******

Колледж космического машиностроения и технологий

ОТЧЕТ

По учебной практике УП.01.01 Разработка программных модулей

программного обеспечения для компьютерных систем

специальность 09.02.03 Программирование в компьютерных системах

Выполнил студент:

Батраков Д. С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

Гусятинер Л. Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(оценка)

Королев, 2020

Содержание отчёта

[**Приложения** 3](#_Toc73444595)

[Техника работы с линейными и разветвляющимися программами 3](#_Toc73444596)

[Техника работы со строками 3](#_Toc73444597)

[Техника работы со списками 3](#_Toc73444598)

[Техника работы с циклом for и генераторами списков 3](#_Toc73444599)

[Техника работы с функциями 4](#_Toc73444600)

[Техника работы со словарями 4](#_Toc73444601)

[Техника работы с множествами 4](#_Toc73444602)

[Техника работы с кортежами 4](#_Toc73444603)

[Техника работы с файлами 4](#_Toc73444604)

[Техника работы с модулями 4](#_Toc73444605)

[Техника работы с классами 5](#_Toc73444606)

[**Раздел 1. Техника решения задач с использованием структурного и объектно-ориентированного программирования.** 5](#_Toc73444607)

[1.1 Установка интерпретатора Python 3 и настройка окружения 5](#_Toc73444608)

[1.2 Техника работы в командной строке и среде IDLE 8](#_Toc73444609)

[1.3 Техника работы с линейными и разветвляющимися программами 9](#_Toc73444610)

[1.4 Техника работы с циклическими программами, цикл while 12](#_Toc73444611)

[1.5 Техника работы с числами 15](#_Toc73444612)

[1.6 Техника работы со строками 18](#_Toc73444613)

[1.7 Техника работы со списками 20](#_Toc73444614)

[1.8 Техника работы с циклом for и генераторами списков 22](#_Toc73444615)

[1.9 Техника работы с функциями 24](#_Toc73444616)

[1.10 Техника работы со словарями 26](#_Toc73444617)

[1.11 Техника работы с множествами 30](#_Toc73444618)

[1.12 Техника работы с кортежами 33](#_Toc73444619)

[1.13 Техника работы с файлами. 34](#_Toc73444620)

[1.14 Техника работы с модулями 34](#_Toc73444621)

[1.15 Техника работы с классами. 43](#_Toc73444622)

[**Раздел 2. Техника решения задач с использованием библиотек** 54](#_Toc73444623)

[2.1 Установка и настройка среды JetBrains PyCharm 54](#_Toc73444624)

[2.2 Техника работы с базами данных 57](#_Toc73444625)

[2.3 Техника работы с библиотекой tkinter 60](#_Toc73444626)

[2.4 Техника работы с библиотекой NumPy 65](#_Toc73444627)

[2.5 Техника работы с библиотекой Matplotlib 67](#_Toc73444628)

[2.6 Элементы работы с библиотекой PyQt 68](#_Toc73444629)

[2.7 Элементы работы с библиотекой PyGame 69](#_Toc73444630)

[**Раздел 3. Разработка проекта с графическим интерфейсом** 73](#_Toc73444631)

[3.1 Изучение входной и выходной документации 73](#_Toc73444632)

[3.2 Разработка требований к проекту. Построение диаграммы использования. 75](#_Toc73444633)

[3.3 Разработка сценария проекта 77](#_Toc73444634)

[3.4 Построение диаграммы классов 78](#_Toc73444635)

[3.5 Разработка базы данных 78](#_Toc73444636)

[3.6 Разработка главного модуля 82](#_Toc73444637)

[3.7 Разработка входящих модулей 83](#_Toc73444638)

[3.8 Тестирование и отладка 89](#_Toc73444639)

[3.9 Дневник 90](#_Toc73444640)

# **Приложения**

## Техника работы с линейными и разветвляющимися программами

**Файлы:**

**К4\_1**

**K4\_2\_1**

**K5\_1\_2**

Техника работы с циклическими программами, цикл while

**Файлы:**

**К5\_2\_1**

**К5\_2\_2**

**К5\_2\_3**

**К5\_2\_4**

**К5\_2\_5**

Техника работы с числами

**Файлы:**

**К6\_1\_1**

**К6\_2\_1**

**К6\_2\_2**

## Техника работы со строками

**Файлы:**

**К7\_1\_1**

**К7\_1\_2**

**К7\_1\_3**

**К7\_1\_4**

## Техника работы со списками

**Файлы:**

**К8\_1\_1**

**К8\_1\_2**

**К8\_2\_1**

**К8\_2\_3**

Техника работы с циклом for и генераторами списков

**Файлы:**

**К9\_2\_1**

**К9\_2\_2**

**К9\_2\_3**

## Техника работы с функциями

**Файлы:**

**К10\_1\_2**

**К10\_1\_3**

**К10\_2\_2**

**К10\_2\_3**

**К10\_2\_4**

## Техника работы со словарями

**Файлы:**

**К11\_1\_1**

**К11\_1\_2**

**К11\_1\_3**

**К11\_2\_1**

**К11\_2\_2**

**K11\_2\_3**

## Техника работы с множествами

**Файлы:**

**К12\_1\_1**

**К12\_1\_2**

**К12\_1\_3**

**К12\_1\_4**

**К12\_1\_5**

**К12\_2\_1**

**К12\_2\_1**

Техника работы с кортежами

**Файлы:**

**К13\_1\_1**

**К13\_1\_2**

Техника работы с файлами

**Файлы:**

**К14\_1\_1**

**К14\_1\_2**

Техника работы с модулями

**Файлы:**

**K15\_1\_1**

**K15\_1\_2**

**K15\_2\_1**

Техника работы с классами

**Файлы:**

**K16\_1**

**K16\_2**

**K16\_3**

**K16\_4**

# **Раздел 1. Техника решения задач с использованием структурного и объектно-ориентированного программирования.**

## Установка интерпретатора Python 3 и настройка окружения

Для установки интерпретатора Python на компьютер, вам нужно скачать дистрибутив. Загрузить его последнюю версию можно с официального сайта, перейдя по ссылке https://www.python.org/downloads/

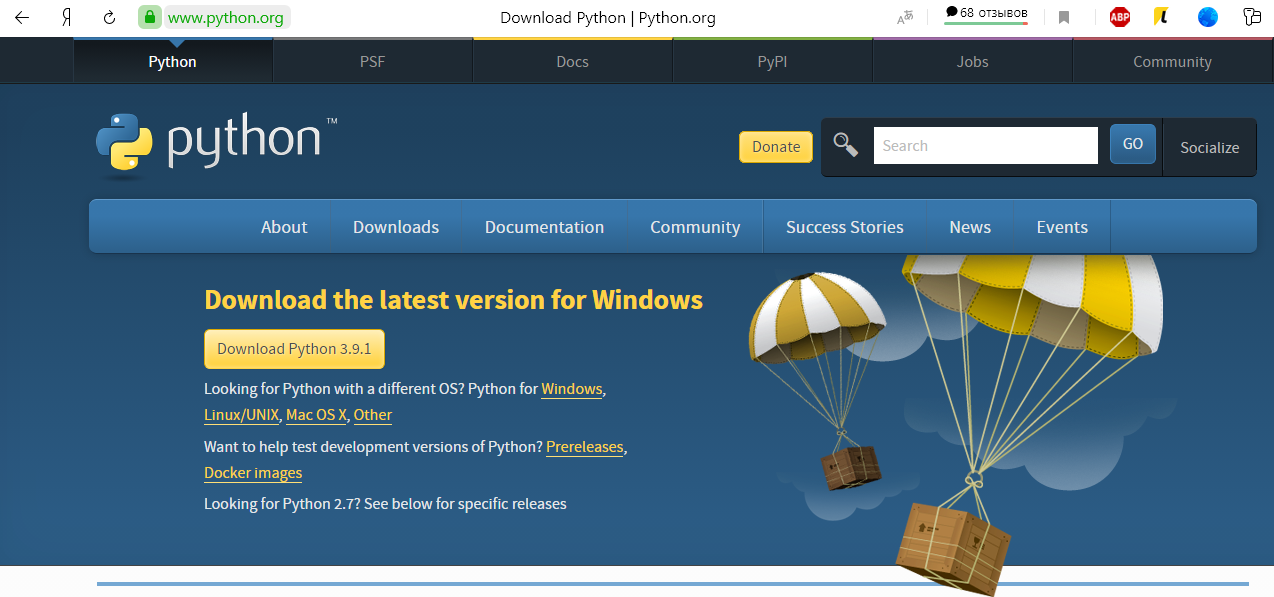


Рис 1. Официальный сайт Python

Порядок установки на Windows:

1. Запустить скачанный установочный файл.

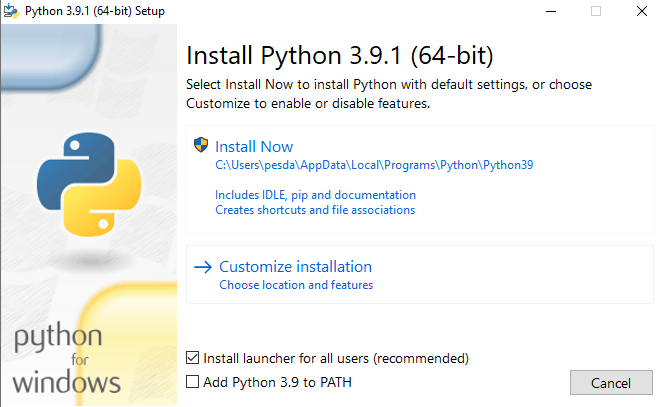
2. Выбрать способ установки.

Рис 2. Установщик Python

3. Отметить необходимые опции установки (доступно при выборе Customize installation)

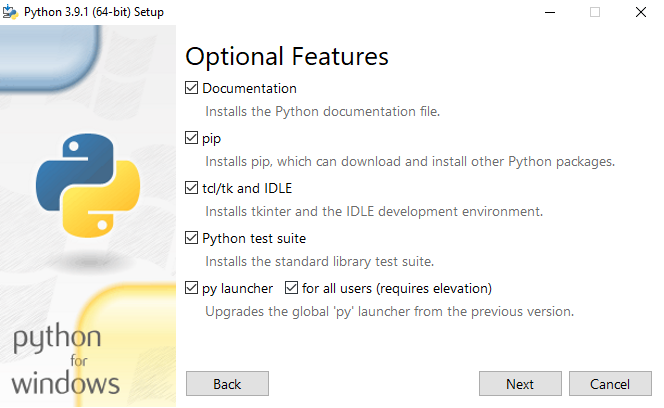


Рис 3. Опции установки

На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Выбираю:

* Documentation – установка документаций.
* pip – установка пакетного менеджера.
* tcl/tk and IDLE – установка интегрированной среды разработки и библиотеки.

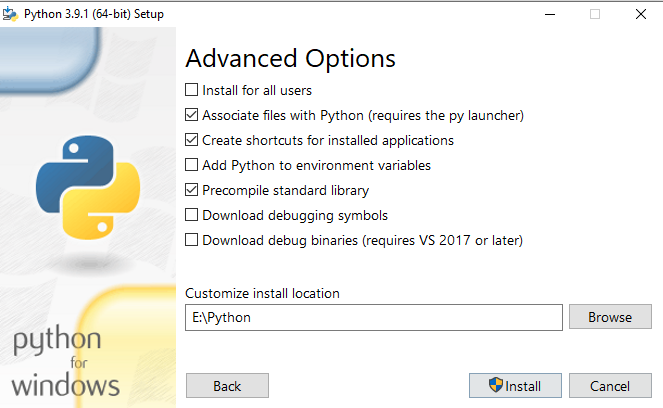
4. Выбираем место установки (доступно при выборе Customize installation)

Рис 4. «Продвинутые» опции установки

5. После успешной установки python:

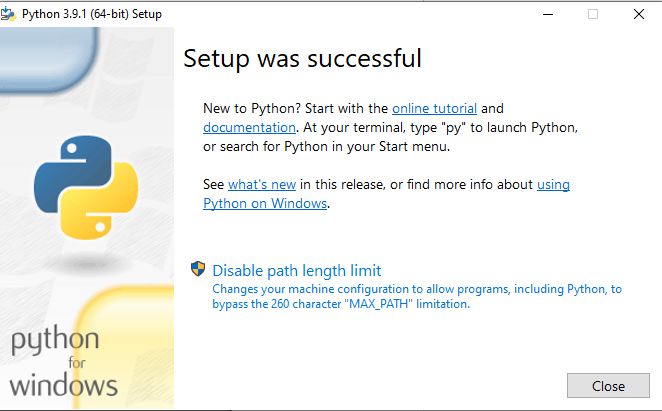


Рис 5. Сообщение об установке python

## Техника работы в командной строке и среде IDLE

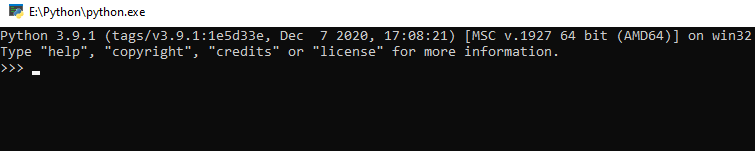
Выполняя (запуская) команду “python” в вашем терминале, вы получаете интерактивную оболочку Python.

Рис 6. Интерактивная оболочка Python

Существует несколько способов закрыть оболочку Python: >>> exit()или же >>> quit()

Теперь напишите в интерактивной оболочке следующий код:

>>> print("Hello, World ")

Нажмите **Enter** на вашей клавиатуре.

>>> print("Hello, World")

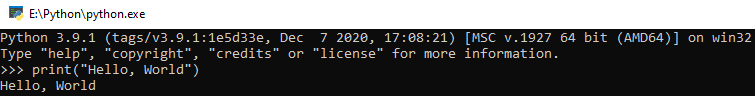
Hello, World

Рис 7. Первая программа

## Техника работы с линейными и разветвляющимися программами

**Листинг 1. K4\_1**

**'''**

- input

Функция input() в Python, ввод данных с клавиатура.

https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-input/

- print

Функция print() в Python, печатает объект.

https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-print/

- форматная строка и метод формат

**'''**

print('Введите имя: ')

a = (input())

print('Hi, {0}, {1}!'.format(a, input()))

**Листинг 2. K4\_2\_1**

**'''**

К4\_2. Техника работы с разветвляющимися программами

Задание 1. Разработать программу для печати даты прописью

Пример ввода: 15.12.1983

Пример вывода: Пятнадцатое декабря одна тысяча девятсот восемьдесят третьего года

**'''**

def get\_date(date):

day\_list = ['первое', 'второе', 'третье', 'четвёртое',

'пятое', 'шестое', 'седьмое', 'восьмое',

'девятое', 'десятое', 'одиннадцатое', 'двенадцатое',

'тринадцатое', 'четырнадцатое', 'пятнадцатое', 'шестнадцатое',

'семнадцатое', 'восемнадцатое', 'девятнадцатое', 'двадцатое',

'двадцать первое', 'двадцать второе', 'двадцать третье',

'двадацать четвёртое', 'двадцать пятое', 'двадцать шестое',

'двадцать седьмое', 'двадцать восьмое', 'двадцать девятое',

'тридцатое', 'тридцать первое']

month\_list = ['января', 'февраля', 'марта', 'апреля', 'мая', 'июня','июля', 'августа', 'сентября', 'октября', 'ноября', 'декабря']

#единицы

year\_list\_unit = ['','первого','второго','третьего', 'четвёртого', 'пятого', 'шестого', 'седьмого', 'восьмого', 'девятого']

year\_list\_unit2 = ['', 'одиннадцатого', 'двенадцатого', 'тринадцатого', 'четырнадцатого', 'пятнадцатого',

'шестнадцатого', 'семнадцатого', 'восемнадцатого', 'девятнадцатого']

#десятки

year\_list\_dec = ['','','двадцать','тридцать','сорок','пятьдесят','шестьдесят','семьдесят','восемьдесят','девяносто']

year\_list\_dec2 = ['', 'десятого', 'двадцатого', 'тридцатого', 'сорокового', 'пятидесятого', 'шестидесятого', 'семидесятого', 'восемидесятого',

'девяностого']

#сотни

year\_list\_hun = ['','сто','двести','триста','четыреста','пятьсот','шестьсот','семьсот','восемьсот','девятьсот']

year\_list\_hun2 = ['','сотого','двухсотого','трёхсотого','четырёхсотого','пятьсотого','шестьсотого','семьсотого','восьмисотого','девятьсотого']

#тысячи

year\_list\_th = ['','тысяча','две тысячи','три тысячи']

year\_list\_th2 = ['','тысячного','двух тысячного','трёх тысячного']

date\_list = date.split('.')

date\_y = [int(date\_list[2])//1000, int(date\_list[2])%1000//100, int(date\_list[2])%100//10,int(date\_list[2])%10]

if(date\_y[2] == 1):

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' +

year\_list\_th[date\_y[0]] + ' ' + year\_list\_hun[date\_y[1]-1] + ' ' + year\_list\_unit2[date\_y[3]] + ' ' + 'года')

elif (date\_y[0] == date\_y[1] == date\_y[2] == date\_y[3] == 0):

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' + 'нулевого года')

elif (date\_y[1] == date\_y[2] == date\_y[3] == 0):

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' +

year\_list\_th2[date\_y[0]] + ' ' + 'года')

elif(date\_y[2] == date\_y[3] == 0):

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' +

year\_list\_th[date\_y[0]] + ' ' + year\_list\_hun2[date\_y[1]] + ' ' + 'года')

elif(date\_y[3] == 0):

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' +

year\_list\_th[date\_y[0]] + ' ' + year\_list\_hun[date\_y[1]] + ' ' + year\_list\_dec2[date\_y[2]] + ' ' + 'года')

return (day\_list[int(date\_list[0]) - 1] + ' ' +

month\_list[int(date\_list[1]) - 1] + ' ' +

year\_list\_th[date\_y[0]] + ' ' + year\_list\_hun[date\_y[1]] + ' ' + year\_list\_dec[date\_y[2]] + ' ' + year\_list\_unit[date\_y[3]] + ' ' + 'года')

date = input()

while(date != 'stop'):

print(get\_date(date))

date = input()

**Листинг 3. K5\_1\_2**

**'''**

Задание 2.

