

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:

Степаненко К.А

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

Гусятинер Л. Б.

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

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

Королев, 2021

**Содержание**

[**Раздел 1. Техника решения задач с использованием структурного и объектно-ориентированного программирования.** 2](#_Toc69926563)

[1.1 Установка интерпретатора Python 3 и настройка окружения 2](#_Toc69926564)

[1.2 Техника работы в командной строке и среде IDLE 5](#_Toc69926565)

[1.3 Техника работы с линейными и разветвляющимися программами 7](#_Toc69926566)

[1.4 Техника работы с циклическими программами, цикл while 16](#_Toc69926567)

[1.5 Техника работы с числами 24](#_Toc69926568)

[1.6 Техника работы со строками 31](#_Toc69926569)

[1.7 Техника работы со списками 36](#_Toc69926570)

[1.8 Техника работы с циклом for и генераторами списков 41](#_Toc69926571)

[1.9 Техника работы с функциями 48](#_Toc69926572)

[1.10 Техника работы со словарями 54](#_Toc69926573)

[1.11 Техника работы с множествами 64](#_Toc69926574)

[1.12 Техника работы с кортежами 72](#_Toc69926575)

[1.13 Техника работы с файлами 78](#_Toc69926576)

[1.14 Техника работы с модулями 86](#_Toc69926577)

[1.15 Техника работы с классами 102](#_Toc69926578)

[**Раздел 2. Техника решения задач с использованием библиотек** 113](#_Toc69926579)

[2.1 Установка и настройка среды JetBrains PyCharm 113](#_Toc69926580)

[2.2 Техника работы с базами данных 119](#_Toc69926581)

[2.3 Техника работы с библиотекой tkinter 123](#_Toc69926582)

[2.4 Техника работы с библиотекой NumPy 126](#_Toc69926583)

[2.5 Техника работы с библиотекой Matplotlib 128](#_Toc69926584)

[2.6 Элементы работы с библиотекой PyQt 130](#_Toc69926585)

[2.7 Элементы работы с библиотекой PyGame 132](#_Toc69926586)

[**Раздел 3. Разработка проекта с графическим интерфейсом** 146](#_Toc69926587)

[3.1 Изучение входной и выходной документации 146](#_Toc69926588)

[3.2 Разработка требований к проекту. Построение диаграммы использования. 148](#_Toc69926589)

[3.3 Разработка сценария проекта 150](#_Toc69926590)

[3.4 Построение диаграммы классов 150](#_Toc69926591)

[3.5 Разработка базы данных 151](#_Toc69926592)

[3.6 Разработка главного модуля 155](#_Toc69926593)

[3.7 Разработка входящих модулей 156](#_Toc69926594)

[3.8 Тестирование и отладка 162](#_Toc69926595)

[3.9 Дневник 163](#_Toc69926596)

# **Раздел 1. Техника решения задач с использованием структурного программирования.**

## 1.1 Установка интерпретатора Python 3 и настройка окружения

Для установки интерпретатора Python на компьютер, первое, что нужно сделать – это скачать дистрибутив. Загрузить его  можно с официального сайта, перейдя по ссылке <https://www.python.org/downloads/>

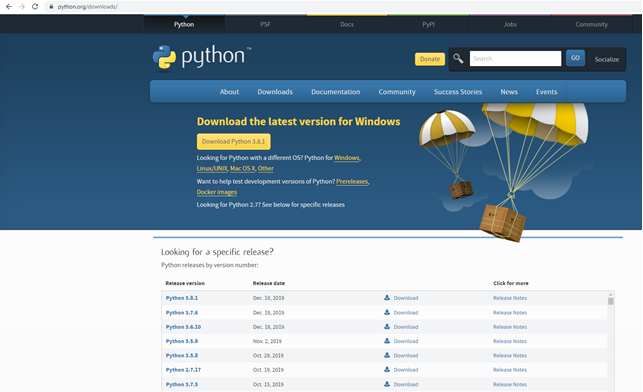


Рисунок 1. Официальный сайт Python

Порядок установки на Windows:

1. Запустить скачанный установочный файл.

2. Выбрать способ установки.



Рисунок 2. Установщик Python

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

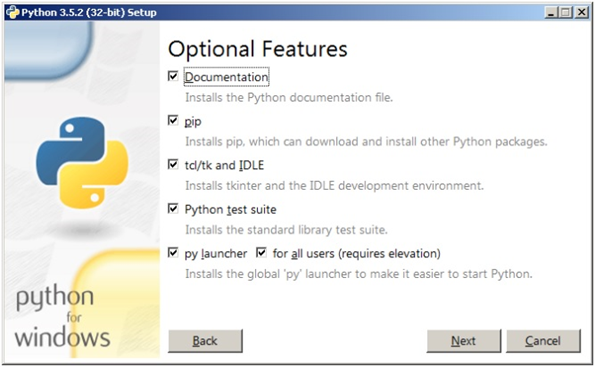


Рисунок 3. Опции установки

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

Documentation – установка документаций.

pip – установка пакетного менеджера pip.

tcl/tk and IDLE – установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).

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

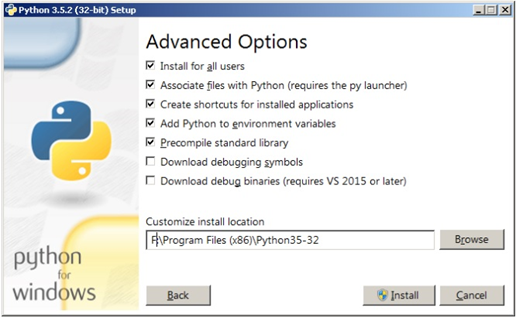


Рисунок 4. Продвинутые опции установки

5. После успешной установки:

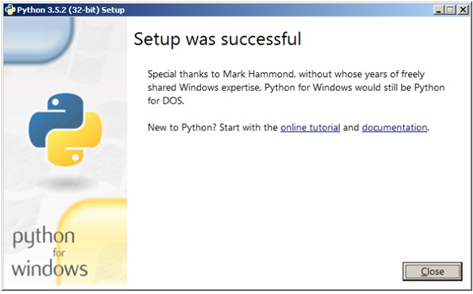


Рисунок 5. Сообщение об установке

## 1.2 Техника работы в командной строке и среде IDLE

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

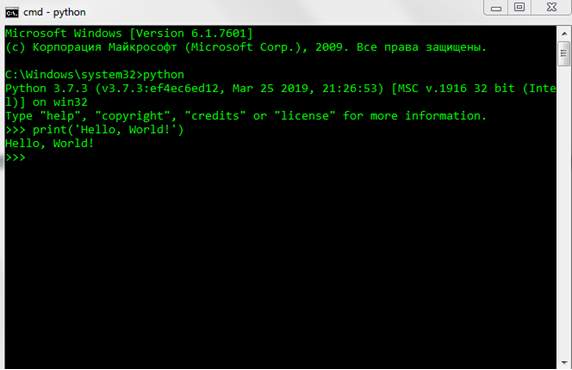


Рисунок 6. Интерактивная оболочка Python

Существует несколько способов закрыть оболочку Python:

>>> exit()

или же

>>> quit()

Кроме того, CTRL + D закроет оболочку и вернет вас в командную строку терминала.

[IDLE](https://docs.python.org/2/library/idle.html) - простой редактор для Python, который поставляется вместе с Python.

Откройте IDLE в вашей системе выбора.

В оболочке есть подсказка из трех прямоугольных скобок:

>>>

Теперь напишите в подсказке следующий код:

>>> print("Hello, World")

Нажмите Enter .

