

# **Лабораторный практикум**

## **Защита информации посредством среды программирования Python**

## СОДЕРЖАНИЕ

Теоретическая часть .....	3
1.1 Знакомство с Python.....	3
1.2 Возможности Python.....	5
1.3 Установка Python .....	6
1.4 Первая программа на Python.....	7
1.5 Переменные Python.....	8
1.6 Условный оператор if.....	9
1.7 Ввод и вывод данных, форматирование вывода.....	11
1.8 Циклы.....	13
1.9 Модули.....	17
1.10 Функции.....	19
Практическая часть .....	21
Лабораторная работа №1 Visual Studio Code.....	21
Лабораторная работа №2 Генератор паролей.....	22
Лабораторная работа №3 Использование своих функций в качестве модулей .....	26
Лабораторная работа №4 Вывод в файлы и шифрование .....	30
Лабораторная работа №5 Создание графического интерфейса PyQt5 ...	35
Лабораторная работа №6 Конвертация .ui в .py .....	43
Лабораторная работа №7 Приложение для распознавания лиц .....	44
Лабораторная работа №8 Контроль целостности файлов .....	62
Лабораторная работа №9 Сканирование портов .....	70
Лабораторная работа №10 Компиляция .py в .exe.....	74

## Теоретическая часть

### 1.1 Знакомство с Python

Язык программирования Python 3 – это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для новичков. С его помощью можно решать задачи различных типов.

Язык Python обладает некоторыми примечательными особенностями, которые обуславливают его широкое распространение. Поэтому прежде, чем изучать Python, следует рассказать о его достоинствах и недостатках.

#### **Python 3: преимущества и недостатки языка:**

- 1) Python – интерпретируемый язык программирования. С одной стороны, это позволяет значительно упростить отладку программ, с другой – обуславливает сравнительно низкую скорость выполнения.
- 2) Динамическая типизация. В Python не надо заранее объявлять тип переменной, что очень удобно при разработке.
- 3) Хорошая поддержка модульности. Вы можете легко написать свой модуль и использовать его в других программах.
- 4) Встроенная поддержка Unicode в строках. В Python необязательно писать всё на английском языке, в программах вполне может использоваться ваш родной язык.
- 5) Поддержка объектно – ориентированного программирования. При этом его реализация в Python является одной из самых понятных.
- 6) Автоматическая сборка мусора, отсутствие утечек памяти. (Стоит отметить то, что Python написан на компилируемых языках программирования Си и С++, что может способствовать утечке памяти, ведь Python имеет огромное количество модулей, лишь в стандартной библиотеке Python на сегодняшний день более 200 модулей. Также

можно установить и другие модули с помощью терминала и `pip`, команда `pip install` осуществляет установку новых модулей. Если смотреть на данный язык с перспективой вперёд, нужно учитывать то, что в дальнейшем придётся читать документацию по каждому используемому модулю и даже читать код данных модулей.)

- 7) Интеграция с C/C++, если возможностей Python недостаточно.
- 8) Понятный и лаконичный синтаксис, способствующий ясному отображению кода. Удобная система функций позволяет при грамотном подходе создавать код, в котором будет легко разобраться другому человеку в случае необходимости. Также вы сможете научиться читать программы и модули, написанные другими людьми.
- 9) Огромное количество модулей, как входящих в стандартную поставку Python 3, так и сторонних. В некоторых случаях для написания программы достаточно лишь найти подходящие модули и правильно их скомбинировать. Таким образом, вы можете думать о составлении программы на более высоком уровне, работая с уже готовыми элементами, выполняющими различные действия.
- 10) Кроссплатформенность. Программа, написанная на Python, будет функционировать совершенно одинаково вне зависимости от того, в какой операционной системе она запущена. Отличия возникают лишь в редких случаях, и их легко заранее предусмотреть благодаря наличию подробной документации.

## 1.2 Возможности Python

### **Вещи, которые умеет делать Python:**

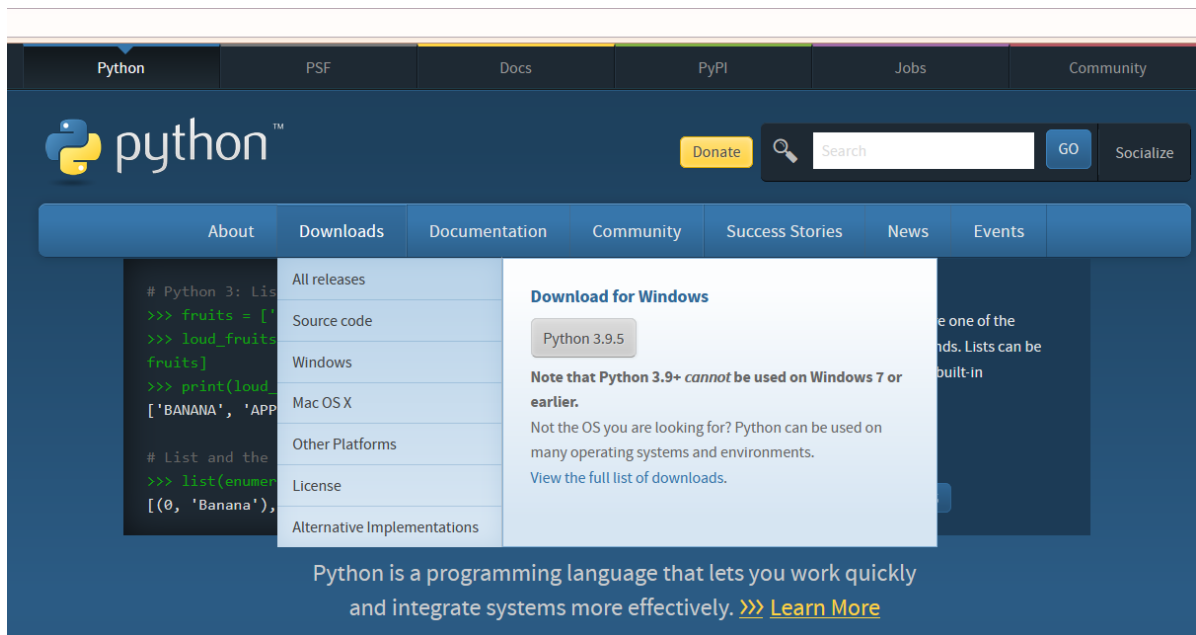
- 1) Работа с xml/html файлами;
- 2) Работа с http запросами;
- 3) GUI (графический интерфейс);
- 4) Создание веб – сценариев;
- 5) Работа с FTP;
- 6) Работа с изображениями, аудио и видео файлами;
- 7) Робототехника;
- 8) Программирование математических и научных вычислений.

И многое другое.

Таким образом, Python подходит для решения львиной доли повседневных задач, будь то резервное копирование, чтение электронной почты, либо же какая –нибудь игрушка. Язык программирования Python практически ничем не ограничен, поэтому также может использоваться в крупных проектах. К примеру, Python интенсивно применяется IT – гигантами, такими как, например, Google и Yandex. К тому же простота и универсальность Python делают его одним из лучших языков программирования.

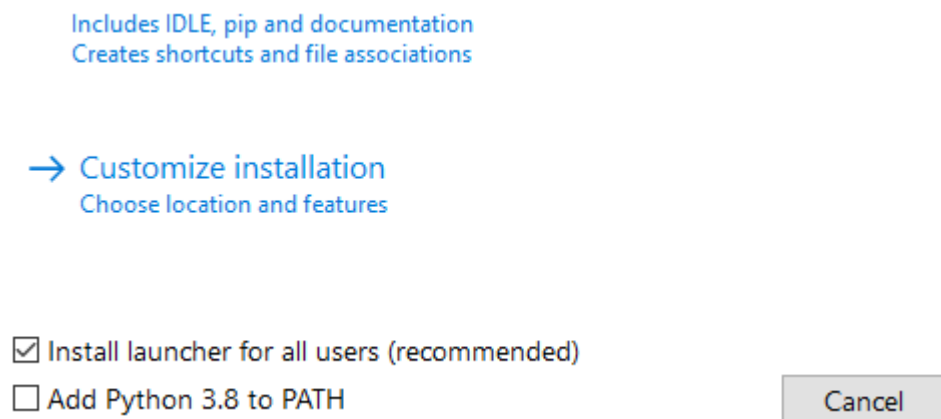
## 1.3 Установка Python

Скачиваем Python с официального сайта: [python.org](https://python.org)



На данный момент актуальная версия Python 3.9.5

Во время установки стоит **поставить галочку** Add Python to PATH. Дабы наш Python был по умолчанию доступен из терминала.



Если во время установки вы сделали всё правильно и добавили Python в PATH. То при открытии командной строки и вводе команды *python* вы увидите следующее:

```
C:\Users\lewdkeqing>python
Python 3.9.5 (tags/v3.9.5:0a7dcdb, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

## 1.4 Первая программа на Python

Для того чтобы запустить Python в интерактивном режиме, открываем командную строку и пишем *python* как было показано выше.

Вводим `print("Hello world!")`

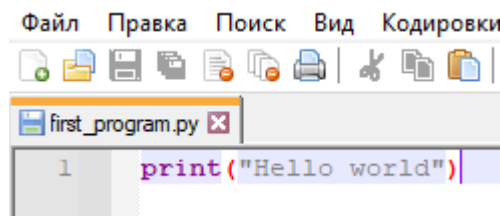
Результат должен быть таким.

```
C:\Users\lewdkeqing>python
Python 3.9.5 (tags/v3.9.5:0a7dcbbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
>>>
```

Чтобы данная программа работала как запускаемый файл, создадим файл с расширением .py

Например `first_program.py`. Напишем в данном файле всё ту же команду `print("Hello world!")`

Результат должен быть таким.



Открываем командную строку и переходим в директорию, где расположен наш файл. В моём случае это рабочий стол.

Если Python был добавлен в PATH, то при написании имени нашего файла `first_program.py` будет выполнена программа.

```
C:\Windows\system32\cmd.exe

C:\Users\lewdkeqing>cd Desktop

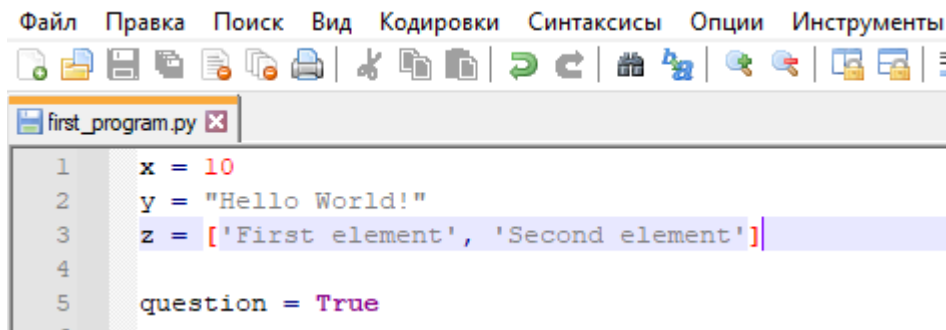
C:\Users\lewdkeqing\Desktop>first_program.py
Hello world

C:\Users\lewdkeqing\Desktop>
```

## 1.5 Переменные Python

Познакомимся с переменными в Python. Открываем созданный нами ранее файл `first_program.py`

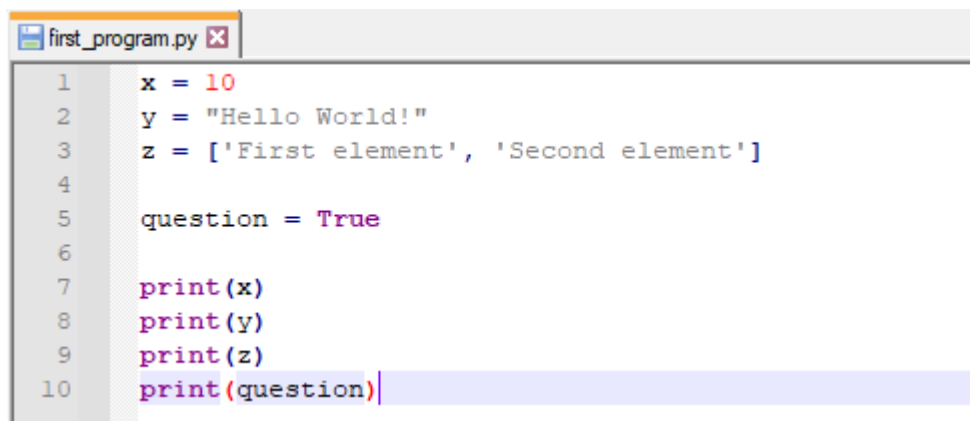
И объявляем переменные.



```
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты
first_program.py
1  x = 10
2  y = "Hello World!"
3  z = ['First element', 'Second element']
4
5  question = True
6
```

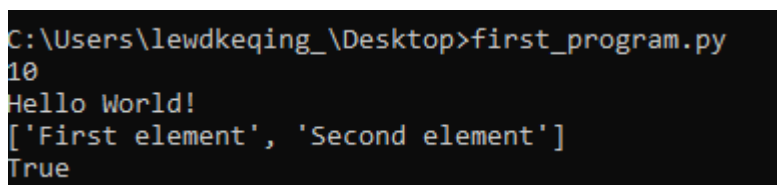
Python – это язык с динамической типизацией, а значит при объявлении переменной не обязательно указывать тип переменных. Для тех, кто забыл в языке C++ при объявлении переменных, нужно указывать тип, а Python это делает за нас.

Также попробуем вывести данные переменные, с помощью уже известной нам команды `print`



```
first_program.py
1  x = 10
2  y = "Hello World!"
3  z = ['First element', 'Second element']
4
5  question = True
6
7  print(x)
8  print(y)
9  print(z)
10 print(question)
```

Результат должен быть таким.



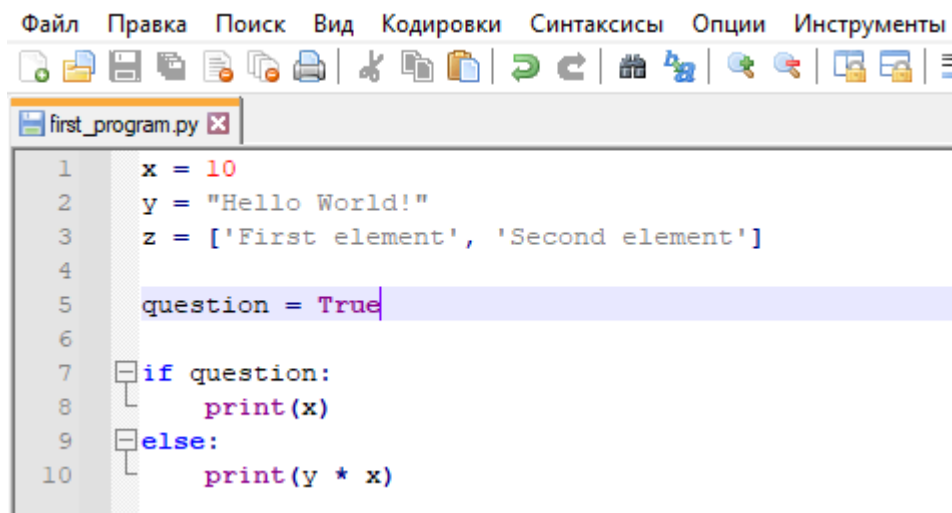
```
C:\Users\lewdkeqing\Desktop>first_program.py
10
Hello World!
['First element', 'Second element']
True
```



## 1.6 Условный оператор if

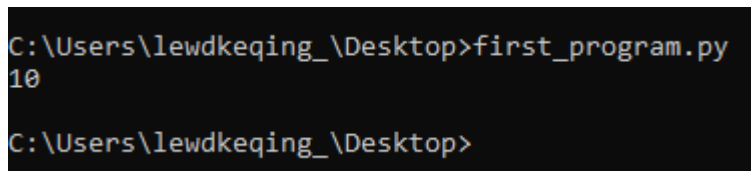
**Условная инструкция if – elif – else** (её ещё иногда называют оператором ветвления) – основной инструмент выбора в Python. Проще говоря, она выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия.

Модифицируем наш файл `first_program.py` вот в такой вид



```
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты
[Icons]
first_program.py
1  x = 10
2  y = "Hello World!"
3  z = ['First element', 'Second element']
4
5  question = True
6
7  if question:
8      print(x)
9  else:
10     print(y * x)
```

Запускаем из терминала, результат должен быть таким.



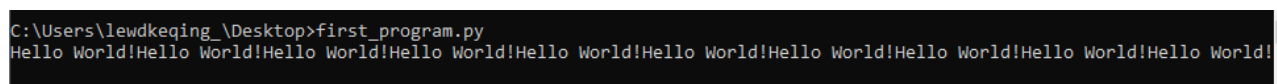
```
C:\Users\lewdkeqing\Desktop>first_program.py
10
C:\Users\lewdkeqing\Desktop>
```

Итак, что же произошло в нашей программе?

Условный оператор `if` проверяет булеву переменную `question`. И если она является правдой (`True`), то выводит на экран нашу переменную `x`. В ином случае выводит на экран переменную `y` помноженную на `x`.

Запишем в нашу булеву переменную `question = False` и посмотрим на результат вывода.

Если результат получился таким:



```
C:\Users\lewdkeqing\Desktop>first_program.py
Hello World!Hello World!Hello World!Hello World!Hello World!Hello World!Hello World!Hello World!Hello World!
```

То вы всё сделали правильно!

Если программа выдаёт ошибку **SyntaxError** или **IndentationError**, стоит проверить себя на **соблюдение синтаксиса**. Везде ли соблюдены отступы. Стоят ли двоеточия около конструкций `if`, `else` и т. п.

В любом случае если вы продвинетесь дальше в написании своего кода, рекомендую ознакомиться с [PEP8 – Руководство по написанию кода](#).

Естественно, условный оператор `if` работает не только с булевыми переменными. Также мы можем проверять условия и других переменных.

Например:

```
1  x = 10
2  y = "Hello World!"
3  z = ['First element', 'Second element']
4
5  question = False
6
7  if x > 8:
8      print(z[1])
9  elif x < 8:
10     print(z[0])
11  else:
12     print(x)
```

Что происходит в данном коде?

Мы проверяем переменную `x`, больше ли она числа 8. Меньше ли числа 8. В ином случае выводим на экран переменную `x`.

В случае, когда `x > 8` мы выводим второй элемент списка `z` с помощью команды `print(z[1])`.

Проверяем работу программы:

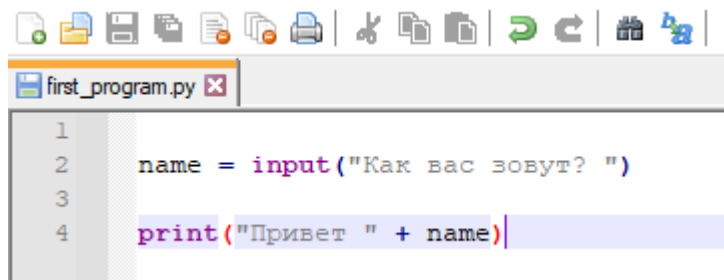
```
C:\Users\lewdkeqing\Desktop>first_program.py
Second element

C:\Users\lewdkeqing\Desktop>
```

## 1.7 Ввод и вывод данных, форматирование вывода

Один из самых простых вариантов ввода данных, осуществляется с помощью `input`.

Напишите программу по типу:



```
1
2 name = input("Как вас зовут? ")
3
4 print("Привет " + name)
```

Проверим работу программы:

При запуске программа спрашивает наше имя.

```
C:\Users\lewdkeqing\Desktop>first_program.py
Как вас зовут? _
```

А затем приветствует нас.

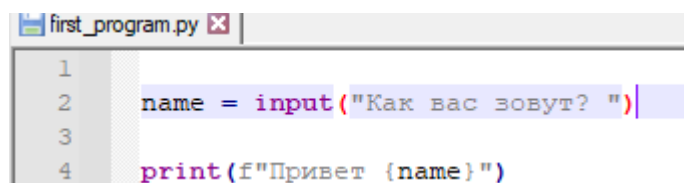
```
C:\Users\lewdkeqing\Desktop>first_program.py
Как вас зовут? Никита
Привет Никита

C:\Users\lewdkeqing\Desktop>
```

Как вы заметили при выводе переменной `name` мы использовали конструкцию `print("Привет " + name)`. Таким выводом можно пользоваться, но удобнее будет `print(f"Привет {name}")`

Также можно воспользоваться `print("Привет {0}".format(name))`

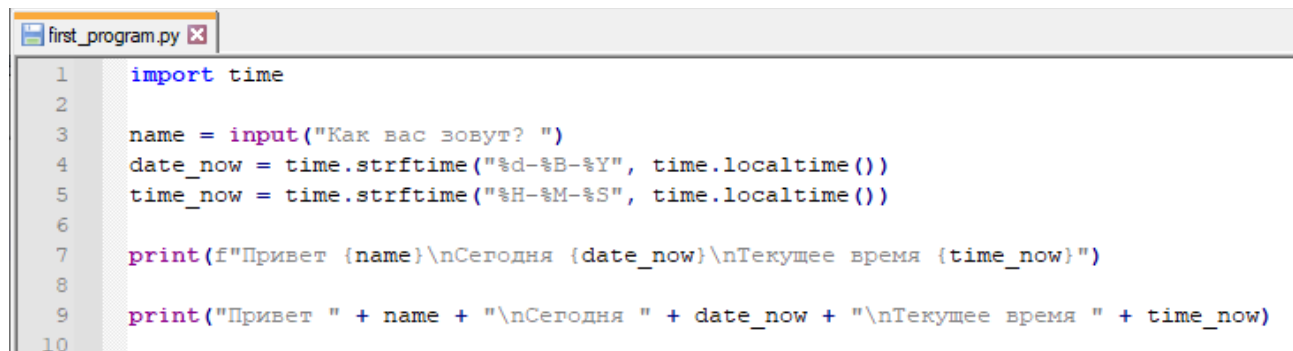
Попробуйте данные способы.