Придумать пример(ы) на использование break / continue /else.

**'''**

while (1):

print('''(P) -> print \n(R) -> repit \n(B) -> break''')

i = input()

if i == 'P':

print("Hello world")

elif i == 'R':

print("Plz repit :)")

continue

elif i == 'B':

break

else:

print("Incorrect Data, try again")

#or

x = input()

i = 0

while x != '.':

if x == '!':

print(i)

break

i += 1

x = input()

else:

print('not found')

## Техника работы с циклическими программами, цикл while

**Листинг 4. К5\_2\_1**

**'''**

Задание 1. Вычислить значение sin(x) с точностью до epsilon при помощи разложения в ряд

**'''**

import math

eps = 1.0

n = 0

while eps + 1 > 1:

eps /= 2

n+=1

eps \*= 2

x = int(input())

s = 0

for i in range(n):

s += math.sin(x)

print(s)

**Листинг 5.К5\_2\_2**

**'''**

К5\_2. Техника работы с циклическими программами \_ цикл while;

Задание 2.

https://stepik.org/lesson/3364/step/11?unit=947

Напишите программу, которая считывает со стандартного ввода целые числа, по одному числу

в строке, и после первого введенного нуля выводит сумму полученных на вход чисел.

**'''**

def main():

a = int(input())

summ = 0

while a != 0:

summ = summ + a

a = int(input())

print(summ)

if \_\_name\_\_ == “\_\_main\_\_”:

main()

**Листинг 6. К5\_2\_3**

**'''**

К5\_2. Техника работы с циклическими программами \_ цикл while;

Задание 3.

Разработать программу для нахождения наибольшего общего делителя

**'''**

def nod(a, b):

assert a >= 0 and b >= 0

if a == 0 or b == 0:

return max(a, b)

return nod(b % a, a)

if \_\_name\_\_ == "\_\_main\_\_":

a, b = map(int, input().split())

print(nod(a, b))

**Листинг 7. К5\_2\_4**

**'''**

К5\_2. Техника работы с циклическими программами \_ цикл while;

Задание 4.

С использованием результата задания 2 разработать программу для нахождения наименьшего

общего кратного

'**''**

def main():

a, b = map(int, input().split())

c = a \* b

while (n != 0 and c != 0):

if (a > b):

a = a % b

else:

b = b % a

print(c // (a + b))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Листинг 8. К5\_2\_5**

**'''**

К5\_2. Техника работы с циклическими программами \_ цикл while;

Задание 5.

https://stepik.org/lesson/3369/step/8?unit=952

Напишите программу, которая выводит часть последовательности 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 ...

(число повторяется столько раз, чему равно).

На вход программе передаётся неотрицательное целое число n — столько элементов

последовательности должна отобразить программа.

На выходе ожидается последовательность чисел, записанных через пробел в одну строку.

Например, если n = 7, то программа должна вывести 1 2 2 3 3 3 4.

**'''**

def main():

n = int(input())

j = num = 1

for i in range(1, n+1):

print(num, end=' ')

if j < num:

j += 1

else:

j = 1

num += 1

print("\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Техника работы с числами

**Листинг 9. К6\_1\_1**

**'''**

Задание 1. Составить и выполнить по 3 примера использования модулей для работы

с дробными числами (fractions), для точных вычислений (decimal).

Задание 2. Подготовить инструкцию по использованию модулей fractions, decimal.

'**''**

from decimal import Decimal

from fractions import Fraction

#decimal

print("Функции <<Decimal>>")

number = Decimal("0.444")

print(number)

number = number.quantize(Decimal("1.00"))

print(number)

number = Decimal("0.555678")

print(number.quantize(Decimal("1.00")))

number = Decimal("0.999")

print()

#fraction

print("Функции <<Fraction>>")

print(number.quantize(Decimal("1.00")))

print(Fraction(1, 3))

print(Fraction(3.1415))

a = Fraction(1, 7)

b = Fraction(1, 3)

print(a + b)

**Листинг 10. К6\_2\_1**

**'''**

К6\_2. Техника работы с числами cmath

**'''**

import cmath

def main():

print("Сложные функции")

print("cmath.polar(complex(1.0, 1.0)) =", cmath.polar(complex(1.0, 1.0)))

print("cmath.phase(complex(1.0, 1.0)) =", cmath.phase(complex(1.0, 1.0)))

print("abs(complex(1.0, 1.0)) =", abs(complex(1.0, 1.0)))

print("cmath.sqrt(complex(25.0, 25.0)) =", cmath.sqrt(complex(25.0, 25.0)))

print("cmath.cos(complex(25.0, 25.0)) =", cmath.cos(complex(25.0, 25.0)))

print()

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

**Листинг 11. К6\_2\_2**

**'''**

К6\_2. Техника работы с числами math

**'''**

import math

def getsin(x):

multiplier = 1

result = 0

for i in range(1, 20, 2):

result += multiplier \* pow(x, i) / math.factorial(i)

multiplier \*= -1

return result

def main():

print("Арифметические функции")

print("math.pow(3, 2) =", math.pow(3, 2))

print("math.pow(9, 0.5) =", math.pow(9, 0.5))

print("math.sqrt(9) =", math.sqrt(9))

print("math.factorial(5) =", math.factorial(5))

print("sin(pi/2) =", getsin(math.pi / 2))

print()

print("Тригонометрические функции")

print("math.sin(math.pi/4) =", math.sin(math.pi / 4))

print("math.cos(math.pi) =", math.cos(math.pi))

print("math.tan(math.pi/6) =", math.tan(math.pi / 6))

print("math.hypot(12,5) =", math.hypot(12, 5))

print("math.atan(0.5773502691896257) =", math.atan(0.5773502691896257))

print("math.asin(0.7071067811865476) =", math.asin(0.7071067811865476))

print()

print("Гиперболические функции")

print("math.asinh(11.548739357257746) =", math.asinh(11.548739357257746))

print("math.acosh(11.591953275521519) =", math.acosh(11.591953275521519))

print("math.atanh(0.99627207622075) =", math.atanh(0.99627207622075))

print("math.sinh(math.pi) =", math.sinh(math.pi))

print("math.cosh(math.pi) =", math.cosh(math.pi))

print()

print("Логарифмические функции")

print("math.log(148.41315910257657) =", math.log(148.41315910257657))

print("math.log(148.41315910257657, 2) =", math.log(148.41315910257657, 2))

print("math.log(148.41315910257657, 10) =", math.log(148.41315910257657,10))

print("math.log(1.0000025) =", math.log(1.0000025))

print("math.log1p(0.0000025) =", math.log1p(0.0000025))

print("math.exp(5) =", math.exp(5))

print("math.e\*\*5 =", math.e \*\* 5)

print()

print("Округление:")

print("math.ceil(1.001) =", math.ceil(1.001))

print("math.floor(1.001) =", math.floor(1.001))

print("math.factorial(10) =", math.factorial(10))

print("math.gcd(10,125) =", math.gcd(10, 125))

print("math.trunc(1.001) =", math.trunc(1.001))

print("math.trunc(1.999) =", math.trunc(1.999))

print()

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

## Техника работы со строками

**Листинг 12. К7\_1\_1**

**'''**

К7\_1. Техника работы со строками;

Задание 1.

https://stepik.org/lesson/201702/step/5?unit=175778

С клавиатуры вводятся строки, последовательность заканчивается точкой.

Выведите буквы введенных слов в верхнем регистре, разделяя их пробелами.

**'''**

a = input()

while a != ".":

print(" ".join(a.upper()))

a = input()

**Листинг 13. К7\_1\_2**

**'''**

Задание 2. https://stepik.org/lesson/201702/step/8?unit=175778

Известно, что для логина часто не разрешается использовать строки содержащие пробелы.

Но пользователю нашего сервиса особенно понравилась какая-то строка.

Замените пробелы в строке на символы нижнего подчеркивания, чтобы строка

могла сгодиться для логина. Если строка состоит из одного слова, менять ничего не нужно.

**'''**

string = str(input())

string = string.replace(' ', '\_')

print(string))

**Листинг 14. К7\_1\_3**

**"""**

Задание 3. https://stepik.org/lesson/201702/step/9?unit=175778

Уберите точки из введенного IP-адреса. Выведите сначала четыре числа через пробел,

а затем сумму получившихся чисел.

**"""**

string = input().split('.')

print(\*string)

s = 0

for i in string:

s += int(i)

print(s)

**Листинг 15. К7\_1\_4**

**"""**

Задание 4. https://stepik.org/lesson/201702/step/14?unit=175778

Программист логирует программу, чтобы хорошо знать,

как она себя ведет (эта весьма распространенная и важная практика).

Он использует разные типы сообщений для вывода ошибок (error),

предупреждений (warning), информации (info) или подробного описания (verbose).

Сообщения отличаются по внешнему виду. Назовем модификаторами такие символы,

которые отличают сообщения друг от друга, позволяя программисту понять, к какому

из типов относится сообщения. Модификаторы состоят из двух одинаковых символов

и записываются по разу в начале и в конце строки.

**"""**

while True:

b = input()

if b == ".":

break

elif "@@" in b:

print("ошибка")

elif "!!" in b:

print("предупреждение")

elif "//" in b:

print("информация")

elif "\*\*" in b:

print("подробное сообщение")

## Техника работы со списками

**Листинг 16. К8\_1\_1**

**"""**

Задание 1. https://pythontutor.ru/lessons/lists/problems/more\_than\_neighbours/

Задача «Больше своих соседей»

Дан список чисел. Определите, сколько в этом списке элементов, которые больше двух

своих соседей, и выведите количество таких элементов. Крайние элементы списка никогда не учитываются, поскольку у них недостаточно соседей.

**"""**

n = [int(i) for i in input().split()]

a = 0

for i in range(2,len(n)):

if n[i-2] < n[i-1] > n[i]:

a += 1

print(a)

**Листинг 17. К8\_1\_2**

**"""**

Задание 2. https://pythontutor.ru/lessons/lists/problems/num\_equal\_pairs/

Задача «Количество совпадающих пар»

Дан список чисел. Посчитайте, сколько в нем пар элементов, равных друг другу.

Считается, что любые два элемента, равные друг другу образуют одну пару, которую необходимо посчитать.

**"""**

n = [int(i) for i in input().split()]

a = 0

for i in range(len(n)-1):

for j in range(i + 1,len(n)):

if n[i] == n[j]:

a += 1

print(a)

**Листинг 18. К8\_2\_1**

**"""**

Задание 1. Array112. Дан массив A размера N.

Упорядочить его по возрастанию методом сортировки

простым обменом: просматривать массив, сравнивая его соседние элементы и меняя их местами, если левый элемент пары больше правого; повторить описанные действия N 1 раз. Для контроля за выполняемыми действиями

выводить содержимое массива после каждого просмотра.

Учесть, что при каждом просмотре количество анализируемых

пар можно уменьшить на 1.

**""**

from random import randint

n = int(input())

a = []

for i in range(n):

a.append(randint(1,100))

print(a)

for i in range(n-1):

for j in range(n-i-1):

if a[j] > a[j+1]:

a[j], a[j+1] = a[j+1], a[j]

print(a)

print(a)

**Листинг 19. К8\_2\_3**

**"""**

Задание 3. Array114. Дан массив A размера N. Упорядочить

его по возрастанию методом сортировки простыми вставками:

сравнить элементы A0 и A1 и, при необходимости меняя их

местами, добиться того, чтобы они оказались упорядоченными

по возрастанию; затем обратиться к элементу A2 и

переместить его в левую (уже упорядоченную) часть массива,

сохранив ее упорядоченность; повторить этот процесс для

остальных элементов, выводя содержимое массива после

обработки каждого элемента (от 1-го до N-1 го).

**"""**

a = []

n = int(input())

for i in range(n):

a.append(int(input()))

for i in range(n - 1):

for j in range(n - 1 - i):

MIN, MAX = min(a[j], a[j + 1]), max(a[j], a[j + 1])

a[j], a[j + 1] = MIN, MAX

print(a)

## Техника работы с циклом for и генераторами списков

**Листинг 19. К9\_2\_1**

**"""**

Задание 1. Array55. Дан целочисленный массив A размера N (<= 15). Переписать в новый целочисленный

массив B все элементы с нечетными порядковыми номерами (1, 3, ...) и вывести размер

полученного массива B и его содержимое. Условный оператор не использовать.

**"""**

import random

n = int(input())

a = [random.randint(1, 20) for x in range(n)]

print('Размер полученного массива A:',len(a))

print(a)

b = [a[i] for i in range(1,len(a),2)]

print('Размер полученного массива B:',len(b))

print(b)

**Листинг 20. К9\_2\_2**

**'''**

Задание 2. Array57. Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B

того же размера вначале все элементы исходного массива с четными номерами,

а затем — с нечетными:

A[0], A[2], A[4], A[6], ..., A[1], A[3], A[5], ... .

Условный оператор не использовать.

**'''**

n = int(input())

a = [int(input()) for \_ in range(n)]

b = []

for i in range(0,n,2):

b.append(a[i])

for i in range(1,n,2):

b.append(a[i])

print(\*b)

**Листинг 21. К9\_2\_3**

**"""**

Задание 3. Array58. Дан массив A размера N. Сформировать новый массив B того же размера по

следующему правилу: элемент B[K] равен сумме элементов массива A с номерами от 0 до K.

**"""**

n = int(input())

a = [int(input()) for \_ in range(n)]

b=[]

summ = 0

for k in range(0,len(a)):

summ += a[k]

b.append(summ)

print(\*b)

## Техника работы с функциями

**Листинг 22. К10\_1\_2**

**'''**

Задание 2. Func6. Описать функцию SumRange(A, B) целого типа, находящую сумму всех целых

чисел от A до B включительно (A и B — целые). Если A > B, то функция возвращает 0.

