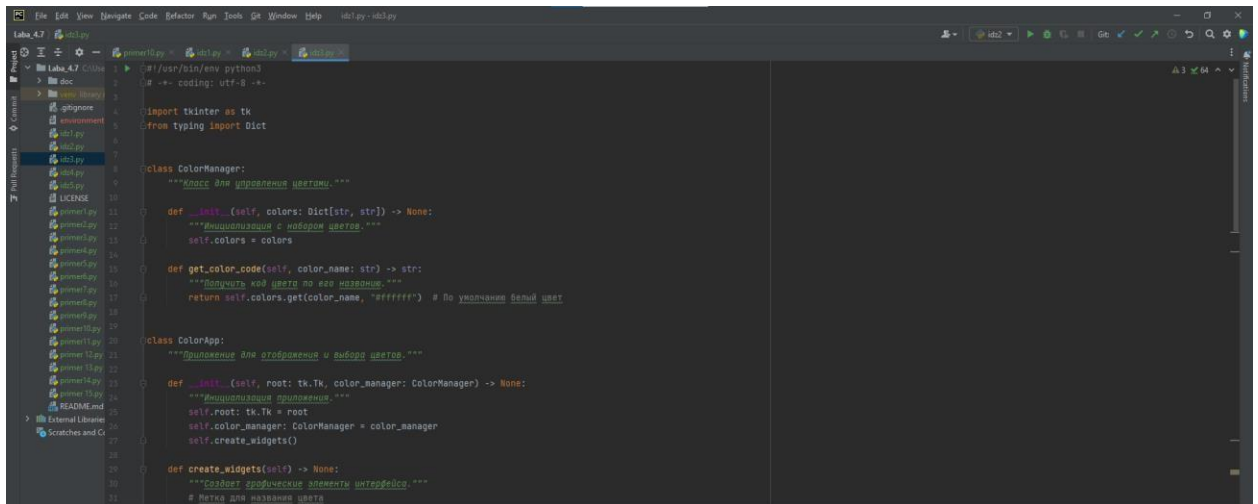


Рисунок 38. Выполнение третьего индивидуального задания

Далее запустим код и проверим его на выполнение.

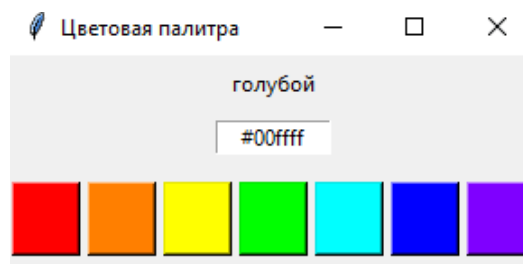


Рисунок 39. Результат третьего индивидуального задания

Создал новый файл под названием idz4.py.

Условие задания: напишите программу, состоящую из однострочного и многострочного текстовых полей и двух кнопок "Открыть" и "Сохранить". При клике на первую должен открываться на чтение файл, чье имя указано в поле класса Entry, а содержимое файла должно загружаться в поле типа Text.

При клике на вторую кнопку текст, введенный пользователем в экземпляр Text, должен сохраняться в файле под именем, которое пользователь указал в однострочном текстовом поле.

Файлы будут читаться и записываться в том же каталоге, что и файл скрипта, если указывать имена файлов без адреса.

Для выполнения практической работы вам понадобится функция `open` языка Python и методы файловых объектов чтения и записи.