>>> print("Hello, World")

Hello, World

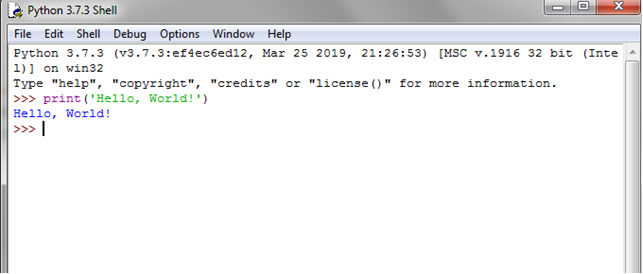


Рисунок 7. Первая программа

## 1.3 Техника работы с линейными и разветвляющимися программами

Листинг 1. K4\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#Задание. Разработать программы по темам  
#- input -print  
#Функция input() в Python, ввод данных с клавиатура.  
  
name = input("What is your name? ")  
age = int(input("How old are you? "))  
home = input("Where are you live? ")  
print("This is",name)  
print('Is ' + str(age))  
print('She/He live in',home)

Листинг 2. K4\_2

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
'''

#Разработать программу для печати даты прописью

import re

import math

#дата

t = input()

#считали дату, взяли кортеж с тремя элементами d, m ,y

r = re.findall(r'(\d{2})\.(\d{2})\.(\d{4})', t)[0]

d, m, y = tuple(map(int, r))

#кортеж для дней

d\_list = (

'первое', 'второе', 'третье', 'четвёртое', 'пятое',

'шестое', 'седьмое', 'восьмое', 'девятое', 'десятое',

'одиннадцатое', 'двенадцатое', 'тринадцатое', 'четырнадцатое', 'пятнадцатое',

'шестнадцатое', 'семнадцатое', 'восемнадцатое', 'девятнадцатое', 'двадцатое')

#кортеж для месяцев

m\_list = (

'января', 'февраля', 'марта', 'апреля',

'мая', 'июня', 'июля', 'августа',

'сентября', 'октября', 'ноября', 'декабря')

#кортеж для годов

y1\_list = (

'одна тысяча', 'две тысячи', 'три тысячи',

'четыре тысячи', 'пять тысяч', 'шесть тысяч',

'семь тысяч', 'восемь тысяч', 'девять тысяч')

y2\_list = (

'', 'сто', 'двести', 'триста',

'четыреста', 'пятьсот', 'шестьсот',

'семьсот', 'восемьсот', 'девятьсот')

y3\_list = (

'первого', 'второго', 'третьего', 'четвёртого', 'пятого',

'шестого', 'седьмого', 'восьмого', 'девятого', 'десятого',

'одиннадцатого', 'двенадцатого', 'тринадцатого', 'четырнадцатого', 'пятнадцатого',

'шестнадцатого', 'семнадцатого', 'восемнадцатого', 'девятнадцатого', 'двадцатого')

y4\_list = (

'тридцатого', 'сорокового', 'пятидесятого', 'шестидесятого',

'семидесятого', 'восьмидесятого', 'девяностого')

y5\_list = (

'двадцать', 'тридцать', 'сорок', 'пятьдесят',

'шестьдесят', 'семьдесят', 'восемьдесят', 'девяносто')

y6\_list = (

'одна тысячного года', 'двухтысячного года', 'трехтысячного года', 'четырехтысячного года',

'пятитысячного года', 'шеститысячного года', 'семитысячного года', 'восьмитысячного года')

#преобразование в письменный вид дня

if d == 30:

print('тридцатое', end=' ')

else:

if d > 30:

print('тридцать', end=' ')

d %= 30

elif d > 20:

print('двадцать', end=' ')

d %= 20

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

i = 0

#преобразование в письменный вид месяца

while i-1 < m:

if i == m:

print(m\_list[i-1], end =' ')

break

i+=1

i = 0

#преобразование в письменный вид года

while i <= y / 1000:

if y / 1000 == 0 and y > 1000 and y % 1000 == i:

print(y6\_list[i])

break

elif i == math.floor(y / 1000):

print(y1\_list[i-1], end = ' ')

j = 0

while j < 10:

if j == math.floor((y % 1000) / 100):

print(y2\_list[j], end = ' ')

g = 0

while g < 20:

if (y % 100) <= 20 and g == math.floor(y % 100):

print(y3\_list[g-1], 'года')

break

elif y % 100 > 20 and math.floor((y % 100) / 10) == g and y % 10 == 0:

print(y4\_list[g-3], 'года')

break

elif y % 100 > 20 and math.floor((y % 100) / 10) ==g:

print(y5\_list[g-2], end =' ')

k = 0

while k < 10:

if y % 10 == k:

print(y3\_list[k-1], 'года')

break

k+=1

g+=1

j+=1

i+=1

## 1.4 Техника работы с циклическими программами, цикл while

Листинг 3. K5\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Задание 1. На плоскости нарисован квадрат заданного размера с левой #нижней

#вершиной в начале координат. В квадрат вписывается окружность.

#Случайным образом в квадрате выбирается 1000 точек.

#а) нужно определить, сколько точек попало внутрь круга

#б) считая количество точек пропорциональным площади, найти отношение #площадей

#круга и квадрата

#в) по этому отношению определить приближённое значение числа пи

#г) определить, насколько найденное значение отличается от #"библиотечного".

import random

import math

count = 100

len\_1 = float(input())

points\_inside\_the\_ring = []

points\_out\_of\_ring = []

r = len\_1 / 2

counter = 0

Pi = 0.0

k = 0.0

i = 0

j = 0

while i < count:

x = random.uniform(0, len\_1)

y = random.uniform(0, len\_1)

points\_koor = (x, y)

point\_with\_num = [i+1, points\_koor]

i += 1

if x >= r and y >= r:

if r\*\*2 >= (x-r)\*\*2 + (y-r)\*\*2:

counter += 1

points\_inside\_the\_ring.append(point\_with\_num)

k+=(x\*\*2 + y\*\*2 < len\_1\*\*2 \*1.0)

j+=1

else:

points\_out\_of\_ring.append(point\_with\_num)

elif x >= r and y <= r:

if r\*\*2 >= (x-r)\*\*2 + (r-y)\*\*2:

counter += 1

points\_inside\_the\_ring.append(point\_with\_num)

k+=(x\*\*2 + y\*\*2 < len\_1\*\*2 \*1.0)

j+=1

else:

points\_out\_of\_ring.append(point\_with\_num)

elif x <= r and y >= r:

if r\*\*2 >= (r-x)\*\*2 + (y-r)\*\*2:

counter += 1

points\_inside\_the\_ring.append(point\_with\_num)

k+=(x\*\*2 + y\*\*2 < len\_1\*\*2 \*1.0)

j+=1

else:

points\_out\_of\_ring.append(point\_with\_num)

elif x <= r and y <= r:

if r\*\*2 >= (r-x)\*\*2 + (r-y)\*\*2:

counter += 1

points\_inside\_the\_ring.append(point\_with\_num)

k+=(x\*\*2 + y\*\*2 < len\_1\*\*2 \*1.0)

j+=1

else:

points\_out\_of\_ring.append(point\_with\_num)

attitude = len\_1\*\*2/(math.pi\*r\*\*2)

Pi = 4\*k/j

print('attitude:', attitude)

print('points inlisde the ring:', counter)

print('Approximate number of pi:', Pi)

print('difference:', math.pi - 4\*k/j)

Листинг 4. K5\_1\_2.

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

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

#Задача.Дан массив чисел ,напечатать этот массив с следующими условиями:

#если число нечетное его не печатать, если четное напечатать 2 раза

Arr = []

len\_arr = int(input())

i = 0

while i < len\_arr:

x = int(input())

Arr.append(x)

i+=1

i = 0

while i < len\_arr:

if Arr[i] % 2 == 1:

i+=1

continue

else:

print(Arr[i], Arr[i])

i+=1

Листинг 5. K5\_2\_2

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

# 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.

a = int(input())

sum = 0

while a != 0:

sum += a

a = int(input())

print(sum)

Листинг 6. K5\_2\_3.

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

Num1 , Num2 = map(int, input().split())

while Num2 != 0:

if Num1 < Num2:

Num1, Num2 = Num2, Num1

Num1 = Num1 % Num2

Num1, Num2 = Num2, Num1

print(Num1)

Листинг 7. K5\_2\_5.

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Напишите программу, которая выводит часть

#последовательности 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 ...

#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.

len1 = int(input())

i = 0

j = 0

k = 0

while k < len1:

j += 1

for i in range(j):

if k < len1:

k += 1

print(j, end= ' ')

## 1.5 Техника работы с числами

Листинг 8. K6\_1.