С помощью этой функции найти суммы чисел от A до B и от B до C, если даны числа A, B, C.

**'''**

def SumRange(a, b):

summ = 0

for i in range(a, b+1):

summ += i

return summ

print("Введите числа:")

a = [int(x) for x in input().split()]

lna = len(a)

summ = 0

for i in range(1, lna):

summ += SumRange(a[i-1], a[i])

print(summ)

**Листинг 23. К10\_1\_3**

**'''**

Задание 3. Func10. Описать функцию IsSquare(K) логического типа, возвращающую True, если целый параметр K (> 0) является квадратом некоторого целого числа, и False в противном случае. С ее помощью найти количество квадратов в наборе из 10 целых

положительных чисел.

**'''**

import math

def IsSquare(k):

return math.sqrt(k).is\_integer()

i = 0

count = 0

while i != 10:

num = int(input())

print(IsSquare(num))

i += 1

**Листинг 24. К10\_2\_2**

**"""**

Задание 2. Использовать map, lambda. Квадраты в обратном порядке. Числа вводятся до точки. Через пробел выведите эти числа в обратном порядке, возводя их в квадрат.

**"""**

a = [int(x)\*\*2 for x in iter(input, '.')]

b = list(map(lambda x: print(x, end=' '), a[::-1]))

print()

**Листинг 25. К10\_2\_3**

**'''**

Задание 3. Использовать lambda, filter.

Array55. Дан целочисленный массив A размера N (<= 15). Переписать в новый целочисленный

массив B все элементы с нечетными порядковыми номерами (1, 3, ...) и вывести размер

полученного массива B и его содержимое. Условный оператор не использовать.

**'''**

n = int(input("Введите размер списка: "))

a = [int(input(f"{x}| ")) for x in range(n)]

b = list(filter(lambda x: x%2, a))

print("\n")

print("Список B:", \*b, sep=" ")

print("Размер списка b:", len(b))

**Листинг 26. К10\_2\_4**

'''

Задание 4.

Быстрая инициализация. Программа получает на вход три числа через пробел — начало и конец

диапазона, а также степень, в которую нужно возвести каждое число из диапазона. Выведите числа получившегося списка через пробел.

'''

a = int(input("Начало: "))

b = int(input("Конец: "))

pow = int(input("Степень: "))

arr = [x\*\*pow for x in range(a, b+1)]

print(\*arr)

## Техника работы со словарями

**Листинг 27. К11\_1\_1**

**"""**

Задание 1. https://pythontutor.ru/lessons/dicts/problems/occurency\_index/

Задача «Номер появления слова»

Условие. В единственной строке записан текст. Для каждого слова из данного текста

подсчитайте, сколько раз оно встречалось в этом тексте ранее.

Словом считается последовательность непробельных символов идущих подряд, слова разделены

одним или большим числом пробелов или символами конца строки.

**"""**

d = dict()

i = 0

for key in input().split():

d[word] = d.get(word, 0) + 1

print(d[word] - 1, end = ' ')

print()

**Листинг 28. К11\_1\_2**

**"""**

Задание 2. https://pythontutor.ru/lessons/dicts/problems/permissions/

Задача «Права доступа»

Условие. В файловую систему одного суперкомпьютера проник вирус, который сломал контроль

за правами доступа к файлам. Для каждого файла известно, с какими действиями можно к

нему обращаться: запись W, чтение R, запуск X.

В первой строке содержится число N — количество файлов содержащихся в данной файловой

системе. В следующих N строчках содержатся имена файлов и допустимых с ними операций,

разделенные пробелами. Далее указано чиcло M — количество запросов к файлам. В последних

M строках указан запрос вида Операция Файл. К одному и тому же файлу может быть применено

любое колличество запросов.

Вам требуется восстановить контроль над правами доступа к файлам (ваша программа для

каждого запроса должна будет возвращать OK если над файлом выполняется допустимая

операция, или же Access denied, если операция недопустима.

**"""**

permissions = {}

n = int(input())

for i in range(n):

s = input().split()

permissions[s[0]] = s[1:]

for i in range(int(input())):

perm, file = input().split()

if perm == 'read':

perm = 'R'

if perm == 'write':

perm = 'W'

if perm == 'execute':

perm = 'X'

if perm in permissions[file]:

print('OK')

else:

print('Access denied')

**Листинг 29. К11\_1\_3**

**"""**

Задание 3.https://pythontutor.ru/lessons/dicts/problems/most\_frequent\_word/

Задача «Самое частое слово»

Условие. Дан текст: в первой строке задано число строк, далее идут сами строки.

Выведите слово, которое в этом тексте встречается чаще всего. Если таких слов несколько,

выведите то, которое меньше в лексикографическом порядке.

**"""**

dictt = {}

max = 0

for \_ in range(int(input())):

for word in input().split():

if word in dictt:

dictt[word] += 1

else:

dictt[word] = 1

if dictt[word] > max:

max = dictt[word]

for key, value in sorted(dictt.items()):

if value == max:

print(key)

break

**Листинг 30. К11\_2\_1**

**'''**

Задание 1. https://stepik.org/lesson/243394/step/4?unit=215740

Телефонная книга. Этап 1. Коля устал запоминать телефонные номера и заказал у Вас

программу, которая заменила бы ему телефонную книгу. Коля может послать программе

два вида запросов: строку, содержащую имя контакта и его номер, разделенные пробелом,

или просто имя контакта. В первом случае программа должна добавить в книгу новый номер,

во втором – вывести номер контакта. Ввод происходит до символа точки. Если введенное

имя уже содержится в списке контактов, необходимо перезаписать номер.

**'''**

dict = {}

for d in iter(input, '.'):

arr = d.split()

if len(arr) == 1:

print(dict[arr[0]])

else:

dict[arr[0]] = arr[1]

**Листинг 31. К11\_2\_2**

**'''**

Задание 2. https://stepik.org/lesson/243394/step/8?unit=215740

Телефонная книга. Этап 2. Коля понял, что у многих из его знакомых есть несколько

телефонных номеров и нельзя хранить только один из них. Он попросил доработать Вашу

программу так, чтобы можно было добавлять к существующему контакту новый номер или даже

несколько номеров, которые передаются через запятую. По запросу телефонного номера

должен выводиться весь список номеров в порядке добавления, номера должны разделяться

запятой. Если у контакта нет телефонных номеров, должна выводиться строка "Не найдено".

**'''**

dict = {}

a = input()

while a!='.':

a = a.replace(',','').split()

if len(a)>1:

if a[0] not in dict.keys():

dict[a[0]]=list(a[1:]) #значение по ключу "первое слово в строке" равно оставшимся словам в строке

else:

for i in range(1,len(a)):

dict[a[0]].append(a[i])

else:

if a[0] in dict.keys(): #если имя есть в списке ключей

print(\*dict.get(a[0]),sep=', ') #вывести значения по ключу

else:

print('Не найдено')

a = input()

**Листинг 32.K11\_2\_3**

**'''**

Задание 3. https://stepik.org/lesson/243394/step/13?unit=215740

Телефонная книга. Этап 3. Коле очень понравилась Ваша программа, однако он стал

замечать, что иногда в его телефонную книгу попадают номера в некорректном формате.

Чтобы не сохранять недействительные номера, он попросил Вас обрабатывать только номера,

соответствующие критериям:

- номер должен начинаться либо с +7, либо с 8 и состоять из 11 цифр.

- блоки цифр могут разделяться пробелами или дефисами.

- вторая, третья и четвертая цифры могут помещаться в скобки.

Если программа встречает некорректный номер, она должна его проигнорировать. В обратном

случае она должна привести номер к виду +7 (900) 800-70-60 и запомнить. Остальной

функционал программы остается без изменений.

**'''**

dict = {}

while True:

line = input()

if line != '.':

end\_name = line.find(' ')

if end\_name != -1:

name, numbers = line[:end\_name], line[end\_name+1:]

for number in numbers.split(','):

number = number.replace(' ', '').replace('-', '').replace('(', '').replace(')', '')

if number.startswith('+7') and len(number) == 12:

new\_number = number[:2] + ' (' + number[2:5] + ') ' + number[5:8] + '-' + number[8:10] + '-' + number[10:]

dict.setdefault(name, []).append(new\_number)

elif number.startswith('8') and len(number) == 11:

new\_number = '+7 (' + number[1:4] + ') ' + number[4:7] + '-' + number[7:9] + '-' + number[9:]

dict.setdefault(name, []).append(new\_number)

else:

print(', '.join(dict.get(line)) if line in dict else 'Не найдено')

else:

break

## 1.11 Техника работы с множествами

**Листинг 33. К12\_1\_1**

**'''**

Задание 1. https://pythontutor.ru/lessons/sets/problems/number\_of\_unique/

Задача «Количество различных чисел»

Условие. Дан список чисел. Определите, сколько в нем встречается различных чисел.

**'''**

print(len(set(input().split())))

**Листинг 34. К12\_1\_2**

**'''**

Задание 2. https://pythontutor.ru/lessons/sets/problems/number\_of\_coincidental/

Задача «Количество совпадающих чисел»

Условие. Даны два списка чисел. Посчитайте, сколько чисел содержится одновременно как

в первом списке, так и во втором.

**'''**

print(len(set(input().split()) & (set(input().split()))))

**Листинг 35. К12\_1\_3**

**'''**

Задание 3. https://pythontutor.ru/lessons/sets/problems/sets\_intersection/

Задача «Пересечение множеств»

Условие. Даны два списка чисел. Найдите все числа, которые входят как в первый,

так и во второй список и выведите их в порядке возрастания.

**'''**

print(\*sorted(set(input().split())&set(input().split()),key=int))

**Листинг 36. К12\_1\_4**

**'''**

Задание 4. https://pythontutor.ru/lessons/sets/problems/number\_of\_words/

Задача «Количество слов в тексте»

Условие. Дан текст: в первой строке записано число строк, далее идут сами строки.

Определите, сколько различных слов содержится в этом тексте.

Словом считается последовательность непробельных символов идущих подряд, слова разделены

одним или большим числом пробелов или символами конца строки.

**'''**

words = set()

for i in range(int(input())):

words.update(input().split())

print(len(words))

**Листинг 37. К12\_1\_5**

**'''**

Задание 5. https://pythontutor.ru/lessons/sets/problems/polyglotes/

Задача «Полиглоты»

Условие. Каждый из некоторого множества школьников некоторой школы знает некоторое

количество языков. Нужно определить сколько языков знают все школьники, и сколько языков

знает хотя бы один из школьников.

В первой строке задано количество школьников. Для каждого из школьников сперва записано

количество языков, которое он знает, а затем - названия языков, по одному в строке.

В первой строке выведите количество языков, которые знают все школьники. Начиная со

второй строки - список таких языков. Затем - количество языков, которые знает хотя бы

один школьник, на следующих строках - список таких языков. Языки нужно выводить в

лексикографическом порядке, по одному на строке.

**'''**

students = [{input() for j in range(int(input()))} for i in range(int(input()))]

known\_by\_everyone, known\_by\_someone = set.intersection(\*students), set.union(\*students)

print(len(known\_by\_everyone), \*sorted(known\_by\_everyone), sep='\n')

print(len(known\_by\_someone), \*sorted(known\_by\_someone), sep='\n')

**Листинг 38. К12\_2\_1**

**'''**

Задание 1. https://stepik.org/lesson/3380/step/3?unit=963

Простейшая система проверки орфографии может быть основана на использовании списка известных слов.

Если введённое слово не найдено в этом списке, оно помечается как "ошибка".

Попробуем написать подобную систему.

На вход программе первой строкой передаётся количество d известных нам слов, после чего

на d строках указываются эти слова.

Затем передаётся количество l строк текста для проверки, после чего l строк текста.

Выведите уникальные "ошибки" в произвольном порядке. Работу производите без учёта регистра.

**'''**

def getset():

n = int(input())

s =set()

while n:

n-=1

for i in input().split():

s.add(i.lower())

return s

dic = getset()

words = getset()

for word in words-dic:

print(word)

**Листинг 39. К12\_2\_2**

**'''**

Задание 2. (Л.Б.) Сессия

В файле disc.txt хранится перечень дисциплин, выносимых на сессию, например,

Теория алгоритмов

МДК.01.01

Основы экономики

...

В файле session.txt хранятся сведения о результатах сессии, например,

Грушников; П2-18; Теория алгоритмов; 5

Константинович; П2-18; Теория алгоритмов; 5

...

Студент считается сдавшим сессию, если у него сданы все предметы и нет оценки "2".

Студент считается "отличником", если у него все пятерки

Требуется сформировать множества:

- студентов, сдавших сессию

- студентов-отличников

- дисциплин, по которым нет задолженностей

Результат вывести в файл output.txt

**'''**

file1 = open("session.txt", "r")

file2 = open("disc.txt", "r")

list = file1.readlines()

session = [i.strip().split('; ') for i in list]

list = file2.readlines()

disc = [i.strip() for i in list]

file1.close()

file2.close()

names = []

for str in session:

if str[0] not in names:

names.append(str[0])

honour = set()

sdav = set()

predmet = {\*disc}

for i in range(len(names)):

count\_5 = 0

count\_2 = 0

for j in range(len(session)):

if(session[j][0] == names[i]):

if(session[j][3] == '5'):

count\_5 += 1

elif(session[j][3] == '2'):

count\_2 += 1

predmet.discard(session[j][2])

if(count\_5 == len(disc)):

honour.add(names[i])

if(count\_2 == 0):

sdav.add(names[i])

# Запись в новый файл output.txt

file3 = open("output.txt","w")

file3.write('Сдали:\n'+'\n'.join(names))

file3.write('\n..............\n')

file3.write('Отличники:\n'+'\n'.join(honour))

file3.write('\n..............\n')

file3.write('Предметы по которым нет задолжности:\n'+'\n'.join(predmet))

file3.close()

## 1.12 Техника работы с кортежами

**Листинг 40. К13\_1\_1**

**'''**

Задание 1. https://stepik.org/lesson/193753/step/4?unit=168148

Вывести чётные

Необходимо вывести все четные числа на отрезке [a; a \* 10].

**'''**

a =int(input())

print(tuple([i for i in range(a+a%2,a\*10+1,2)]))

**Листинг 41. К13\_1\_2**

**'''**

Задание 2. https://stepik.org/lesson/193753/step/5?unit=168148

Убывающий ряд.

С клавиатуры вводятся целые числа a > b. Выведите убывающую последовательность чисел

по одному числу в строке.

**'''**

print(\*tuple([i for i in range(int(input()), int(input()), -1)]), sep='\n')

1.13 Техника работы с файлами.

**Листинг 42. K14\_1\_1**

**'''**

Дана строка S и текстовый файл. Добавить строку S в конец файла.

**'''**

a = open('text.txt', 'a')

a.write(input()+'\n')

a.close()

**Листинг 43. K14\_1\_2**

**'''**

Дана строка S и текстовый файл. Заменить в файле все пустые строки на строку S.

**'''**

file =open('text2.txt', 'r')

file1 =open('text3.txt', 'a')

s = input()

for line in file:

if line == '\n':

file1.write(s + '\n')

else:

file1.write(line)

print(line)

file.close()

file1.close()

1.14 Техника работы с модулями

**Листинг 44. K15\_1\_1**

#Класс deque() модуля collections в Python.

#Двусторонняя очередь в Python.

#Класс collections.deque() это обобщение стеков и очередей и представляет собой

#двустороннюю очередь. Двусторонняя очередь deque() поддерживает

#поточно-ориентированные, эффективные по памяти операции добавления и

#извлечения элементов последовательности с любой стороны с примерно одинаковой

#производительностью O(1) в любом направлении.

#Списки поддерживают аналогичные операции, но они оптимизирован только для

#быстрых операций с последовательностями фиксированной длины и требуют затрат

#O(n) на перемещение памяти для операций pop(0) и insert(0, v), которые

#изменяют как размер, так и положение базового представления данных.