```
1
2 name = input("Как вас зовут? ")
3
4 print(f"Привет {name}")
```

Плюсом такого подхода является то, что при выводе сразу нескольких переменных нам не придётся писать постоянно +.

Посмотрите на 7 и 9 строчку на скриншоте ниже, обе строчки выполняют одну и ту же функцию, но насколько сильно они отличаются по написанию.

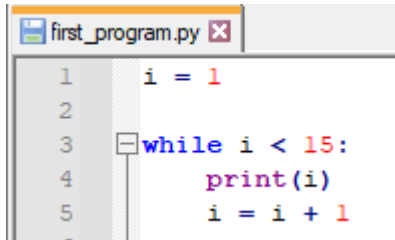


```
1  import time
2
3  name = input("Как вас зовут? ")
4  date_now = time.strftime("%d-%B-%Y", time.localtime())
5  time_now = time.strftime("%H-%M-%S", time.localtime())
6
7  print(f"Привет {name}\nСегодня {date_now}\nТекущее время {time_now}")
8
9  print("Привет " + name + "\nСегодня " + date_now + "\nТекущее время " + time_now)
10
```

## 1.8 Циклы

Существует два типа циклов, таких как for и while.

**Цикл while** – один из самых универсальных циклов в Python. Выполняет тело цикла до тех пор, пока условие истинно.

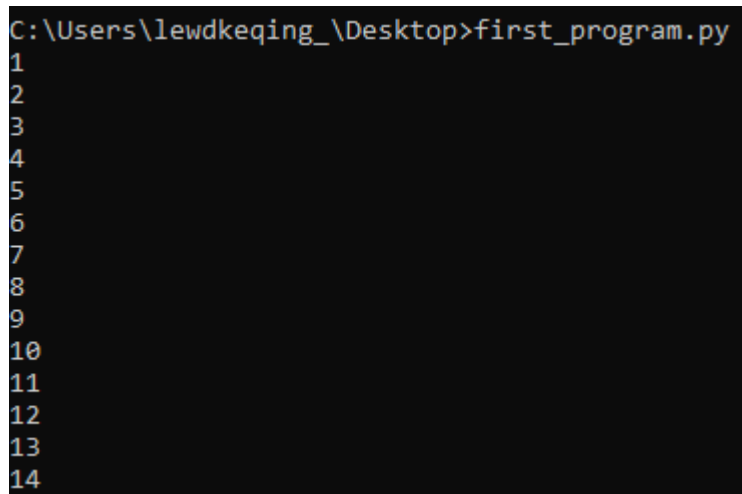


```
1 i = 1
2
3 while i < 15:
4     print(i)
5     i = i + 1
```

В самом начале работы мы присваиваем переменной *i* значение равное 1 после начинается тело цикла с условием выполнения: *Пока i < 15*:

Выводим значение *i* и добавляем к *i* + 1.

Результат работы:



```
C:\Users\lewdkeqing\Desktop>first_program.py
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Таким образом наша программа отработывает 14 раз, на 15 раз условие проверки является ложью  $15 < 15$  (False), Цикл перестаёт работать.

Если мы желаем, чтобы программа **отработывала 15 раз**, в начале мы можем присвоить *i* = 0 или поменять условие цикла на while i <= 15.

**Цикл for** – этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
first_program.py X
1 list = ['one', 'two', 'three', 'four', 'five']
2
3 for i in list:
4     print(i)
```

В данном примере у нас есть список list из слов: one, two, three, four, five

В данном случае цикл `for` будет работать до тех пор, пока не закончатся элементы списка.

```
C:\Users\lewdkeqing\Desktop>first_program.py
one
two
three
four
five
```

Также можно поступать и со строками.

```
first_program.py X
1 word = "strange word"
2
3 for i in word:
4     print(i)
```

```
C:\Users\lewdkeqing\Desktop>first
s
t
r
a
n
g
e
w
o
r
d
```

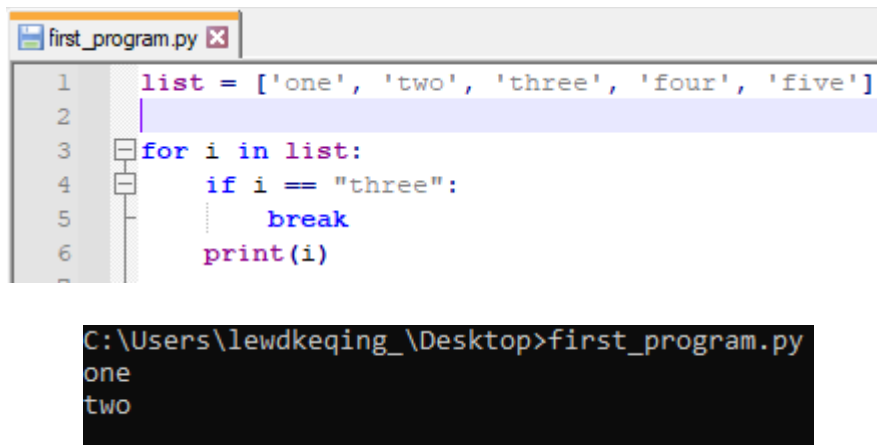
**Оператор `continue`** — начинает следующий проход цикла, минуя оставшееся тело цикла (`for` или `while`).

Например, в данном условии мы проверяем элементы списка на совпадение со строкой “three”, если такое встречается, то мы пропускаем данный элемент с помощью *continue* и продолжаем работу цикла выводя элементы:

```
first_program.py X
1 list = ['one', 'two', 'three', 'four', 'five']
2
3 for i in list:
4     if i == "three":
5         continue
6     print(i)
```

```
C:\Users\lewdkeqing\Desktop>first_program.py
one
two
four
five
```

**Оператор break** – досрочно прерывает цикл.



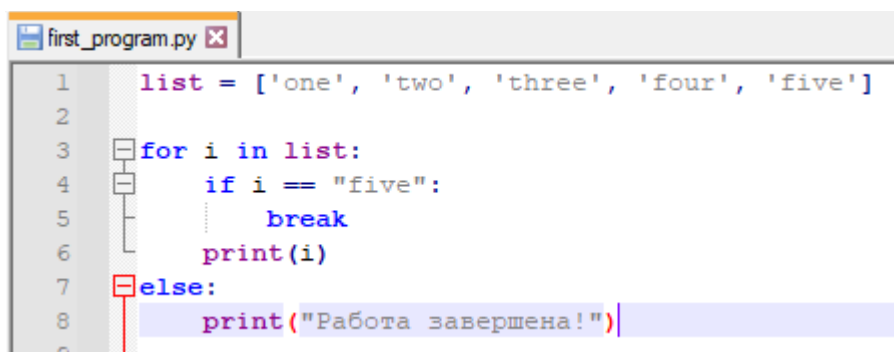
```
first_program.py X
1 list = ['one', 'two', 'three', 'four', 'five']
2
3 for i in list:
4     if i == "three":
5         break
6     print(i)
```

```
C:\Users\lewdkeqing\Desktop>first_program.py
one
two
```

**Else в циклах** – проверяет, был ли произведен выход из цикла инструкцией break, или же «естественным» образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошёл без помощи break.

**Пример работы Else в циклах.**

В данном примере мы выходим из цикла с помощью break, соответственно блок else не выполняется.

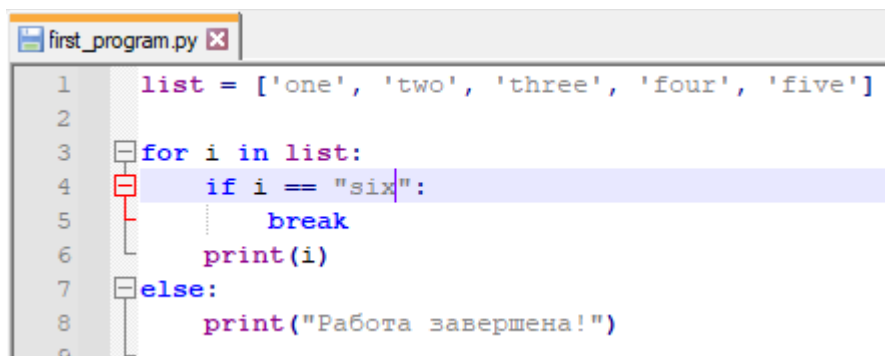


```
first_program.py X
1 list = ['one', 'two', 'three', 'four', 'five']
2
3 for i in list:
4     if i == "five":
5         break
6     print(i)
7 else:
8     print("Работа завершена!")
```

```
C:\Users\lewdkeqing\Desktop>first_program.py
one
two
three
four

C:\Users\lewdkeqing\Desktop>
```

В данном примере цикл завершается, происходит выход из цикла естественным образом и блок `else` выполняется.



```
1 list = ['one', 'two', 'three', 'four', 'five']
2
3 for i in list:
4     if i == "six":
5         break
6     print(i)
7 else:
8     print("Работа завершена!")
9
```

```
C:\Users\lewdkeqing\Desktop>first_program.py
one
two
three
four
five
Работа завершена!

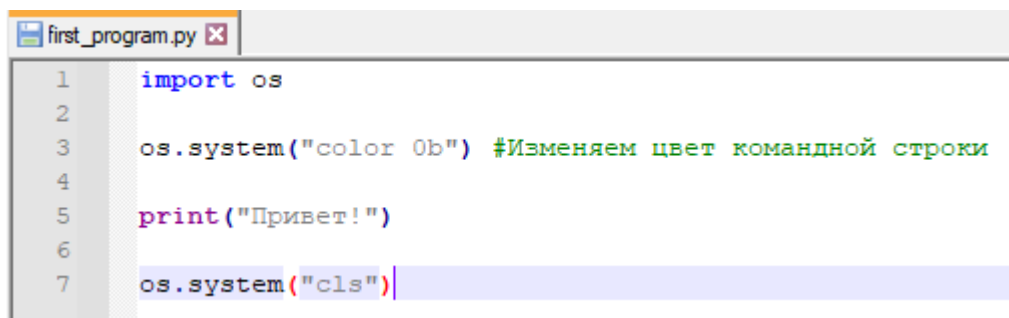
C:\Users\lewdkeqing\Desktop>
```



## 1.9 Модули

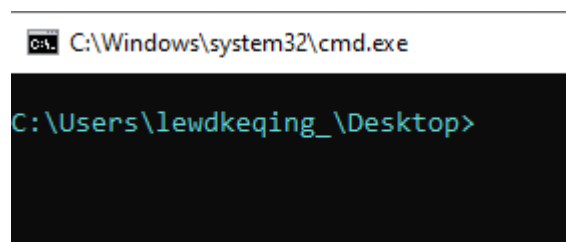
Модулем в Python называется любой файл с программой. Каждая программа может импортировать модуль и получить доступ к его классам, функциям и объектам. Нужно заметить, что модуль может быть написан не только на Python, а например, на C или C++.

Подключить модуль можно с помощью инструкции `import`. К примеру, подключим модуль `os`.

A screenshot of a text editor window titled 'first\_program.py'. The code is as follows:

```
1 import os
2
3 os.system("color 0b") #Изменяем цвет командной строки
4
5 print("Привет!")
6
7 os.system("cls")
```

Результат запуска данной программы.

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\system32\cmd.exe'. The prompt is 'C:\Users\lewdkeqing\Desktop>'. The text 'Привет!' is displayed in blue, and the prompt 'C:\Users\lewdkeqing\Desktop>' is also in blue, indicating that the 'color 0b' command was successful.

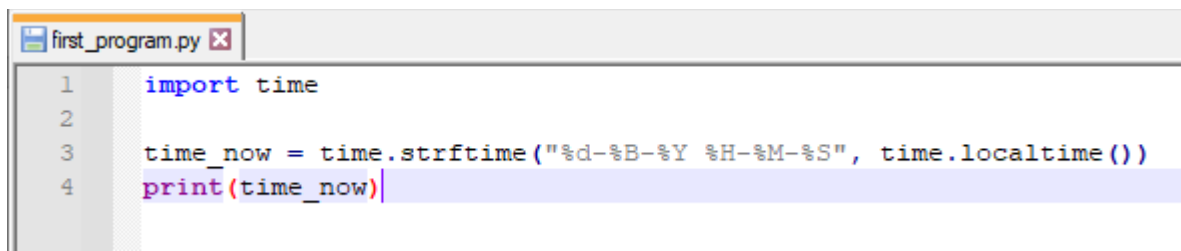
Что произошло? Мы импортировали модуль `os`. С помощью данного модуля и команды `os.system` мы сменили цвет консоли на `0b`. По сути, команда `os.system` – исполняет системную команду. В обычной консоли точно также можно поменять цвет с помощью `color 0b`.

Затем мы вывели в консоль слово “Привет!”, а после очистили консоль с помощью `os.system("cls")`.

### Модуль `time`.

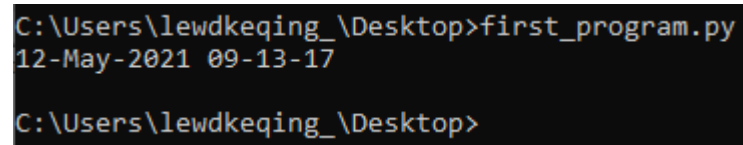
Данный модуль служит для вывода даты и времени в различных форматах.

Напишем простую программу по типу.



```
1 import time
2
3 time_now = time.strftime("%d-%B-%Y %H-%M-%S", time.localtime())
4 print(time_now)
```

Результат работы программы.



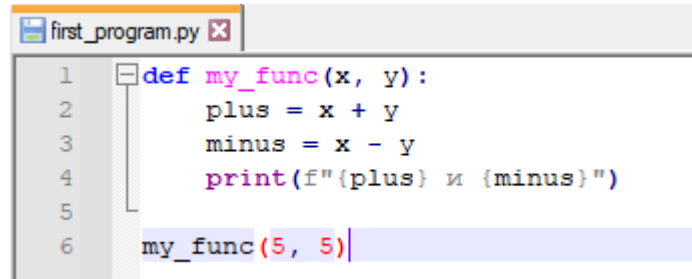
```
C:\Users\lewdkeqing\Desktop>first_program.py
12-May-2021 09-13-17

C:\Users\lewdkeqing\Desktop>
```

## 1.10 Функции

Функция в Python – объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции `def`.

Напишем простую функцию.

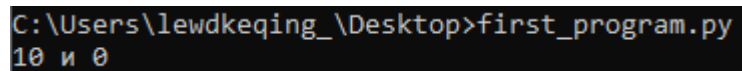


```
first_program.py X
1 def my_func(x, y):
2     plus = x + y
3     minus = x - y
4     print(f"{plus} и {minus}")
5
6 my_func(5, 5)
```

Функция начинается с `def`, затем мы задаём имя функции `my_func`, а после в скобках указываем передаваемые аргументы.

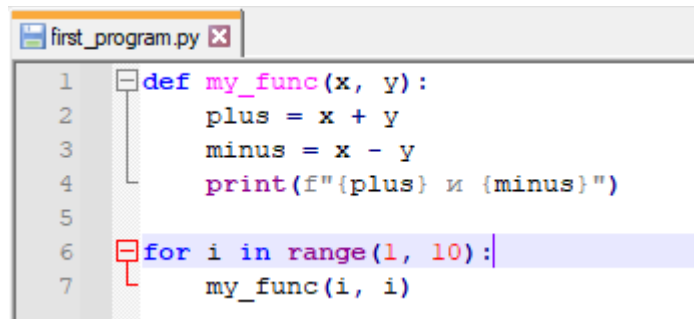
Внутри функции мы складываем и вычитаем `x` и `y`, а затем выводим их на экран.

С помощью `my_func(5, 5)` мы вызываем функцию со значениями 5,5. Результат работы программы.



```
C:\Users\lewdkeqing\Desktop>first_program.py
10 и 0
```

Также мы можем вызвать данную функцию несколько раз. Например так.



```
first_program.py X
1 def my_func(x, y):
2     plus = x + y
3     minus = x - y
4     print(f"{plus} и {minus}")
5
6 for i in range(1, 10):
7     my_func(i, i)
```

Результат работы программы.

```
C:\Users\lewdkeqing\Desktop>first_program.py
2 и 0
4 и 0
6 и 0
8 и 0
10 и 0
12 и 0
14 и 0
16 и 0
18 и 0
```

Также в функциях существует инструкция `return` которая возвращает из функции значения.

```
1 def my_func(x, y):
2     return x + y
3
4 one = my_func(5, 5)
5 two = my_func(2, 10)
6 three = my_func(one, two)
7
8 print(f"{one}, {two}, {three}")
```

Данная функция возвращает сумму `x` и `y`, при вызове функции мы указываем переменные, в эти переменные сохраняются наши результаты вызова, при вызове происходит возвращение значений, а затем мы просто выводим результаты переменных.

```
C:\Users\lewdkeqing\Desktop>first_program.py
10, 12, 22
```

**По окончании теоретической части, рекомендуется пройти тест. Который располагается рядом с практикумом «Тест Python»\index.html.**

## Практическая часть

### Лабораторная работа №1

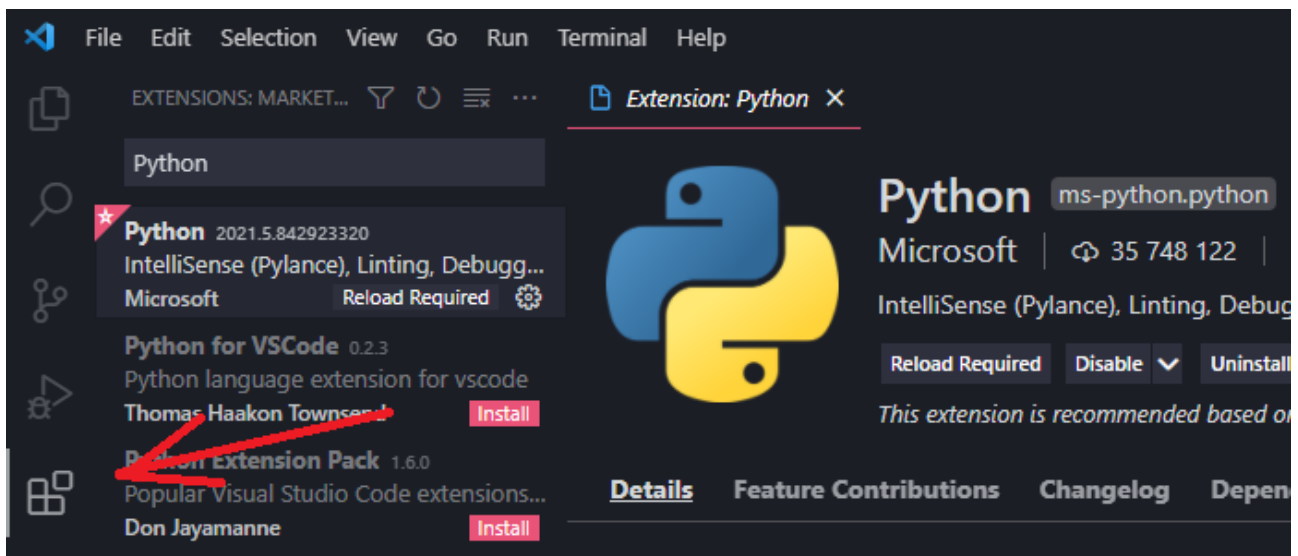
#### Visual Studio Code

Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб и облачных приложений.

Так как мы уже установили Python, для дальнейшего удобного написания кода будем использовать VSCode и несколько удобных расширений.

Устанавливаем VSCode с официального сайта <https://code.visualstudio.com/> После установки запускаем программу и переходим во вкладку Extensions.

Сделать это можно с помощью горячих клавиш Ctrl+Shift+X или с помощью бокового меню как показано на скриншоте ниже



В поиске расширений пишем “Python” и устанавливаем расширение от Microsoft, также для удобства рекомендую поставить Tabnine Autocomplete AI, данное расширение помогает писать код быстрее.

Также есть удобное расширение Python Docstring Generator, которое поможет комментировать код.

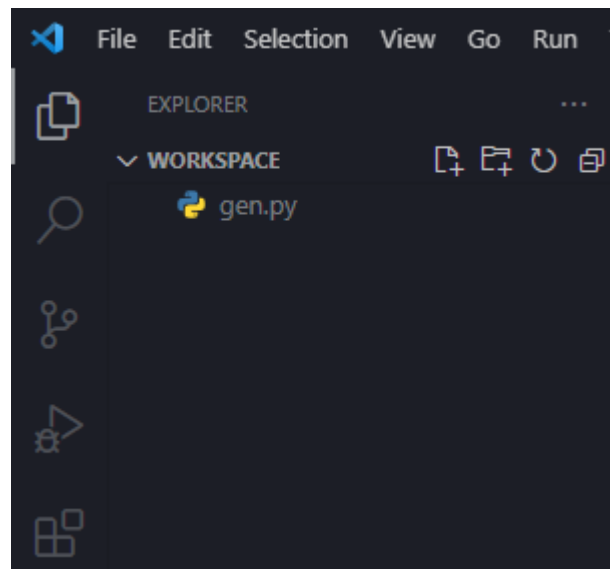
## Лабораторная работа №2

### Генератор паролей

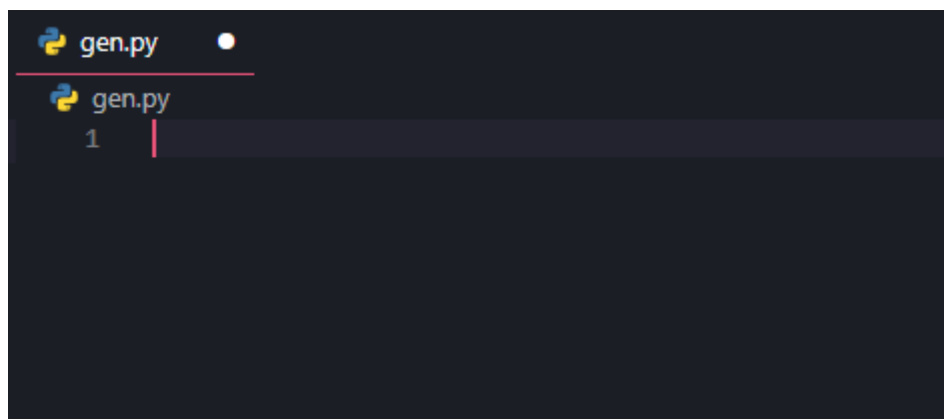
В рамках данного лабораторного практикума напишем функцию для генерации паролей.

После установки расширений создаём папку в любом удобном месте и запускаем VSCode.

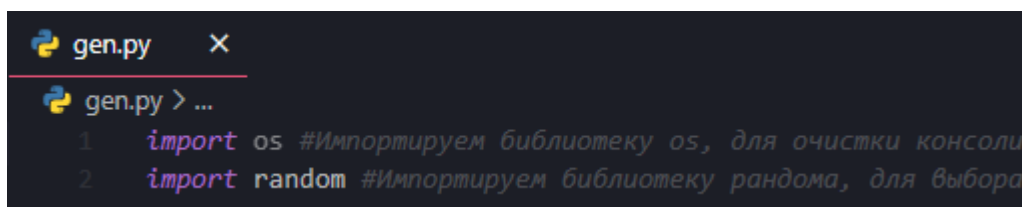
Слева сверху нажимаем File > Open Folder и открываем нашу папку. Слева в обозревателе нажимаем правой кнопкой мыши и нажимаем New File, назовём его gen.py



После создания файла кликаем по нему, откроется окно редактора кода.

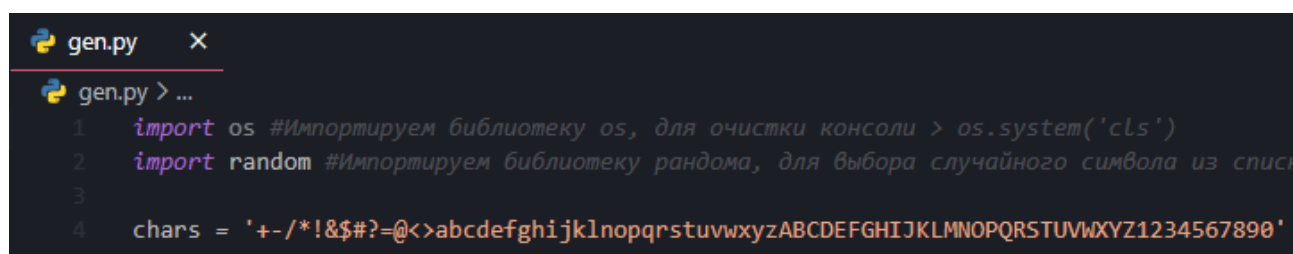


Импортируем две библиотеки `import os` и `import random`, первая будет использоваться в случае, когда нам нужно будет очистить командную строку с помощью `os.system("cls")`, а вторая библиотека для выбора случайного символа для будущего пароля.