# Выполнил:Степаненко Кирилл Алексеевич  
# Группа: П2-18  
# Fractions  
# Этот модуль пригодится в тех случаях, когда вам необходимо выполнить вычисления с дробями  
# Или когда результат должен быть выражен в формате дроби.  
  
from fractions import Fraction as frac  
print(Fraction(33.33))  
2345390243441541/70368744177664  
print(Fraction('33.33'))  
3333/100  
  
# Модуль Fraction особенно полезен, потому что он автоматически уменьшает дробь. Выглядит это вот так:  
  
Fraction(153, 272)  
Fraction(9, 16)  
  
# Кроме того, вы можете выполнять бинарные (двоичные) операции над дробью также просто,как вы используете int или float.  
# Просто добавьте две фракции:  
  
Fraction(1, 2) + Fraction(3, 4)  
Fraction(5, 4)  
  
# Теперь давайте попробуем возвести дробь в степень:  
  
Fraction(1, 8) \*\* Fraction(1, 2)  
0.3535533905932738  
# Decimal- вычисления с заданной точностью  
# Модуль Decimal незаменим, если нужно считать деньги: с его помощью вы сможете подсчитать точную сумму, вплоть до копеек.  
  
# При работе с числами с плавающей точкой (то есть float) мы сталкиваемся с тем, что в результате вычислений мы получаем не совсем верный результат:  
  
number = 0.1 + 0.1 + 0.1  
print(number) # 0.30000000000000004  
  
# Проблему может решить использование функции round(), которая округлит число.  
# Однако есть и другой способ, который заключается в использовании встроенного модуля decimal.  
# Ключевым компонентом для работы с числами в этом модуле является класс Decimal.  
# Для его применения нам надо создать его объект с помощью конструктора.  
# В конструктор передается строковое значение, которое представляет число:  
  
from decimal import Decimal  
  
number = Decimal("0.1")  
  
# После этого объект Decimal можно использовать в арифметических операциях:  
  
from decimal import Decimal  
  
number = Decimal("0.1")  
number = number + number + number  
print(number) # 0.3  
  
# В операциях с Decimal можно использовать целые числа:  
  
number = Decimal("0.1")  
number = number + 2  
  
# Однако нельзя смешивать в операциях дробные числа float и Decimal:  
  
number = Decimal("0.1")  
number = number + 0.1 # здесь возникнет ошибка  
  
# С помощью дополнительных знаков мы можем определить, сколько будет символов в дробной части числа:  
  
number = Decimal("0.10") # Строка "0.10" определяет два знака в дробной части, даже если последние символы будут представлять ноль. Соответственно "0.100" представляет три знака в дробной части.  
number = 3 \* number  
print(number) # 0.30

## 1.6 Техника работы со строками

Листинг 9. K7\_1\_1.

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#С клавиатуры вводятся строки, последовательность заканчивается точкой.

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

a = input()

while a != ".":

print(' '.join(list(a.upper())))

a = input()

Листинг 10. K7\_1\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

a = input()

b = a.replace(" ", "\_")

print(b)

Листинг 11. K7\_1\_3.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Уберите точки из введенного IP-адреса.

#Выведите сначала четыре числа через пробел, а затем сумму получившихся #чисел.

a = input()

sum1 = 0

a = a.split('.')

i = 0

for i in a:

sum1 += int(i)

print(i, end = ' ')

print()

print(sum1)

Листинг 12. K7\_1\_4.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

#@@ обозначает ошибку

#!! обозначает предупреждение

#// обозначает информационное сообщение

#\*\* обозначает подробное сообщение

#Напишите программу, которая принимает строки до точки и выводит,

#какого типа это сообщение.

#Если сообщение не содержит модификаторов, проигнорируйте его.

st = input()

while st != ".":

if st[0] == st[1] == st[-1] == st[-2] == '!':

print("предупреждение")

elif st[0] == st[1] == st[-1] == st[-2] == '@':

print("ошибка")

elif st[0] == st[1] == st[-1] == st[-2] == '/':

print("информация")

elif st[0] == st[1] == st[-1] == st[-2] == '\*':

print("подробное сообщение")

st = input()

Листинг 13. K7\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Форматирование строк с помощью метода format (Инструкция)

#Если для подстановки требуется только один аргумент, то значение - сам аргумент:

#Пример №1

print('Hello, {}!'.format('World'))

#А если несколько, то значениями будут являться все аргументы со строками

#подстановки (обычных или именованных):

#Пример №2

print('{0}, {1}, {2}'.format('a', 'b', 'c'))

#Однако метод format умеет большее. Вот его синтаксис:

#поле замены : "{" [имя поля] ["!" преобразование] [":" спецификация] #"}"

#имя поля : arg\_name ("." имя атрибута | "[" индекс "]")\*

#преобразование : "r" (внутреннее представление) | "s" (человеческое #представление)

#спецификация : см. ниже

#Пример №3

print("Units destroyed: {players[0]}".format(players = [1, 2, 3]))

print("Units destroyed: {players[0]!r}".format(players = ['1', '2', '3']))

#Cпецификация формата:

#спецификация : [[fill]align][sign][#][0][width][,][.precision][type]

#заполнитель : символ кроме '{' или '}'

#выравнивание : "<" , ">" , "=" , "^"

#знак : "+" , "-" , " "

#ширина : integer

#точность : integer

#тип : "d" ,"o" ,"x" ,"X" ,"e" ,"E" ,"f" ,"g" ,"G" ,

# "c" ,"s" ,"%" ,"x" ,"X" ,"%"

#Выравнивание производится при помощи символа-заполнителя. Доступны #следующие варианты выравнивания:

# < - Символы-заполнители будут справа (выравнивание объекта по левому #краю) (по умолчанию).

# > - Выравнивание объекта по правому краю.

# = - Заполнитель будет после знака, но перед цифрами. Работает только с #числовыми типами.

# ^ - Выравнивание по центру.

#Опция "знак" используется только для чисел и может принимать следующие #значения:

#"+" - Знак должен быть использован для всех чисел.

#"-" - Для отрицательных, ничего для положительных.

#" " - Для положительных.

#Поле "тип" может принимать следующие значения:

#'d', 'i', 'u' - Десятичное число.

#'o' - Число в восьмеричной системе счисления.

#'x' - Число в шестнадцатеричной системе счисления (буквы в нижнем #регистре).

#'X' - Число в шестнадцатеричной системе счисления (буквы в верхнем #регистре).

#'e' - Число с плавающей точкой с экспонентой (экспонента в нижнем #регистре).

#'E' - Число с плавающей точкой с экспонентой (экспонента в верхнем #регистре).

#'f' - Число с плавающей точкой (обычный формат).

#'g' - Число с плавающей точкой. с экспонентой (экспонента в нижнем #регистре), если она меньше, чем -4 или точности, иначе обычный формат.

#'G' - Число с плавающей точкой. с экспонентой (экспонента в верхнем #регистре), если она меньше, чем -4 или точности, иначе обычный формат.

#'c' - Символ (строка из одного символа или число - код символа).

#'s' - Строка.

#'%' - Число умножается на 100, отображается число с плавающей #точкой, а за ним знак %.

## 1.7 Техника работы со списками

Листинг 14. K8\_1\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

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

counter = 0

i = 1

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

if a[i - 1] < a[i] > a[i + 1]:

counter += 1

print(counter)

Листинг 15. K8\_1\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

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

counter = 0

i = 0

for i in range(len(a)):

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

if a[i] == a[j]:

counter += 1

print(counter)

Листинг 16. K8\_2\_1.py

#Выполнил:Степаненко Кирилл Алексеевич  
#Группа: П2-18

#Array112. Дан массив A размера N.

#Упорядочить его по возрастанию методом сортировки

#простым обменом («пузырьковой» сортировкой):

#просматривать массив, сравнивая его соседние элементы

#(A0 и A1, A1 и A2 и т. д.) 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())

lst = []

for i in range(n):

lst.append(randint(1,100))

print(lst)

for i in range(n-1):

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

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

lst[j], lst[j+1] = lst[j+1], lst[j]

print(lst)

Листинг 17. K8\_2\_3.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Array113. Дан массив A размера N.