#import collections

#dq = collections.deque([iterable[, maxlen]])

#Возвращаемое значение:

#новый объект deque().

#Описание:

#Класс deque() модуля collections возвращает новый объект deque(),

#инициализированный слева направо данными из итерируемой последовательности iterable.

#При создании объекта очереди класс использует метод dq.append() для добавления

#элементов из итерации iterable. Если итерация не указана, новая очередь deque() будет пуста.

from collections import deque

dq = deque('abcd')

dq

print(dq)

print()

#Если аргумент maxlen не указан или равен None, количество хранимых записей в

#объекте deque() может увеличиваться до произвольной длины. В противном случае,

#объект deque() ограничивает количество хранимых элементов в своем контейнере

#максимальной длиной maxlen.

#При добавлении новых элементов, когда заполнение очереди deque() становится

#больше значения maxlen, избыточное количество элементов удаляется/сбрасывается

#с противоположного конца. Заполнение очереди на определенную длину

#обеспечивают функциональность, аналогичную команде bash tail в Unix.

#Такое поведение полезно для отслеживания транзакций и других пулов данных,

#где интерес представляют только самые последние изменения или действия.

#Атрибуты и методы класса Deque:

#Метод dq.append() добавляет x к правой стороне (в конец) контейнера deque().

dq.append('123')

print(dq)

print()

#Метод dq.appendleft() добавляет x к левой стороне (в начало) контейнера deque().

dq.appendleft('456')

print(dq)

print()

#Метод dq.copy() создает мелкую копию контейнера deque().

dq\_copy = dq.copy()

print(dq\_copy)

print()

#Метод dq.clear() удаляет все элементы из контейнера deque(),

#оставляя его длиной 0.

dq\_copy.clear()

print(dq\_copy)

print()

#Метод dq.count() подсчитывает количество элементовконтейнера

#deque(), равное значению x.

dq.append('456')

print(dq.count('456'))

print()

#Метод dq.extend() расширяет правую сторону (с конца) контейнера

#deque(), добавляя элементы из итерируемого аргумента iterable.

dq.extend('12gf')

print(dq)

print()

#Метод dq.extendleft() расширяет левую сторону (с начала) контейнера

#deque(), добавляя элементы из итерируемого аргумента iterable.

dq.extendleft('45zv')

print(dq)

print()

#Метод dq.index() вернет позицию (индекс) первого совпадения значения

#аргумента x в контейнере deque(), расположенного после необязательного

#аргумента start и до необязательного аргумента stop.

print(dq.index('4', 1))

print()

#Метод dq.insert() вставляет значение аргумента x в позицию i контейнера

#deque().

#Если вставка значение аргумента x приведет к тому, что ограниченный контейнер

#deque() выйдет за пределы maxlen, будет вызвано исключение IndexError.

dq.insert(2, 'dc')

print(dq)

#Метод dq.pop() удаляет и возвращает элемент с правой стороны (с конца)

#контейнера deque(). Если элементы отсутствуют, возникает ошибка IndexError.

dq.pop()

print(dq)

print()

#Метод dq.popleft() удаляет и возвращает элемент с левой стороны (с начала)

#контейнера deque(). Если элементы отсутствуют, возникает ошибка IndexError.

print(dq.popleft())

print(dq)

print()

#Метод dq.remove() удаляет первое вхождение значения value в контейнер

#deque(). Если значение value не найдено, возникает ошибка IndexError.

dq.remove('1')

print(dq)

print()

#Метод dq.reverse() разворачивает элементы контейнера deque() на месте

#и возвращает None.

dq.reverse()

print(dq)

print()

#Метод dq.rotate() разворачивает контейнер deque() на n шагов вправо. Если

#аргумент n имеет отрицательное значение, то разворачивает контейнер налево.

#Когда контейнер не пуст, вращение на один шаг вправо эквивалентно

#dq.appendleft(d.pop()), а вращение на один шаг влево эквивалентно

#dq.append(d.popleft()).

dq.rotate(3)

print(dq)

dq.rotate(-5)

print(dq)

print()

#Свойство dq.maxlen() возвращает максимальный размер maxlen контейнера deque(), если параметр maxlen не задан, то возвращает None.

#Пример использования

deq = deque()

x = input()

#добавлять в обратном порядке до 1-го вхождения ',' и в конце

#вывести содержимое

while x != '.':

deq.appendleft(x)

x= input()

print(deq)

**Листинг 45. K15\_1\_2**

#Подготовить инструкцию по использованию модулей Counter.

#класс collections.Counter() предназначен для удобных и быстрых подсчетов

#количества появлений неизменяемых элементов в последовательностях.

#rom collections import Counter

#cnt = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])

#dict(cnt)

#{'blue': 3, 'red': 2, 'green': 1}

#Синтаксис

import collections

#cnt = collections.Counter([iterable-or-mapping])

#Параметры:

#iterable-or-mapping - итерируемая последовательность или словарь.

#Возвращаемое значение:

#объект Counter().

#Описание:

#Класс Counter() модуля collections - это подкласс словаря dict

#для подсчета хеш-объектов (неизменяемых, каких как строки, числа, кортежи

#и т.д.).Это коллекция, в которой элементы хранятся в виде словарных ключей,

#а их счетчики хранятся в виде значений словаря.

#Счетчик может быть любым целочисленным значением, включая ноль или

#отрицательное число.

#Класс collections.Counter() похож на мультимножества в других языках

#программирования.

#Элементы считываются из итерируемой последовательности, инициализируются

#из другого словаря или счетчика Counter():

#from collections import Counter

# новый пустой счетчик

# cnt = Counter()

# новый счетчик из последовательности

# cnt = Counter('gallahad')

# новый счетчик из словаря

# cnt = Counter({'red': 4, 'blue': 2})

# новый счетчик из ключевых слов 'args'

# cnt = Counter(cats=4, dogs=8)

#Счетчики collections.Counter() имеют интерфейс словаря, за исключением

#того,

#что они возвращают 0 для отсутствующих элементов вместо вызова #исключения

#KeyError:

#cnt = Counter(['eggs', 'ham'])

#cnt['bacon']

# 0

#Установка счетчика в ноль не удаляет элементы из счетчика. Используйте

#инструкцию del, чтобы полностью удалить ключ счетчика:

# запись счетчика с нулевым счетом

#cnt['sausage'] = 0

# удаление счетчика с нулевым счетом

#del cnt['sausage']

#В качестве подкласса dict(), класс Counter() унаследовал возможность

#запоминания порядка вставки.

#Математические операции над объектами Counter() также сохраняют #порядок.

#Результаты упорядочены в соответствии с тем, когда элемент сначала

#встречается в левом операнде, а затем в порядке, в котором #встречается

#правыйоперанд.

#Атрибуты и методы класса Counter():

#Метод cnt.elements()

from collections import Counter

cnt = Counter(a=0, b=2, c=3,)

sorted(cnt.elements())

# ['b', 'b', 'c', 'c', 'c']

#Метод cnt.most\_common() возвращает список из n наиболее #распространенных

#элементов и их количество от наиболее распространенных до наименее.

#Если n опущено или None, метод cnt.most\_common() возвращает все #элементы

#в счетчике.Элементы с равным количеством упорядочены в порядке, в #котором они

#встречаются первыми:

from collections import Counter

Counter('Assistance').most\_common(2)

# [('s', 3), ('a', 2)]

#Метод cnt.subtract() вычитает элементы текущего счетчика cnt и #итерируемой

#последовательности или другого словаря или другого счетчика #Counter().

#Подобно методу словаря dict.update(), но вычитает количество #(значения

#ключей), а не заменяет их.

#Значения ключей как у счетчика так и у словаря могут быть нулевыми #или отрицательными.

from collections import Counter

c = Counter(a=0, b=4)

d = Counter(a=1, b=2)

c.subtract(d)

c

# Counter({'a': -1, 'b': 2})

#Метод cnt.update() складывает элементы текущего счетчика cnt и #итерируемой

#последовательности или другого словаря или другого счетчика #Counter().

#Подобно методу словаря dict.update(), но складывает количество #(значения ключей), а не заменяет их.

#Кроме того, ожидается, что итерация будет последовательностью #элементов,

#а не последовательностью двойных кортежей (key, value).

from collections import Counter

c = Counter(a=1, b=8, c=6)

d = Counter(a=1, b=2, c=3)

c.update(d)

c

# Counter({'a': 2, 'b': 10, 'c': 9, })

#Пример:

from collections import Counter

s1 ='aabbbcccdeff'

c1 = Counter(s1)

print("c1 :", c1)

#Счетчик, используемый в Списке для поиска частот всех его уникальных

#элементов списка

L1 =[1, 2, 1, 1, 4, 4, 4, 5, 6, 6, 3, 3, 0, 0]

t1 = Counter(L1)

print("t1 :", t1)

#c1 : Counter({'b': 3, 'c': 3, 'a': 2, 'f': 2, 'e': 1, 'd': 1})

#t1 : Counter({1: 3, 4: 3, 0: 2, 3: 2, 6: 2, 2: 1, 5: 1})

**Листинг 46. K15\_2\_1**

#Словарь со значениями по умолчанию.

#Класс defaultdict() модуля collections ни чем не отличается от #обычного

#словаря за исключением того, что по умолчанию всегда вызывается #функция,

#которая возвращает значение по умолчанию для новых значений. Другими

#словами Класс defaultdict() представляет собой словарь со значениями #по

#умолчанию.

#Параметры:

#default\_factory - тип данных или функция, которая возвращает значение #по умолчанию для новых значений.

#Описание:

#Класс defaultdict() модуля collections возвращает новый словарь-#подобный

#объект. Defaultdict является подклассом встроенного класса dict().

#Он переопределяет один метод и добавляет одну доступную для записи

#переменную экземпляра.

#Остальная функциональность такая же, как и для класса dict(), и здесь #она не описана.

#Первый аргумент предоставляет начальное значение для атрибута

#default\_factory. По умолчанию None.

#Все остальные аргументы обрабатываются так же, как если бы они были

#переданы конструктору dict(), включая ключевые аргументы.

#Дополнительный метод класса defaultdict():

#\_\_missing\_\_(key):

#Если атрибут default\_factory равен None, то это вызывает исключение

#KeyError с ключом key в качестве аргумента.

#Если default\_factory не равен None, то метод \_\_missing\_\_() вызывается #без

#аргументов для предоставления значения по умолчанию для данного #ключа,

#это значение вставляется в словарь для ключа key.

#Если вызов default\_factory вызывает исключение, это исключение

#распространяется без изменений.

#Метод \_\_missing\_\_() вызывается методом \_\_getitem\_\_() класса dict(), #когда

#запрошенный ключ key не найден.

#Все, что он возвращает или поднимает,

#затем возвращается или вызывается методом \_\_getitem\_\_().

#Обратите внимание, что метод \_\_missing\_\_() не вызывается ни для каких

#операций, кроме как \_\_getitem\_\_().

#Это означает, что [метод defaultdict.get()],

#как и обычные словари, будет возвращать None - как значение по #умолчанию,

#а не использовать default\_factory.

#Переменная экземпляра класса defaultdict():

#default\_factory:

#Этот атрибут используется методом \_\_missing \_\_(). Он инициализируется #из

#первого аргумента, переданного в конструктор,

#если он есть или устанавливается в None, если он отсутствует.

#Пример:

from collections import defaultdict

s = 'Cucumber'

d = defaultdict(int)

for k in s:

d[k] += 1

sorted(d.items())

# [('c', 2), ('m', 1), ('b', 1),('e', 1),'r', 1), ('u', 2)]

1.15 Техника работы с классами.

**Листинг 47. K16\_1**

#Задание 1. Создание класса

#Задание 2. Создание объекта

#Задание 3. Функция init

#Задание 4. Методы объектов

#Задание 5. Параметр self

#Задание 6. Изменение свойств объекта

#Задание 7. Удалить свойства объекта

#Задание 8. Удаление объектов

#Для того, чтобы создать класс, используйте ключевое слово class.

#Создадим класс с именем MyClass и свойством x:

class Myclass:

x = 5

#Создание объекта

#Теперь мы можем использовать класс под названием myClass для создания #объектов.

#Создадим объект под названием p1, и выведем значение x:

p1 = Myclass()

print(p1.x)

#Функция init

#Приведенные выше примеры — это классы и объекты в их простейшей форме и

#не очень полезны в приложениях.

#Чтобы понять значение классов, нам нужно понять встроенную функцию #\_\_init\_\_.

#У всех классов есть функция под названием \_\_init\_\_(), которая всегда #выполняется при

#создании объекта. Используйте функцию \_\_init\_\_() для добавления #значений #свойствам объекта

#или других операций, которые необходимо выполнить, при создании объекта.

#Для создания класса под названием Person, воспользуемся функцией #\_\_init\_\_(),

#что бы добавить значения для имени и возраста:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# Объекты также содержат методы. Методы в объектах — это

# функции, принадлежащие объекту.

# Параметр self

# Его не обязательно называть self, вы можете называть его как #хотите,

# но он должен быть первым параметром любой функции в классе.

def myFunc(self):

print('Hello, my name is ' + self.name)

p1 = Person('Vasya', 36)

print(p1.name)

print(p1.age)

#вызов метода

p1.myFunc()

#Изменение свойств объекта

p1.age = 40

print(p1.age)

#Удалить свойства объекта

del p1.age

#print(p1.age) выдаст ошибку

#Удаление объектов

del p1

**Листинг 48. K16\_2**

#Создание классов

#Оператор class создает новое определение класса. Имя класса сразу следует

#за ключевым словом class, после которого ставиться двоеточие:

#Пример создания класса

class Employee:

'''Базовый класс для всех сотрудников'''

# кол-во сотрудников

emp\_count = 0

# это конструктор класса

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.emp\_count += 1

# методы класса

def display\_count(self):

print('Всего сотрудников : %d' % Employee.emp\_count)

def display\_employee(self):

print('Имя: {}, зарплата: {}'.format(self.name, self.salary))

#Создание экземпляров класса

# Это создаст первый объект класса Employee

emp1 = Employee('Andrey', 2000)

print(emp1)

#Доступ к атрибутам

#Получите доступ к атрибутам класса, используя оператор '.' после объекта #класса.

#Доступ к классу можно получить используя имя переменой класса:

emp2 = Employee('Maria', 5000)

emp1.display\_employee()

emp2.display\_employee()

print('All people: %d' % Employee.emp\_count)

#Вместо использования привычных операторов для доступа к атрибутам вы #можете

#использовать эти функции:

#getattr(obj, name [, default]) — для доступа к атрибуту объекта.

#hasattr(obj, name) — проверить, есть ли в obj атрибут name.

#setattr(obj, name, value) — задать атрибут. Если атрибут не существует, #он

# будет создан.

#delattr(obj, name) — удалить атрибут

print()

setattr(emp1, 'age', 8) #устанавливает атрибут 'age' на 8

print(emp1.age)

print(hasattr(emp1, 'age')) # возвращает true если атрибут 'age' существует

print(getattr(emp1, 'age')) # возвращает значение атрибута 'age'

delattr(emp1, 'age') # удаляет атрибут 'age'

#Встроенные атрибуты класса

#Каждый класс Python хранит встроенные атрибуты, и предоставляет к ним

#доступ через оператор ., как и любой другой атрибут

#\_\_dict\_\_ — словарь, содержащий пространство имен класса.

#\_\_doc\_\_ — строка документации класса. None если, документация отсутствует.

#\_\_name\_\_ — имя класса.