```
gen.py X
gen.py > ...
1  import os #Импортируем библиотеку os, для очистки консоли
2  import random #Импортируем библиотеку рандома, для выбора
```

Далее создаём переменную, и пишем в неё последовательности символов, из которых будет строиться наш пароль.

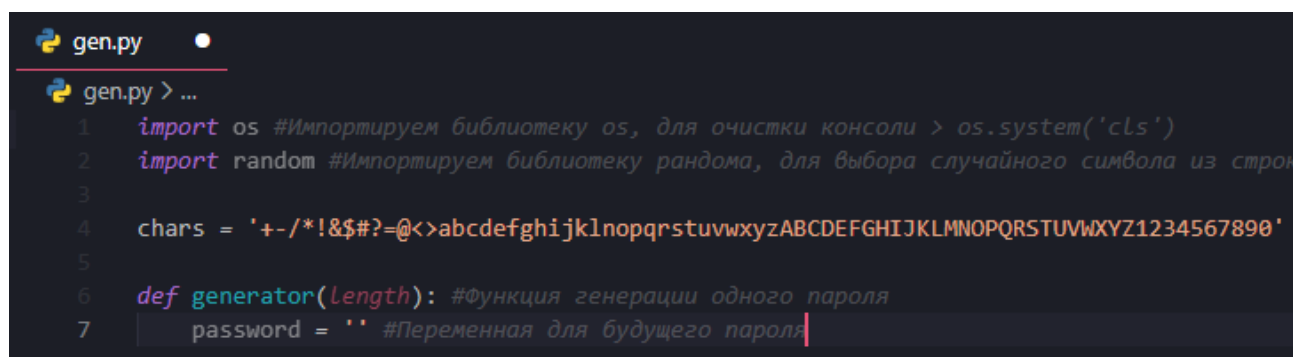


```
gen.py X
gen.py > ...
1  import os #Импортируем библиотеку os, для очистки консоли > os.system('cls')
2  import random #Импортируем библиотеку рандома, для выбора случайного символа из спис
3
4  chars = '+-/*!&$#?=@<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
```

Вы можете задать любую последовательность символов, в моём случае это — знаки, буквы латинского алфавита в нижнем и верхнем регистре, а также цифры.

Далее создаём функцию генератора и указываем одну передаваемую переменную `length`, которая будет отвечать за длину генерируемого пароля.

Внутри функции создаём пустую строчную переменную `password`.



```
gen.py •
gen.py > ...
1  import os #Импортируем библиотеку os, для очистки консоли > os.system('cls')
2  import random #Импортируем библиотеку рандома, для выбора случайного символа из строк
3
4  chars = '+-/*!&$#?=@<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
5
6  def generator(length): #Функция генерации одного пароля
7      password = '' #Переменная для будущего пароля
```

Далее создаём цикл, который будет работать по длине пароля. Внутри цикла пишем упрощённую конструкцию `password += random.choice(chars)`,


данная конструкция работает точно так же, как и `password = password + random.choice(chars)`. После выхода из цикла возвращаем сгенерированный пароль с помощью `return password`.

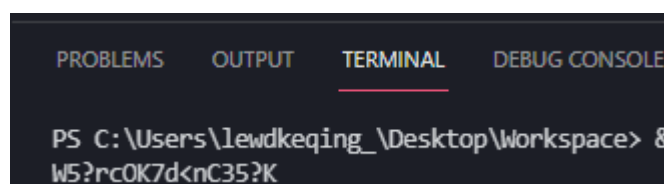
```
gen.py > generator
1  import os #Импортируем библиотеку os, для очистки консоли > os.system('cls')
2  import random #Импортируем библиотеку рандома, для выбора случайного символа из строк
3
4  chars = '+-/*!&$#?=@<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
5
6  def generator(length): #Функция генерации одного пароля
7      password = '' #Переменная для будущего пароля
8      for n in range(length):
9          password += random.choice(chars)
10     return password
```

Итоговая программа должна выглядеть так.

Чтобы убедиться в работе программы, напишем чуть ниже простой вызов функции и вывод результата на экран с помощью команды `print(generator(16))`.

```
gen.py > ...
1  import os #Импортируем библиотеку os, для очистки консоли > os.system('cls')
2  import random #Импортируем библиотеку рандома, для выбора случайного символа из строк
3
4  chars = '+-/*!&$#?=@<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
5
6  def generator(length): #Функция генерации одного пароля
7      password = '' #Переменная для будущего пароля
8      for n in range(length):
9          password += random.choice(chars)
10     return password
11
12  print(generator(16))
```

Нажимаем справа сверху кнопку запуска программы  или с помощью контекстного меню (правой кнопкой мыши) > Run Python File in Terminal.



PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Users\lewdkeqing\Desktop\Workspace> &  
W5?rc0K7d<nC35?K

Программа сгенерировала пароль



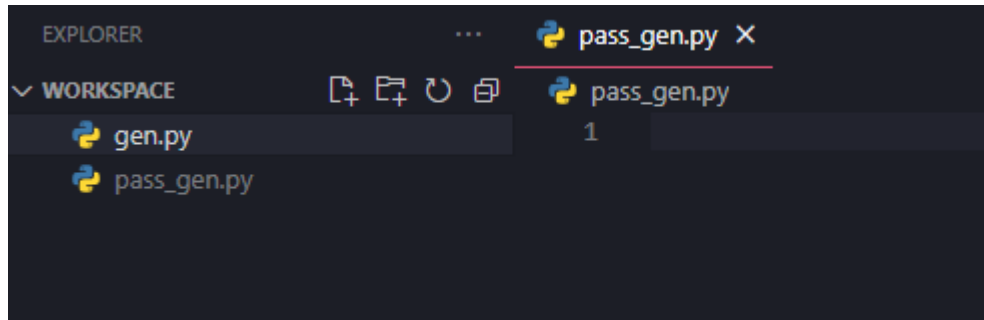
В данном случае мы передаём в функцию `generator` число 16, которое указывает длину будущего пароля. После формирования пароля функция возвращает нам результат в виде уже готового сгенерированного пароля, а мы с помощью `print` её выводим. Получается так, что внутри `print` мы осуществили вызов функции.

**ВАЖНО:** Невозможно осуществить вызов функции до самого описания функции, все функции должны быть написаны до их вызова.

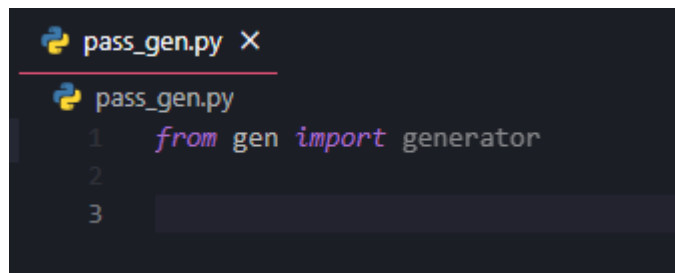
## Лабораторная работа №3

### Использование своих функций в качестве модулей

Создадим в папке еще один файл

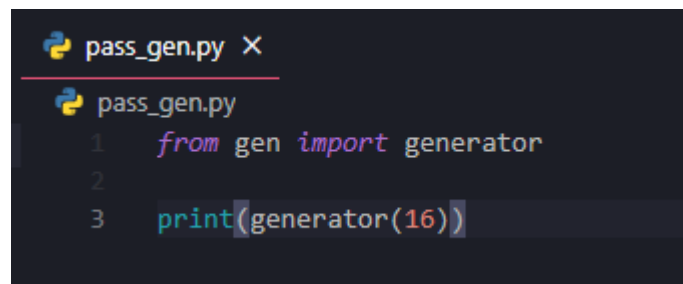


Импортируем из прошлого файла, функцию `generator` с помощью команды `from gen import generator`, таким образом мы можем дополнять свои программы написанными ранее функциями.



В таком же ключе работают и другие модули, мы просто добавляем их в свою программу и пользуемся уже готовыми написанными функциями. В этом и прелесть языка Python. Программу можно написать, просто прописывая логику работы от модуля к модулю.

Попробуем проверить работу. Напишем `print(generator(16))` и запустим программу.



```
PS C:\Users\lewdkeqing\Desktop\Workspace>
69teoPuKn?>AL1/h
```

Теперь попробуем написать генератор сразу нескольких паролей по желанию, при запуске программы.

Создаем переменную `number_of_passwords` и `length_of_passwords`, в которых просим ввести данные с помощью ранее изученного метода `input`.

```
pass_gen.py > ...
1  from gen import generator
2
3  number_of_passwords = int(input("Введите количество паролей: "))
4  length_of_passwords = int(input("Введите длину паролей: "))
5
```

Как вы могли заметить в данном случае мы обернули метод `input` в скобки, а перед скобками указали `int` – это метод конвертации введенных значений в целые числа.

Далее создаём функцию и передаём в неё сразу две переменные, созданные нами ранее.

```
pass_gen.py > many_generator
1  from gen import generator
2
3  number_of_passwords = int(input("Введите количество паролей: "))
4  length_of_passwords = int(input("Введите длину паролей: "))
5
6  def many_generator(number_of_passwords, length_of_passwords):
```

Затем объявляем переменную `passwords = []` и пишем цикл, в котором будем генерировать определённое количество паролей.

```

pass_gen.py > many_generator
1  from gen import generator
2
3  number_of_passwords = int(input("Введите количество паролей: "))
4  length_of_passwords = int(input("Введите длину паролей: "))
5
6  def many_generator(number_of_passwords, length_of_passwords):
7      passwords = []
8      for n in range(number_of_passwords):
9          passwords.append(generator(length_of_passwords))
10     return passwords

```

Данный цикл работает по переменной количества паролей, внутри цикла при каждой итерации происходит добавление в список `passwords` нового пароля, при каждой итерации используется наш ранее написанный модуль `gen` и функция `generator`. Метод `.append` позволяет добавить новый пароль в список.


По окончании работы цикла, мы возвращаем список из сгенерированных паролей.

Проверим работу программы добавив ниже строку вывода паролей.

```

1  from gen import generator
2
3  number_of_passwords = int(input("Введите количество паролей: "))
4  length_of_passwords = int(input("Введите длину паролей: "))
5
6  def many_generator(number_of_passwords, length_of_passwords):
7      passwords = []
8      for n in range(number_of_passwords):
9          passwords.append(generator(length_of_passwords))
10     return passwords
11
12  print(many_generator(number_of_passwords, length_of_passwords))

```

Нажимаем справа сверху кнопку запуска программы  или с помощью контекстного меню (правой кнопкой мыши) > Run Python File in Terminal.

```

PS C:\Users\lewdkeqing\Desktop\Workspace>
Введите количество паролей: 30

```

```

PS C:\Users\lewdkeqing\Desktop\Workspace>
Введите количество паролей: 30
Введите длину паролей: 16

```

Получаем список из 30 паролей, длиной в 16 символов.

```
PS C:\Users\lewdkeqing\Desktop\Workspace>
Введите количество паролей: 30
Введите длину паролей: 16
['3BoMa8C8a&-GmHvD', 'Z*dv$$X8LAjouU!e', '
z52Bj<Izs0fW*', 'DA6a0COA$a*k#+qK', 'NPaPC
@wxn$P/H', 'zxocF5GLbvkUV@H2', 'yC/1IuWoT8
```

Если хотим вывести каждый пароль на новой строке в консоли, можно использовать такой метод.

```
pass_gen.py > ...
1  from gen import generator
2
3  number_of_passwords = int(input("Введите количество паролей: "))
4  length_of_passwords = int(input("Введите длину паролей: "))
5
6  def many_generator(number_of_passwords, length_of_passwords):
7      passwords = []
8      for n in range(number_of_passwords):
9          passwords.append(generator(length_of_passwords))
10     return passwords
11
12 for psswr in many_generator(number_of_passwords, length_of_passwords):
13     print(psswr)
```

```
PS C:\Users\lewdkeqing\Desktop\Workspace>
Введите количество паролей: 16
Введите длину паролей: 30
=3>p&S1T$r+r&002&I8JX=evEuk-bXp
?4ywlpc*63@ae0qPkD3C1E3nuLj*=@
T=0@e4P2T+!2K8N/>sPNp&dos<9A0T
```

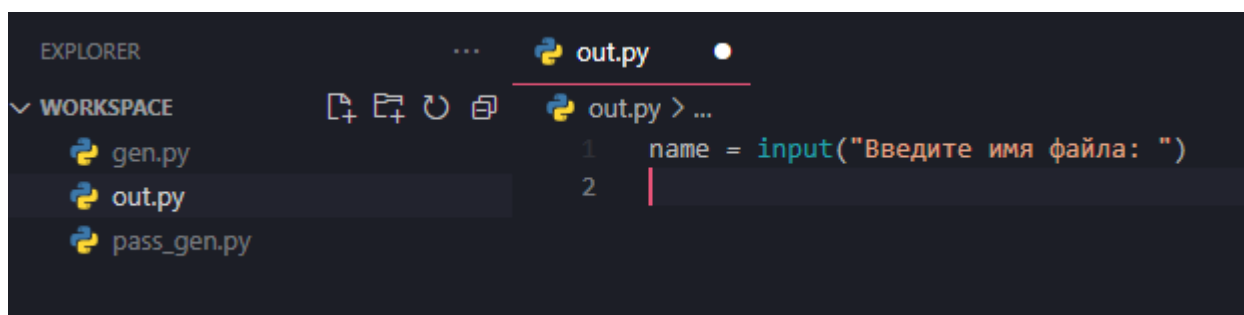
**Попробуйте придумать другой метод вывода паролей с новой строки.**

## Лабораторная работа №4

### Вывод в файлы и шифрование

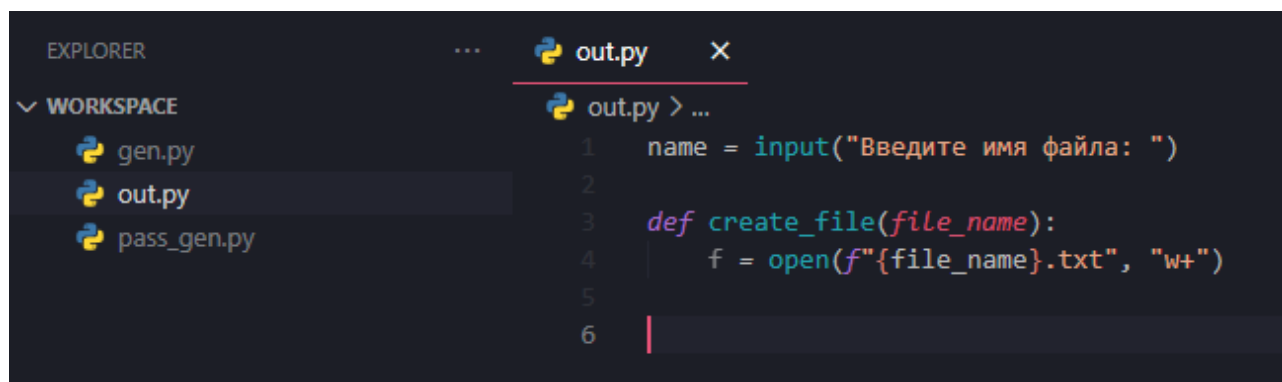
В этой лабораторной работе мы подробнее познакомимся с выводом информации в файлы, а также разберемся с модулем для шифрования.

Создадим новый .py файл, например out.py в первой строке попросим ввести имя будущего файла.



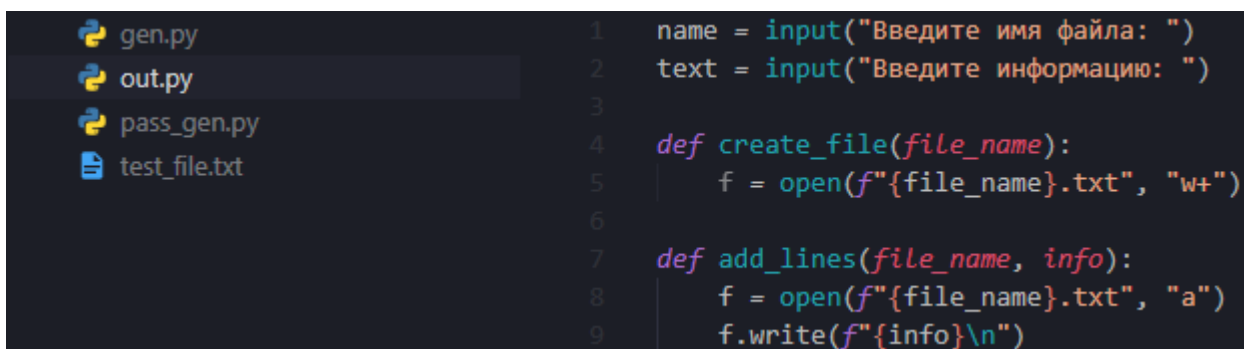
```
1 name = input("Введите имя файла: ")
2 |
```

Затем прописываем функцию, которая будет создавать пустой файл с нашим именем.



```
1 name = input("Введите имя файла: ")
2 |
3 def create_file(file_name):
4     f = open(f"{file_name}.txt", "w+")
5 |
6 |
```

Затем прописываем функцию для добавления записей в файл и добавляем ввод записываемых строк.



```
1 name = input("Введите имя файла: ")
2 text = input("Введите информацию: ")
3 |
4 def create_file(file_name):
5     f = open(f"{file_name}.txt", "w+")
6 |
7 def add_lines(file_name, info):
8     f = open(f"{file_name}.txt", "a")
9     f.write(f"{info}\n")
10 |
```

Если вы уже знакомы с методом вывода информации в файл, у вас мог возникнуть вопрос, для чего существует функция создания пустого файла, ведь при вызове второй функции на добавление записей, файл автоматически будет создан.

Ответ прост, данная функция будет полезна в случае, если мы захотим очистить файл или просто его переписать. Удобно иметь функции на все случаи.

Проверим работу программы, добавив вызов функций.

```
out.py > ...
1  name = input("Введите имя файла: ")
2  text = input("Введите информацию: ")
3
4  def create_file(file_name):
5      f = open(f"{file_name}.txt", "w+")
6
7  def add_lines(file_name, info):
8      f = open(f"{file_name}.txt", "a")
9      f.write(f"{info}\n")
10
11  create_file(name)
12  add_lines(name, text)
```

Запускаем программу.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
PS C:\Users\lewdkeqing\Desktop\Workspace>
Введите имя файла: test_file
Введите информацию: Hello!
PS C:\Users\lewdkeqing\Desktop\Workspace>
```

```
out.py  test_file.txt X
test_file.txt
1  Hello!
2
```

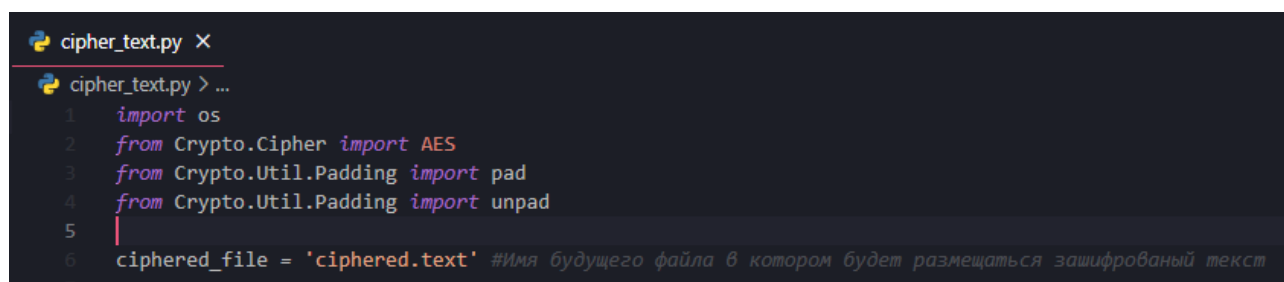
Запустите программу ещё раз и убедитесь, что новая информация не записывается в файл, вызовите функцию так, чтобы происходило добавление новых строк в файл test\_file.txt

Перейдите в командную строку и установите модуль `pycryptodome` с помощью команды: `pip install pycryptodome`.

Для установки модуля необходимо подключение к интернету!

Создайте новый файл с именем `cipher_text.py`

Импортируйте модули, как показано на скриншоте ниже

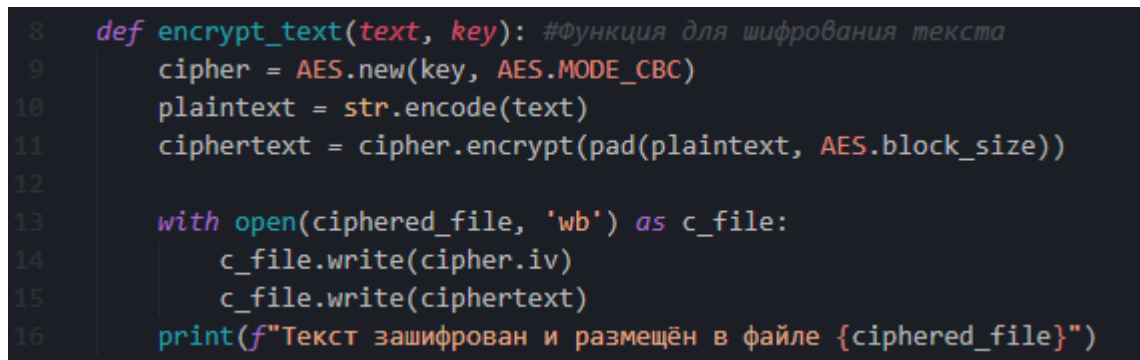


```
1 import os
2 from Crypto.Cipher import AES
3 from Crypto.Util.Padding import pad
4 from Crypto.Util.Padding import unpad
5
6 ciphered_file = 'ciphered.text' #Имя будущего файла в котором будет размещаться зашифрованный текст
```

Также создаём переменную с именем будущего файла, в котором будет храниться зашифрованный текст.

Создаём функцию `encrypt_text(text, key)`

Прописываем логику внутри функции, с помощью которой будет производиться шифрование текста.



```
8 def encrypt_text(text, key): #Функция для шифрования текста
9     cipher = AES.new(key, AES.MODE_CBC)
10    plaintext = str.encode(text)
11    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
12
13    with open(ciphered_file, 'wb') as c_file:
14        c_file.write(cipher.iv)
15        c_file.write(ciphertext)
16    print(f"Текст зашифрован и размещён в файле {ciphered_file}")
```

Для того чтобы понять логику работы данной функции, необходимо изучить сам модуль `pycryptodome` и необходимую документацию. В рамках данного лабораторного практикума мы не будем вдаваться в подробности, наша задача реализация готовых функций для работы программы. В документации автора модуля есть инструкции, которым мы просто следуем.



Коротко, переменная cipher содержит в себе копию функциональности из модуля и ключ, а переменная plaintext строку, которую она получает при вызове нашей функции. Далее происходит преобразование текста в переменной ciphertext, а затем мы открываем файл и записываем в него зашифрованный текст.

Теперь создаём функцию decrypt\_text(key) для дешифрования текста.

```
19 def decrypt_text(key): #Функция для дешифрования текста
20     with open(ciphered_file, 'rb') as c_file:
21         iv = c_file.read(16)
22         ciphertext = c_file.read()
23         cipher = AES.new(key, AES.MODE_CBC, iv)
24         plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
25         print(f"{ciphered_file}: " + plaintext.decode())
```

Данная функция служит для того, чтобы открыть файл с зашифрованным текстом, прочитать необходимые строки, а затем произвести дешифрование.