#Упорядочить его по возрастанию методом сортировки простым

#выбором: найти максимальный элемент массива и поменять его

#местами с последним (N-1 м) элементом; выполнить описанные

#действия N 1 раз, каждый раз уменьшая на 1 количество

#анализируемых элементов и выводя содержимое массива.

from random import randint

n = int(input())

lst = []

for i in range(n):

lst.append(randint(1, 100))

i = -1

print(lst)

while i > -n:

k = i

j = i - 1

while j >= -n:

if lst[j] > lst[k]:

k = j

j -= 1

lst[i], lst[k] = lst[k], lst[i]

i -= 1

print(lst)

## 1.8 Техника работы с циклом for и генераторами списков

Листинг 18. K9\_2\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

# 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 = random.randrange(2, 24)  
a = [i for i in range(n)]  
b = a[1::2]  
print(n)  
print(a)  
print(len(b))  
print(b)

Листинг 19. K9\_2\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

import random  
n = random.randrange(2, 22)  
a = [i for i in range(n)]  
b = a[0::2] + a[1::2]  
print(n)  
print(a)  
print(len(b))  
print(b)

Листинг 20. K9\_3\_4.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
# 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.  
  
from random import random  
n = int(input("n = "))  
a = []  
b = []  
for i in range(n):  
 c = int(random() \* 50)  
 a.append(c)  
print(a)  
s = 0  
for i in range(n-1,-1,-1):  
 s = s+a[i]  
 b.insert(0,s)  
print(b)

Листинг 21. K9\_2\_4.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Matrix3. Даны целые положительные числа M, N и набор из M чисел.

#Сформировать матрицу размера M x N, у которой в каждом столбце

#содержатся все числа из исходного набора (в том же порядке).

from random import randint

M = int(input())

N = int(input())

mass\_a = []

mass\_b = [[]]

for i in range(M):

mass\_a.append(randint(1,100))

print(mass\_a)

for i in range(M):

#for j in range(N):

mass\_b[i][0] = mass\_a[i]

print(mass\_b)

1.9 Техника работы с функциями.

Листинг 22. K10\_1\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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(x, y):  
 if x > y:  
 return 0  
 else:  
 return sum(range(x, y + 1))  
  
  
a = int(input('Введите первое число: '))  
b = int(input('Введите второе число: '))  
c = int(input('Введите третье число. Если не хотите его использовать - введите нуль: '))  
if (a and b > 0) and (c == 0):  
 print("a =", a)  
 print("b =", b)  
 print('Сумма чисел от', a, 'до', b, 'равна', SumRange(a, b))  
elif a and b and c > 0:  
 print("a =", a)  
 print("b =", b)  
 print("c =", c)  
 print('Сумма чисел от', a, 'до', b, 'равна', SumRange(a, b))  
 print('Сумма чисел от', b, 'до', c, 'равна', SumRange(b, c))

Листинг 23. K10\_1\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#Описать функцию SortInc3(X), меняющую содержимое списка X из трех  
#вещественных элементов таким образом, чтобы их значения оказались упорядоченными по  
#возрастанию (функция возвращает None). С помощью этой функции упорядочить по  
#возрастанию два данных списка X и Y.  
  
import random as rnd  
  
  
def SortInc3(x):  
 x.sort()  
  
  
a = [rnd.uniform(0, 100) for i in range(3)]  
b = [rnd.uniform(0, 100) for i in range(3)]  
  
SortInc3(a)  
SortInc3(b)  
print(a)  
print(b)

Листинг 24. K10\_2\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#Использовать map, lambda  
#Квадраты в обратном порядке. Числа вводятся до точки. Через пробел выведите эти числа в  
#обратном порядке, возводя их в квадрат.  
#Sample Input:  
#5  
#16  
#20  
#1  
#9  
#.  
#Sample Output:  
#81 1 400 256 25  
  
v = [int(x) for x in iter(input, '.')]  
b = list(map(lambda x: print(x \* x, end=' '), v[::-1]))

Листинг 25. K10\_2\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
# Использовать map, lambda  
# Квадраты в обратном порядке. Числа вводятся до точки. Через пробел выведите эти числа в  
# обратном порядке, возводя их в квадрат.  
  
lst = []  
lst\_2 = []  
s = input()  
while s != ".":  
 lst.append(int(s))  
 s = input()  
lst\_2 = list(map(lambda x: x\*\*2, lst[::-1]))  
print(\*lst\_2)

Листинг 26. K10\_2\_3.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
# 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())  
a = [int(input()) for y in range(n)]  
b = list(filter(lambda x: x%2, a))  
print(b, len(b))

Листинг 27. K10\_2\_4.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#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, b, c = map(int, input().split())  
a = [i for i in range(a, b+1)]  
print(\*list(map(lambda x: x\*\*c, a)))

1.10 Техника работы с словарями

Листинг 28. K11\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

# 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.

a = {}  
for i in input().split():  
 a[i] = a.get(i , 0) + 1  
 print(a[i] - 1 , end = ' ')

Листинг 29. K11\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#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.

d = dict()

mass = []

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

mass = s.split()

if len(mass) == 1:

print(d[mass[0]])

else:

d[mass[0]] = mass[1]

1.11 Техника работы с множествами.

Листинг 30. K12\_1\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

# Задача «Количество различных чисел»  
# Условие. Дан список чисел. Определите, сколько в нем встречается различных чисел.

s = set(input().split())

print(len(s))

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

Листинг 31. K12\_1\_2.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

# Задача «Количество совпадающих чисел»  
# Условие. Даны два списка чисел. Посчитайте, сколько чисел содержится одновременно как  
# в первом списке, так и во втором.

s1 = set(input().split())

s2 = set(input().split())

print(set.intersection(s1,s2))

Листинг 32. K12\_1\_3.py

# Выполнил:Степаненко Кирилл

# Группа: П2-18  
# 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.  
a = set(input().split())  
b = set(input().split())  
c = list(a & b)  
for i in range (len(c)):  
 c[i] = int(c[i])  
c.sort()  
print(' '.join([str(i) for i in c]))

Листинг 33. K12\_1\_4.py

# Выполнил:Степаненко Кирилл  
# Группа: П2-18  
# 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.  
  
n = int(input())  
words = set()  
for i in range(n):  
 for x in input().split():  
 words.add( x )  
print(len(words))

Листинг 34. K12\_1\_5.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#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.  
  
n = int(input())  
lang = set()  
  
for i in range(n):  
 m = int(input())  
 a = set()  
 for j in range(m):  
 a.update(input().split())  
 if len(lang) == 0:  
 lang = a  
 c = lang & a  
 b = lang | a  
  
print(len(c))  
print(\*sorted(c))  
print(len(b))  
print(\*sorted(b))

Листинг 35. K12\_2\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18  
#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 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.  
  
n = int(input())  
a = set()  
for i in range(n):  
 s = input().lower()  
 a.add(s)  
e= set()  
l = int(input())  
for i in range(l):  
 s = input().lower().split()  
 for j in s:  
 if(j not in a and j not in error):  
 e.add(j)  
print('\n'.join(e))

1.12 Техника работы с кортежами.

Листинг 36. K13\_1\_1.py

#Выполнил:Степаненко Кирилл  
#Группа: П2-18

#Вывести чётные

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

n = int(input())

k = n \* 10

if n % 2 == 0:

s1 = tuple(n for n in range(k+1))

s2 = s1[0:len(s1):2]

print(s2)

else:

s1 = tuple(n for n in range(k+1))

s2 = s1[2:len(s1):2]

print(s2)

Листинг 37. K13\_2.py

#Именованные кортежи в Python.

#Именованные кортежи присваивают имя каждому значению элемента в кортеже #и тем самым создают более читаемый код.

#Они могут использоваться везде, где используются обычные кортежи и #добавляют возможность доступа к полям по

#имени вместо индекса позиции.

#Синтаксис:

#import collections

#ntuple = collections.namedtuple(typename, field\_names, \*, \

# rename=False, defaults=None, module=None)

#Параметры:

#typename - строка, имя именованного кортежа,

#field\_names - последовательность строк, имена элементов кортежа,

#rename - bool, авто-переименование повторяющихся имен элементов,

#defaults=None - итерируемая последовательность, значения по умолчанию #имен кортежа,

#module=None - атрибут \_\_module\_\_ именованного кортежа.

#Возвращаемое значение: новый подкласс кортежа с именем typename.