#\_\_module\_\_ — имя модуля, в котором определяется класс. Этот атрибут #\_\_main\_\_

# в интерактивном режиме.

#\_\_bases\_\_ — могут быть пустые tuple, содержащие базовые классы, в порядке

# их появления в списке базового класса.

print()

print('Employee.\_\_doc\_\_:', Employee.\_\_doc\_\_)

print('Employee.\_\_name\_\_:', Employee.\_\_name\_\_)

print('Employee.\_\_module\_\_:', Employee.\_\_module\_\_)

print('Employee.\_\_bases\_\_:', Employee.\_\_bases\_\_)

print('Employee.\_\_dict\_\_:', Employee.\_\_dict\_\_)

#Уничтожение объектов (сбор мусора)

#Python автоматически удаляет ненужные объекты (встроенные типы или #экземпляры

#классов), чтобы освободить пространство памяти. С помощью процесса #‘Garbage

#Collection’ Python периодически восстанавливает блоки памяти, которые #больше

#не используются.

#Сборщик мусора Python запускается во время выполнения программы и тогда,

#когда количество ссылок на объект достигает нуля. С изменением количества

#обращений к нему, меняется количество ссылок.

#Когда объект присваивают новой переменной или добавляют в контейнер #(список,

#кортеж, словарь), количество ссылок объекта увеличивается. Количество #ссылок

#на объект уменьшается, когда он удаляется с помощью del, или его ссылка

#выходит за пределы видимости. Когда количество ссылок достигает нуля, #Python

#автоматически собирает его.

a = 40 # создали объект <40>

b = a # увеличивает количество ссылок <40>

c = [b] # увеличивает количество ссылок <40>

del a # уменьшает количество ссылок <40>

b = 100 # уменьшает количество ссылок <40>

c[0] = -1 # уменьшает количество ссылок <40>

#классом можно реализовать специальный метод \_\_del\_\_(),

#называемый деструктором.

class Point:

def \_\_init\_\_(self, x=0, y=0):

self.x = x

self.y = y

def \_\_del\_\_(self):

class\_name = self.\_\_class\_\_.\_\_name\_\_

print('{} destroyed'.format(class\_name))

pt1 = Point()

pt2= pt1

pt3 = pt1

print()

print(id(pt1), id(pt2), id(pt3))

del pt1

del pt2

del pt3

**Листинг 49. K16\_3**

#Наследование класса

#Вместо того, чтобы начинать с нуля, вы можете создать класс, на основе #уже

#существующего. Укажите родительский класс в круглых скобках после имени #нового

#класса.

#Класс наследник наследует атрибуты своего родительского класса. Вы можете

#использовать эти атрибуты так, как будто они определены в классе #наследнике.

#Он может переопределять элементы данных и методы родителя.

#Пример наследования класса в Python

class Parent:

parent\_attr = 100

def \_\_init\_\_(self):

print('Вызов родительского конструктора')

def parent\_method(self):

print('Вызов родительского метода')

def set\_attr(self, attr):

Parent.parent\_attr = attr

def get\_attr(self):

print('Атрибут родителя: {}'.format(Parent.parent\_attr))

def my\_method(self):

print('Вызов родительского метода')

class child(Parent):

def \_\_init\_\_(self):

print('Вызов конструктора класса наследника')

def child\_method(self):

print('Вызов метода класса наследника')

def my\_method(self):

print('Вызов метода наследника')

c = child()

c.child\_method()

c.parent\_method()

c.set\_attr(300)

c.get\_attr()

print()

#Вы можете использовать функции issubclass() или isinstance() для

#проверки отношений двух классов и экземпляров.

#Логическая функция issubclass(sub, sup) возвращает значение True,

#если данный подкласс sub действительно является подклассом sup.

#Логическая функция isinstance(obj, Class) возвращает True, если obj

#является экземпляром класса Class или является экземпляром подкласса #класса.

#Переопределение методов

#Вы всегда можете переопределить методы родительского класса. В вашем #подклассе

#могут понадобиться специальные функции. Это одна из причин переопределения

#родительских методов.

c.my\_method()

print()

#Популярные базовые методы

#В данной таблице перечислены некоторые общие функции. Вы можете

#переопределить их в своих собственных классах.

# 1)\_\_init\_\_(self [, args...]) — конструктор (с любыми необязательными

#аргументами)

# obj = className(args)

# 2)\_\_del\_\_(self) — деструктор, удаляет объект

#del obj

# 3)\_\_repr\_\_(self) — оценочное строковое представление

#repr(obj)

# 4)\_\_str\_\_(self) — печатное строковое представление

#str(obj)

#Пример

class Vector:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_str\_\_(self):

return 'Vector ({}, {})'.format(self.a, self.b)

def \_\_add\_\_(self, other):

return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2, 10)

v2 = Vector(5, -2)

print(v1 + v2)

print()

#Приватные методы и атрибуты класса

#Атрибуты класса могут быть не видимыми вне определения

#класса. Вам нужно указать атрибуты с \_\_ вначале, и эти атрибуты не будут

#вызваны вне класса.

#Пример приватного атрибута

class JustCounter:

\_\_secret\_count = 0

def count(self):

self.\_\_secret\_count += 1

print(self.\_\_secret\_count)

counter = JustCounter()

counter.count()

counter.count()

#print(counter.\_\_secret\_count) выдаст ошибку

#Вы можете получить доступ к таким атрибутам, так object.\_className\_\_attrName.

print(counter.\_JustCounter\_\_secret\_count)

**Листинг 50. K16\_4**

#Задание 1. Придумать собственный класс

#Задание 2. Неформально описать функционал класса

#Задание 3. Реализовать класс в модуле

#Задание 4. Разработать скрипт для демонстрации работы с классом #(импортировать

#модуль, создать экземпляры, вызвать методы)

class Machine:

'''

Общая характеристика машин

'''

# кол-во машин

\_\_all\_types = 0

def \_\_init\_\_(self, doors, wheels, atype, body):

self.doors = doors

self.wheels = wheels

self.atype = atype

self.body = body

Machine.\_\_all\_types += 1

#Общие сведения

def machine\_info(self):

print('Тип: {}'.format(self.atype))

print('Кузов: {}'.format(self.body))

print('Кол-во дверей: {}'.format(self.doors))

print('Кол-во: {}'.format(self.wheels))

print()

def what\_is\_this(self):

print('Machine')

print()

#Экземпляр №1

machine1 = Machine(4, 4, 'w2001', 'car')

machine1.machine\_info()

print(machine1.\_Machine\_\_all\_types)

#Экземпляр #2

machine2 = Machine(2, 3, 'w21001', 'car')

machine2.machine\_info()

print(machine1.\_Machine\_\_all\_types)

class Motorcycle(Machine):

'''

Общая характеристика мотоциклов

'''

def \_\_init\_\_(self, doors, wheels, atype, body, engine, gas\_tank\_volume, engine\_power, max\_speed, acceliration, fuel\_consuption):

self.doors = doors

self.wheels = wheels

self.atype = atype

self.body = body

self.engine = engine

self.gas\_tank\_volume = gas\_tank\_volume

self.engine\_power = engine\_power

self.max\_speed = max\_speed

self.acceliration = acceliration

self.fuel\_consuption = fuel\_consuption

#подробное описание

def motorcycle\_info(self):

print('Тип: {}'.format(self.atype))

print('Кузов: {}'.format(self.body))

print('Кол-во дверей: {}'.format(self.doors))

print('Кол-во: {}'.format(self.wheels))

print('Двигатель: {}'.format(self.engine))

print('Объем бензобака: {}'.format(self.gas\_tank\_volume))

print('Мощность двигателя: {}'.format(self.engine\_power))

print('Макс. скорость: {}'.format(self.max\_speed))

print('Разгон до 100: {}'.format(self.acceliration))

print('Расход топлива: {}'.format(self.fuel\_consuption))

print()

def what\_is\_this(self):

print('Motorcycle')

print()

#Экземпляр №3

motorcycle1 = Motorcycle('None', 2, 'w21002', 'motorcycle', 'v6', '20', '120 hp', '180 km/h', '3.01 s', 3.5)

motorcycle1.machine\_info()

motorcycle1.motorcycle\_info()

class Cars(Machine):

'''

Общая характеристика автомобилей

'''

def \_\_init\_\_(self, doors, wheels, atype, body, engine, gas\_tank\_volume, engine\_power, max\_speed, acceliration, fuel\_consuption):

self.doors = doors

self.wheels = wheels

self.atype = atype

self.body = body

self.engine = engine

self.gas\_tank\_volume = gas\_tank\_volume

self.engine\_power = engine\_power

self.max\_speed = max\_speed

self.acceliration = acceliration

self.fuel\_consuption = fuel\_consuption

#подробное описание

def car\_info(self):

print('Тип: {}'.format(self.atype))

print('Кузов: {}'.format(self.body))

print('Кол-во дверей: {}'.format(self.doors))

print('Кол-во: {}'.format(self.wheels))

print('Двигатель: {}'.format(self.engine))

print('Объем бензобака: {}'.format(self.gas\_tank\_volume))

print('Мощность двигателя: {}'.format(self.engine\_power))

print('Макс. скорость: {}'.format(self.max\_speed))

print('Разгон до 100: {}'.format(self.acceliration))

print('Расход топлива: {}'.format(self.fuel\_consuption))

print()

def what\_is\_this(self):

print('car')

print()

#Экземпляр №4

car1 = Cars(5, 6, 'w2002', 'SUV', 'v8', '40', '160 hp', '195 km/h', '4.23 s', 4.5)

car1.machine\_info()

car1.car\_info()

#Нам больше не нужна харатеристика "объём бака" у car1

del car1.gas\_tank\_volume

#Теперь при обращение и при печати будет выдавать ошибку

#print(car1.gas\_tank\_volume) ошибка

#В подклассах можно изменять методы классов

machine1.what\_is\_this()

car1.what\_is\_this()

#Мы можем увидеть, что представляет собой тот или иной класс и его названия

#с помощью встроенных атрибутов класса

print('Machine documentation:', Machine.\_\_doc\_\_)

print('Machine name:', Machine.\_\_name\_\_)

# **Раздел 2. Техника решения задач с использованием библиотек**

## Установка и настройка среды JetBrains PyCharm

PyCharm - это кросс-платформенная среда разработки.

Для скачивания PyCharm переходим по этой ссылке <https://www.jetbrains.com/ru-ru/pycharm/>. Для скачивания доступно две версии: профессиональная(платная) и версия для сообщества(бесплатная). Скачиваем бесплатную.

Запускаем скачанный .exe файл, в установщике кликаем «Next»:

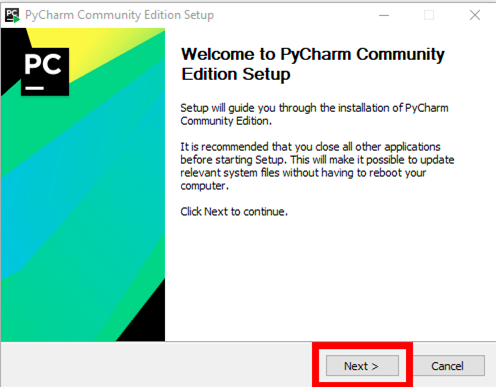


Рисунок 8. Установщик PyCharm



Рисунок 9. Место установки PyCharm



Рисунок 10. Стартовое окно PyCharm

В следующем окне будет два поля. В первом укажите, где будет располагаться новый проект. Второе поле должно быть заполненным

поумолчанию. Там содержится путь к установленному ранее Python.

Дальше откроется окно самого редактора. Теперь вы готовы начать писать программы на Python!

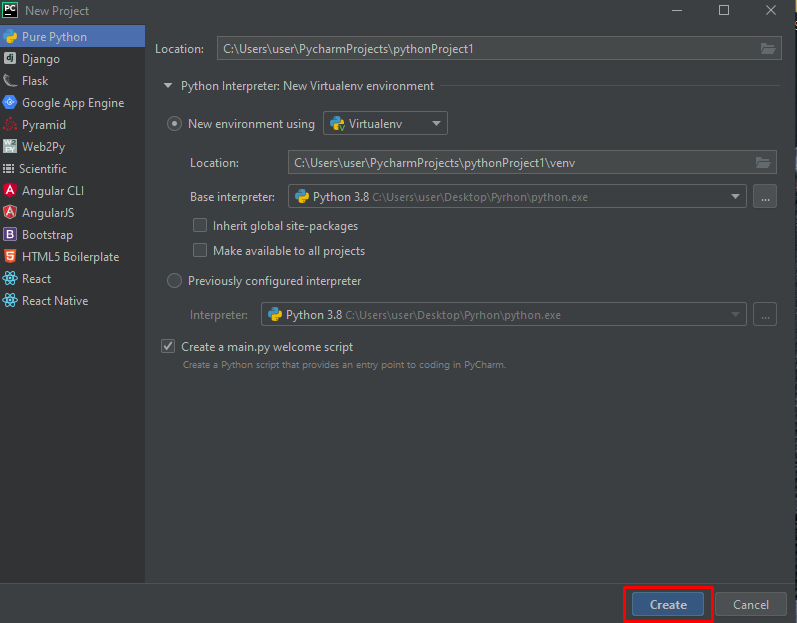


Рисунок 11. Создание нового файла

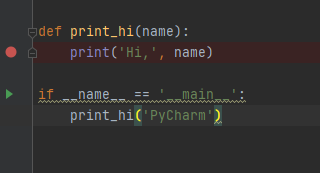


Рисунок 12. Написание программы

Теперь запустим ее. Для этого нужно сделать следущее:

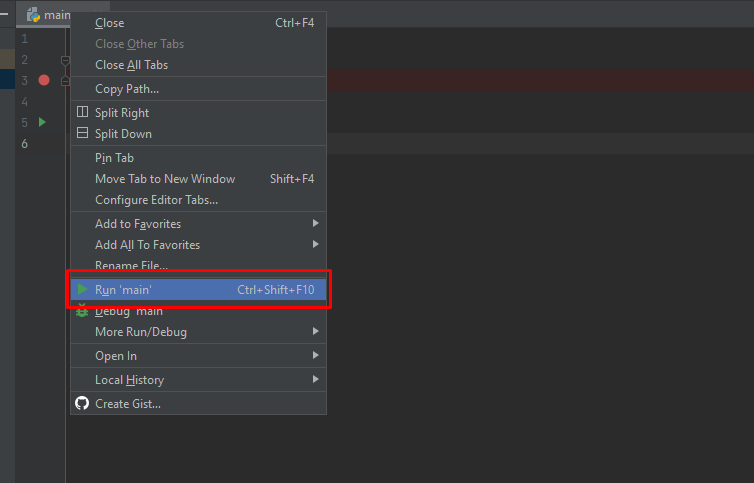


Рисунок 13. Запуск программы

Нажимаем на название нашей программы ПКМ, далее нажимаем на кнопку ‘run’ и видим как в консоль выводится результат.

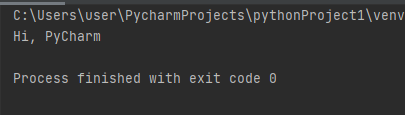


Рисунок 14. Результат нашей программы.

## 2.2 Техника работы с базами данных

Python SQLite поставляется вместе с любой установленной версией Python, поэтому нет необходимости устанавливать его с помощью pip. Чтобы импортировать его, поскольку мы будем использовать Python3, мы импортируем модуль sqlite3.

После завершения кода в той же папке где находится python-file появится файл с БД, но просто так его не посмотреть

Для того чтобы увидеть БД, необходимо будет скачать DB Browser (SQLite). Переходим по ссылке и скачиваем программу <https://sqlitebrowser.org/dl/>.

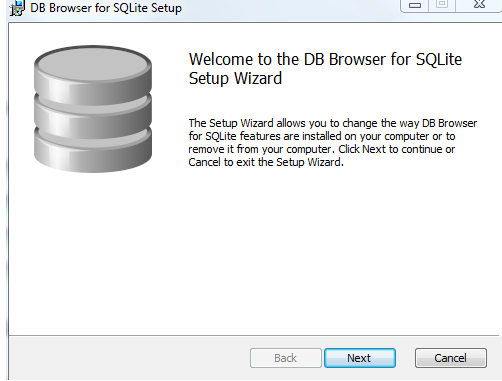


Рисунок 15. Установщик DB Browser

После установки, открываем DB Browser для работы с ним.