Теперь, когда наши функции готовы, создаём бесконечный цикл:

```
27 while True:
28     os.system("cls")
29     print(f" 1 - Зашифровать текст в файл {ciphered_file}")
30     print(f" 2 - Расшифровать текст из файла {ciphered_file}")
31     print(" 0 - Выйти")
32     question = input("\n Выберите действие: ")
33     if question == "1":
34         os.system("cls")
35         c_text = input("Введите текст, для шифрования: ")
36         os.system("cls")
37         keyA = str.encode(input("Введите ключ (16-бит): "))
38         os.system("cls")
39         try:
40             encrypt_text(c_text, keyA)
41         except ValueError:
42             os.system("cls")
43             print("Неверный ключ!\nКлюч должен быть строго длиной в 16-бит! Длина введенного ключа " + str(len(keyA)))
44             x = input("Нажмите любую клавишу, чтобы продолжить")
45     elif question == "2":
46         os.system("cls")
47         keyA = str.encode(input("Введите ключ (16-бит): "))
48         try:
49             decrypt_text(keyA)
50         except ValueError:
51             os.system("cls")
52             print("Неверный ключ!\nКлюч должен быть строго длиной в 16-бит! Длина введенного ключа " + str(len(keyA)))
53             x = input("Нажмите любую клавишу, чтобы продолжить")
54     elif question == "0":
55         os.system("cls")
56         exit()
57     else:
58         os.system("cls")
59         print("Выберите правильный вариант!")
60         x = input("Нажмите любую клавишу, чтобы продолжить")
```

**Объясните для чего служит данный бесконечный цикл.**

В итоге должен получиться рабочий скрипт, который работает вот так:

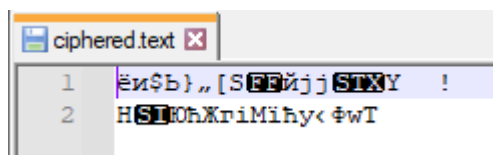
```
1 - Зашифровать текст в файл ciphered.text
2 - Расшифровать текст из файла ciphered.text
0 - Выйти
```

Выберите действие: 1

Введите текст, для шифрования: Привет!

Введите ключ (16-бит): mysecretpassword

Текст зашифрован и размещён в файле ciphered.text  
Нажмите любую клавишу, чтобы продолжить



```
1  ëи$б}„[SFFйjjSTXY !
2  HSt0hXrImiHy<fwT
```

```
1 - Зашифровать текст в файл ciphered.text
2 - Расшифровать текст из файла ciphered.text
0 - Выйти
```

Выберите действие: 2

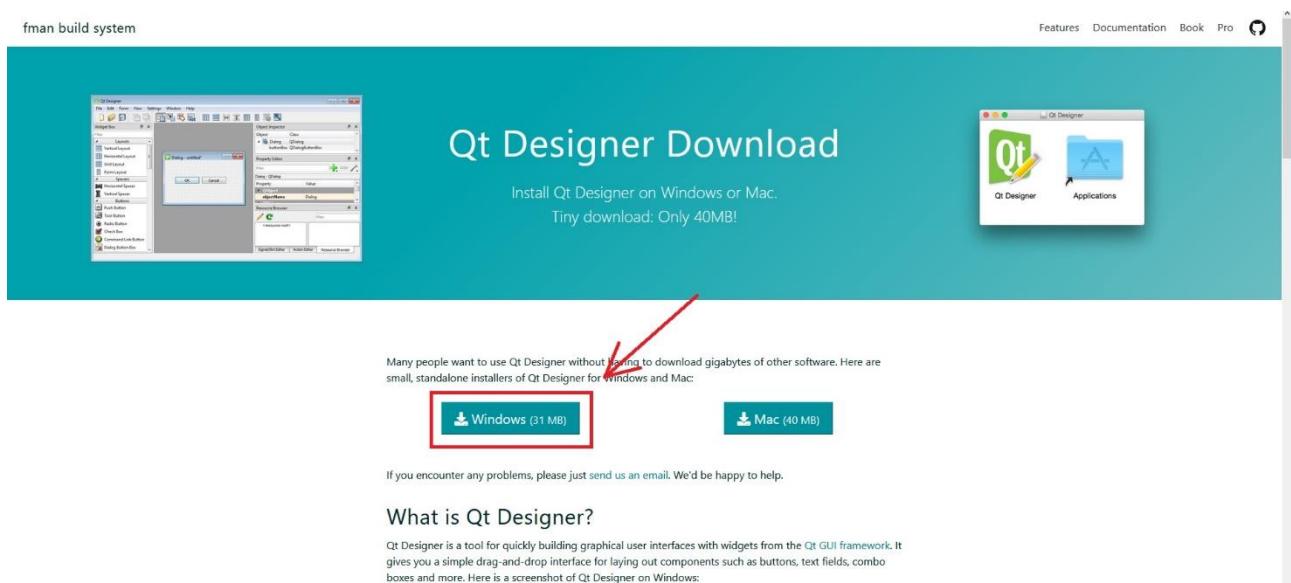
Введите ключ (16-бит): mysecretpassword  
ciphered.text: Привет!  
Нажмите любую клавишу, чтобы продолжить

## Лабораторная работа №5

### Создание графического интерфейса PyQt5

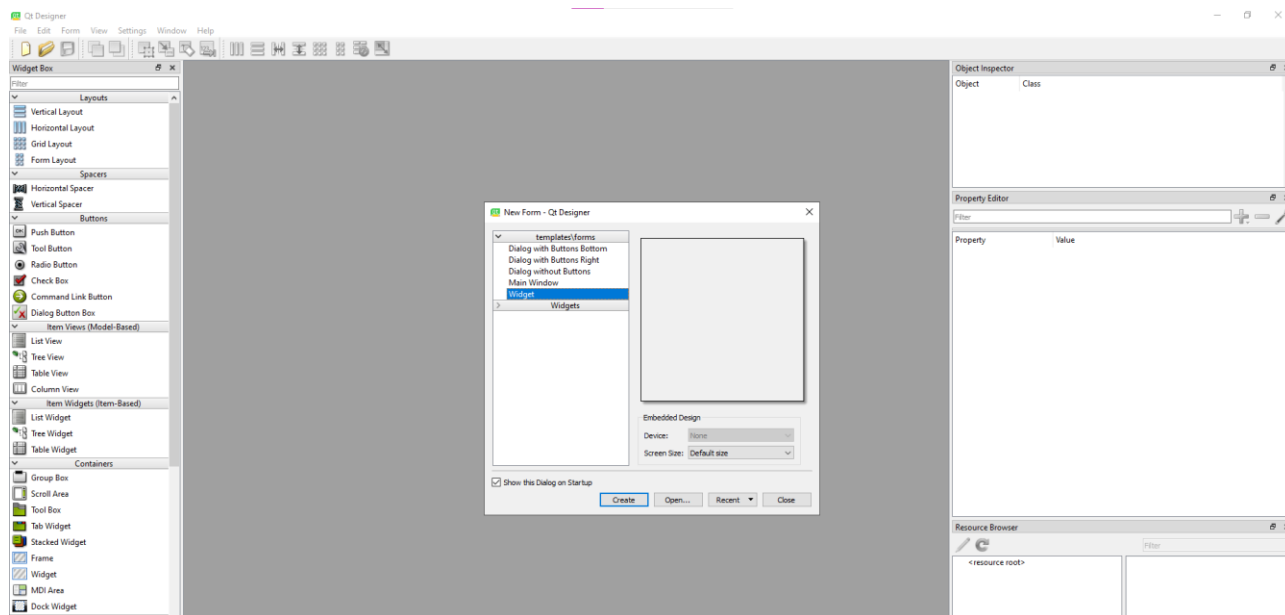
Для создания графических интерфейсов в Python существует несколько библиотек, таких как tkinter, PyQt5 и другие. В рамках данного лабораторного практикума будем использовать PyQt5, плюсом данного модуля служит то, что мы можем использовать конструктор интерфейса Qt Designer, который поможет нам расположить элементы.

Для начала работы установите Qt Designer по ссылке: <https://build-system.fman.io/qt-designer-download>.

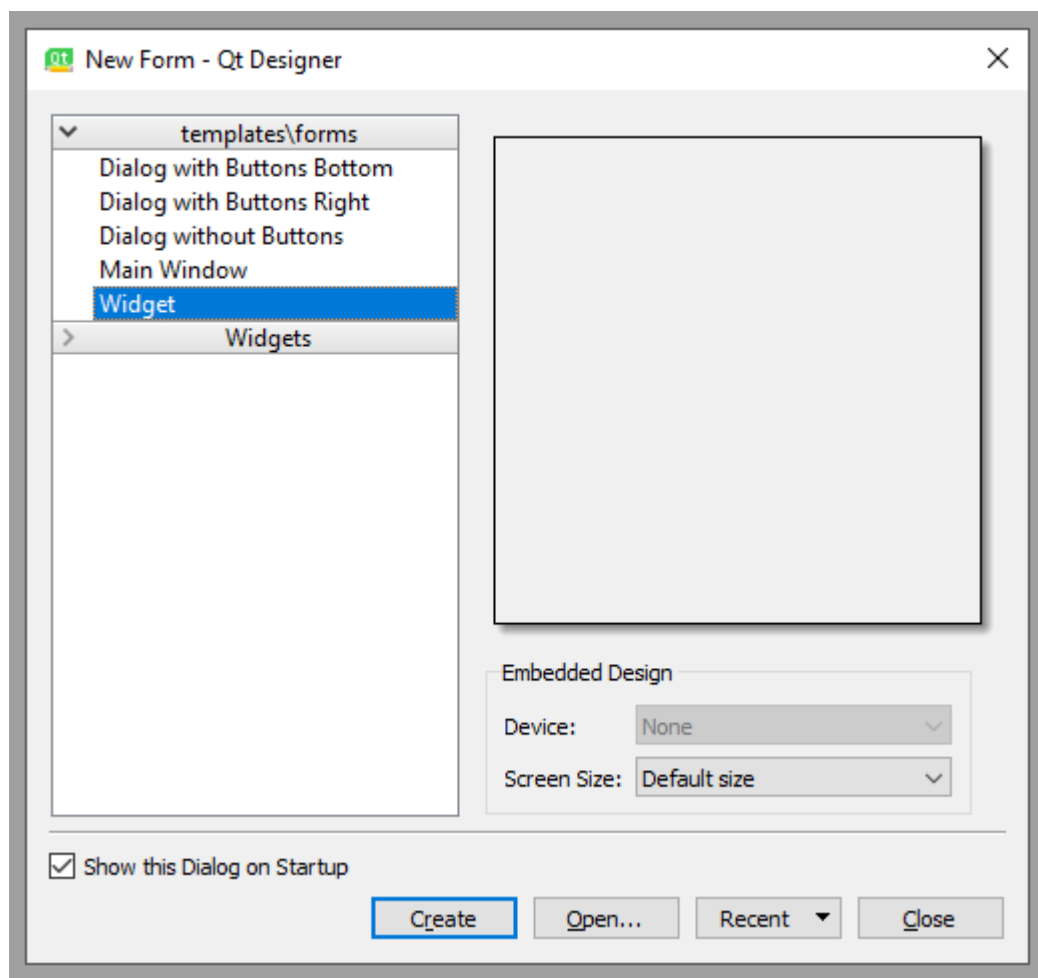


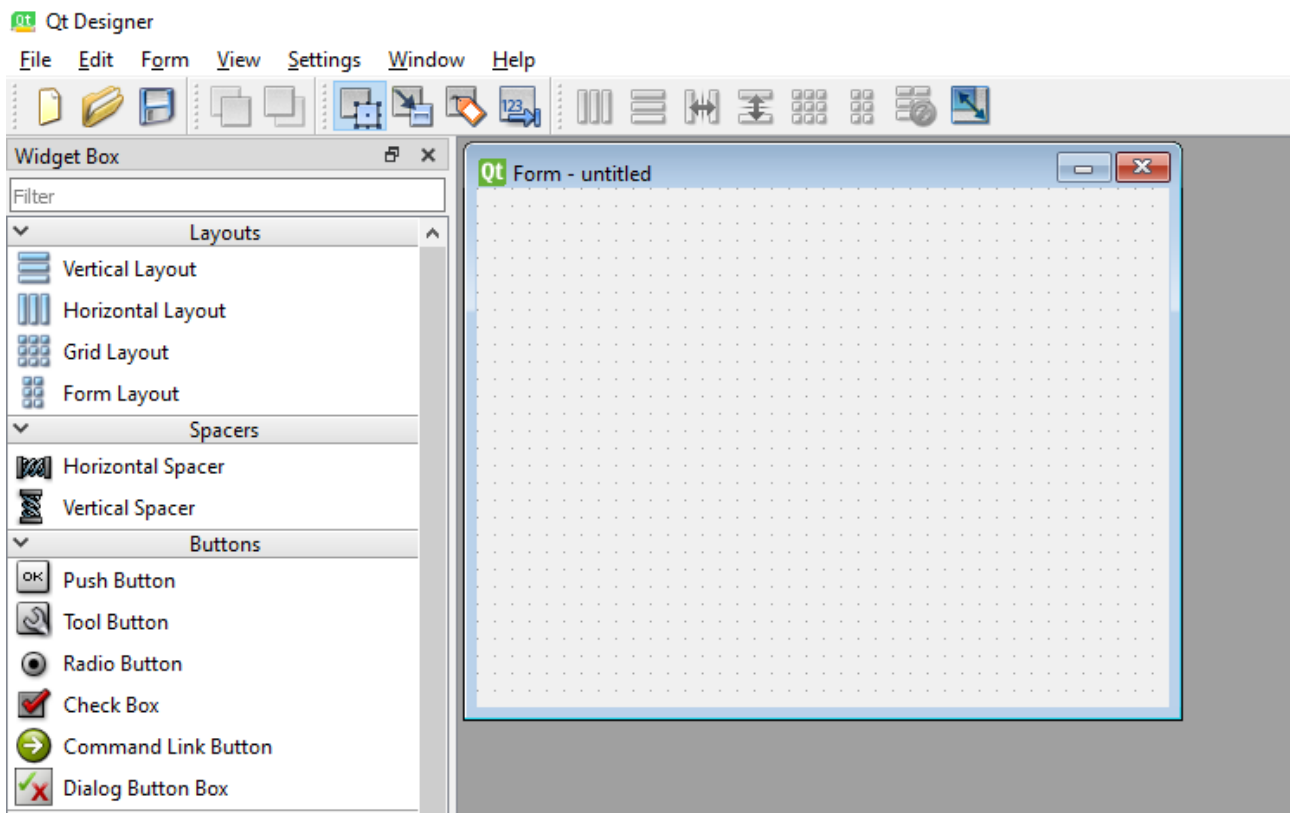
После установки Qt Designer, перейдите в командную строку и с помощью команд `pip install PyQt5` и `pip install PyQt5-tools` установите модуль PyQt5 и необходимые инструменты.

После установки модулей запускаем Qt Designer.



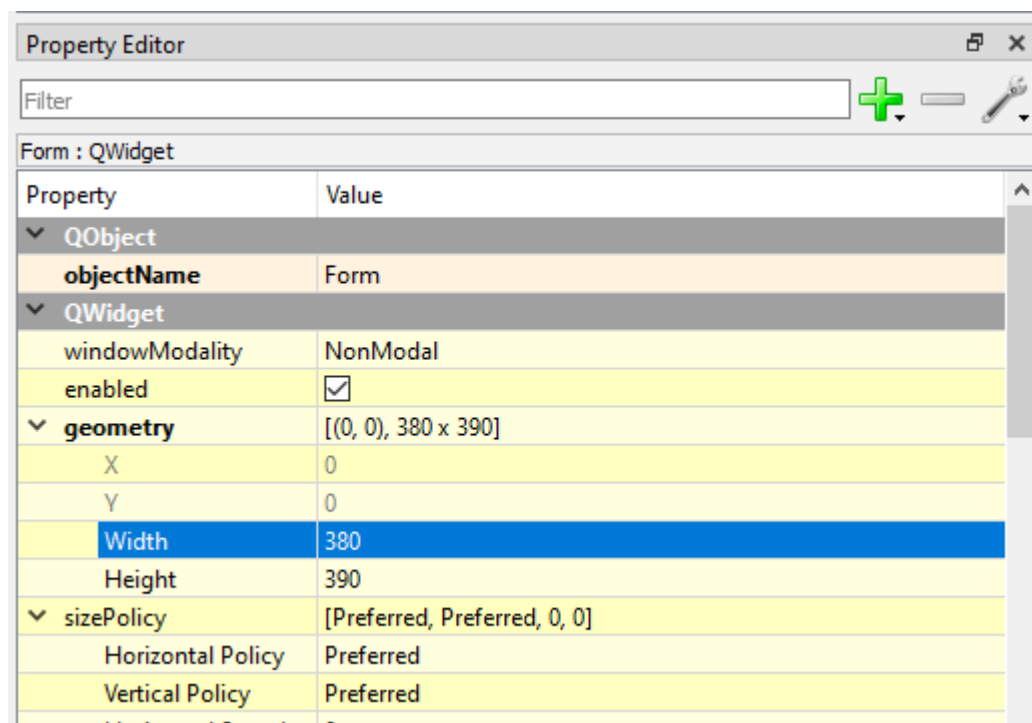
В начальном окне выбираем Widget и нажимаем Create



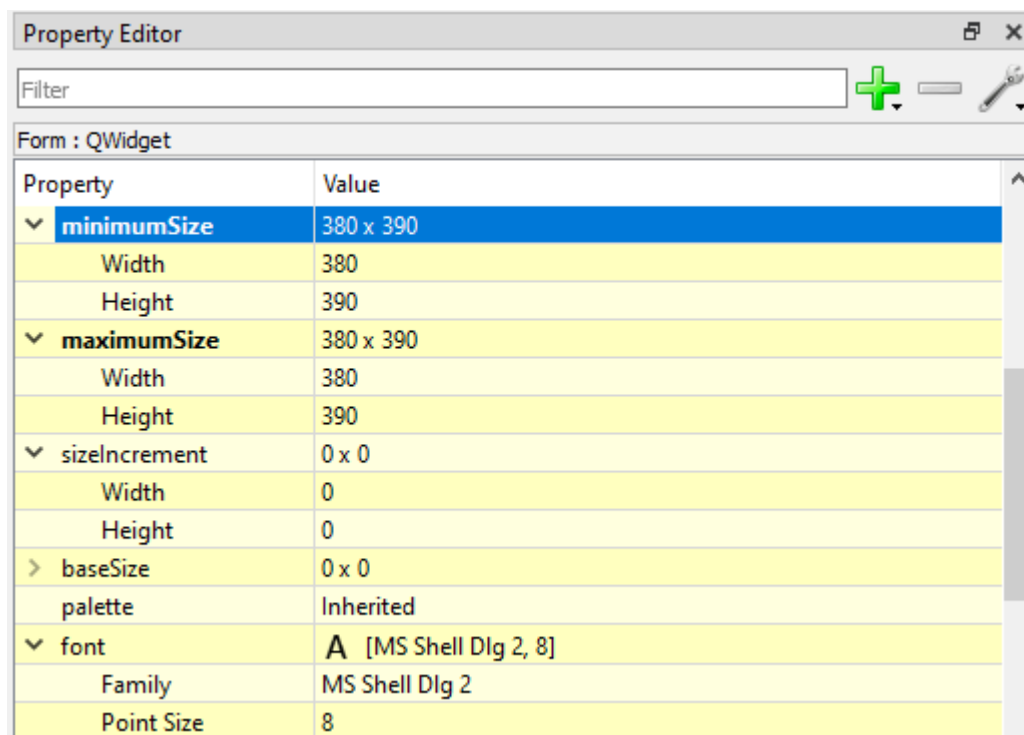


Это наш конструктор интерфейса, в меню слева мы можем выбирать элементы и перетаскивать их в наше главное окно, а в правой колонке редактировать элементы.

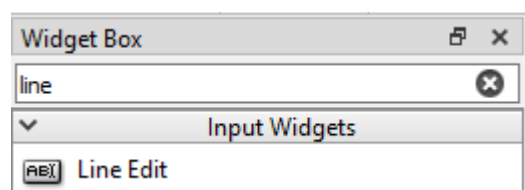
Задаём ширину и высоту окна (380x390) в правой колонке.



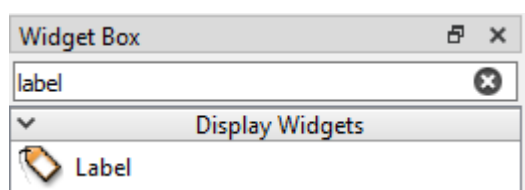
Также задаём `minimumSize` и `maximumSize` (380x390)



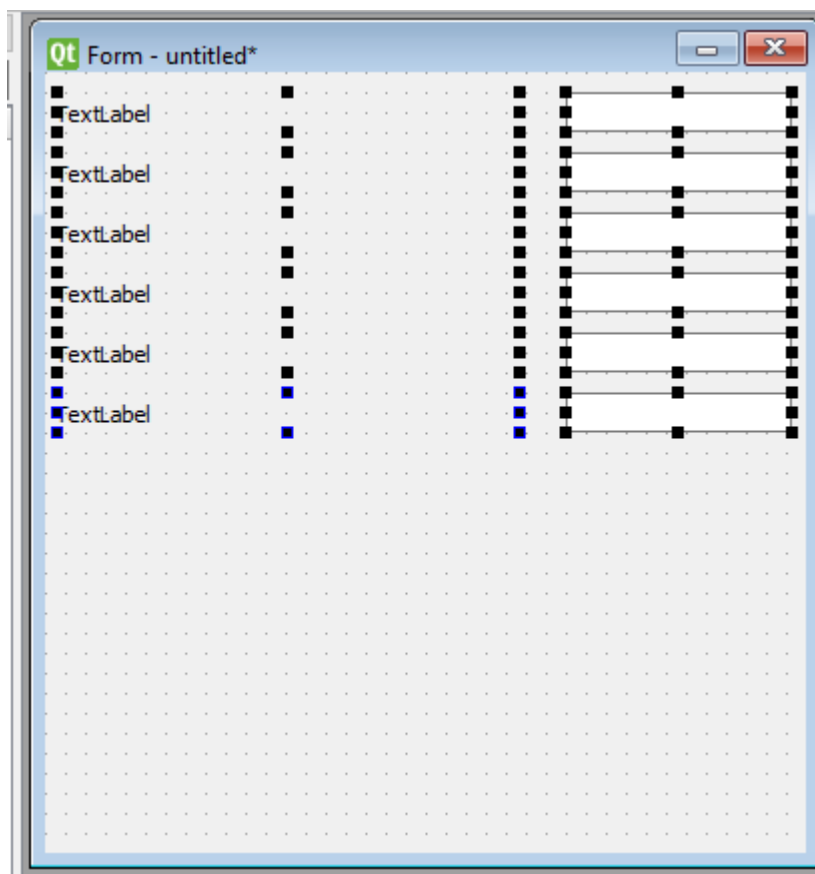
В колонке слева находим `lineEdit` и переносим 6 штук в наше окно.