#Описание:

#Класс namedtuple() модуля collections возвращает новый подкласс кортежа #с именем typename. Новый подкласс

#используется для создания объектов, похожих на кортежи, которые имеют #индексируемые и итерируемые поля, доступные

#для поиска по атрибутам. Экземпляры подкласса также имеют полезную строку #документации с typename и field\_names,

#а так же метод \_\_repr\_\_(), который перечисляет содержимое кортежа в #формате name=value.

#Имена полей field\_names представляют собой последовательность строк, #таких как ['x', 'y']. В качестве альтернативы,

#field\_names может быть одной строкой, в которой каждое имя поля разделено #пробелами и/или запятыми, например,

#'x y' или 'x, y'.

#Для имен полей (элементов кортежа) может использоваться любой #действительный идентификатор Python, за исключением

#имен, начинающихся с подчеркивания. Допустимые идентификаторы состоят из #букв, цифр и символов подчеркивания,

#но не начинаются с цифры или символа подчеркивания и не могут быть #ключевыми словами, такими как class, for,

#return, global, pass и т. д.

#Если аргумент rename=True, то недопустимые имена полей автоматически #заменяются позиционными именами. Например

#['abc', 'def', 'ghi', 'abc'] преобразуется в ['abc', '\_1', 'ghi', '\_3'], # последовательностью. Поскольку аргумента со значением

#по умолчанию должны идти после любых обязательных #аргументов, то #значения по умолчанию будут применяются к #самым

#правым параметрам. Например, если имена полей #именованного кортежа это #['x', 'y', 'z'], а значения по #умолчанию

#(1, 2), то тогда x будет обязательным аргументом, y по #умолчанию будет #1, а z #будет 2.

#Если аргумент module определен, то атрибуту именованного #кортежа #\_\_module\_\_ присваивается значение module.

#Экземпляры именованных кортежей не имеют словарей, поэтому они #легковесны #и требуют не больше памяти, чем обычные

#кортежи.

#Пример

from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])

# создаем с позиционным или именованным параметром

p = Point(11, y=22)

# можно обращаться по индексу

# как к обычному кортежу

print(p[0] + p[1])

# 33

# распаковать как обычный кортеж

x, y = p

print(x, y)

# (11, 22)

# поля также доступны по названию

print(p.x + p.y)

# 33

# человеко-читаемый \_\_repr\_\_

print(p)

#Именованные кортежи поддерживают функцию getattr():

print(getattr(p, 'x'))

#Атрибуты и методы класса namedtuple():

#Метод ntuple.\_make() создает новый экземпляр класса namedtuple() из #существующей последовательности или

#итерации iterable.

t = [11, 22]

print(Point.\_make(t))

#Метод ntuple.\_asdict() вернет новый словарь dict, который отображает имена #полей в соответствии с их значениями:

p = Point(x=11, y=22)

print(p.\_asdict())

#Метод ntuple.\_replace() вернет новый экземпляр именованного кортежа, #заменив указанные поля новыми значениями:

p = Point(x=11, y=22)

print(p.\_replace(x=33))

#Свойство ntuple.\_fields вернет кортеж строк, перечисляющий имена полей. #Полезно для самоанализа и для создания

#новых именованных типов кортежей из существующих именованных кортежей.

print(p.\_fields)

Color = namedtuple('Color', 'red green blue')

Pixel = namedtuple('Pixel', Point.\_fields + Color.\_fields)

print(Pixel(11, 22, 128, 255, 0))

#Свойство ntuple.\_field\_defaults вернет словарь, который сопоставляет #имена полей со значениями по умолчанию.

Account = namedtuple('Account', ['type', 'balance'], defaults=[0])

print(Account.\_field\_defaults)

print(Account('premium'))

#Примеры использования именованного кортежа:

#Вот как добавить вычисляемое поле и формат печати фиксированной ширины:

class Box:

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

self.\_\_weight = 0

@property

def weight(self):

return self.\_\_weight

@weight.setter

def weight(self, new\_weight):

if new\_weight < 0:

raise ValueError('negative weight')

self.\_\_weight = new\_weight

b = Box()

b.weight = 100

print(b.weight)

b.weight = -100

print(b.weight)

1.13 Техника работы с файлами.

Листинг 38. K14\_1\_1.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

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

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

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

a.close()

Листинг 39. K14\_1\_2.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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 40. K15\_1\_1.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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.

#Синтаксис:

#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 исользования

deq = deque()

x = input()

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

#вывыести содержимое

while x != '.':

deq.appendleft(x)

x= input()

print(deq)

Листинг 41. K15\_1\_2.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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 42. K15\_2\_1.py

#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 43. K16\_1.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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 44. K16\_2.py

#Выполнил:Степаненко Кирилл

#Группа: П2-18

#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 45. K16\_3.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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 46. K16\_8.py

#Выполнил:Степаненко Кирилл   
#Группа: П2-18

#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\_\_)

# **Техника решения задач с использованием библиотек**

## Установка и настройка среды JetBrains PyCharm

PyCharm - это кросс-платформенная среда разработки.

Переходим на страницу скачивания PyCharm <https://www.jetbrains.com/ru-ru/pycharm/>. Для скачивания доступно две версии: профессиональная и версия для сообщества. Версия для сообщества бесплатная. Ее и скачаем.

Запускаем скачанный .exe файл. В первом окне нас приветствует сам установщик. Смело кликаем «Next»:



Рисунок 8. Установщик PyCharm



Рисунок 9.Место установки PyCharm



Рисунок 10. Процесс установки PyCharm



Рисунок 11. Стартовое окно PyCharm

В следующем окне будет два поля. В первом укажите, где будет располагаться новый проект. Второе поле должно быть заполненным помолчанию. Там содержится путь к установленному ранее Python 

Рисунок 12. Выбор места для нового проекта и выбор интерпретатора

Дальше откроется окно самого редактора.

Теперь вы готовы начать писать программы на Python!

Кликните правой кнопкой на название вашего проекта и в раскрывающемся списке выберите пункт «New > Python file»: 

Рисунок 13. Создание нового Phython файла

Появится окно, в котором вы можете задать имя файлу. Задайте какое-нибудь имя и нажмите «OK».

Справа откроется сам файл. Пока он пустой. Пишем:



Рисунок 14. Написание программы

Теперь нужно запустить нашу небольшую (1 строка) программу. Для этого в верхнем меню перейдите в «Run > Run…».

﷟HYPERLINK "https://neuralnet.info/wp-content/uploads/2017/09/PyCharm\_4.png"

Рисунок 15. Запуск программы

В появившемся окне щелкните по названию вашего файла. Теперь программа запустится. В нижней части редактора должна появиться консоль с результатом



Рисунок 16. Выполнение программы

# Техника работы с базами данных

Приложения: sql\_bd.py

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.

Листинг 47 sql\_bd.py

import sqlite3  
  
  
class Sqliter:  
  
 *# Функция \_\_init\_\_ служит инициализацией БД* def \_\_init\_\_(self, name\_db):  
 self.connection = sqlite3.connect(name\_db)  
 self.cursor = self.connection.cursor()  
  
 *# Функция create\_table создаёт таблицу в нашей БД* def create\_table(self, table):  
 with self.connection:  
 table = ''.join(chr for chr in table if chr.isalnum())  
 if table != '':  
 self.cursor.execute(f"CREATE TABLE IF NOT EXISTS `{table}` (\  
 id integer PRIMARY KEY,\  
 `fio` text, `group` text, `direction` text)")  
 self.save()  
  
 *# Функция add\_student добавляет студента в БД, если его не существует* def add\_student(self, \*\*kwargs):  
 with self.connection:  
 data = kwargs  
 if kwargs.get('data') != None:  
 data = kwargs['data']  
 self.cursor.execute("INSERT INTO `students`\  
 (`fio`, `group`, `direction`) VALUES (?, ?, ?)",  
 (data['fio'], data['group'], data['direction']))  
 self.save()  
  
 *# Функция get\_id возвращает id записи в таблице* def get\_id(self, \*\*kwargs):  
 with self.connection:  
 try:  
 data = kwargs  
 if kwargs.get('data') != None:  
 data = kwargs.get('data')  
 return self.cursor.execute("SELECT \* FROM `students` WHERE \  
 `fio` = ? AND `group` = ? AND `direction` = ?",  
 (data['fio'], data['group'], data['direction'])).fetchall()[0][0]  
 except:  
 return -1  
  