Рисунок 16. Окно DB Browser

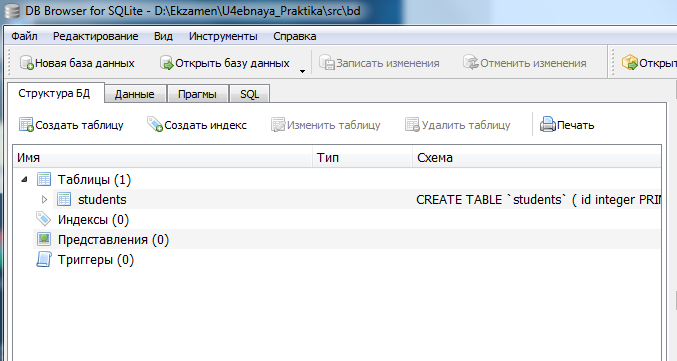
В открывшееся окно DB Browser’a перетаскиваем файл с БД.

Рисунок. 17. Таблица students в DB Browser

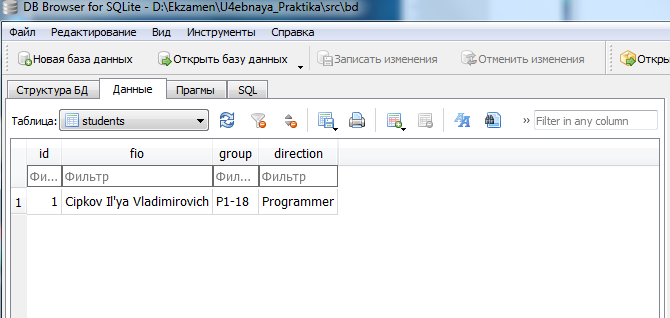
Чтобы просмотреть содержимое БД необходимо нажать на кнопку «Данные».

Рисунок. 18. Данные таблицы students

Видно, что в таблицу занеслись корректные данные.

2.3 Техника работы с библиотекой tkinter

Библиотека Tkinter установлена в Python в качестве стандартного модуля, поэтому нам не нужно устанавливать что-либо для его использования. Чтобы импортировать его, пропишем import tkinter. В качестве примера для демонстрации tkinter будет приведена программа, которая принимает данные из формы и добавляет их в текстовый файл, также в программе есть возможность очищать весь файл полностью.

**Листинг 51. Управление Заказами БД.**

from tkinter import \*

from tkinter import messagebox

from tkinter import filedialog

import sqlite3

TEXT\_BUTTON\_DEL = "Удалить"

TEXT\_CREATE\_FILE = "Создать базу данных"

TEXT\_OPEN\_FILE = "Открыть базу данных"

nameDB = "database.db"

BUTTON\_COLOR = "#E6E6FA"

WINDOW\_COLOR = "#FF8C00"

ACT\_BUTTON\_COLOR = "#B0C4DE"

LABEL\_BG = "#FF8C00"

class Database:

def \_\_init\_\_(self, nameDB):

self.db = sqlite3.connect(nameDB)

self.cursor = self.db.cursor()

def create\_table(self, table):

self.cursor.execute(f"CREATE TABLE IF NOT EXISTS '{table}' (\

order\_id INTEGER, \

order\_name TEXT, \

customer\_name TEXT, \

price INTEGER)")

def save(self):

self.db.commit()

def get\_table(self, table):

return [i for i in self.cursor.execute(f"SELECT \* FROM {table}")]

def delete\_order\_by\_id():

db = sqlite3.connect(nameDB)

cursor = db.cursor()

db.execute('DELETE FROM {0} WHERE order\_id = {1}'.format("Orders", window.id\_order\_del.get()))

db.commit()

def insert\_data():

db = Database(nameDB)

db.create\_table("Orders")

db.cursor.execute("INSERT INTO orders (order\_id, order\_name, customer\_name, price) VALUES (?,?,?,?)",

(window.id\_order.get(), window.name\_order.get(), window.customer.get(), window.price.get()))

db.save()

def creat\_db(): # диалоговое окно, где в поле имя файлов надо ввести имя файла которое вы хотите создать

global nameDB

path\_db = filedialog.asksaveasfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*")))

db = sqlite3.connect(path\_db)

nameDB = path\_db

def open\_db(): # диалоговое окно, где пользователь выбирает какой файл ему нужно открыть

global nameDB

path\_db = filedialog.askopenfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*"))) # op является путём и его надо передать в функции добавления и удаления из файла

db = sqlite3.connect(path\_db)

nameDB = path\_db

class Window:

def \_\_init\_\_(self):

self.window = Tk()

self.window.title("Управление заказами")

self.window.geometry("640x250")

self.window["bg"] = WINDOW\_COLOR

self.id\_order = StringVar() # Присваивает строку из виджета поля для ввода текста

self.name\_order = StringVar()

self.customer = StringVar()

self.price = StringVar()

self.id\_order\_del = StringVar()

self.top\_menu()

self.main\_menu()

def top\_menu(self):

self.menu = Menu(self.window) # иницилизирую объект меню для секций

self.menu.add\_cascade(label=TEXT\_CREATE\_FILE, command=creat\_db)

self.menu.add\_cascade(label=TEXT\_OPEN\_FILE, command=open\_db)

self.menu.add\_cascade(label="О программе", command=lambda: messagebox.showinfo("О программе", "Разработку ведет Батраков Данила.")) # Меню о программе добавляется на панель меню

self.window.config(menu=self.menu) # доступ к атрибуту объекта после его инициализации

def create\_label(self, text\_label, x\_label, y\_label): # Функция принимает строку и координаты

Label(text=text\_label, bg=LABEL\_BG).place(x=x\_label, y=y\_label) # Отображает текст в окне по переданным координатам

def main\_menu(self):

self.create\_label("ID заказа: ", 0, 3) # Отображаем текст хранящейся в переменной

Entry(width=20, textvariable=self.id\_order).place(x=0, y=20) # Определим элемент Entry который представляет собой поле для ввода текста

self.create\_label("Заказ: ", 160, 3)

Entry(width=20, textvariable=self.name\_order).place(x=160, y=20)

self.create\_label("Заказчик:", 320, 3)

Entry(width=20, textvariable=self.customer).place(x=320, y=20)

self.create\_label("Цена: ", 480, 3)

Entry(width=20, textvariable=self.price).place(x=480, y=20)

Button(self.window, text="Добавить", command=insert\_data, bg=BUTTON\_COLOR, activebackground=ACT\_BUTTON\_COLOR).place(x=542, y=45)

self.create\_label("Удаление по ID: ", 480, 120)

Entry(width=20, textvariable=self.id\_order\_del, ).place(x=480, y=140)

Button(self.window, text=TEXT\_BUTTON\_DEL, width=10, command=delete\_order\_by\_id, bg=BUTTON\_COLOR, activebackground=ACT\_BUTTON\_COLOR).place(x=525, y=165)

window = Window()

mainloop() # Отображает главное окно со всеми виджетами

Форма имеет вид:

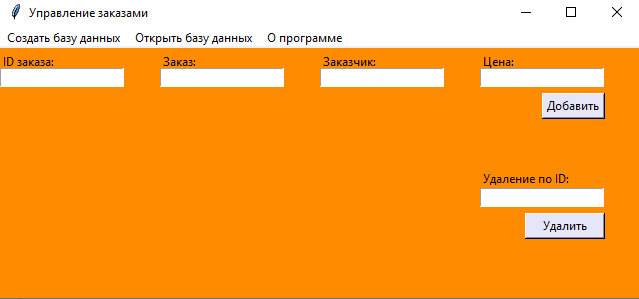


Рисунок. 19. Вид формы

Заполним форму данными:

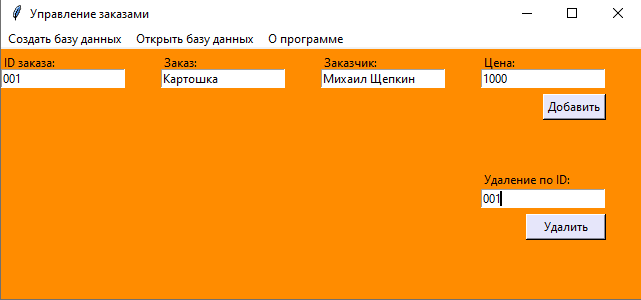


Рисунок. 20. Заполненная форма

Кнопка “Создать базу данных ” создаёт базу данных сразу открывая её.

Кнопка “Открыть базу данных ” позволяет выбрать базу данных для открытия.

Нажмём кнопку “Добавить”. В открытой базе данных сохраняются все данные в реальном времени.

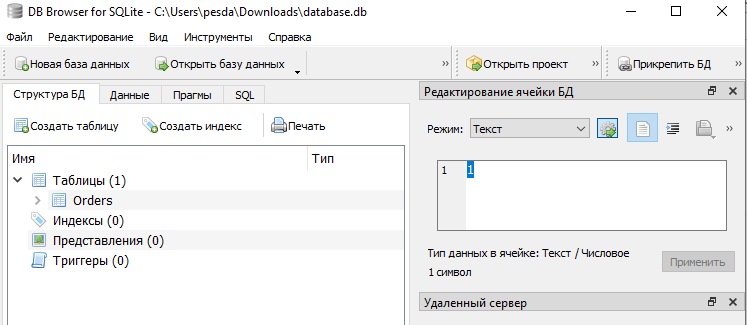


Рисунок. 21. Вывод в файл

## 2.4 Техника работы с библиотекой NumPy

NumPy это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций.

Установка:

• Через терминал командой: “pip install numpy”

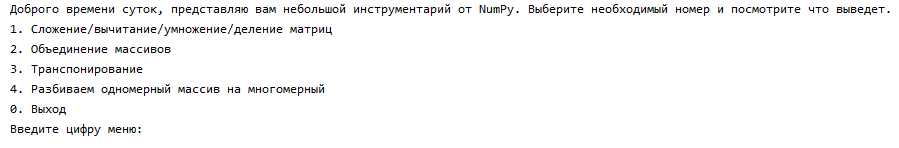
• По инструкции на сайте <https://www.scipy.org/install.html>

Главной особенностью numpy является объект array. Массивы схожи со списками в python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных, как float и int. С массивами можно проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее чем со списками.

**Листинг 52. primer.py**

import numpy  
  
  
def main():  
 *# Бесконечно выводим меню в консоль, пока пользователь не введёт 0* while(True):  
 menu()  
 print("Введите цифру меню: ")  
 num = int(input())  
 if(num == 1):  
 print("Введите функцию(-, +, \*, /):")  
 s = input()  
 function\_1(s)  
 print("\n")  
 elif(num == 2):  
 function\_2()  
 print("\n")  
 elif(num == 3):  
 function\_3()  
 print("\n")  
 elif(num == 4):  
 function\_4()  
 print("\n")  
 elif(num == 0):  
 *# Выход из программы* exit(0)  
 else:  
 print("Вы ввели не корректный номер функции. Попробуйте ещё раз\n")  
  
  
*# Функция menu обеспечивает вывод меню в консоль*def menu():  
 print("Доброго времени суток, представляю вам небольшой\  
 инструментарий от NumPy. Выберите необходимый номер и посмотрите что выведет.")  
 print("1. Сложение/вычитание/умножение/деление матриц")  
 print("2. Объединение массивов")  
 print("3. Транспонирование")  
 print("4. Разбиваем одномерный массив на многомерный")  
 print("0. Выход")  
  
*# Функция function\_1 выводит в консоль пример сложения/вычитания/деления/умножения матриц*def function\_1(s):  
 A = numpy.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
 B = A.copy()[::-1, ::-1]  
 if(s == "+"):  
 print("A\n", A)  
 print("B\n", B)  
 print("A + B\n", A + B)  
 elif(s == "-"):  
 print("A\n", A)  
 print("B\n", B)  
 print("A - B\n", A - B)  
 elif(s == "\*"):  
 print("A\n", A)  
 print("B\n", B)  
 print("A \* B\n", A \* B)  
 elif(s == "/"):  
 print("A\n", A)  
 print("B\n", B)  
 print("A / B\n", A // B)  
  
*# Функция function\_2 выводит в консоль пример объединения двух матриц в одну*def function\_2():  
 A = numpy.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
 B = A[::-1]  
 C = A[:, ::-1]  
 print("A\n", A)  
 print("\nB\n", B)  
 print("\nC\n", C)  
 print("\nstack(A, B, C)\n", numpy.stack((A, B, C)))  
  
*# Функция function\_3 выводит в консоль пример транспонирования матрицы*def function\_3():  
 A = numpy.array([[1, 2, 3], [4, 5, 6]])  
 print('A\n', A)  
  
 B = A.T  
 print('\nА транспонированный\n', B)  
  
*# Функция function\_4 выводит в консоль пример разбиения одномерного массива на многомерный*def function\_4():  
 A = numpy.arange(24)  
 B = A.reshape(4, 6)  
 C = A.reshape(4, 3, 2)  
 print("Массив размером 1х1\n", A)  
 print('\nМассив размером 4х6\n', B)  
 print('\nМассив размером 4х3х2\n', C)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':

main()

Рисунок. 22. Вывод меню в консоль

## 2.5 Техника работы с библиотекой Matplotlib

Библиотека matplotlib в Python помогает нам отображать данные на графиках в простейшем виде.

Библиотеку matplotlib легко установить с помощью pip:

pip install matplotlib

Теперь мы готовы создать несколько примеров, используя эту библиотеку визуализации данных.

**Листинг 53. matpotlib.py**

import matplotlib.pyplot as plt  
*# Координаты точек по x*x = [1, 5, 10, 15, 20]  
*# Координаты точек графика first по оси y*y1 = [1, 7, 3, 5, 11]  
*# Координаты точек графика second по оси y*y2 = [4, 3, 1, 8, 12]  
plt.figure(figsize=(7, 4)) *# Размер рисунка  
# Построение графика first*plt.plot(x, y1, 'o-r', alpha=0.7, label="first", lw=5, mec='b', mew=2, ms=10)  
*# Построение графика second*plt.plot(x, y2, 'v-.g', label="second", mec='r', lw=2, mew=2, ms=12)  
plt.legend() *# Отображение легенды*plt.grid(True) *# Отображение сетки*plt.show() *# Отображение графика*

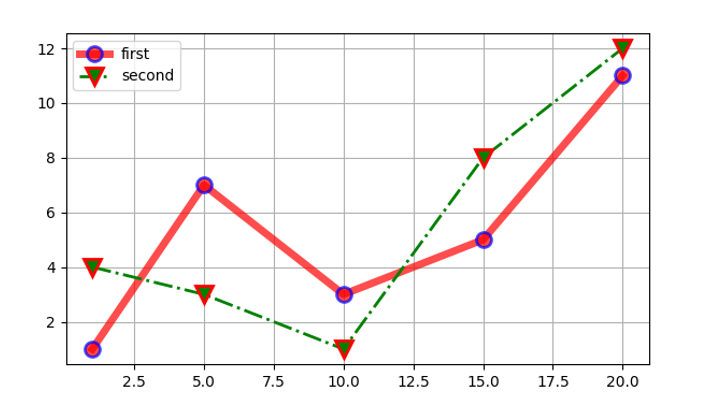


Рисунок. 23. Вывод программы

## 2.6 Элементы работы с библиотекой PyQt

PyQt — набор расширений [графического](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [фреймворка](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [Qt](https://ru.wikipedia.org/wiki/Qt) для [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Python](https://ru.wikipedia.org/wiki/Python), выполненный в виде [расширения](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD) Python. Это мульти-платформенный инструментарий, который запускается на большинстве операционных систем, среди которых Unix, Windows и MacOS.