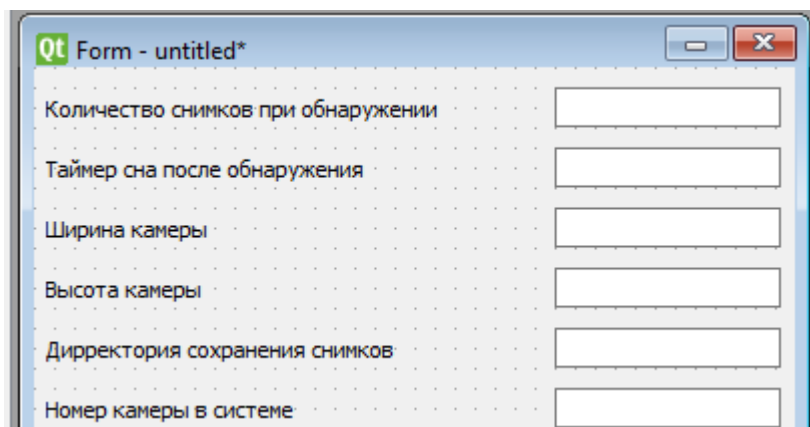
Таким же образом находим `label` и переносим 6 штук в наше окно.



Должно получиться вот так:

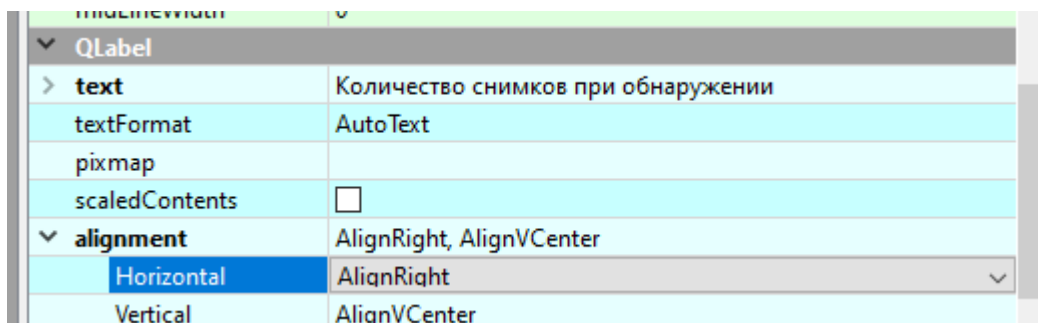


Кликаем двойным щелчком мыши по каждому TextLabel и задаём текст.

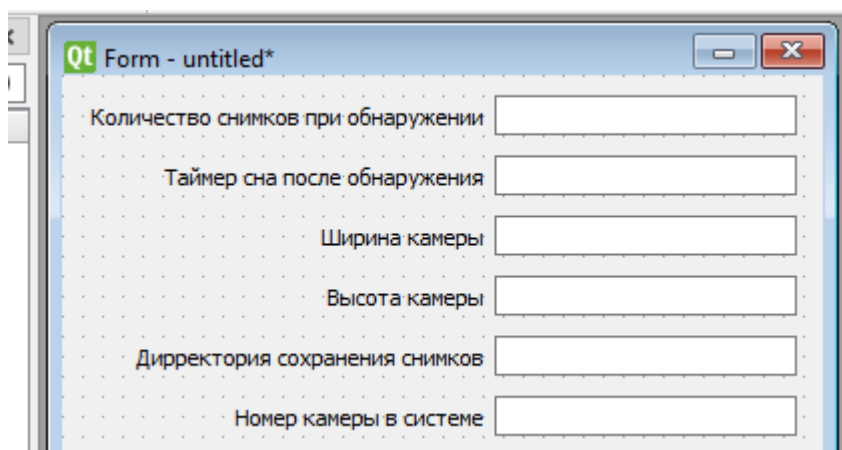


Для того чтобы текст прилегал к правому краю, в колонке справа кликнув по нужным QLabel, находим alignment и выбираем Horizontal: Align Right.

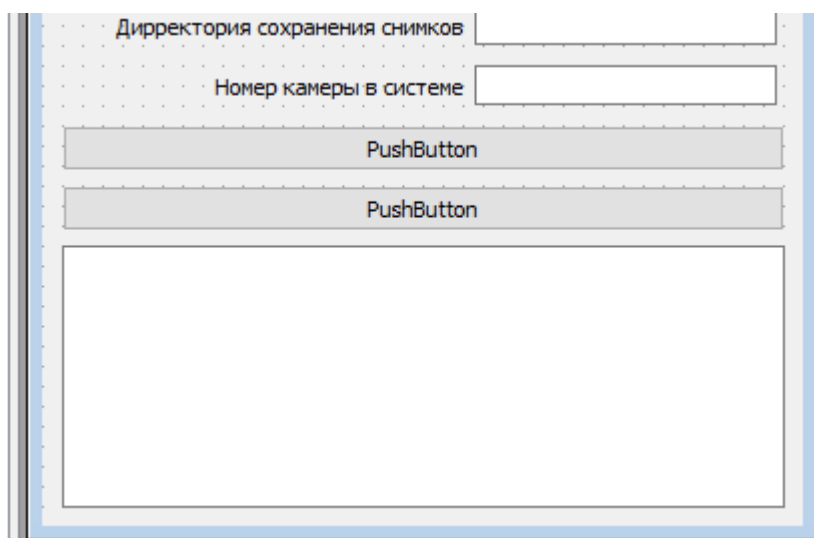
Проделываем это с каждым QLabel.



Должно получиться вот так:



Добавляем две Push Button кнопки и один Text Browser, чтобы получилось вот так:



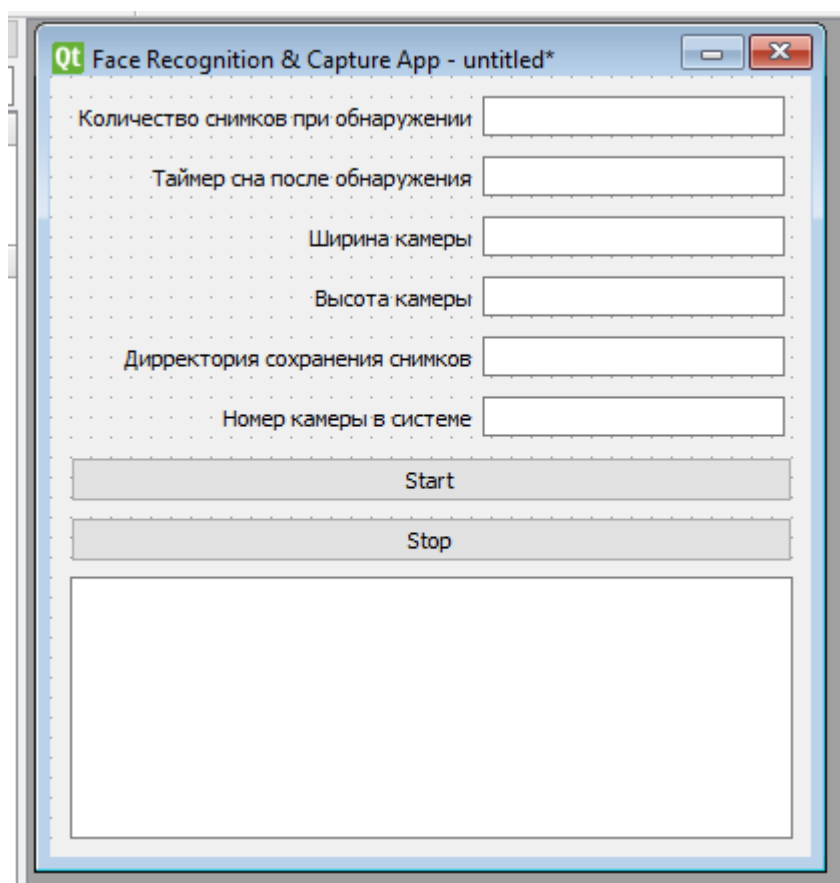


Переименовываем кнопки двойным нажатием в Start и Stop.

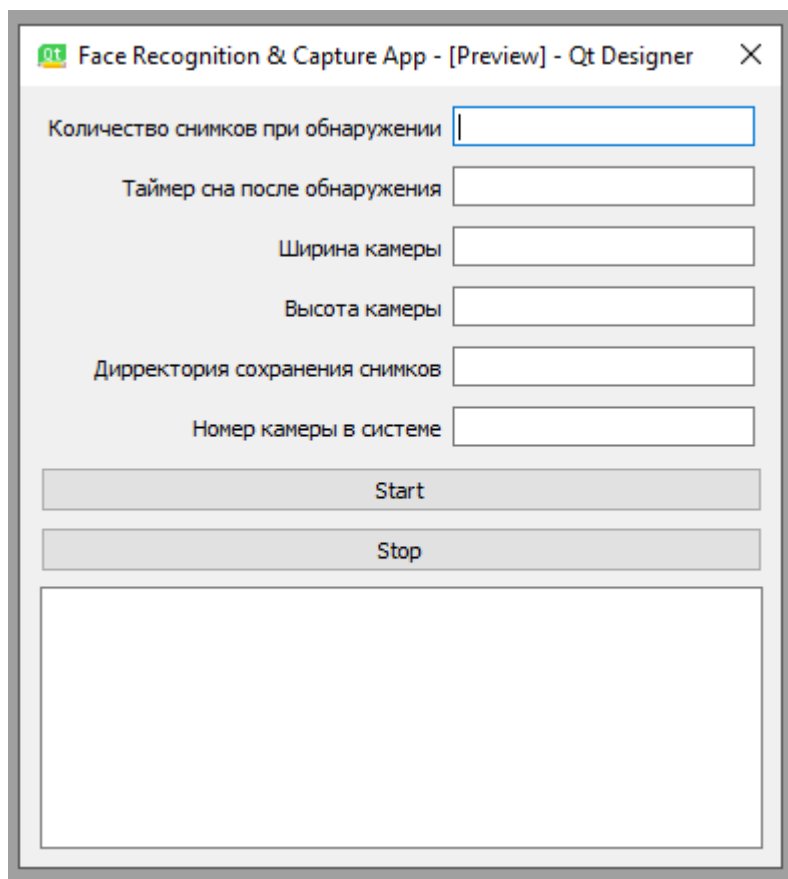
Также кликаем на главное окно и переименовываем его.

Form : QWidget	
Property	Value
Strikeout	<input type="checkbox"/>
Kerning	<input checked="" type="checkbox"/>
Antialiasing	PreferDefault
cursor	Arrow
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
> windowTitle	Face Recognition & Capture App
▼ windowIcon	

Исходный результат должен быть таким.

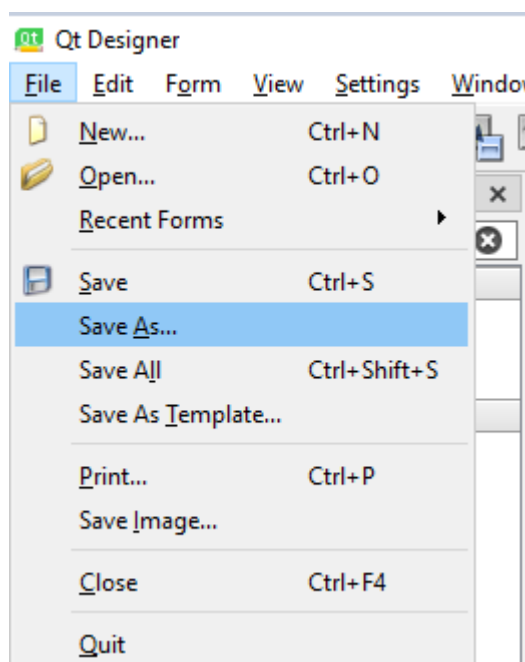


*Мы можем проверить работу программы нажав на сочетание **Ctrl+R***



Пока что, это просто оболочка будущего функционала программы.

Сохраняем наш дизайн в .ui файл с помощью:

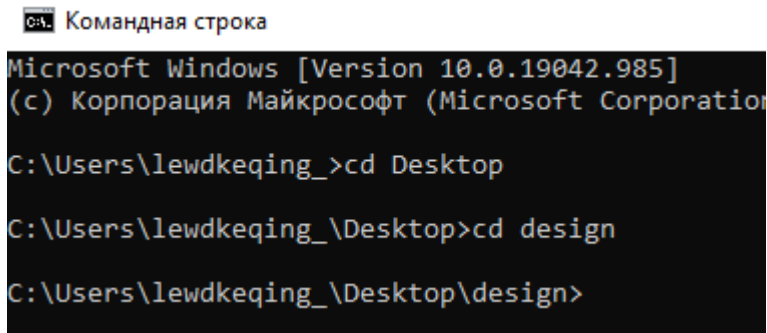


После сохранения файла можете закрыть Qt Designer.

## Лабораторная работа №6

### Конвертация .ui в .py

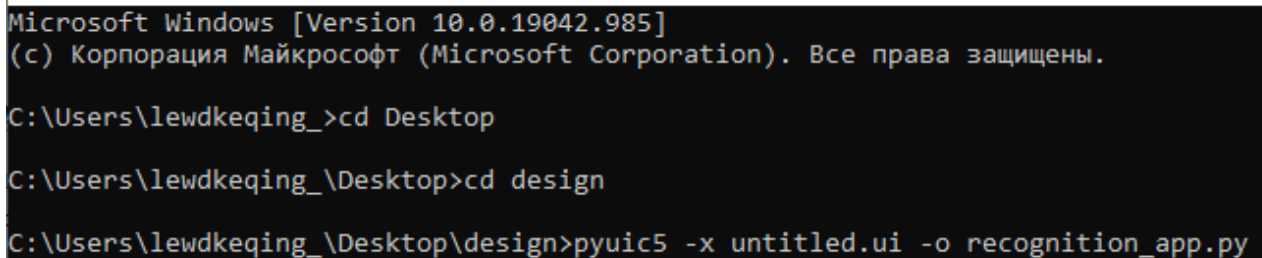
После того, как мы сохранили наш .ui файл воспользуемся установленными инструментами pyqt5–tools. Для этого переходим в командную строку и директорию куда мы сохранили наш .ui файл.



```
Microsoft Windows [Version 10.0.19042.985]
(c) Корпорация Майкрософт (Microsoft Corporation)

C:\Users\lewdkeqing>cd Desktop
C:\Users\lewdkeqing\Desktop>cd design
C:\Users\lewdkeqing\Desktop\design>
```

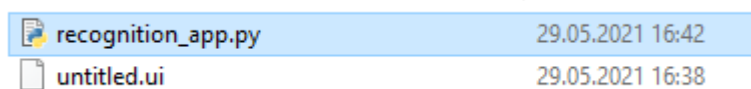
Используем команду: `pyuic5 -x имя_файла.ui -o recognition_app.py`



```
Microsoft Windows [Version 10.0.19042.985]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\lewdkeqing>cd Desktop
C:\Users\lewdkeqing\Desktop>cd design
C:\Users\lewdkeqing\Desktop\design>pyuic5 -x untitled.ui -o recognition_app.py
```

Переходим в нашу директорию, в которой находился файл .ui



И видим файл .py при запуске которого, будет запускаться наше приложение.

## Лабораторная работа №7

### Приложение для распознавания лиц

Запускаем Visual Studio Code и открываем .py файл из прошлой лабораторной работы recognition\_app.py.

```
recognition_app.py > ...
1  -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'untitled.ui'
4  #
5  # Created by: PyQt5 UI code generator 5.15.4
6  #
7  # WARNING: Any manual changes made to this file will be lost when pyuic5 is
8  # run again. Do not edit this file unless you know what you are doing.
9
10
11 from PyQt5 import QtCore, QtGui, QtWidgets
12
13
14 class Ui_Form(object):
15     def setupUi(self, Form):
16         Form.setObjectName("Form")
17         Form.resize(380, 390)
18         Form.setMinimumSize(QtCore.QSize(380, 390))
19         Form.setMaximumSize(QtCore.QSize(380, 390))
20         self.lineEdit = QtWidgets.QLineEdit(Form)
21         self.lineEdit.setGeometry(QtCore.QRect(216, 10, 151, 20))
22         self.lineEdit.setObjectName("lineEdit")
23         self.lineEdit_2 = QtWidgets.QLineEdit(Form)
24         self.lineEdit_2.setGeometry(QtCore.QRect(216, 40, 151, 20))
25         self.lineEdit_2.setObjectName("lineEdit_2")
26         self.lineEdit_3 = QtWidgets.QLineEdit(Form)
27         self.lineEdit_3.setGeometry(QtCore.QRect(216, 70, 151, 20))
```

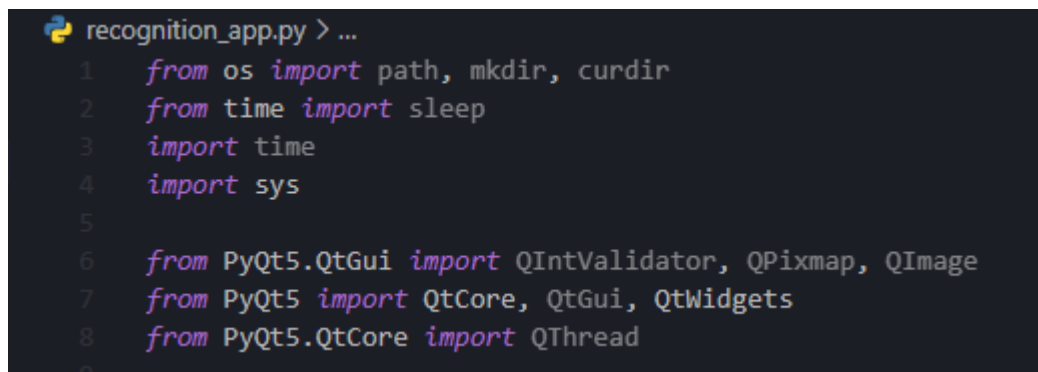
Это наш сгенерированный файл, который содержит в себе классы и функции для отображения и работы приложения.

Внесём в него изменения, для начала удалим закомментированные строки в начале файла.

```
recognition_app.py > ...
1
2 from PyQt5 import QtCore, QtGui, QtWidgets
3
4
5 class Ui_Form(object):
6     def setupUi(self, Form):
7         Form.setObjectName("Form")
8         Form.resize(380, 390)
9         Form.setMinimumSize(QtCore.QSize(380, 390))
10        Form.setMaximumSize(QtCore.QSize(380, 390))
11        self.lineEdit = QtWidgets.QLineEdit(Form)
12        self.lineEdit.setGeometry(QtCore.QRect(216, 10, 151, 20))
13        self.lineEdit.setObjectName("lineEdit")
14        self.lineEdit_2 = QtWidgets.QLineEdit(Form)
15        self.lineEdit_2.setGeometry(QtCore.QRect(216, 40, 151, 20))
```

Далее импортируем необходимые для работы функции.

Как показано на скриншоте ниже.



```
recognition_app.py > ...
1  from os import path, mkdir, curdir
2  from time import sleep
3  import time
4  import sys
5
6  from PyQt5.QtGui import QIntValidator, QPixmap, QImage
7  from PyQt5 import QtCore, QtGui, QtWidgets
8  from PyQt5.QtCore import QThread
9
```

Разберемся, что и для чего импортируется.

- `os` – будет работать для определения текущей директории, а также для создания новых папок;
- `sleep` – будет работать для задержки;
- `time` – будет работать для получения форматированного времени;
- `sys` – будет служить для выхода из приложения (`sys.exit(app.exec_())`);
- `QIntValidator` – Проверяет введенные значения в наши поля `lineEdit`;
- `QPixmap, QImage` – служат для отображения картинок внутри нашего приложения;
- `QtCore, QtGui, QtWidgets` – интерфейс программы;
- `QThread` – создание потока, чтобы приложение не зависало при вызове другого класса имеющего бесконечный цикл.

После импорта необходимых функций приступаем к написанию приложения.

```

6  from PyQt5.QtGui import QIntValidator, QPixmap, QImage
7  from PyQt5 import QtCore, QtGui, QtWidgets
8  from PyQt5.QtCore import QThread
9
10 dir_now = path.abspath(curdir)
11
12 def current_milli_time(): #Функция для получения миллисекунд
13     return round(time.time() * 1000)

```

Добавляем переменную `dir_now`, для получения текущей директории, а также пропишем первую функцию, получения миллисекунд.

Вносим изменения в функцию `setupUi`

```

def setupUi(self, Form):
    Form.setObjectName("Form")
    Form.resize(380, 390)
    Form.setMinimumSize(QtCore.QSize(380, 390))
    Form.setMaximumSize(QtCore.QSize(380, 390))
    icon = QtGui.QIcon()
    icon.addPixmap(QtGui.QPixmap("icon.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)

    Form.setWindowIcon(icon)
    self.label_2 = QtWidgets.QLabel(Form)
    self.label_2.setGeometry(QtCore.QRect(10, 40, 201, 25))
    font = QtGui.QFont()
    font.setPointSize(8)
    self.label_cam = QtWidgets.QLabel(Form)
    self.label_cam.setGeometry(QtCore.QRect(380, 10, 657, 370))
    font_for_cam = QtGui.QFont()
    font_for_cam.setPointSize(16)
    self.label_cam.setFont(font_for_cam)
    self.label_cam.setStyleSheet("")
    self.label_cam.setTextFormat(QtCore.Qt.AutoText)
    self.label_cam.setAlignment(QtCore.Qt.AlignCenter)
    self.label_cam.setObjectName("label")
    _translate = QtCore.QCoreApplication.translate
    Form.setWindowTitle(_translate("Form", "Form"))
    self.label_cam.setText(_translate("Form", "Последний захваченный снимок"))
    self.label_2.setFont(font)
    self.label_2.setLayoutDirection(QtCore.Qt.LeftToRight)
    self.label_2.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
    self.label_2.setObjectName("label_2")
    self.label_5 = QtWidgets.QLabel(Form)
    self.label_5.setGeometry(QtCore.QRect(10, 130, 201, 25))
    font = QtGui.QFont()
    font.setPointSize(8)
    self.label_5.setFont(font)

```

```

        self.label_5.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label_5.setAlignment(QtCore.Qt.AlignRight / QtCore.Qt.AlignTrailing / QtCo
re.Qt.AlignVCenter)
        self.label_5.setObjectName("label_5")
        self.label_3 = QtWidgets.QLabel(Form)
        self.label_3.setGeometry(QtCore.QRect(10, 70, 201, 25))
        font = QtGui.QFont()
        font.setPointSize(8)
        self.label_3.setFont(font)
        self.label_3.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label_3.setAlignment(QtCore.Qt.AlignRight / QtCore.Qt.AlignTrailing / QtCo
re.Qt.AlignVCenter)
        self.label_3.setObjectName("label_3")
        self.lineCamHeight = QtWidgets.QLineEdit(Form)
        self.lineCamHeight.setGeometry(QtCore.QRect(220, 100, 150, 25))
        self.lineCamHeight.setObjectName("lineCamHeight")
        self.pushButtonStart = QtWidgets.QPushButton(Form)
        self.pushButtonStart.setGeometry(QtCore.QRect(9, 190, 361, 25))
        self.pushButtonStart.setObjectName("pushButtonStart")
        self.lineCamCaptures = QtWidgets.QLineEdit(Form)
        self.lineCamCaptures.setGeometry(QtCore.QRect(220, 10, 150, 25))
        self.lineCamCaptures.setObjectName("lineCamCaptures")
        self.lineDirectory = QtWidgets.QLineEdit(Form)
        self.lineDirectory.setGeometry(QtCore.QRect(220, 130, 150, 25))
        self.lineDirectory.setObjectName("lineDirectory")
        self.textBrowser = QtWidgets.QTextBrowser(Form)
        self.textBrowser.setGeometry(QtCore.QRect(10, 250, 361, 131))
        self.textBrowser.setObjectName("textBrowser")
        self.pushButtonStop = QtWidgets.QPushButton(Form)
        self.pushButtonStop.setGeometry(QtCore.QRect(9, 220, 361, 25))
        self.pushButtonStop.setObjectName("pushButtonStop")
        self.label = QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(10, 10, 201, 25))
        font = QtGui.QFont()
        font.setPointSize(8)
        self.label.setFont(font)
        self.label.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label.setAlignment(QtCore.Qt.AlignRight / QtCore.Qt.AlignTrailing / QtCore
.Qt.AlignVCenter)
        self.label.setObjectName("label")
        self.label_6 = QtWidgets.QLabel(Form)
        self.label_6.setGeometry(QtCore.QRect(10, 160, 201, 25))
        font = QtGui.QFont()
        font.setPointSize(8)
        self.label_6.setFont(font)
        self.label_6.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label_6.setAlignment(QtCore.Qt.AlignRight / QtCore.Qt.AlignTrailing / QtCo
re.Qt.AlignVCenter)
        self.label_6.setObjectName("label_6")
        self.lineCamTimerCaptures = QtWidgets.QLineEdit(Form)
        self.lineCamTimerCaptures.setGeometry(QtCore.QRect(220, 40, 150, 25))

```

```

self.lineCamTimerCaptures.setObjectName("lineCamTimerCaptures")
self.label_4 = QtWidgets.QLabel(Form)
self.label_4.setGeometry(QtCore.QRect(10, 100, 201, 25))
font = QtGui.QFont()
font.setPointSize(8)
self.label_4.setFont(font)
self.label_4.setLayoutDirection(QtCore.Qt.LeftToRight)
self.label_4.setAlignment(QtCore.Qt.AlignRight /QtCore.Qt.AlignTrailing/QtCore.Qt.AlignVCenter)
self.label_4.setObjectName("label_4")
self.lineCamID = QtWidgets.QLineEdit(Form)
self.lineCamID.setGeometry(QtCore.QRect(220, 160, 150, 25))
self.lineCamID.setObjectName("lineCamID")
self.lineCamWidth = QtWidgets.QLineEdit(Form)
self.lineCamWidth.setGeometry(QtCore.QRect(220, 70, 150, 25))
self.lineCamWidth.setObjectName("lineCamWidth")
self.retranslateUi(Form)
QtCore.QMetaObject.connectSlotsByName(Form)
self.add_functions() #Регистрация функций кнопок
self.pushButtonStop.setEnabled(False) #Отключаем кнопку

```

Суть данных изменений в том, что мы переименовали наши QLineEdit в удобные нам lineCamID и т.п. А также исправили размеры полей, добавили иконку приложения icon.ico, а также создали ещё один QLineEdit в котором будут размещаться кадры, при захвате лица.