 *# Функция save сохраняет изменения в БД* def save(self):  
 self.connection.commit()  
 print(f"{self.cursor.rowcount} отредактированно строк")  
  
 *# Функция close закрывает БД* def close(self):  
 self.connection.close()  
  
*# Функция input\_student возвращает данные о студенте в нужном для БД формате*def input\_student(\*\*kwargs):  
 data = kwargs  
 if kwargs.get('data') != None:  
 data = kwargs.get('data')  
 student = {  
 'fio': data['fio'],  
 'group': data['group'],  
 'direction': data['direction']  
 }  
 return student  
  
*# Создание БД*bd = Sqliter("bd")  
*# Создание таблицы students*bd.create\_table('students')  
  
  
student = input\_student(fio="Cipkov Il'ya Vladimirovich",\  
 group="P1-18", direction="Programmer")  
*# Если студента не находит в БД, то мы его добавляем в БД*if bd.get\_id(data=student) == -1:  
 bd.add\_student(data=student)  
*# Если такой студент существует, то в консоль выводит "Ne mogu("*else:  
 print('Ne mogu(')  
  
*# Прекращение работы с БД*bd.close()

После завершения кода в той же папке где находится python-file появится файл с БД и в ней должна появиться запись «*1 Stepanenko Kirill Alexeevich P2-18 Programmer*». Просто так его открыть и посмотреть нельзя.

Для того чтобы увидеть БД, необходимо будет скачать DB Browser (SQLite). Переходим на страницу скачивания программы <https://sqlitebrowser.org/dl/>.

Рисунок 17. Установщик DB Browser



Рисунок18. Окно DB Browser

После установки, открываем DB Browser для работы с ним.

# Техника работы с библиотекой tkinter

Приложения: MyTable.py

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 48 MyTable.py

from tkinter import \*  
  
def main():  
 *#Функция save сохраняет введённые данные в строку и записывает строчку в файл* def save():  
 str = get\_name() + ' ' + group\_get() + ' ' + show\_parametrs() + m\_g()  
 with open("test.txt", "a") as f:  
 f.write(str + '\n')  
  
 *# Функция group\_get возвращает выбранную группу пользователем* def group\_get():  
 return my\_var.get()  
  
 *# Функция show\_parametrs возвращает все языки, которые выбрал пользователь* def show\_parametrs():  
 str = ""  
 for name, var in parametrs:  
 if var.get():  
 str += name + ' '  
 return str  
  
 *# Функция get\_name возвращает имя введённое пользователем* def get\_name():  
 return my\_var2.get()  
  
 *# Функция delete очищает текстовоый файл полностью* def delete():  
 f = open('test.txt', 'w+')  
 f.seek(0)  
 f.close()  
  
 *# Функция m\_g возвращает выбранный пол пользователем* def m\_g():  
 if v.get() == 1:  
 return "Мужской"  
 else:  
 return "Женский"  
  
 *# Создание объекта окна верхнего уровня (на нём будут распологаться все элементы* root = Tk()  
 *# Задание размера окна и его координат расположения* root.geometry("300x300+700+300")  
 *# StringVar используется для легкого отслеживания изменений в переменных tkinter* my\_var = StringVar('')  
 my\_var2 = StringVar('')  
 parametrs = (("Java", IntVar()), ("C#", IntVar()), ("C++", IntVar()),\  
 ("Python", IntVar()), ("Pascal", IntVar()))

*# Заполнение и задание расположения CheckBox'ов* x, y = 2, 0  
 for name, var in parametrs:  
 Checkbutton(root, text=f"{name}", variable=var).grid(row=x, column=y)  
 x += 1  
  
 *# Задаём местоположение текстовой метке и полю для ввода* lb1 = Label(text="Имя:").grid(row=0, column=0)  
 en = Entry(width=30, textvariable=my\_var2).grid(row=0, column=1, columnspan=3)  
  
 *# Задаём местоположение текстовой метке и SpinBox'у(в котором находится список групп)* lb2 = Label(text="Группа:").grid(row=1, column=0, columnspan=1)  
 mas = ('П1-18', 'П2-18')  
 sb = Spinbox(width=7, values=mas, textvariable=my\_var,\  
 command=group\_get).grid(row=1, column=1)

*# Задаём местоположение кнопкам Save и Delete*  
b = Button(root, text="Save", width=25, height=2, command=save).grid(row=9, column=1, columnspan=2)  
 b1 = Button(root, text="Delete", width=25, height=2,\  
 command=delete).grid(row=10, column=1, columnspan=2)  
  
 *# Создаём и задаём местоположение текстовой метке и RadioButton'ам*  
v = IntVar()  
 lb3 = Label(text="Пол:").grid(row=8, column=0)  
 rb1= Radiobutton(root, text="Мужской", value=1,\  
 variable=v).grid(row=8, column=1, columnspan=1)  
 rb2 = Radiobutton(root, text="Женский", value=2,\  
 variable=v).grid(row=8, column=2, columnspan=1)  
  
 *# Запуск цикла обработки событий*  
mainloop()  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Техника работы с библиотекой NumPy

Приложения: primer.py

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>

Импорт модуля numpy:

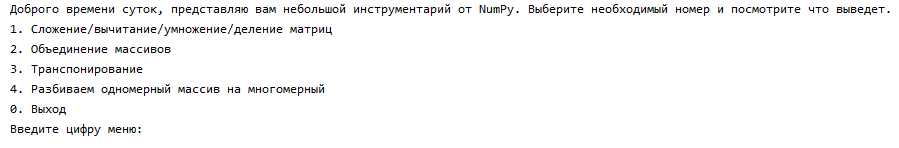
import numpy

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 49 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 19. Вывод меню в консоль

# Техника работы с библиотекой Matplotlib

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

Чтобы установить библиотеку Matplotlib введите в терминале команды:

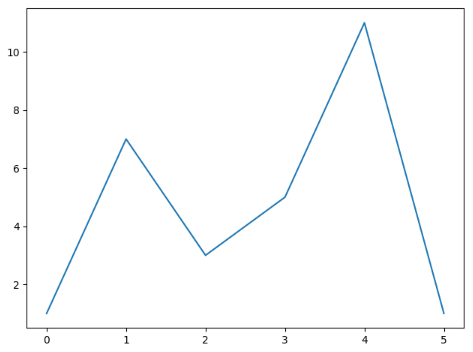
python -m pip install -U pip

python -m pip install -U matplotlib

1. **Построение графиков.**
   1. Построение одного графика.

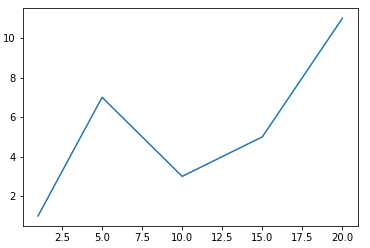
**import** matplotlib.pyplot **as** plt  
plt.plot([1, 7, 3, 5, 11, 1])  
plt.show()

Если в качестве параметра функции plot() передать список, то значения из этого списка будут отложены по оси ординат (ось y), а по оси абсцисс (ось x) будут отложены индексы элементов массива:



Для того, чтобы задать значения по осям x и y необходимо в plot() передать два списка:

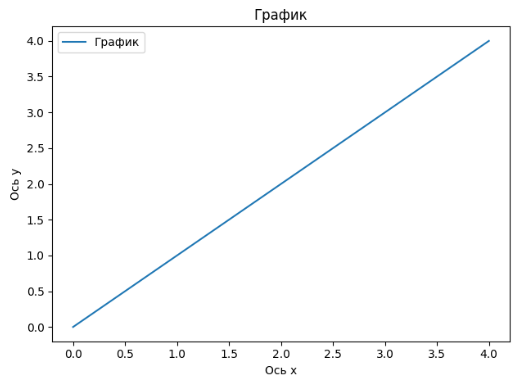
plt.plot([1, 5, 10, 15, 20], [1, 7, 3, 5, 11])



Оформление графика:

Листнинг 50 mat1.py

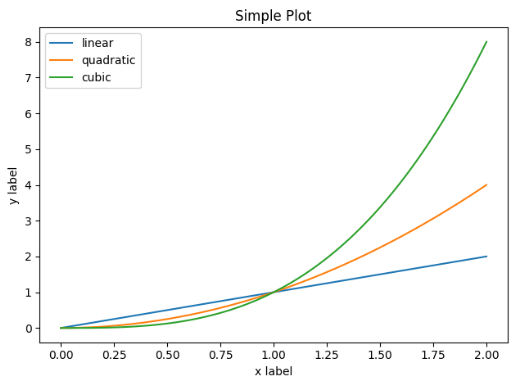
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
n = [0, 1, 2, 3, 4]  
fig, ax = plt.subplots() *# Получение фигуры и оси*ax.plot(n, n, label=**"График"**) *# Начертить график*ax.set\_xlabel(**"Ось x"**) *# Подписи осей*ax.set\_ylabel(**"Ось y"**)  
ax.set\_title(**"График"**) *# Заголовок фигуры*ax.legend() *# Показать легенду*plt.show()



* 1. Построение нескольких графиков на одной фигуре

Листнинг 51 mat2.py

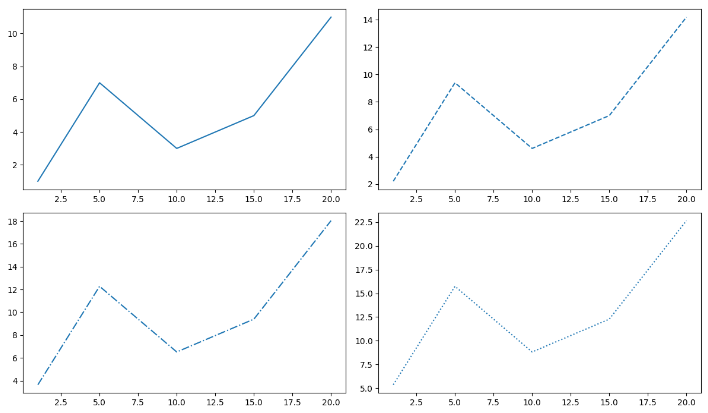
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
n = np.linspace(0, 2, 100) *# 100 чисел в интервале [0, 2]*fig, ax = plt.subplots()  
ax.plot(n, n, label=**"linear"**) *# 1 график*ax.plot(n, n\*\*2, label=**"quadratic"**) *# 2 график*ax.plot(n, n\*\*3, label=**"cubic"**) *# 3 график*ax.set\_xlabel(**"x label"**)  
ax.set\_ylabel(**"y label"**)  
ax.set\_title(**"Simple Plot"**)  
ax.legend()  
plt.show()



* 1. Размещение графиков на разных полях

Листнинг 52 mat3.py

**import** matplotlib.pyplot **as** plt  
*# Исходный набор данных*x = [1, 5, 10, 15, 20]  
y1 = [1, 7, 3, 5, 11]  
y2 = [i\*1.2 + 1 **for** i **in** y1]  
y3 = [i\*1.2 + 1 **for** i **in** y2]  
y4 = [i\*1.2 + 1 **for** i **in** y3]  
*# Настройка размеров подложки*plt.figure(figsize=(12, 7))  
*# Вывод графиков*plt.subplot(2, 2, 1)  
plt.plot(x, y1, **'-'**)  
plt.subplot(2, 2, 2)  
plt.plot(x, y2, **'--'**)  
plt.subplot(2, 2, 3)  
plt.plot(x, y3, **'-.'**)  
plt.subplot(2, 2, 4)  
plt.plot(x, y4, **':'**)  
plt.show()

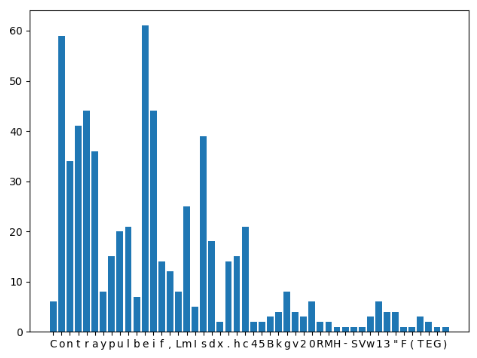


1. **Построение гистограммы.**

Напишем программу, которая подсчитывает количество различных символов и выводит гистограмму с их частотами.

Листнинг 53 mat4.py

**import** matplotlib.pyplot **as** plt  
d = {}  
**with** open(**"data.txt"**, **"r"**) **as** f:  
 **for** line **in** f:  
 **for** c **in** line:  
 **if** c != **' ' and** c != **'\n'**:  
 **if** c **not in** d:  
 d.update({c : 1})  
 **else**:  
 d[c] += 1  
  
fig, ax = plt.subplots()  
plt.bar(d.keys(), d.values())  
plt.show()

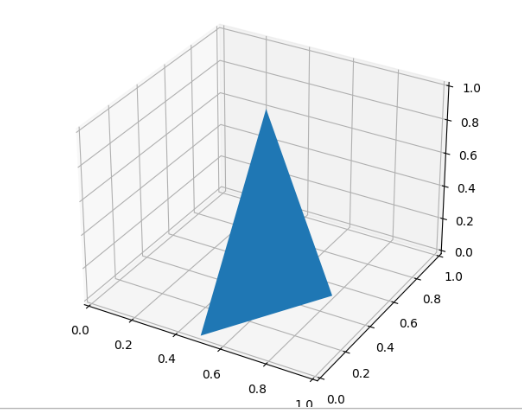


1. **Построение 3D.**

Построение треугольника в 3D фигуре:

Листнинг 54 mat5.py

**from** mpl\_toolkits.mplot3d **import** Axes3D  
**import** matplotlib.pyplot **as** plt  
**from** mpl\_toolkits.mplot3d.art3d **import** Poly3DCollection  
fig = plt.figure()  
ax = Axes3D(fig)  
triangle = [((0.8,0.5,0), (0.5,0,0), (0.5, 0.5, 1))]  
ax.add\_collection(Poly3DCollection(triangle))  
plt.show()



## 2.6 Техника работы с библиотекой PyQt.

Листинг 55 pyqt.py

import sys

from PyQt5.QtWidgets import QApplication

from PyQt5.QtWidgets import QLabel

from PyQt5.QtWidgets import QPushButton

from PyQt5.QtWidgets import QVBoxLayout

from PyQt5.QtWidgets import QWidget

def greeting():

if msg.text():

msg.setText("")

else:

msg.setText("321")

app = QApplication(sys.argv)

window = QWidget()

window.setWindowTitle('Signals and slots')

layout = QVBoxLayout()

btn = QPushButton('Press me!')

btn.clicked.connect(greeting)

layout.addWidget(btn)

msg = QLabel('')

layout.addWidget(msg)

window.setLayout(layout)

window.show()

sys.exit(app.exec\_())

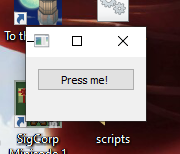


Рисунок 20. Запущенная программа с кнопкой “Press me!”.

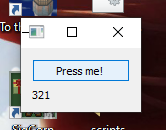


Рисунок 21. При нажатии на кнопку “Press me!” отображается число 321, при повторном нажатии число исчезает.