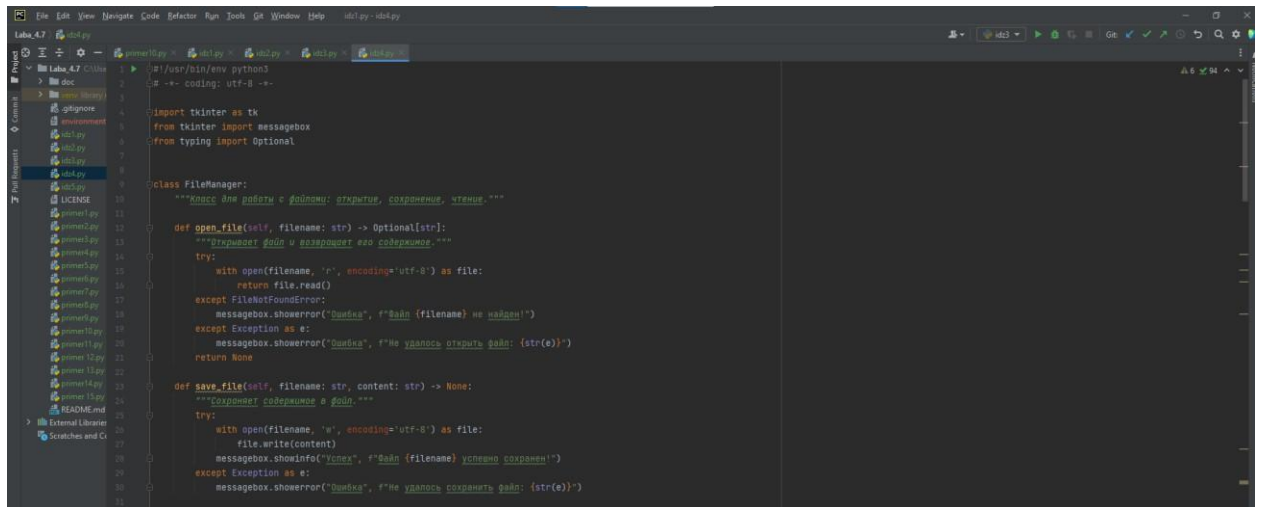


Рисунок 40. Выполнение четвертого индивидуального задания

Далее запустим код и проверим его на выполнение.

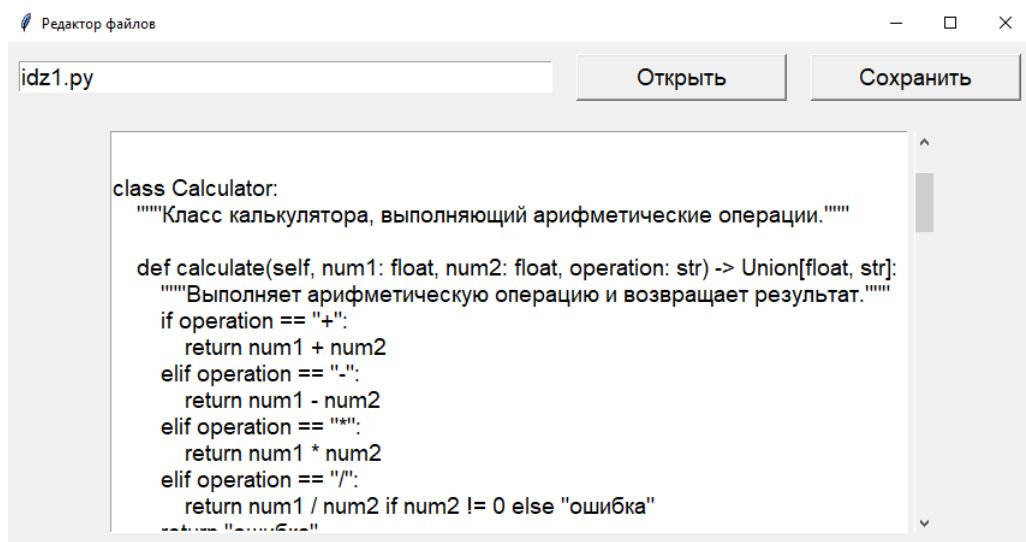


Рисунок 41. Результат четвертого индивидуального задания

Создал новый файл под названием idz5.py.

Условие задания: напишите программу, в которой имеется несколько объединенных в группу радиокнопок, индикатор которых выключен (`indicatoron=0`). Если какая-нибудь кнопка включается, то в метке должна отображаться соответствующая ей информация. Обычных кнопок в окне быть не должно.