**Листинг 54. Антон ?**

from PyQt5 import QtCore, QtWidgets

from PyQt5.QtWidgets import QMainWindow, QLabel, QGridLayout, QWidget, qApp, QAction

from PyQt5.QtCore import QSize

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

QMainWindow.\_\_init\_\_(self)

self.setMinimumSize(QSize(100, 100))

self.setWindowTitle("Скоро отчисление !?")

central\_widget = QWidget(self)

self.setCentralWidget(central\_widget)

grid\_layout = QGridLayout(self)

central\_widget.setLayout(grid\_layout)

title = QLabel("Антон отчислен !!!", self)

title.setAlignment(QtCore.Qt.AlignCenter)

grid\_layout.addWidget(title, 0, 0)

exit\_action = QAction("&Выход", self)

exit\_action.setShortcut('Q') # Задаём хоткей для выхода

exit\_action.triggered.connect(qApp.quit)

file\_menu = self.menuBar() # Устанавливаем в панель меню данный Action.

file\_menu.addAction(exit\_action)

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

mw = MainWindow()

mw.show()

sys.exit(app.exec())

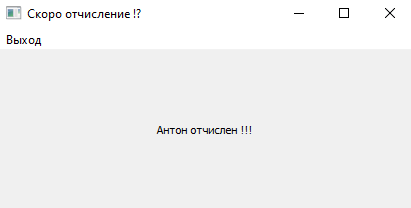


Рисунок. 23. Вывод программы

## 2.7 Элементы работы с библиотекой PyGame

Pygame – это библиотека модулей для языка Python, созданная для разработки 2D игр.

Pygame не входит в стандартную библиотеку Python, то есть не поставляется с установочным пакетом, а требует отдельной установки.

**Листинг 55. Змейка**

import pygame

import random

window = pygame.display.set\_mode((402, 402))

pygame.display.set\_caption("Zmey 3000")

screen = pygame.Surface((402, 402))

done = True

import pygame

import time

import random

pygame.init()

white = (255, 255, 255)

yellow = (255, 255, 102)

black = (0, 0, 0)

red = (213, 50, 80)

green = (0, 255, 0)

blue = (50, 153, 213)

dis\_width = 600

dis\_height = 400

dis = pygame.display.set\_mode((dis\_width, dis\_height))

pygame.display.set\_caption('Snake Game by Pythonist')

clock = pygame.time.Clock()

snake\_block = 10

snake\_speed = 15

font\_style = pygame.font.SysFont("bahnschrift", 25)

score\_font = pygame.font.SysFont("comicsansms", 35)

def Your\_score(score):

value = score\_font.render("Your Score: " + str(score), True, yellow)

dis.blit(value, [0, 0])

def our\_snake(snake\_block, snake\_list):

for x in snake\_list:

pygame.draw.rect(dis, black, [x[0], x[1], snake\_block, snake\_block])

def message(msg, color):

mesg = font\_style.render(msg, True, color)

dis.blit(mesg, [dis\_width / 6, dis\_height / 3])

def gameLoop():

game\_over = False

game\_close = False

x1 = dis\_width / 2

y1 = dis\_height / 2

x1\_change = 0

y1\_change = 0

snake\_List = []

Length\_of\_snake = 1

foodx = round(random.randrange(0, dis\_width - snake\_block) / 10.0) \* 10.0

foody = round(random.randrange(0, dis\_height - snake\_block) / 10.0) \* 10.0

while not game\_over:

while game\_close == True:

dis.fill(blue)

message("You Lost! Press C-Play Again or Q-Quit", red)

Your\_score(Length\_of\_snake - 1)

pygame.display.update()

for event in pygame.event.get():

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_q:

game\_over = True

game\_close = False

if event.key == pygame.K\_c:

gameLoop()

for event in pygame.event.get():

if event.type == pygame.QUIT:

game\_over = True

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

x1\_change = -snake\_block

y1\_change = 0

elif event.key == pygame.K\_RIGHT:

x1\_change = snake\_block

y1\_change = 0

elif event.key == pygame.K\_UP:

y1\_change = -snake\_block

x1\_change = 0

elif event.key == pygame.K\_DOWN:

y1\_change = snake\_block

x1\_change = 0

if x1 >= dis\_width or x1 < 0 or y1 >= dis\_height or y1 < 0:

game\_close = True

x1 += x1\_change

y1 += y1\_change

dis.fill(blue)

pygame.draw.rect(dis, green, [foodx, foody, snake\_block, snake\_block])

snake\_Head = []

snake\_Head.append(x1)

snake\_Head.append(y1)

snake\_List.append(snake\_Head)

if len(snake\_List) > Length\_of\_snake:

del snake\_List[0]

for x in snake\_List[:-1]:

if x == snake\_Head:

game\_close = True

our\_snake(snake\_block, snake\_List)

Your\_score(Length\_of\_snake - 1)

pygame.display.update()

if x1 == foodx and y1 == foody:

foodx = round(random.randrange(0, dis\_width - snake\_block) / 10.0) \* 10.0

foody = round(random.randrange(0, dis\_height - snake\_block) / 10.0) \* 10.0

Length\_of\_snake += 1

clock.tick(snake\_speed)

pygame.quit()

quit()

gameLoop()

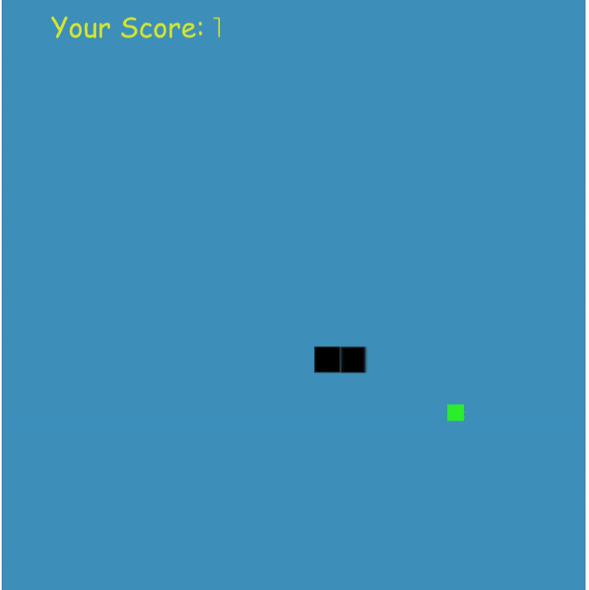


Рисунок. 23. Вывод программы

# **Раздел 3. Разработка проекта с графическим интерфейсом**

В данном проекте мы работаем в основе с двумя библиотеками “tkinter” и “sqlite3”.

Библиотека Tkinter установлена в Python в качестве стандартного модуля, поэтому нам не нужно устанавливать что-либо для его использования.

Чтобы импортировать его, пропишем

import tkinter

Python SQLite поставляется вместе с любой установленной версией Python, поэтому нет необходимости устанавливать его с помощью pip. Чтобы импортировать его, поскольку мы будем использовать Python3, мы импортируем модуль sqlite3.

import sqlite3

3.1 Изучение входной и выходной документации

Входные данные:

1. Документ на оплату

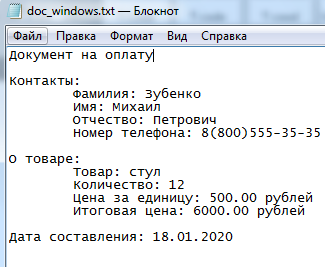


Рисунок. 24. Документ на оплату

1. Сумма денег клиента на оплату и вид оплаты (нал/безнал)

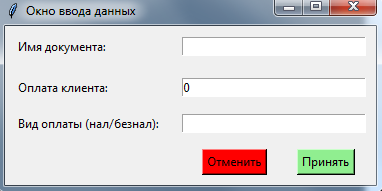
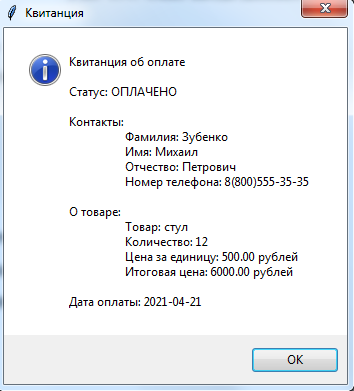


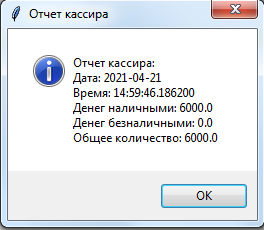
Рисунок. 25. Окно ввода данных

Выходные данные

1. Квитанцию об оплате

  
Рисунок. 26. Квитанция об оплате

1. Отчёт кассира

 Рисунок. 27. Отчёт кассира

1. База данных

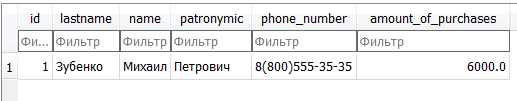


Рисунок. 28. Таблица “clients”

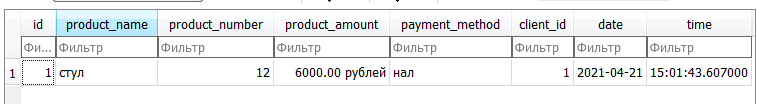


Рисунок. 29. Таблица “registry”

3.2 Разработка требований к проекту. Построение диаграммы использования.

Требования к проекту:

1. Быстрое оформление документов
2. Правильность оформления документов
3. Печать документов

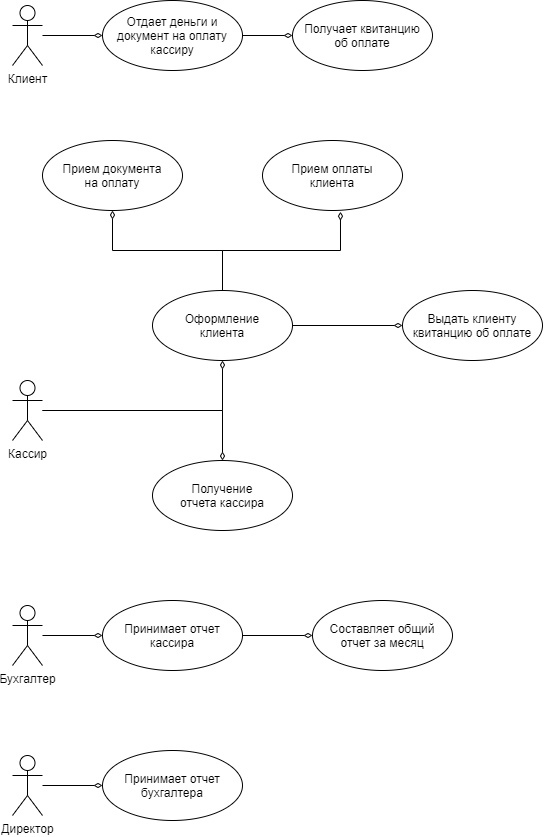


Рисунок. 30. Диаграмма использования

* 1. Разработка сценария проекта

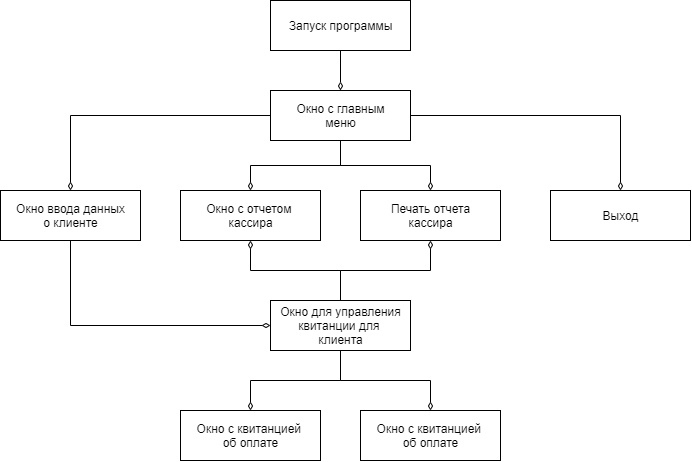
В данном разделе приведен сценарий использования программы пользователем (Рисунок 31).

Рисунок. 31. Сценарий проекта

* 1. Построение диаграммы классов

В данном разделе находятся диаграммы классов приложения.(Рисунок 32)

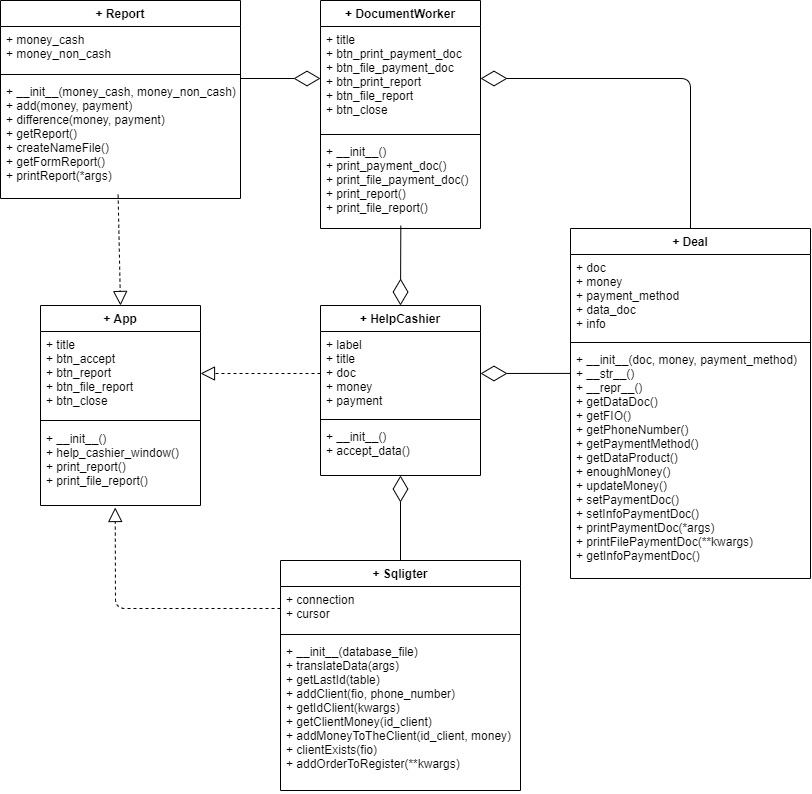


Рисунок. 32. Диаграмма классов

* 1. Разработка базы данных

В проекте в качестве базы данных используется **SQLite**. Причины использования:

1. **Минимальные затраты ресурсов.** Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера: база данных SQLite представляет собой обычный файл. Когда БД не используется, она не расходует процессорное время.
2. **Надежность и быстрота.** Базы данных SQLite поддерживают транзакции баз данных. Кроме того, операции чтения и записи данных реализуются на оптимизированном коде С.

Для работы с бд мы решили написать отдельный класс SQLigter.