После внесённых изменений, добавляем функцию app\_change\_text\_browser(self, text):

```

105 self.label_4.setObjectName("label_4")
106 self.lineCamID = QtWidgets.QLineEdit(Form)
107 self.lineCamID.setGeometry(QtCore.QRect(220, 160, 150, 25))
108 self.lineCamID.setObjectName("lineCamID")
109 self.lineCamWidth = QtWidgets.QLineEdit(Form)
110 self.lineCamWidth.setGeometry(QtCore.QRect(220, 70, 150, 25))
111 self.lineCamWidth.setObjectName("lineCamWidth")
112 self.retranslateUi(Form)
113 QtCore.QMetaObject.connectSlotsByName(Form)
114 self.add_functions() #Регистрация функций кнопок
115 self.pushButtonStop.setEnabled(False) #Отключаем кнопку
116
117 def app_change_text_browser(self, text): #Функция для вывода информации в textBrowser
118     self.textBrowser.append(str(text))
119     self.textBrowser.moveCursor(QtGui.QTextCursor.End)
120

```



Далее изменяем функцию retranslateUi:

```
def retranslateUi(self, Form): #Именуем
    _translate = QtCore.QCoreApplication.translate
    Form.setWindowTitle(_translate("Form", "Face Recognition & Capture App"))
    self.label_2.setText(_translate("Form", "Таймер сна после обнаружения"))
    self.label_5.setText(_translate("Form", "Дирректория для сохранения снимков
"))
    self.label_3.setText(_translate("Form", "Ширина камеры"))
    self.lineCamHeight.setText(_translate("Form", "720"))
    self.pushButtonStart.setText(_translate("Form", "Start"))
    self.lineCamCaptures.setText(_translate("Form", "1"))
    self.lineDirectory.setText(_translate("Form", (dir_now + "\Captures")))
    self.pushButtonStop.setText(_translate("Form", "Stop"))
    self.label.setText(_translate("Form", "Количество снимков при обнаружении")
)

    self.label_6.setText(_translate("Form", "Номер камеры в системе"))
    self.lineCamTimerCaptures.setText(_translate("Form", "15"))
    self.label_4.setText(_translate("Form", "Высота камеры"))
    self.lineCamID.setText(_translate("Form", "0"))
    self.lineCamWidth.setText(_translate("Form", "1280"))

    self.onlyInt = QIntValidator()
    self.lineCamHeight.setValidator(self.onlyInt)
    self.lineCamWidth.setValidator(self.onlyInt)
    self.lineCamTimerCaptures.setValidator(self.onlyInt)
    self.lineCamID.setValidator(self.onlyInt)
    self.lineCamCaptures.setValidator(self.onlyInt)
```

```
117     def app_change_text_browser(self, text): #Функция для вывода информации в textBrowser
118         self.textBrowser.append(str(text))
119         self.textBrowser.moveCursor(QtGui.QTextCursor.End)
120
121     def retranslateUi(self, Form): #Именуем
122         _translate = QtCore.QCoreApplication.translate
123         Form.setWindowTitle(_translate("Form", "Face Recognition & Capture App"))
124         self.label_2.setText(_translate("Form", "Таймер сна после обнаружения"))
125         self.label_5.setText(_translate("Form", "Дирректория для сохранения снимков"))
126         self.label_3.setText(_translate("Form", "Ширина камеры"))
127         self.lineCamHeight.setText(_translate("Form", "720"))
128         self.pushButtonStart.setText(_translate("Form", "Start"))
129         self.lineCamCaptures.setText(_translate("Form", "1"))
130         self.lineDirectory.setText(_translate("Form", (dir_now + "\Captures")))
131         self.pushButtonStop.setText(_translate("Form", "Stop"))
132         self.label.setText(_translate("Form", "Количество снимков при обнаружении"))
133         self.label_6.setText(_translate("Form", "Номер камеры в системе"))
134         self.lineCamTimerCaptures.setText(_translate("Form", "15"))
135         self.label_4.setText(_translate("Form", "Высота камеры"))
136         self.lineCamID.setText(_translate("Form", "0"))
137         self.lineCamWidth.setText(_translate("Form", "1280"))
138
139         self.onlyInt = QIntValidator()
140         self.lineCamHeight.setValidator(self.onlyInt)
141         self.lineCamWidth.setValidator(self.onlyInt)
142         self.lineCamTimerCaptures.setValidator(self.onlyInt)
143         self.lineCamID.setValidator(self.onlyInt)
144         self.lineCamCaptures.setValidator(self.onlyInt)
```

Суть данных изменений в том, что мы добавили стандартные значения в наше приложение, а также добавили проверку вводимых значений закрепив их за функцией `QIntValidator()`.

Далее добавляем функции `add_functions(self)`, `resize_form(self, width, height)`, `set_pixmap(self, pixmap)`:

```
139     self.onlyInt = QIntValidator()
140     self.lineCamHeight.setValidator(self.onlyInt)
141     self.lineCamWidth.setValidator(self.onlyInt)
142     self.lineCamTimerCaptures.setValidator(self.onlyInt)
143     self.lineCamID.setValidator(self.onlyInt)
144     self.lineCamCaptures.setValidator(self.onlyInt)
145
146     def add_functions(self): #Добавляем функции кнопкам
147         self.pushButtonStart.clicked.connect(lambda: self.recognition_app_start(1))
148         self.pushButtonStop.clicked.connect(lambda: self.recognition_app_start(0))
149
150     def resize_form(self, width, height):
151         Form.resize(width, height)
152         Form.setMinimumSize(QtCore.QSize(width, height))
153         Form.setMaximumSize(QtCore.QSize(width, height))
154
155     def set_pixmap(self, pixmap):
156         self.label_cam.setGeometry(QtCore.QRect(380, 10, 657, 370))
157         pixmap = pixmap.scaledToHeight(370)
158         self.label_cam.setPixmap(QPixmap(pixmap))
```

Разберемся для чего служат эти три новые функции:

- `add_functions(self)` – назначает для кнопок start, stop, действия по передаче данных в будущие функции по запуску потока.
- `resize_form` – служит для изменения ширины и высоты окна.
- `set_pixmap(self, pixmap)` – служит для обновления картинки внутри формы pyqt5.

Добавляем функцию `recognition_app_start(self, state)`, которая будет запускать новый класс.

```
155     def set_pixmap(self, pixmap):
156         self.label_cam.setGeometry(QtCore.QRect(380, 10, 657, 370))
157         pixmap = pixmap.scaledToHeight(370)
158         self.label_cam.setPixmap(QPixmap(pixmap))
159
160     def recognition_app_start(self, state): #Проверяем переданные значения и запускаем
161
162         if state == 1:
163             self.resize_form(1047, 390)
164             self.state = True
165             self.pushButtonStart.setEnabled(False) #Активность кнопок
166             self.pushButtonStop.setEnabled(True)
167             self.rec_app = StartRecognition(self.lineCamCaptures.text(),
168                                           self.lineCamTimerCaptures.text(),
169                                           self.lineCamWidth.text(),
170                                           self.lineCamHeight.text(),
171                                           self.lineDirectory.text(),
172                                           self.lineCamID.text()) #Запуск потока
173
174             self.rec_app.start() #Стартуем
175
176         else:
177             self.resize_form(380, 390)
178             self.pushButtonStart.setEnabled(True) #Активность кнопок
179             self.pushButtonStop.setEnabled(False)
180
181             self.rec_app.stop()
182
```

Суть данной функции в том, что при вызове данной функции и передаче в неё значения state, она проверяет данное значение, если значение = 1, то мы вызываем функцию `resize_form` и меняем ширину и высоту окна. Выключаем активность кнопки Start, и включаем активность кнопки Stop.

Затем происходит вызов класса `StartRecognition` в переменной `rec_app` и передача в него значений из наших полей `line*`.

В ином случае мы обратно меняем ширину и высоту окна, меняем активности кнопок и вызываем функцию для остановки созданного потока.

Создаём новый класс StartRecognition(QThread):

```
176     else:
177         self.resize_form(380, 390)
178         self.pushButtonStart.setEnabled(True) #Активность кнопок
179         self.pushButtonStop.setEnabled(False)
180
181         self.rec_app.stop()
182
183     class StartRecognition(QThread): #Новый поток запускаем cv2
184
185     def __init__(self, captures, time_sleep, cam_width, cam_height, directory, cam_id):
186         self.captures = int(captures)
187         self.time_sleep = int(time_sleep)
188         self.cam_width = int(cam_width)
189         self.cam_height = int(cam_height)
190         self.directory = str(directory)
191
192         ui.app_change_text_browser("Проверяю наличие дирректории.") #Просто лог в textBrowser
193
194         if path.exists(self.directory): #Проверка наличия дирректории
195             ui.app_change_text_browser(f"Дирректория {self.directory} существует.")
196             pass
197
198         else:
199             ui.app_change_text_browser(f"Дирректория {self.directory} не существует.")
200             ui.app_change_text_browser(f"Создаю дирректорию.")
201             mkdir(self.directory)
202             ui.app_change_text_browser(f"Дирректория создана.")
203
204         self.cam_id = int(cam_id)
205         self.state = True
206         QThread.__init__(self)
```

При написании класса мы задействуем функцию QThread, которая сообщает программе, что данный класс будет работать как новый поток.

Функция `__init__` внутри класса, всегда исполняется во время вызова, это значит то, что мы можем внутри данной функции получать данные.

Добавим функцию `img_sender(self, pixmap)`, для отправки изображения внутрь pyqt5 интерфейса.

```
205         self.state = True
206         QThread.__init__(self)
207
208     def img_sender(self, pixmap):
209         height, width, channel = pixmap.shape
210         bytesPerLine = 3 * width
211         qImg = QImage(pixmap.data, width, height, bytesPerLine, QImage.Format_RGB888)
212         ui.set_pixmap(qImg)
```

Создаём функцию run(self):

```
208     def img_sender(self, pixmap):
209         height, width, channel = pixmap.shape
210         bytesPerLine = 3 * width
211         qImg = QImage(pixmap.data, width, height, bytesPerLine, QImage.Format_RGB888)
212         ui.set_pixmap(qImg)
213
214     def run(self):
215
216         import os
217         import time
218
219         import cv2
220
221         i = 1
```

**ВАЖНО:** Внутри данной функции импортируем библиотеки os, time, cv2.

Для импорта библиотеки cv2 необходимо её установить! С помощью команды: *pip install opencv-python*.

После установки opencv, мы можем продолжить данную лабораторную работу.

```
221         i = 1
222
223         ui.app_change_text_browser("Создаю поток.")
224
225         face_cascade_db = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
226
227         cap = cv2.VideoCapture(self.cam_id) #Выбираем источник изображения
228
229         cap.set(cv2.CAP_PROP_FRAME_WIDTH, self.cam_width) #Задаем размеры источника
230         cap.set(cv2.CAP_PROP_FRAME_HEIGHT, self.cam_height)
231         path = self.directory
232
233         ui.app_change_text_browser("Поток создан.")
```

Что тут происходит?

- Загружаем обученные каскады в переменную face\_cascade\_db.
- Производим захват изображения с камеры в переменную cap.
  - self.cam\_id – целое число указывающее на номер камеры. Если камеры нету, можно скачать OBS и использовать плагин OBS VirtualCam 2.0.5, в качестве виртуальной камеры.

- Также можно заменить строку `cap = cv2.VideoCapture(self.cam_id),` на `cap = cv2.VideoCapture("some_image.jpg"),` чтобы производить распознавание лица с картинки.
- `cap.set` — устанавливаем ширину и высоту источника захвата изображения.
- `path` — записываем директорию указанную в форме `pyqt5`.

```

233 ui.app_change_text_browser("Поток создан.")
234
235 while self.state:
236     success, img = cap.read() #Читаем изображение
237     if success:
238         img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Переводим в оттенки серого
239         faces = face_cascade_db.detectMultiScale(img_gray, 1.1, 19) #Детектим лица
240
241         for (x, y, w, h) in faces: #Это наши лица, отрисовываем квадраты
242
243             cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255, 0), 4) #Квадрат и выбор его цвета
244
245             milli_time = current_milli_time() #Получаем миллисекунды
246             time_now = time.strftime(f"%d %B %Y %H-%M-%S-{milli_time}", time.localtime()) #Получаем дату и время
247
248             ui.app_change_text_browser(f"[{milli_time}] Лицо обнаружено, создаю запись.")
249             cv2.imwrite(os.path.join(path, f"{time_now} Person.jpg"), img) #Логируем (сохраняем фрейм)
250
251             img_to_send = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
252             self.img_sender(img_to_send)
253
254             i += 1

```

Далее начинается бесконечный цикл `while self.state`:

Переносим кадр в переменную `img` и записываем в переменную `success` булево значение (`True`, `False`). Конвертируем изображение в оттенки серого с помощью `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`.

Определяем лица и переносим их в переменную `faces`, а затем в цикле `for` по списку `faces` отрисовываем квадраты на лицах с помощью `cv2.rectangle`.

Получаем время в переменные `milli_time`, `time_now`. Сохраняем кадры с помощью `cv2.imwrite`, переводим изображение из `BGR` в `RGB` и отправляем в интерфейс `pyqt5` с помощью `self.img_sender`.

```

254         i += 1
255
256         if i > self.captures: #Данная конструкция служит для быстрого выхода из потока при нажатии на кнопку стоп
257             ui.app_change_text_browser(f"Лицо захвачено, задержка захвата на {self.time_sleep} с.")
258             x = 0.0
259
260             while x < self.time_sleep:
261                 sleep(0.1)
262                 x += 0.1
263                 if self.state == False:
264                     ui.app_change_text_browser("Поток остановлен.")
265                     break
266
267             i = 1
268         else:
269             ui.app_change_text_browser("Поток остановлен.")
270             ui.app_change_text_browser(f"\nОшибка чтения изображения.\nНомер камеры {self.cam_id} в системе не найден!\n")
271             self.state == False
272
273             ui.resize_form(380, 390)
274             ui.pushButtonStart.setEnabled(True) #Активность кнопок
275             ui.pushButtonStop.setEnabled(False)
276
277             break
278
279 cap.release() #Свободу источнику изображения!

```

- Проверяем с помощью if количество захваченных снимков. Обнуляем x.
- Запускаем цикл while и проверяем на сколько поток должен «уснуть».
- Блок else в данном случае не обязателен, он срабатывает при неверном вводе номера камеры.
- Освобождаем камеру cap.release()

Прописываем функцию stop(self): для остановки потока.

```


279         cap.release() #Свободу источнику изображения!
280
281         def stop(self): #Функция остановки потока
282             self.quit()
283             self.state = False
284             ui.app_change_text_browser("Остановка потока.")
285
286         if __name__ == "__main__": #Инициализация приложения
287             app = QtWidgets.QApplication(sys.argv)
288             Form = QtWidgets.QWidget()
289             ui = Ui_Form()
290             ui.setupUi(Form)
291             Form.show()
292             sys.exit(app.exec_())

```

Если вы написали всё верно, то программа должна работать!

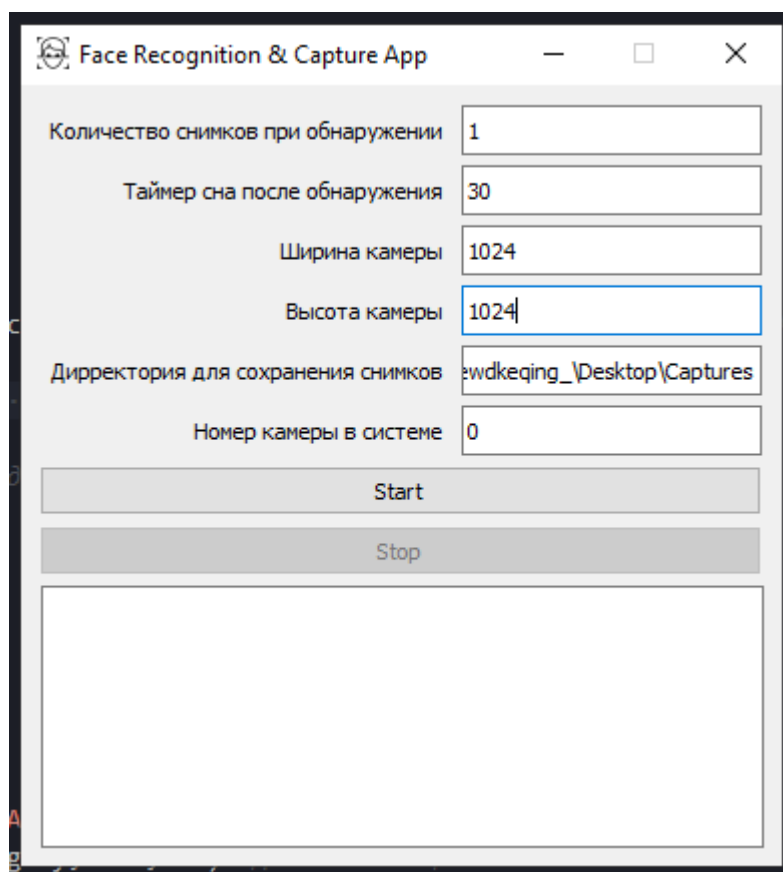
Проверяем работу программы с помощью камеры, указав в интерфейсе номер камеры 0, если у вас есть камера или если это виртуальная камера OBS.

Если нет ни того, ни другого измените строчку кода, на имя изображения!

```
226  
227 |  cap = cv2.VideoCapture("holo.jpg") - #Выбираем источник изображения  
228
```

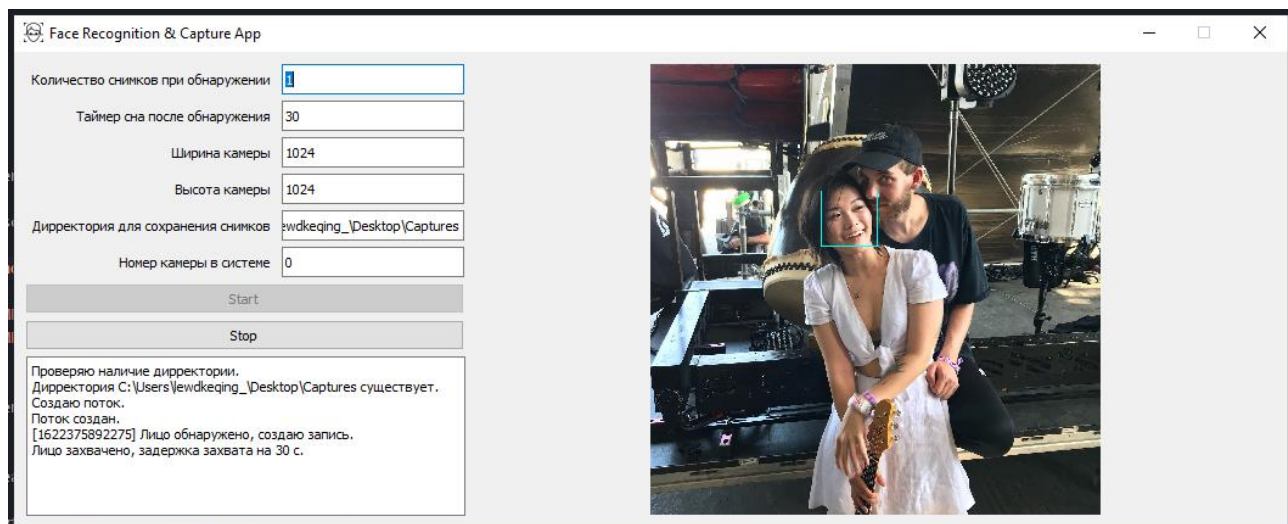
Изображение должно лежать в той же директории, где и наш .ру файл.