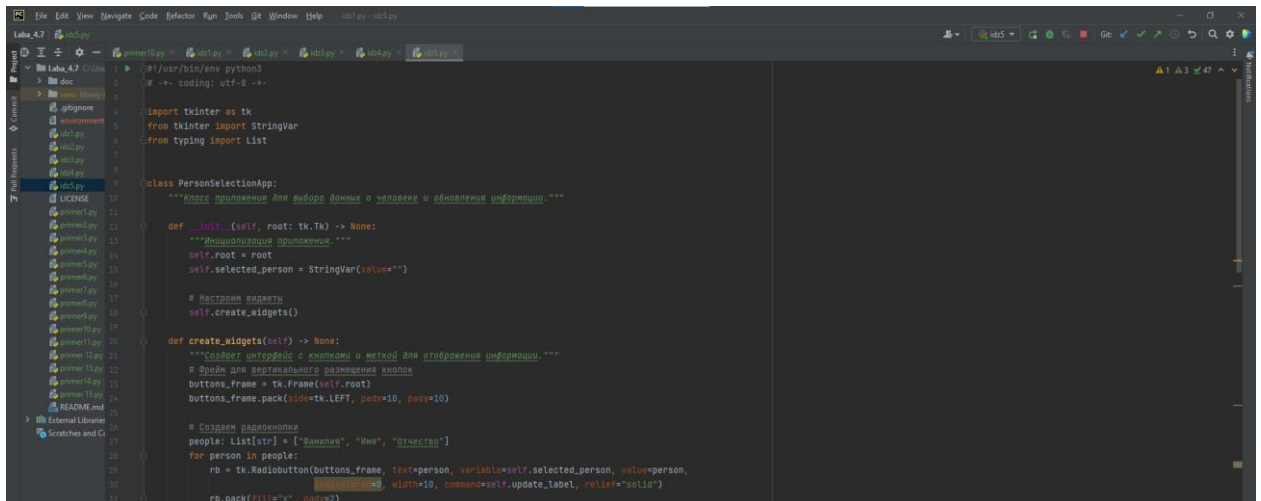


Рисунок 42. Выполнение пятого индивидуального задания

Далее запустим код и проверим его на выполнение.

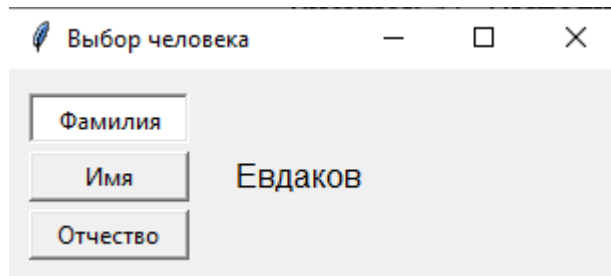


Рисунок 42. Результат пятого индивидуального задания

Задание 5.

После выполнения работы на ветке develop, слил ее с веткой main и отправил изменения на удаленный сервер.

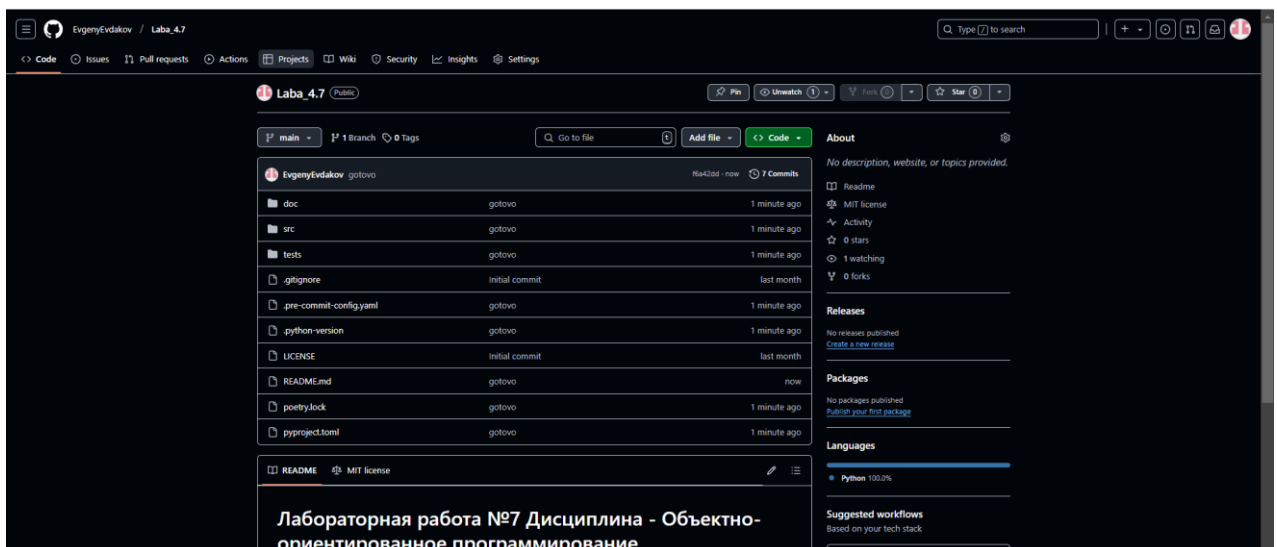


Рисунок 43. Готовый репозиторий

Ссылка: https://github.com/EvgenyEvdakov/Laba_4.7

Ответы на контрольные вопросы:

1. Какие существуют средства в стандартной библиотеке Python для построения графического интерфейса пользователя?

В стандартной библиотеке Python для построения графического интерфейса пользователя (GUI) существует несколько инструментов:

- Tkinter: Наиболее популярный и основной инструмент для создания GUI.
- Pygame: Используется для разработки игр, но также подходит для создания графического интерфейса.
- curses: Для создания текстовых интерфейсов в терминале (не графический, но часто используется для CLI).
- IDLE: Это встроенная среда разработки, использующая Tkinter для создания интерфейса.
- Turtle: Используется для обучения программированию через рисование и графику.

2. Что такое Tkinter?