**Листинг 56: sqlighter.py**

import sqlite3

from sqlite3.dbapi2 import connect

class SQLigter:

def \_\_init\_\_(self, database\_file):

"""Подключаемся к БД и сохраняем курсор соединения"""

self.connection = sqlite3.connect(database\_file)

self.cursor = self.connection.cursor()

# ! Обобщенные функции (для всех таблиц)

def translateData(self, args):

"""Переводит данные из словаря или из списка в список значений"""

if (len(args) == 1) and (type(args[0]) == dict):

return args[0].values()

elif type(args) == dict:

return args.values()

return args

def getLastId(self, table):

"""Вернет последний индекс"""

with self.connection:

try:

#return self.cursor.execute(f"SELECT `id` FROM `{table}` ORDER BY `id` DESC LIMIT 1").fetchall()[0][0]

return self.cursor.execute("SELECT `id` FROM `" + table + "` ORDER BY `id` DESC LIMIT 1").fetchall()[0][0]

except Exception:

return 0

# ! Работа с таблицей данных клиента `clients`

def addClient(self, fio, phone\_number):

"""Добавляем нового пользователя в базу"""

with self.connection:

#lastname, name, patronymic = self.translateData(fio)

need\_id = self.getLastId('clients')+1

print(fio)

result = self.cursor.execute("INSERT INTO `clients` (`id`, `lastname`, `name`, `patronymic`, `phone\_number`, `amount\_of\_purchases`) VALUES (?,?,?,?,?,?)", (need\_id, fio['lastname'], fio['name'], fio['patronymic'], phone\_number, 0.0)).fetchall()

self.connection.commit()

return result

def getIdClient(self, kwargs):

"""Вернет id клиента"""

#lastname, name, patronymic = self.translateData(args)

with self.connection:

return self.cursor.execute(\

"SELECT \* FROM `clients` WHERE `lastname` = ? AND `name` = ? AND `patronymic` = ?", \

(kwargs['lastname'], kwargs['name'], kwargs['patronymic'])\

).fetchall()[0][0]

def getClientMoney(self, id\_client):

"""Вернет сумму денег клиента из таблицы `clients`"""

with self.connection:

return self.connection.execute("SELECT \* FROM `clients` WHERE `id` = ?", (id\_client,)).fetchall()[0][-1]

def addMoneyToTheClient(self, id\_client, money):

"""Добавляет в таблицу `clients` в позицую `money` деньги"""

with self.connection:

amount\_summ = self.getClientMoney(id\_client) + money

self.connection.execute("UPDATE `clients` SET `amount\_of\_purchases` = ?", (amount\_summ,))

return amount\_summ

def clientExists(self, fio):

"""Существует ли клиент"""

with self.connection:

try:

result = self.cursor.execute(\

"SELECT \* FROM `clients` WHERE `lastname` = ? AND `name` = ? AND `patronymic` = ?", \

(fio['lastname'], fio['name'], fio['patronymic'])\

).fetchall()

except:

result = ''

return bool(len(result))

# ! Работа с таблицей реестра заказов `registry`

def addOrderToRegister(self, \*\*kwargs):

"""Добавляем в реестр заказов новый заказ"""

with self.connection:

# product\_name, product\_number, product\_amount, payment\_method, client\_id, date, time

need\_id = self.getLastId('registry') + 1

# [data\_for\_add.append(x) for x in self.translateData(args)]

kwargs['id'] = need\_id

result = self.cursor.execute(\

"INSERT INTO `registry` (\

`id`, `product\_name`, `product\_number`, `product\_amount`, `payment\_method`, `client\_id`, `date`, `time`\

) VALUES (?,?,?,?,?,?,?,?)", \

(kwargs['id'], kwargs['product\_name'], kwargs['product\_number'], kwargs['product\_amount'], kwargs['payment\_method'], \

kwargs['client\_id'], kwargs['date'], kwargs['time'])\

).fetchall()

self.connection.commit()

return result

* 1. Разработка главного модуля

Главный модуль состоит из класса App. Он выполняется сразу при запуске программы. Его также можно назвать главным меню.

**Листинг 57. Главный модуль App**

from tkinter import \*

from tkinter import messagebox

import tkinter as tk

from datetime import datetime

from Cashier import Report

cashier = Report(0, 0)

class App(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Мини-кассир")

# self.geometry('450x500')

# Верхний отступ

tk.Label(self, text='').pack(pady=5)

# Текст окна

tk.Label(self, text='Ваш мини-кассир', font=(80), pady=20).pack(pady=7)

# Кнопки в окне

self.btn\_accept = tk.Button(self, text='Принять документ и деньги', command=self.help\_cashier\_window)

self.btn\_report = tk.Button(self, text='Отчет кассира', command=self.print\_report)

self.btn\_file\_report = tk.Button(self, text='Печатать отчета кассира', command=self.print\_file\_report)

self.btn\_close = tk.Button(self, text='Выход', command=self.destroy)

# Позиционирование кнопок

self.btn\_accept.pack(padx=30, ipadx=11)

self.btn\_report.pack(ipadx=54)

self.btn\_file\_report.pack(ipadx=20)

self.btn\_close.pack(ipadx=80)

# Нижний отступ

tk.Label(self, text='').pack(pady=50)

def help\_cashier\_window(self):

help\_cashier = HelpCashier(self)

help\_cashier.grab\_set()

def print\_report(self):

messagebox.showinfo("Отчет кассира", cashier.getFormReport())

cashier.printReport()

print()

def print\_file\_report(self):

name\_file = cashier.createNameFile()

messagebox.showinfo("Имя квитанции", name\_file)

print("Отчет в файле: ", name\_file)

print()

* 1. Разработка входящих модулей

Описание спецификаций к модулям

Всего разработано 2 модуля:

* Окно ввода данных (HelpCashier)
* Меню для управления квитанцией клиента (DocumentWorker)

Взаимодействие между модулями можно наглядно увидеть по диаграммам классов из раздела 2.4.

Описание модулей:

**Окно ввода данных** – на данном окне находится форма для формирования квитанции клиенту. Для этого нужно заполнить следующие поля:

* Наименование документа
* Денежная сумма, которую дает клиент
* Метод оплаты клиента

Рассмотрим функцию обработки квитанции на оплату и её блок-схему:

**Листинг 58. Функция обработки квитанции на оплату**

def accept\_data(self):

data = {

"doc": self.doc.get(),

"money": self.money.get(),

"payment": self.payment.get()

}

try:

global deal

deal = Deal(data['doc'], data['money'], data['payment'])

# Если не хватило денег

if deal.enoughMoney() == False:

print()

print("Недостаточно")

messagebox.showinfo("Error", "Недостаточно средств")

# Денег хватает

else:

print()

print("Докумет принят")

self.destroy()

deal.updateMoney() # Обновляем сумму денег на стоимость товара

data['money'] = deal.money

cashier.add(data['money'], data['payment']) # Добавляем деньги в отчет

d\_fio = deal.getFIO() # Получаем ФИО клиента

# Если его нет в бд с клиентами, добавляем

if db.clientExists(d\_fio) == False:

db.addClient(d\_fio, deal.getPhoneNumber())

s = ''

for key in d\_fio:

s += d\_fio[key] + ' '

print("Добавлен новый клиент: ", s)

client\_id = db.getIdClient(d\_fio)

db.addMoneyToTheClient(client\_id, data['money']) # Добавляем деньги в общий список покупок клиента

# Добавляем в реестр заказов новый заказ

d = deal.getDataProduct()

date, time = str(datetime.today()).split()

db.addOrderToRegister(\

product\_name=d['product'], \

product\_number=d['number'], \

product\_amount=d['total\_price'], \

payment\_method=deal.getPaymentMethod(), \

client\_id=client\_id, \

date=date, \

time=time

)

data = [d['product'], d['number'], d['total\_price'], deal.getPaymentMethod(), client\_id, date, time]

s = str(db.getLastId('registry')) + ' | '

for x in data:

s += str(x) + ' | '

print("Добавлен в реестр: ", s)

print()

deal.setPaymentDoc()

doc\_work = DocumentWorker()

except:

print("Error")

messagebox.showinfo("Error", "Ошибка")

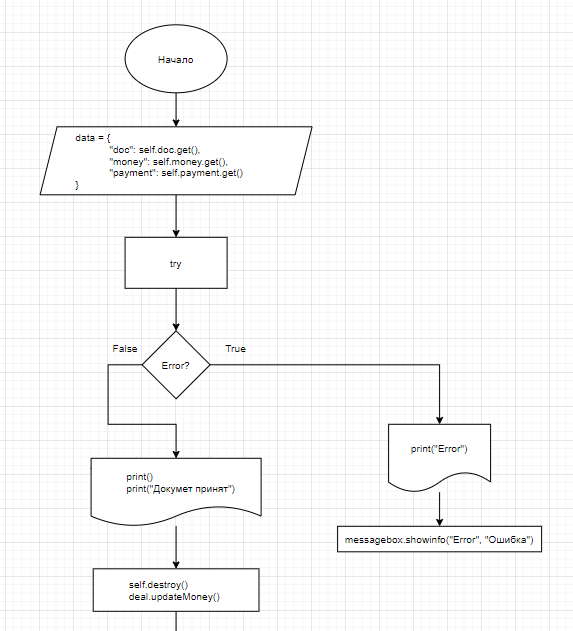


Рисунок. 33. Блок-схема к функции accept\_data. Часть 1

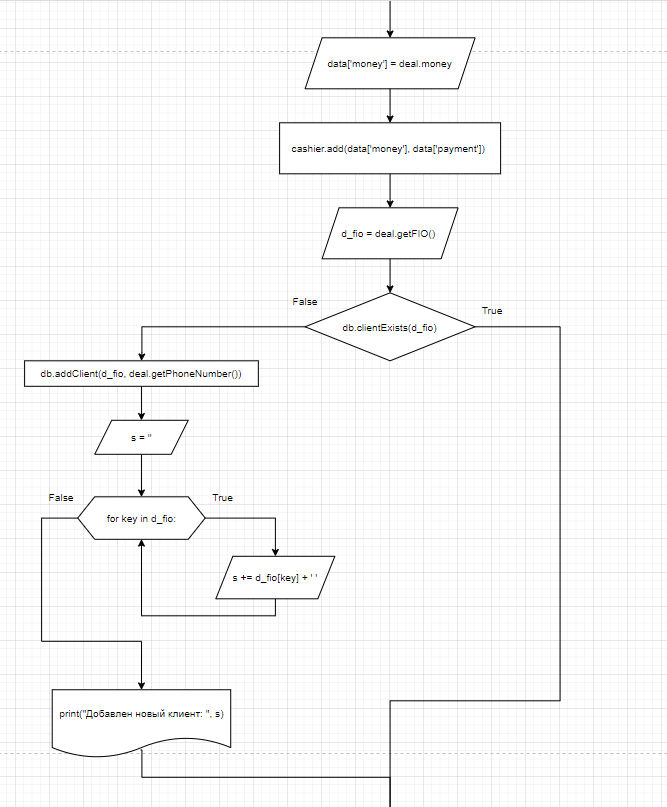


Рисунок. 34. Блок-схема к функции accept\_data. Часть 2

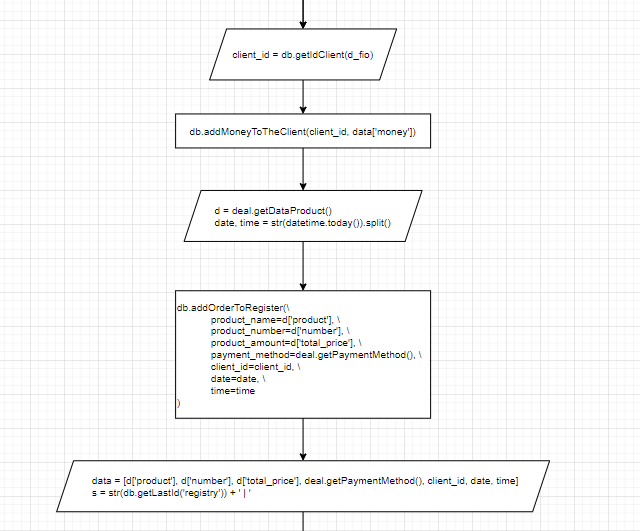


Рисунок. 35. Блок-схема к функции accept\_data. Часть 3.

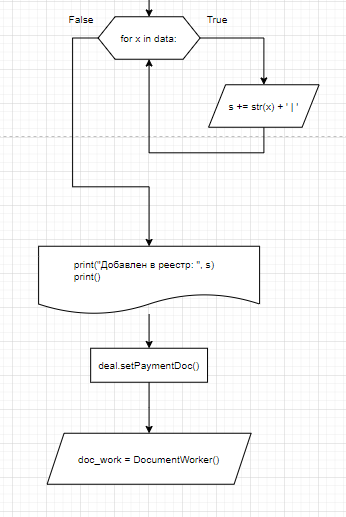


Рисунок. 36. Блок-схема к функции accept\_data. Часть 4

**Меню для управления квитанцией клиента** – на этом окне располагается меню для работы с квитанцией клиента об оплате. В этом меню есть следующие кнопки:

* Квитанция об оплате
* Печатать квитанции об оплате
* Отчет кассира
* Печатать отчета кассира
* Закрыть окно
  1. Тестирование и отладка

В ходе написания проекта при попытке запустить программу были получены ошибки (Рисунок 37):

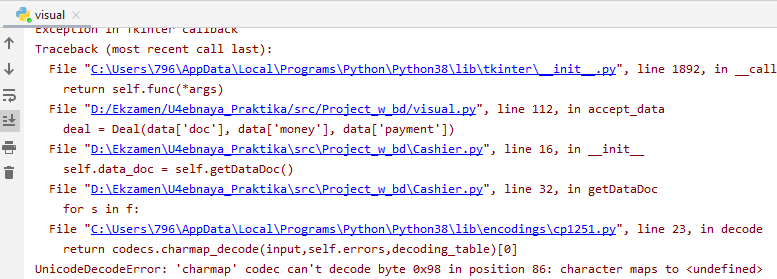


Рисунок. 37. Ошибки

При проверке кода были исправлены найденные ошибки, в результате при запуске программы ошибок не было (Рисунок 38):

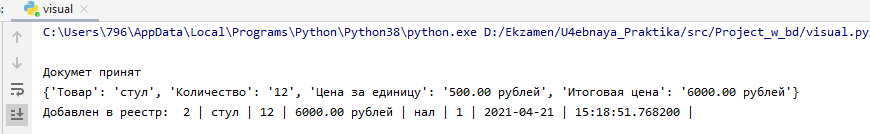


Рисунок. 38. Успешная сборка

* 1. Дневник

|  |  |  |
| --- | --- | --- |
| Дата | Содержание работ | Отметка о выполнении |
| 10.12.2020 | **Техника работы с линейными программами. Техника работы с разветвляющимися программами. Техника работы с циклическими программами.** |  |
| 11.12.2020 | **Техника работы с циклическими программами. Техника работы с числами.** |  |
| 12.12.2020 | **Техника работы со строками. Техника работы со списками.** |  |
| 14.12.2020 | **Техника работы со списками. Техника работы с циклом for и генераторами списков.** |  |
| 15.12.2020 | **Техника работы с функциями. Техника работы со словарями.** |  |
| 16.12.2020 | **Техника работы со словарями. Техника работы с множествами.** |  |
| 17.12.2020 | **Техника работы с кортежами. Техника работы с файлами.** |  |
| 18.12.2020 | **Техника работы с файлами. Техника работы с модулями.** |  |
| 19.12.2020 | **Техника работы с модулями. Техника работы с классами.** |  |
| 21.12.2020 | **Техника работы с классами.** |  |
| 30.03.2021 | Установка и настройка среды JetBrains PyCharm. Техника работы с базами данных. |  |
| 31.03.2021 | Техника работы с базами данных. Техника работы с библиотекой tkinter. |  |
| **01.04.2021** | Техника работы с библиотекой tkinter |  |
| 02.04.2021 | Техника работы с библиотекой tkinter. Техника работы с библиотекой NumPy. |  |
| 05.04.2021 | Техника работы с библиотекой NumPy. Техника работы с библиотекой Matplotlib. |  |
| 06.04.2021 | Техника работы с библиотекой Matplotlib. Элементы работы с библиотекой PyQt. |  |
| 07.04.2021 | Элементы работы с библиотекой PyQt. |  |
| 08.04.2021 | Элементы работы с библиотекой PyQt. Элементы работы с библиотекой PyGame. |  |
| 09.04.2021 | Элементы работы с библиотекой PyGame. Выполнение отчёта и презентации по использованию библиотек. |  |
| 12.04.2021 | Выполнение отчёта и презентации по использованию библиотек. |  |
| 13.04.2021 | Изучение входной и выходной документации. Разработка требований к проекту. Построение диаграммы использования |  |
| 14.04.2021 | Разработка требований к проекту. Построение диаграммы использования. Разработка сценария проекта. |  |
| 15.04.2021 | Разработка сценария проекта. Построение диаграммы классов. |  |
| 16.04.2021 | Построение диаграммы классов. Разработка базы данных. |  |
| 19.04.2021 | Разработка базы данных. Разработка главного модуля. |  |
| 20.04.2021 | Разработка главного модуля. Разработка входящих модулей. |  |
| 21.04.2021 | Разработка входящих модулей. |  |
| 22.04.2021 | Разработка входящих модулей. Тестирование и отладка. Разработка документации. |  |
| 23.04.2021 | Разработка документации. Защита проекта. |  |
| 26.04.2021 | Защита проекта. Сдача зачёта по практике. |  |