### Запускаем программу!



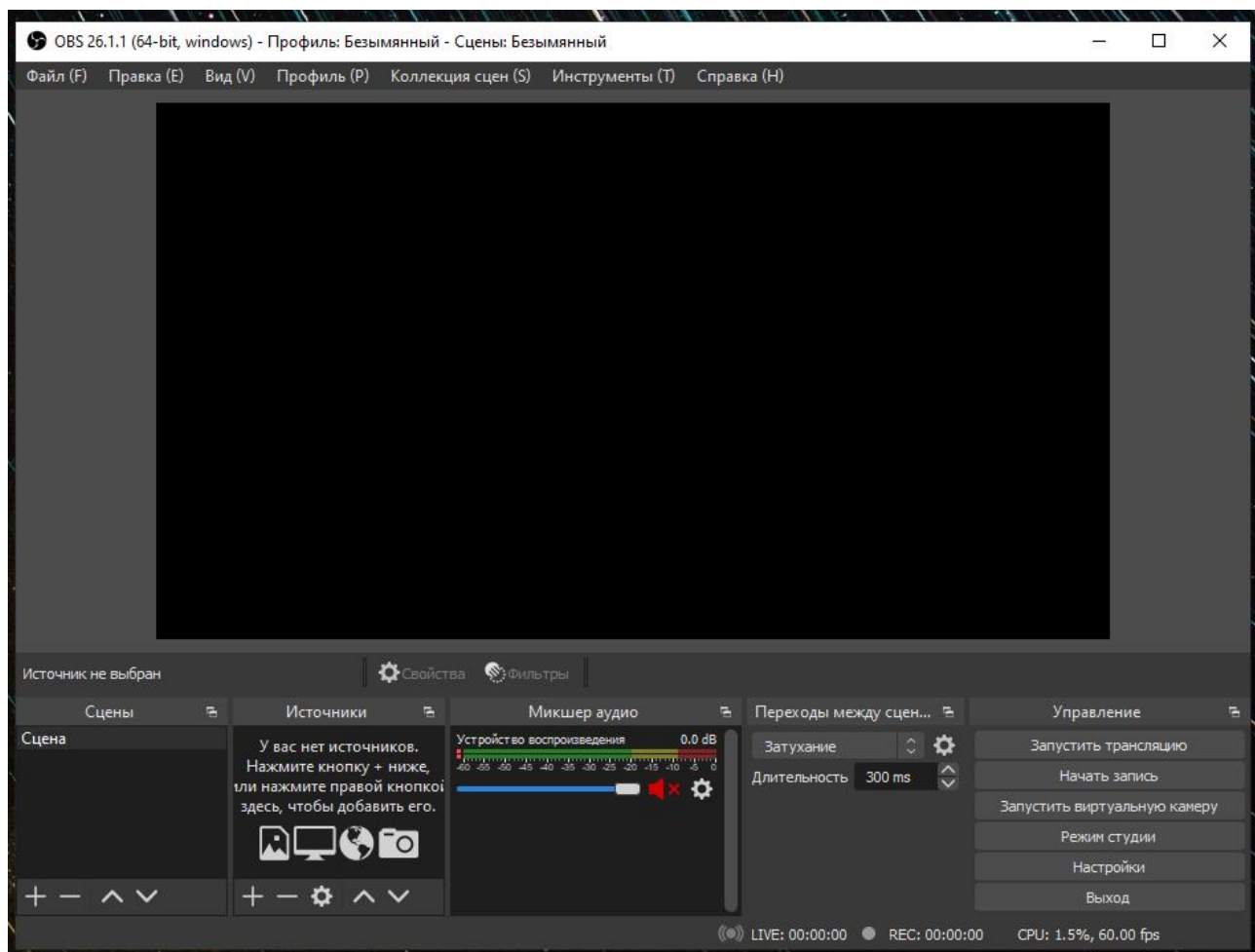
**Нажимаем Start**





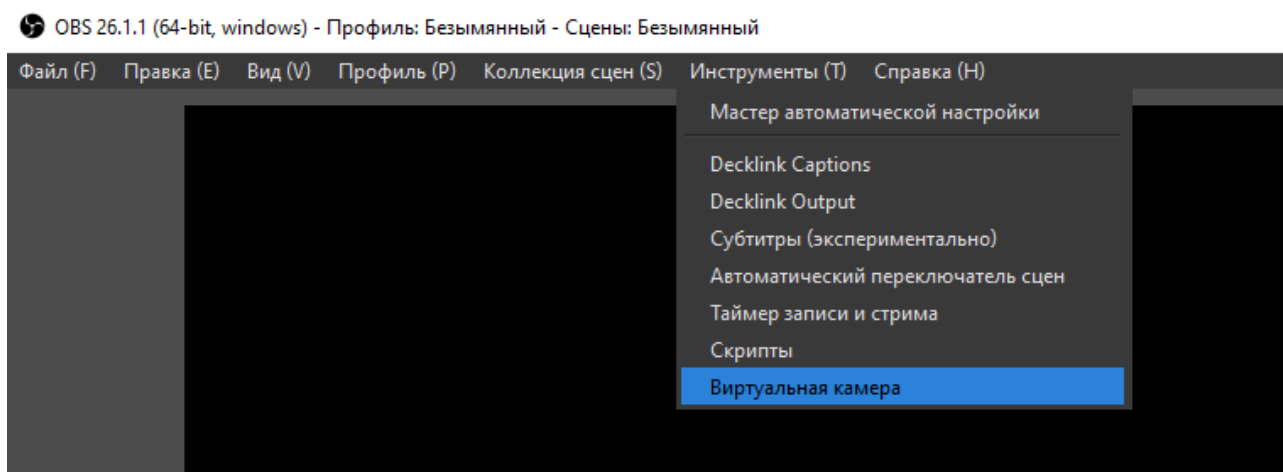
**Программа работает и распознала лицо на изображении.**

Пример работы с виртуальной камерой OBS, ниже на скриншотах.

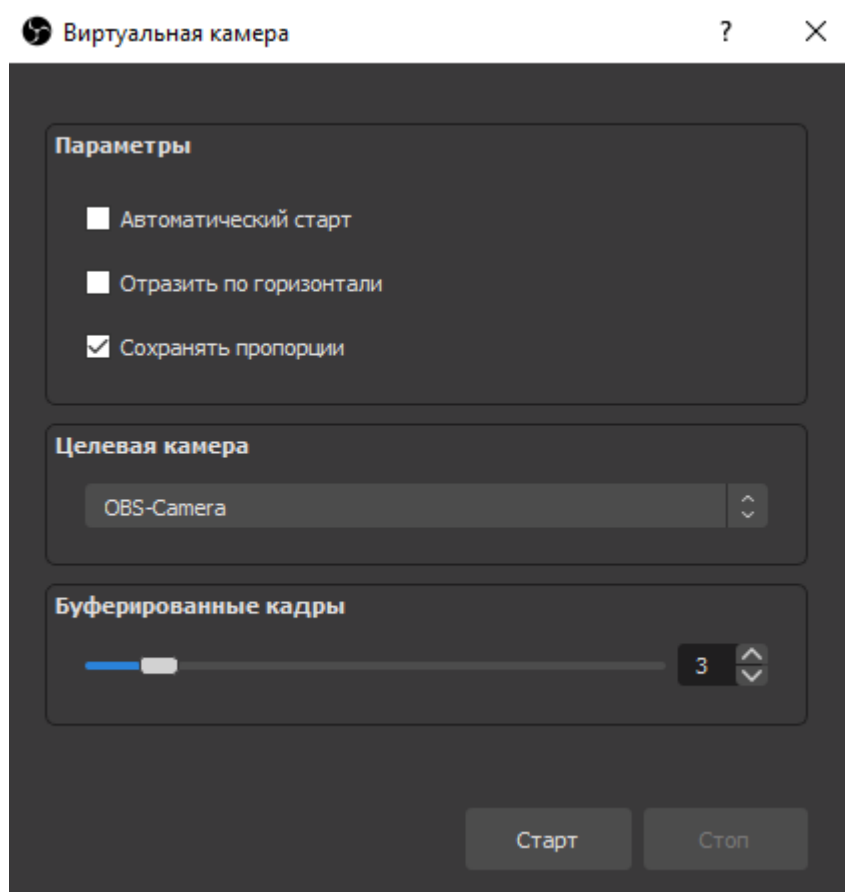


[OBS версии 26.1.1](#) и установленный плагин [OBS Virtual Cam 2.0.5](#)

1. Переходим на вкладку Инструменты в верхней панели и выбираем Виртуальная камера.

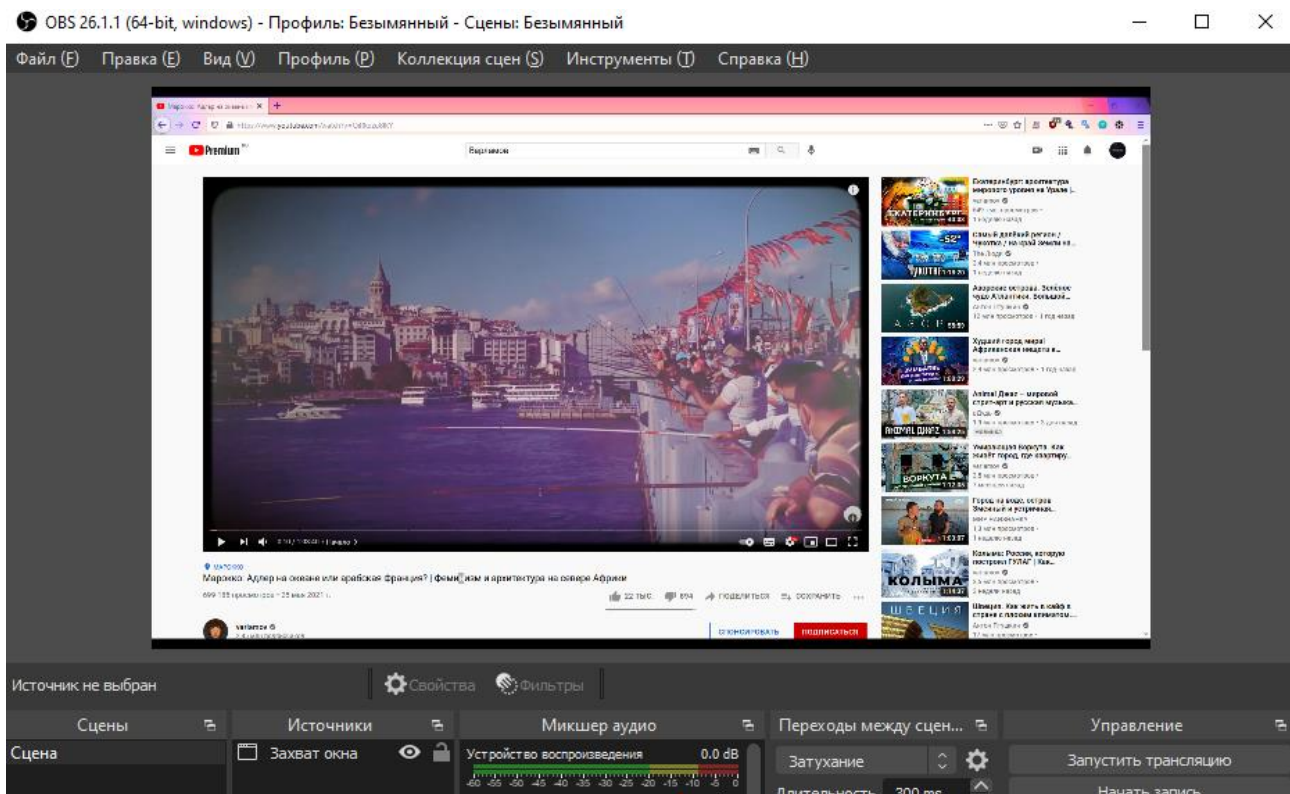
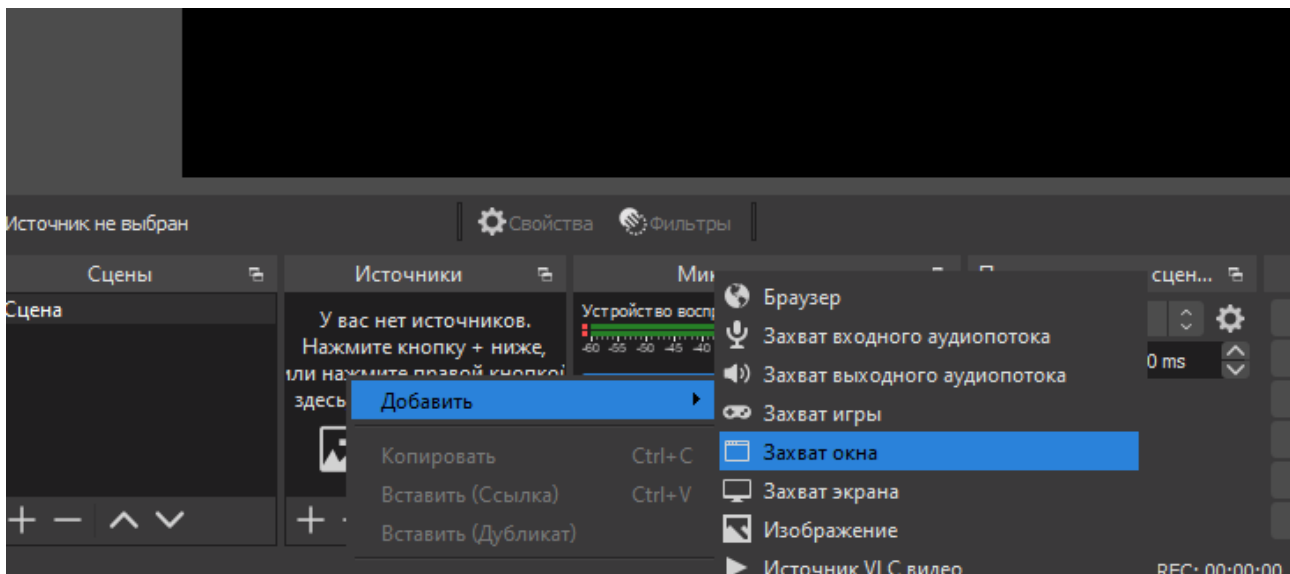


2. Нажимаем старт.



3. Закрываем окно виртуальной камеры.

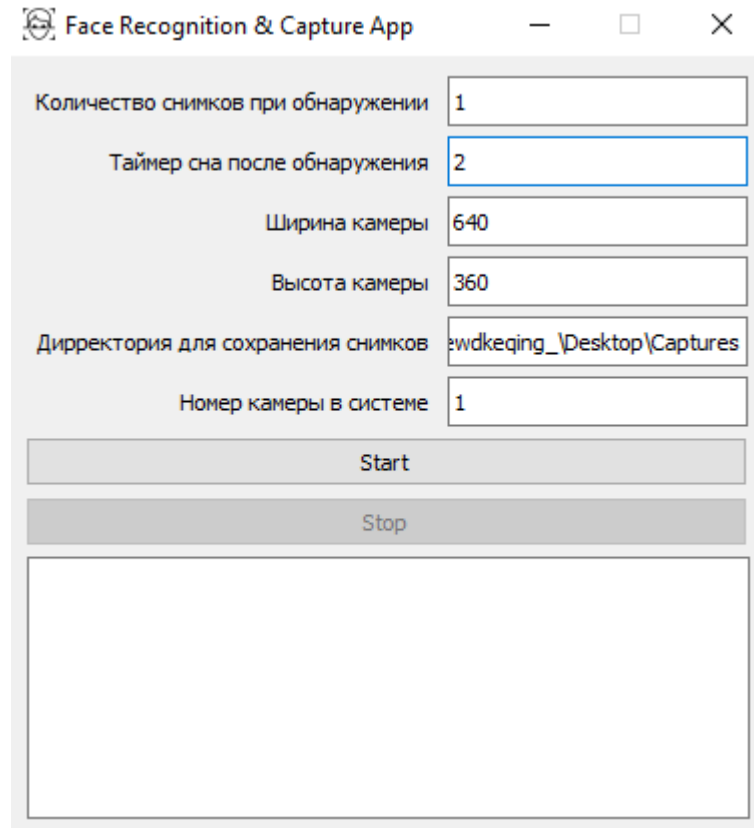
#### 4. Добавляем источник изображения, например окно браузера.



#### 5. Запускаем наше приложение и указываем источник камеры, не забыв поменять в коде строку, если мы её меняли

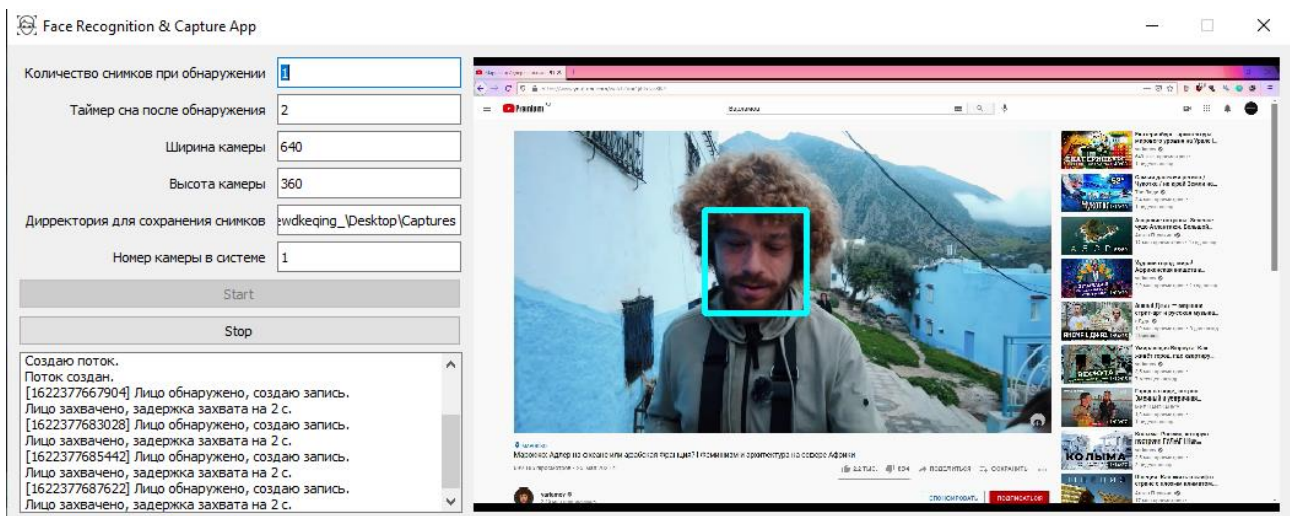
```
226  
227     cap = cv2.VideoCapture(self.cam_id) #Выбираем источник изображения  
228
```

6. Указываем номер камеры в системе. Т.к. у меня есть веб – камера в ноутбуке, и она числится под цифрой 0, то виртуальная камера OBS, будет работать под номером 1.



Мои настройки.

7. Нажимаем Start.



Захват лица работает и сохраняет наши кадры в папку Captures.

Captures

Поиск: Captures

30 May 2021  
17-28-12-162237  
7692659  
Person.jpg

30 May 2021  
17-28-10-162237  
7690469  
Person.jpg

30 May 2021  
17-28-07-162237  
7687622  
Person.jpg

30 May 2021  
17-28-05-162237  
7685442  
Person.jpg

30 May 2021  
17-28-03-162237  
7683028  
Person.jpg

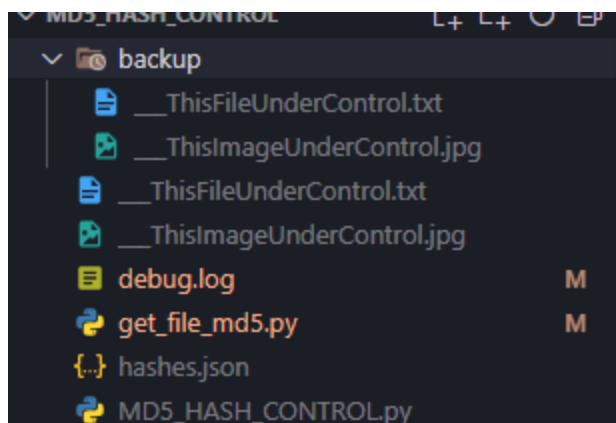
## Лабораторная работа №8

### Контроль целостности файлов

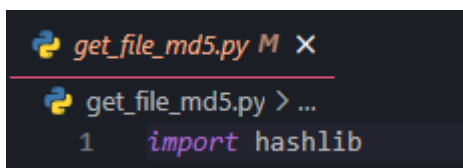
Создаём папку MD5\_HASH\_Control и создаём в ней следующие файлы:

- `get_file_md5.py` – для получения хэшей файлов;
- `MD5_HASH_CONTROL.py` – наш будущий скрипт для контроля целостности файлов.
- `debug.log` – для логирования при изменении файлов или их удалении;
- `hashes.json` – для хранения эталонов хэшей контролируемых файлов;
- любой текстовый файл с текстом, например `___ThisFileUnderControl.txt` для контроля данного файла;
- любую картинку, для контроля данной картинки;
- папку `backup`, в которой будут размещаться копии файлов;

Итого должно получиться вот так:



Открываем файл `get_file_md5.py` и импортируем модуль `hashlib`.



Создаём функцию `get_hash_md5(filename)`:

```
get_file_md5.py M X
get_file_md5.py > ...
1  import hashlib
2
3  def get_hash_md5(filename):
4      with open(filename, 'rb') as f:
5          m = hashlib.md5()
6          while True:
7              data = f.read(8192) #Читаем блоки
8              if not data:
9                  break
10             m.update(data)
11             print(m.hexdigest()) #Возвращаем хэш
12             x = input("\nНажмите любую клавишу...")
13
14 | name_of_file = input("\nВведите имя файла: ")
15 | get_hash_md5(name_of_file)
16 |
```

При запуске данной функции происходит открытие указанного файла на чтение, затем в переменную `m` передаются функции из модуля `hashlib.md5()`. После, в бесконечном цикле, происходит чтение блоков из файла и добавление их в переменную с помощью `m.update`. По окончании работы цикла, выводим на экран хэш с помощью `m.hexdigest()`.

Проверим работу и запустим.



Запишем в наш текстовый файл пару строчек и отправим программе на вычисление хэша.

```
get_file_md5.py M  __ThisFileUnderControl.txt X
__ThisFileUnderControl.txt
1  This text under control. Huh.
```



```
PS C:\Users\lewdkeqing\Desktop\MD5_Hash_Control>

Введите имя файла: ___ThisFileUnderControl.txt
c10e73bff48e039730a35ff606e18e39

Нажмите любую клавишу...
```

### Хэш вычислен!

Отправим на вычисление хэша картинку, которую будем контролировать.

```
PS C:\Users\lewdkeqing\Desktop\MD5_Hash_Control>

Введите имя файла: ___ThisImageUnderControl.jpg
84aac28119ba402eff1f7eeb1179ca23

Нажмите любую клавишу...
```

Запишем эти хэши в файл hashes.json вот в таком формате:

```
{..} hashes.json > ...
1  {
2      "___ThisImageUnderControl.jpg": [
3          "84aac28119ba402eff1f7eeb1179ca23"
4      ],
5      "___ThisFileUnderControl.txt": [
6          "c10e73bff48e039730a35ff606e18e39"
7      ]
8  }
```

Также скопируйте контролируемые файлы в папку backup.

Переходим к написанию скрипта, для проверки целостности файла, открываем файл MD5\_HASH\_CONTROL.py.

Импортируем модули hashlib, shutil, codecs, time, json, os.

```
MD5_HASH_CONTROL.py > ...
1  import hashlib
2  import shutil
3  import codecs
4  import time
5  import json
6  import os
```

- hashlib – для вычисления хэшей файлов;



- `shutil` – для копирования эталонов файлов;
- `codecs` – для корректного отображения кириллицы;
- `time` – для получения времени;
- `json` – для чтения файлов `.json` формата;
- `os` – для очистки вывода в консоль.

```

6  import os
7
8  update_secs = 1 #Периодичность проверок в секундах.
9  backup_dir = "backup" #Директория
10

```

Добавляем две переменные, `update_secs` – будет отвечать за периодичность сравнения хэшей, а `backup_dir` – будет содержать имя директории в которой будут храниться эталоны файлов.

```

9  backup_dir = "backup" #Директория
10
11  file_name = []
12  file_hash = []
13

```

Затем добавляем две переменные в виде списков, которые будут содержать имена файлов и хэши файлов.

```

12  file_hash = []
13
14  with codecs.open('hashes.json', 'r', encoding='utf-8') as f: #Открываем hashes.json
15      templates = json.load(f) #Загружаем имя контролируемого файла и его хэш
16

```

Затем с помощью библиотеки `codecs` и метода `codecs.open` открываем файл `hashes.json` содержащий имена файлов и эталоны хэшей.