Tkinter — это стандартная библиотека для создания графических интерфейсов пользователя в Python. Tkinter является оберткой вокруг библиотеки Tk, которая представляет собой набор инструментов для создания оконных приложений. Tkinter предоставляет доступ к множеству виджетов (кнопки, метки, текстовые поля и т.д.), которые можно использовать для создания GUI.

3. Какие требуется выполнить шаги для построения графического интерфейса с помощью Tkinter?

Основные шаги для создания графического интерфейса с использованием Tkinter:

- Импортировать Tkinter.
- Создать основное окно приложения (`root = tk.Tk()`).
- Добавить виджеты (например, кнопки, метки, текстовые поля) в окно.

- Разместить виджеты в окне (с помощью методов `pack()`, `grid()`, `place()`).
- Запустить главный цикл обработки событий (`root.mainloop()`).

4. Что такое цикл обработки событий?

Цикл обработки событий (или главный цикл) — это механизм, который позволяет программе реагировать на действия пользователя, такие как нажатия кнопок, движение мыши, изменения в полях ввода и т.д. Цикл обработки событий постоянно ожидает событий и вызывает соответствующие обработчики (функции) для этих событий.

Главный цикл запускается методом `root.mainloop()` и продолжается до тех пор, пока окно не будет закрыто.

5. Каково назначение экземпляра класса Tk при построении графического интерфейса с помощью Tkinter?

Экземпляр класса Tk является главным окном вашего приложения. Это окно, которое будет содержать все виджеты, с которыми взаимодействует пользователь. Он инициализирует внутренние механизмы Tkinter, включая главный цикл событий, и управляет отображением всех элементов интерфейса.

6. Для чего предназначены виджеты Button, Label, Entry и Text?

Button: Кнопка, которая реагирует на действия пользователя (например, клики). С помощью неё можно вызвать функцию или выполнить команду.

Label: Метка (ярлык), отображающая текст или изображение, используется для вывода информации.

Entry: Однострочное текстовое поле, в котором пользователь может ввести данные.

Text: Многострочное текстовое поле, в котором можно редактировать большой объем текста (с возможностью прокрутки).

7. Каково назначение метода pack() при построении графического интерфейса пользователя?

Метод `pack()` используется для размещения виджетов в окне. Он автоматически размещает виджеты в контейнере (например, в окне или фрейме) по определенным правилам. Виджеты могут быть размещены вертикально или горизонтально, в зависимости от параметров.

8. Как осуществляется управление размещением виджетов с помощью метода `pack()`?

Метод `pack()` имеет несколько параметров для управления размещением виджетов:

- `side`: Определяет сторону контейнера (например, TOP, BOTTOM, LEFT, RIGHT).
- `fill`: Устанавливает, как виджет должен растягиваться по оси (например, X, Y, BOTH).
- `expand`: Если установлено в True, виджет будет расширяться, чтобы занять доступное пространство.

9. Как осуществляется управление полосами прокрутки в виджете `Text`?

Для управления полосами прокрутки в виджете `Text` используется виджет `Scrollbar`. Полоса прокрутки связывается с виджетом `Text` через параметр `yscrollcommand`.

10. Для чего нужны тэги при работе с виджетом `Text`?

Тэги в `Text` используются для выделения или изменения атрибутов текста (например, цвет, стиль, шрифт). С помощью тэгов можно управлять стилями текста в определенных областях виджета `Text`.

11. Как осуществляется вставка виджетов в текстовое поле?

Вставка виджетов (например, кнопок или меток) в текстовое поле осуществляется с помощью метода `window_create()`. С помощью этого метода можно вставить объект Tkinter (например, виджет) в определенную позицию в тексте.

12. Для чего предназначены виджеты `Radiobutton` и `Checkbutton`?

Radiobutton: Предназначен для выбора одного из нескольких вариантов. Все радиокнопки в одной группе взаимодействуют друг с другом, и только одна из них может быть выбрана одновременно.

Checkbox: Предназначен для выбора нескольких вариантов (можно поставить несколько флажков одновременно). Каждый флажок может быть как выбран, так и не выбран.

13. Что такое переменные Tkinter и для чего они нужны?

Переменные Tkinter (например, StringVar, IntVar, BooleanVar) — это особые объекты, которые используются для связи данных между виджетами и программой. Они позволяют отслеживать изменения в виджетах и обновлять их состояние.

14. Как осуществляется связь переменных Tkinter с виджетами Radiobutton и Checkbox?

Для связи переменных с виджетами Radiobutton и Checkbox используется параметр `variable`. Переменная хранит текущее состояние виджета (выбран/не выбран или значение радиокнопки).

Вывод: приобрел навыки по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.