А с помощью библиотеки `json` и метода `json.load` загружаем в переменную `templates` имена файлов и их эталоны хэшей.

```

15     templates = json.load(f)  #Загружаем имя контролируемого файла и его хэш
16
17     for section, commands in templates.items(): #Сохраняем имя файла и его хэш в переменные
18         file_name.append(section)
19         file_hash.append('\n'.join(commands))
20

```

Далее в цикле читаем переменную `templates` с помощью метода `templates.items`, разделяя их. Далее с помощью метода `append()` добавляем в переменные `file_name` и `file_hash` строки, формируя массивы.

```

19     file_hash.append('\n'.join(commands))
20
21     def debug(name): #Данная функция формирует лог файл
22         try:
23             f = open(f'debug.log', 'a', encoding='utf8')
24             f.write("\n[" + time.strftime("%d-%B-%Y %H-%M-%S") + name)
25         except PermissionError:
26             print(f"[ВНИМАНИЕ] Ошибка получения прав для записи в файл debug.log")
27         return None

```

Далее создаём функцию `debug`, которая будет отвечать за логирование в файл `debug.log` при изменении состояния контролируемых файлов.

При вызове данной функции с помощью стандартного метода `open()`, мы откроем файл `debug.log` на добавление ('a') с кодировкой `utf-8`.

Записываем в файл информацию с временем `time.strftime + name`, где `name` – это текст, передаваемый в функцию `debug`. Также данная конструкция обернута в `try`, `except`. Для перехвата ошибки, в случае если доступа на дозапись в файл `debug.log` не окажется.

```

27         return None
28
29 def replace_file(name): #Данная функция достаёт резервные копии из дирректории
30     try: #Обрабатываем исключения если резервная копия не найдена!
31         src = f"{backup_dir}\\{name}"
32     except FileNotFoundError:
33         print(f"Не удалось найти резервную копию файла {name}")
34         debug(f"Не уадлось найти резервную копию файла {name}")
35         return None
36     dst = f"{name}"
37     try: #Обрабатываем исключения если резервная копия не найдена!
38         shutil.copy2(src, dst) #Копируем файл
39     except FileNotFoundError:
40         print(f"Не удалось найти резервную копию файла {name}")
41         debug(f"Не уадлось найти резервную копию файла {name}")
42         return None

```

Далее создаём функцию `replace_file(name)`, которая будет отвечать за восстановление файлов при их изменении.

Создаём переменную `src`, в которой будем указывать на файл, который будет восстанавливаться, с помощью `print` и написанной функции `debug` логируем.

Создаём переменную `dst`, в которой будем указывать имя файла.

Далее с помощью библиотеки `shutil` и метода `shutil.copy2` восстанавливаем файл из указанной в переменной `src` папки `backup`.

```

42         return None
43
44     num_of_files = len(file_name)
45

```

Получаем количество файлов в переменную `num_of_files`.

```

44 num_of_files = len(file_name)
45
46 while True: #Цикл
47     os.system("cls")
48     for i in range(0, num_of_files):
49         def get_hash_md5(filename):
50             try:
51                 with open(filename, 'rb') as f:
52                     m = hashlib.md5()
53                     while True:
54                         data = f.read(8192) #Читаем блоки
55                         if not data:
56                             break
57                         m.update(data)
58                     return m.hexdigest() #Возвращаем хэш
59             except FileNotFoundError:
60                 replace_file(f"{file_name[i]}") #Если файл не найден, вызываем функцию для восстановления файла из резервной копии
61
62         now_hash = get_hash_md5(f"{file_name[i]}") #Сохраняем измененный хэш в переменную
63
64         if now_hash == file_hash[i]: #Проверяем файл на соответствие
65             print(f"[+] {file_name[i]}")
66         else:
67             print(f"[X] {file_name[i]} - не прошёл проверку или не найден... Восстанавливаю файл...")
68             debug(f"[X] {file_name[i]} - не прошёл проверку... Текущий хэш: {now_hash}. Требуемый хэш: {file_hash[i]}. Восстанавливаю файл.")
69             replace_file(f"{file_name[i]}")
70
71     time.sleep(update_secs) #Периодичность проверок, отправляем программу спать на N-ое количество секунд

```

Создаём бесконечный цикл while True:

Далее в цикле for i in range(0, num\_of\_files), проходимся по файлам. Если возникает ошибка и файл не найден мы его восстанавливаем с помощью функции replace\_file.

В случае если всё хорошо и файл был найден, вычисляем хэш в переменную now\_hash.

Далее с помощью if проверяем текущий хэш с эталоном и выводим информацию.

В конце цикла программа засыпает на определённое количество секунд указанное в переменной update\_secs.

**Запускаем файл MD5\_HASH\_CONTROL.py и проверяем работу.**

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[+] ___ThisImageUnderControl.jpg
[+] ___ThisFileUnderControl.txt

```

Программа работает и показывает, что с файлами всё в порядке.

Попробуем изменить текстовый файл \_\_\_ThisFileUnderControl.txt

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[+] ___ThisImageUnderControl.jpg
[X] ___ThisFileUnderControl.txt - не прошёл проверку или не найден... Восстанавливаю файл...
```

При изменении файла программа пишет, что файл не прошёл проверку.

Удалим второй файл \_\_\_ThisImageUnderControl.jpg

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[X] ___ThisImageUnderControl.jpg - не прошёл проверку или не найден... Восстанавливаю файл...
[+] ___ThisFileUnderControl.txt
```

При удалении картинки программа также пишет, что файл не прошёл проверку  
или не найден.

Проверяем debug.log

```
debug.log
1
2 [31-May-2021 16:50:51] [X] ___ThisFileUnderControl.txt - не прошёл проверку... Текущий хэш: f1ed767244003fc483bc12ccbeb08da0. Требуемый хэш: c10e73b
3 [31-May-2021 16:50:55] [X] ___ThisFileUnderControl.txt - не прошёл проверку... Текущий хэш: 5b71c9f4c57bd735b024c0607cecd87b. Требуемый хэш: c10e73b
4 [31-May-2021 16:51:59] [X] ___ThisImageUnderControl.jpg - не прошёл проверку... Текущий хэш: None. Требуемый хэш: 84aac28119ba402eff1f7eeb1179ca23.
5 [31-May-2021 16:52:03] [X] ___ThisImageUnderControl.jpg - не прошёл проверку... Текущий хэш: None. Требуемый хэш: 84aac28119ba402eff1f7eeb1179ca23.
6 [31-May-2021 16:52:08] [X] ___ThisImageUnderControl.jpg - не прошёл проверку... Текущий хэш: None. Требуемый хэш: 84aac28119ba402eff1f7eeb1179ca23.
```

Информация логируется и записывается в файл.

## Лабораторная работа №9

### Сканирование портов

Открываем Visual Studio Code и создаём два .py файла, один называем scanner.py, а второй main.py.

Открываем файл scanner.py и импортируем модуль socket.

```
scanner.py > ...  
1  import socket  
2  |
```

Далее создаём класс PortScanner и функцию инициализации \_\_init\_\_.

```
scanner.py > ...  
1  import socket  
2  |  
3  class PortScanner:  
4      def __init__(self, ip, ports):  
5          self.ip = ip  
6          self.ports = ports
```

Данная функция инициализации принимает переменные ip и ports, а затем она их присваивает к переменным в «своём окружении» (self.ip, self.ports.)

Создаём функцию scan\_tcp\_port(self, port):

```
6      self.ports = ports  
7  
8      def scan_tcp_port(self, port):  
9          connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
10         connection.settimeout(0.3)  
11         yield connection.connect_ex((self.ip, port)), port  
12         connection.close()
```

Данная функция служит для создания соединения с хостом и последующего получения значения при соединении. Конструкция connection.connect\_ex – возвращает значение 0, если порт открыт.

yield – это альтернатива return, которая работает аналогичным образом, однако yield позволяет сэкономить память и реализовать взаимодействие между

несколькими циклами. С помощью `yield` функция возвращает значение без уничтожения локальных переменных, кроме того, при каждом последующем вызове функция начинает своё выполнение с оператора `yield`.

Создаём функцию `scan_ports(self)`:

```
11         yield connection.connect_ex((self.ip, port)), port
12         connection.close()
13
14     def scan_ports(self):
15         for port in self.ports:
16             yield from self.scan_tcp_port(port)
```

Данная функция отправляет значения из списка в функцию `scan_tcp_port()`.

Создаём функцию `host_up(self)`:

```
15         for port in self.ports:
16             yield from self.scan_tcp_port(port)
17
18     def host_up(self):
19         try:
20             connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21             connection.connect_ex((self.ip, 80))
22             return True
23         except socket.timeout:
24             return True
25         except socket.error:
26             return False
```

Данная функция отвечает за проверку работы хоста. При возникновении ошибок срабатывает эксерт и в зависимости от ошибки функция возвращает True/False.

Создаём функцию `scan(ip, ports=range(1, 65536))`:

```

23         except socket.timeout:
24             return True
25         except socket.error:
26             return False
27
28
29     def scan(ip, ports=range(1, 65536)):
30         scanner = PortScanner(ip, ports)
31         if not scanner.host_up():
32             print("Хост не отвечает.")
33             return
34
35         for connection, port in scanner.scan_ports():
36             connection = "ОТКРЫТ" if connection == 0 else "ЗАКРЫТ или ФИЛЬТРУЕТСЯ"
37             print(f"Порт {port} {connection}")
38

```

Данная функция отвечает за запуск сканирования и вывод результатов. Также для переменной `ports` заданы стандартные значения. Если значение `ports` не будет передано, то в таком случае, сканируем все возможные порты (1–65535).

Открываем `main.py` и импортируем наш файл `scanner.py` в качестве модуля, с помощью `from scanner import scan`.

С помощью NMAP или любого другого способа, узнаём IP адрес любого сайта. Например [youtube.com](https://youtube.com).

```

Starting Nmap 7.70 ( https://nmap.org ) at 2021-06-04 05:05 UTC
Nmap scan report for youtube.com (142.250.64.78)
Host is up (0.00091s latency).
Other addresses for youtube.com (not scanned): 2607:f8b0:4006:806::200e
rDNS record for 142.250.64.78: lga34s30-in-f14.1e100.net

PORT      STATE      SERVICE
21/tcp    filtered  ftp
22/tcp    filtered  ssh
23/tcp    filtered  telnet
80/tcp    open      http
110/tcp   filtered  pop3
143/tcp   filtered  imap
443/tcp   open      https
3389/tcp   filtered  ms-wbt-server

Nmap done: 1 IP address (1 host up) scanned in 1.26 seconds

```

Теперь мы получили IP – адрес и порты, проверим работу нашего скрипта.



```
main.py
1  from scanner import scan
2
3  scan("142.250.64.78", [21, 22, 23, 80, 110, 143, 443, 3389])
```

С помощью вызова функции scan из нашего модуля scanner, передаём в неё значения IP адреса и сканируемых портов.

**ВАЖНО:** Порты должны быть записаны в качестве списка [21, 22, и т.д.], если мы желаем просканировать сразу несколько портов.

Запускаем скрипт.

```
PS C:\Users\lewdkeqing\Desktop\Workspace> &
Порт 21 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
Порт 22 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
Порт 23 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
Порт 80 ОТКРЫТ
Порт 110 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
Порт 143 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
Порт 443 ОТКРЫТ
Порт 3389 ЗАКРЫТ или ФИЛЬТРУЕТСЯ
PS C:\Users\lewdkeqing\Desktop\Workspace>
```

Скрипт работает!

## Лабораторная работа №10

### Компиляция .py в .exe

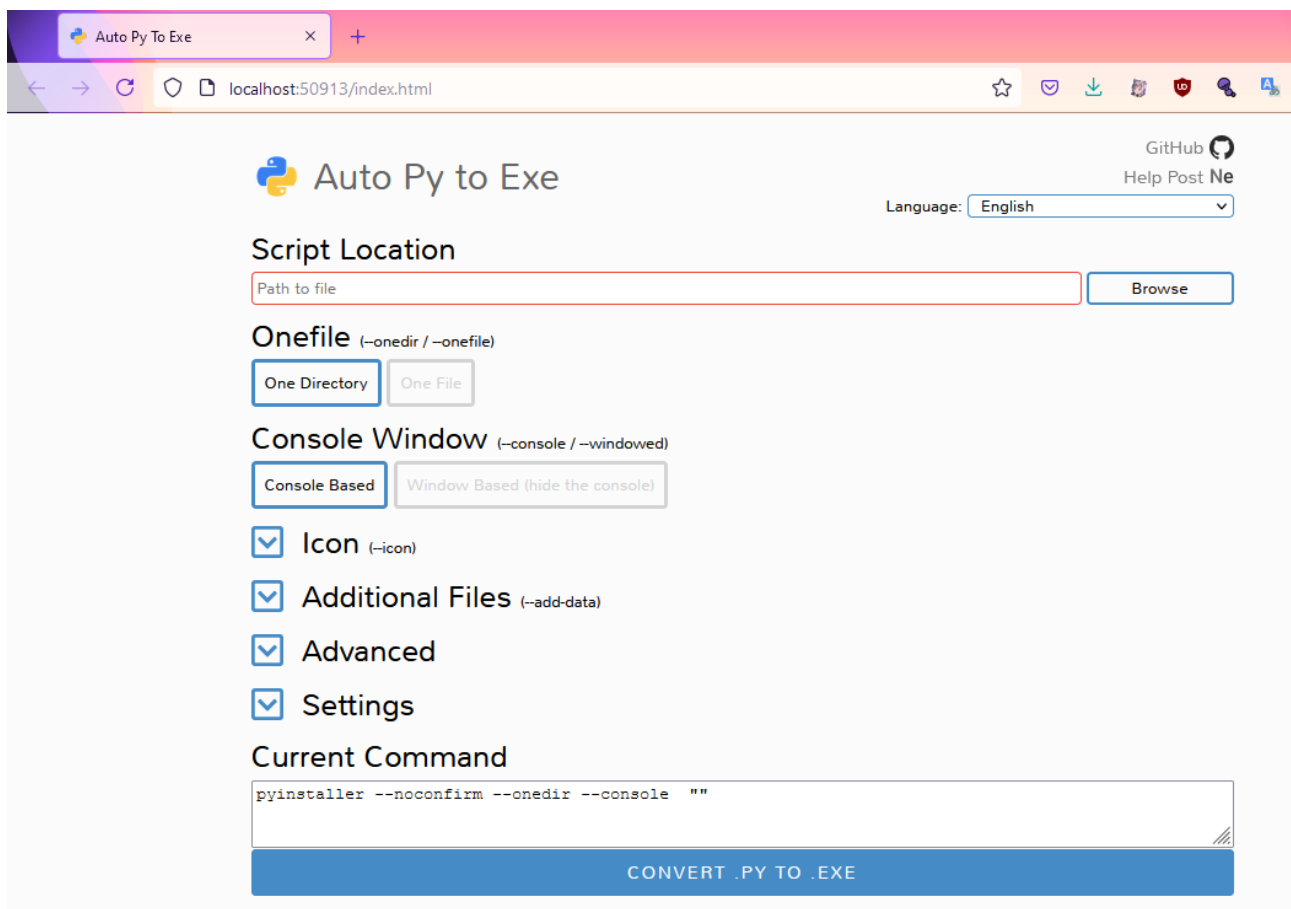
Для компиляции скриптов в исполняемые файлы, воспользуемся модулем `auto-py-to-exe`.

Переходим в командную строку и с помощью команды: `pip install auto-py-to-exe` устанавливаем модуль.

Прописываем в консоль команду `auto-py-to-exe`.

```
C:\Windows\system32\cmd.exe - auto-py-to-exe  
  
C:\Users\lewdkeqing\Desktop>auto-py-to-exe
```

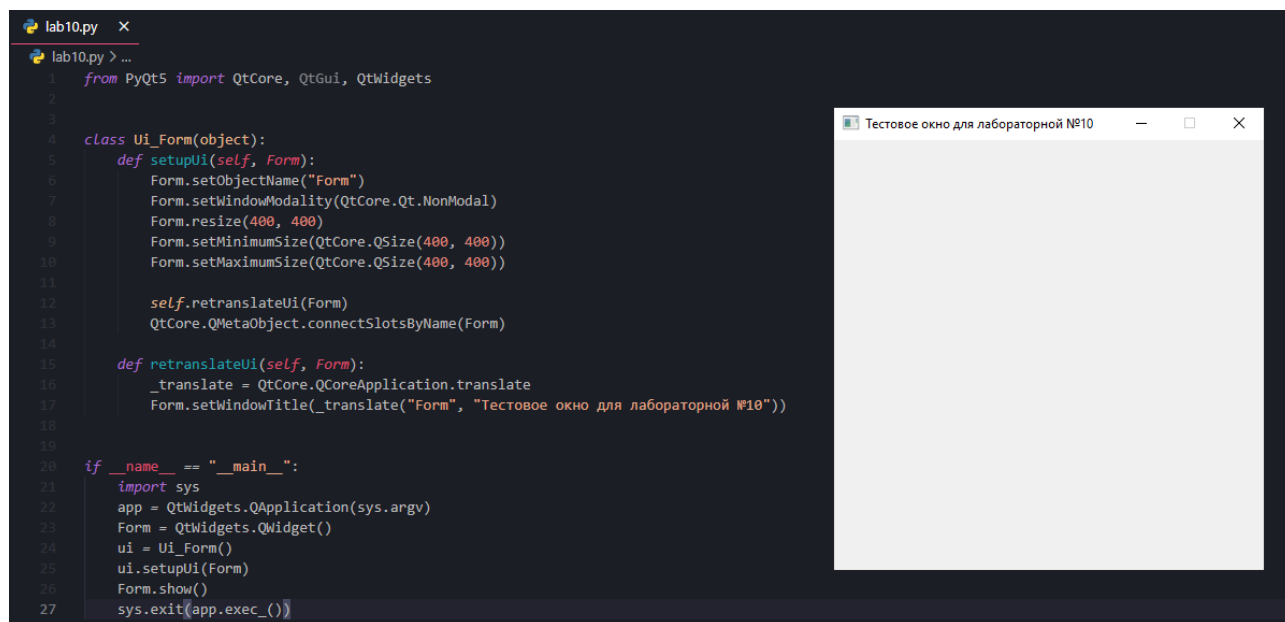
Открывается окно в браузере с локальной страницей (localhost).



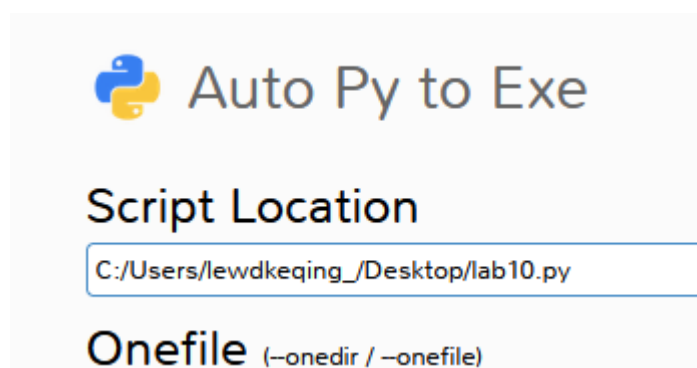
Данный модуль – это просто удобная обёртка для модуля `pyinstaller`.

В окне Current Command мы можем наблюдать получившуюся команду в ходе настройки.

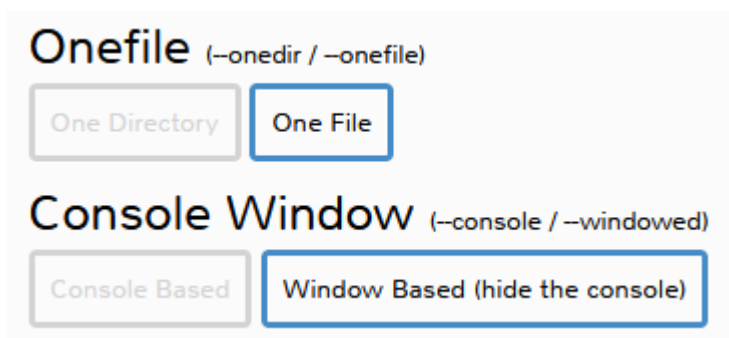
Добавим любой скрипт написанный на Python, например простой скрипт, который будет запускать окно PyQt5 интерфейса.



Добавляем файл lab10.py в окне браузера.



Указываем опции



Данные опции отвечают за то, как будет работать наше приложение.

- Onefile – означает что все нужные компоненты для работы приложения, будут лежать внутри исполняемого файла и при необходимости автоматически помещаться во временную директорию %Temp%.
- Window Based – означает что при запуске нашего исполняемого файла, будет скрыт терминал (командная строка).

Также рекомендую выставить опцию --clean.

Данная опция отвечает за чистую компиляцию все дополнительные/вспомогательные файлы по окончании компиляции будут удалены и останется только наше приложение.


Как видно на скриншоте, наша Current Command изменилась. Нажимаем CONVERT.PY TO .EXE.

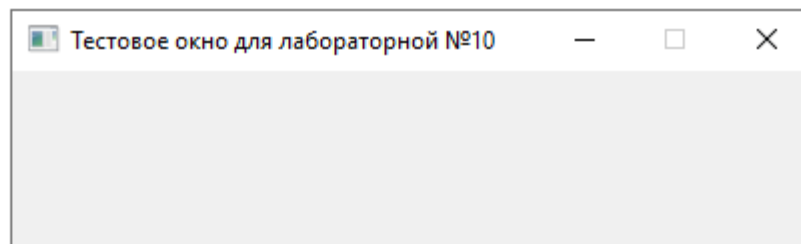
## Output

```
Running auto-py-to-exe v2.9.0
Building directory: C:\Users\LEWDKE~1\AppData\Local\Temp\tmp26lgdhpn
Provided command: pyinstaller --noconfirm --onefile --windowed --clean "C:/Users/lewdkeqing_/Deskt
Recursion Limit is set to 5000
Executing: pyinstaller --noconfirm --onefile --windowed --clean C:/Users/lewdkeqing_/Desktop/lab10.

1404140 INFO: PyInstaller: 4.3
1404154 INFO: Python: 3.9.5
1404175 INFO: Platform: Windows-10-10.0.19042-SP0
1404185 INFO: wrote C:\Users\LEWDKE~1\AppData\Local\Temp\tmp26lgdhpn\lab10.spec
1404195 INFO: UPX is not available.
1404197 INFO: Removing temporary files and cleaning cache in C:\Users\lewdkeqing_\AppData\Local\pyi
1404228 INFO: Extending PYTHONPATH with paths
['C:\\Users\\lewdkeqing_\\Desktop',
 'C:\\Users\\LEWDKE~1\\AppData\\Local\\Temp\\tmp26lgdhpn']
1404293 INFO: checking Analysis
1404295 INFO: Building Analysis because Analysis-00.toc is non existent
1404299 INFO: Initializing module dependency graph...
1404312 INFO: Caching module graph hooks...
1404337 WARNING: Several hooks defined for module 'win32ctypes.core'. Please take care they do not
1404366 INFO: Analyzing base_library.zip ...
```

CONVERTING...

Имя	Дата изменения	Тип	Па
 lab10.exe	05.06.2021 17:24	Приложение	



Готовое рабочее приложение в виде исполняемого файла