# 2.7 Элементы работы с библиотекой PyGame

Приложения: main.py

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. В Ubuntu и родственных дистрибутивах это можно сделать двумя способами – с помощью pip и apt-get:

python3 -m pip install -U pygame --user

или

sudo apt-get install python3-pygame

Для Windows:

Открыв командную строку (cmd) прописываем:

pip install pygame

Убедиться, что библиотека загружена, и что все установилось нормально, можно опять же в командной строке (cmd). Следует вписать следующие две строчки:

Python

Import pygame

Листинг 56 main.py

import pygame, sys, random, platform

from pygame.locals import \*

SIZE = 20

WINDOWWIDTH = 20\*SIZE

WINDOWHEIGHT = 21\*SIZE + 1

FPS = 10

def terminate():

pygame.quit()

sys.exit()

def changeColour(x, y, colour): #Изменяем цвет области в размере квадрата 20\*20 пикслей

for i in range(x\*SIZE, (x+1)\*SIZE):

for j in range(y\*SIZE, (y+1)\*SIZE):

pixArray[i][j] = colour

return None

def getFoodLocation(): #Рандомный спавн еды для змейки

x = random.randint(0, 19)

y = random.randint(0, 19)

while(pixArray[x\*SIZE][y\*SIZE] != windowSurface.map\_rgb(WHITE)):

x = random.randint(0, 19)

y = random.randint(0, 19)

changeColour (x, y, RED)

return x, y

def drawText(text, font, surface, x, y):

textobj = font.render(text, 1, TEXTCOLOUR)

textrect = textobj.get\_rect()

textrect.bottomright = (x, y)

surface.blit(textobj, textrect)

UP = "U"

DOWN = "D"

LEFT = "L"

RIGHT = "R"

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

RED = (233, 86, 78)

GREEN = (0, 255, 0)

BLUE = (78, 174, 233)

TEXTCOLOUR = BLACK

pygame.init()

mainClock = pygame.time.Clock()

windowSurface = pygame.display.set\_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)

pygame.display.set\_caption('S N A K E')

windowSurface.fill(WHITE)

if(platform.system() == 'Windows'):

font = pygame.font.SysFont("Consolas", 19)

elif(platform.system() == 'Linux'):

font = pygame.font.SysFont("Ubuntu", 19)

pygame.display.update()

pixArray = pygame.PixelArray(windowSurface)

topScore = '000'

#Общий игровой цикл

while True:

#Рестарт игры

snake = [[2, 0, RIGHT], [1, 0, RIGHT], [0, 0, RIGHT]]

swapper = ''

windowSurface.fill(WHITE)

for i in snake:

changeColour(i[0], i[1], BLUE)

x, y = getFoodLocation()

#Игровой цикл который запускает игру

while True:

#Получаем готовность пользователя

pixArray = pygame.PixelArray(windowSurface)

for event in pygame.event.get():

if event.type == QUIT:

terminate()

if event.type == KEYDOWN:

if event.key == K\_LEFT or event.key == ord('a') :

swapper = LEFT

elif event.key == K\_RIGHT or event.key == ord('d'):

swapper = RIGHT

elif event.key == K\_DOWN or event.key == ord('s'):

swapper = DOWN

elif event.key == K\_UP or event.key == ord('w'):

swapper = UP

orig = swapper

#Предотвращение движения хвоста змеи в любом направлении

if ((snake[0][2] == UP and swapper == DOWN) or (snake[0][2] == DOWN and swapper == UP) or (snake[0][2] == RIGHT and swapper == LEFT) or (snake[0][2] == LEFT and swapper == RIGHT)):

swapper = snake[0][2]

#Управление змейкой

for i in range(len(snake)):

snake[i][2], swapper = swapper, snake[i][2]

if snake[i][2] == RIGHT:

snake[i][0] += 1

elif snake[i][2] == LEFT:

snake[i][0] -= 1

elif snake[i][2] == DOWN:

snake[i][1] += 1

elif snake[i][2] == UP:

snake[i][1] -= 1

if snake[i][0] > 19:

snake[i][0] = 0

elif snake[i][0] < 0:

snake[i][0] = 19

elif snake[i][1] < 0:

snake[i][1] = 19;

elif snake[i][1] > 19:

snake[i][1] = 0

swapper = orig #Очень важная строка , если ее удалить то можно сломать управление змейкой

#Game over

if [snake[0][0], snake[0][1], UP] in snake[1:] or [snake[0][0], snake[0][1], DOWN] in snake[1:] or [snake[0][0], snake[0][1], RIGHT] in snake[1:] or [snake[0][0], snake[0][1], LEFT] in snake[1:]:

break

#Блок кода который помогает змейке "телепортировать себя" при проходе через боковые стенки

if snake[0][0] == x and snake[0][1] == y:

x, y = getFoodLocation()

if snake[len(snake) - 1][2] == DOWN:

if snake[len(snake) - 1][1] >= 1:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]-1, DOWN])

else:

if snake[len(snake) - 1][0] >= 1:

snake.append([snake[len(snake)-1][0]-1, snake[len(snake)-1][1], RIGHT])

else:

snake.append([snake[len(snake)-1][0]+1, snake[len(snake)-1][1], LEFT])

if snake[len(snake) - 1][2] == UP:

if snake[len(snake) - 1][1] <= 18:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]+1, UP])

else:

if snake[len(snake) - 1][0] >= 1:

snake.append([snake[len(snake)-1][0]-1, snake[len(snake)-1][1], RIGHT])

else:

snake.append([snake[len(snake)-1][0]+1, snake[len(snake)-1][1], LEFT])

if snake[len(snake) - 1][2] == LEFT:

if snake[len(snake) - 1][0] <= 18:

snake.append([snake[len(snake)-1][0]+1, snake[len(snake)-1][1], LEFT])

else:

if snake[len(snake) - 1][1] >= 1:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]-1, DOWN])

else:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]+1, UP])

if snake[len(snake) - 1][2] == RIGHT:

if snake[len(snake) - 1][0] >= 1:

snake.append([snake[len(snake)-1][0]-1, snake[len(snake)-1][1], RIGHT])

else:

if snake[len(snake) - 1][0] >= 1:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]-1, DOWN])

else:

snake.append([snake[len(snake)-1][0], snake[len(snake)-1][1]+1, UP])

windowSurface.fill(WHITE)

for i in snake:

changeColour(i[0], i[1], BLUE)

changeColour(x, y, RED)

for i in range(400):

pixArray[i][400] = BLACK

del pixArray

score = str(len(snake))

while len(score) < 3:

score = '0' + score

while len(topScore) < 3:

topScore = '0' + topScore

if int(score) > int(topScore):

topScore = score

drawText(score, font, windowSurface, 399, 420)

drawText(topScore, font, windowSurface, 170, 420)

drawText("SCORE : ", font, windowSurface, 369, 420)

drawText("HIGH SCORE : ", font, windowSurface, 140, 420)

pygame.display.update()

mainClock.tick(FPS)

# **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. Название книги, автор, год выпуска книги, цена.

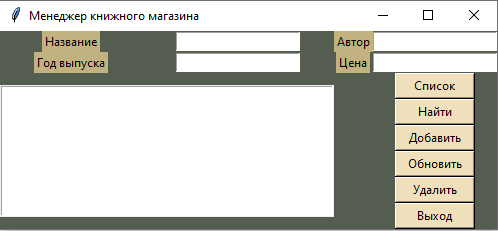


Рисунок 22. Окно ввода данных

Выходные данные

1. Список книг из БД
2. База данных

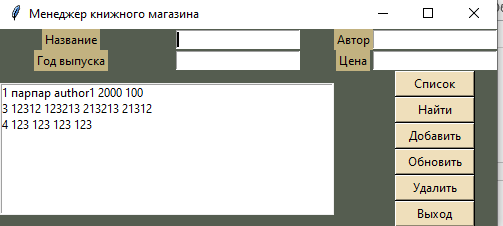


Рисунок 23. База данных в окне программы

3.2 Разработка требований к проекту. Построение диаграммы использования.

Требования к проекту:

1. Быстрая работа с БД
2. Правильность оформления документов
3. Простота просмотра БД и ее редактирование

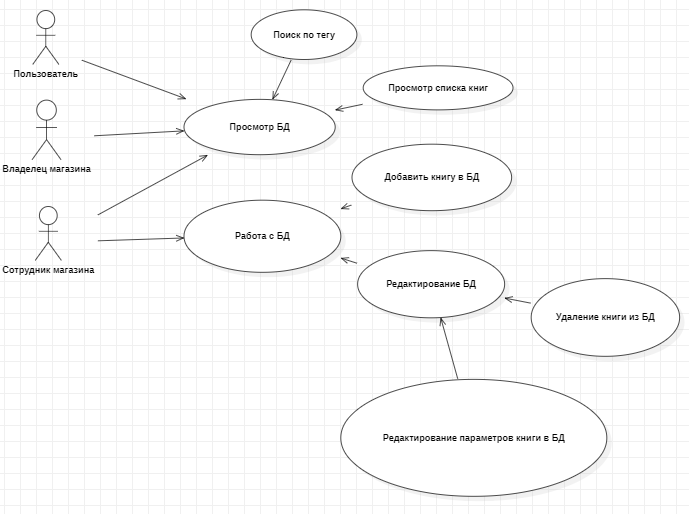


Рисунок 24. Диаграмма использования

* 1. Разработка сценария проекта

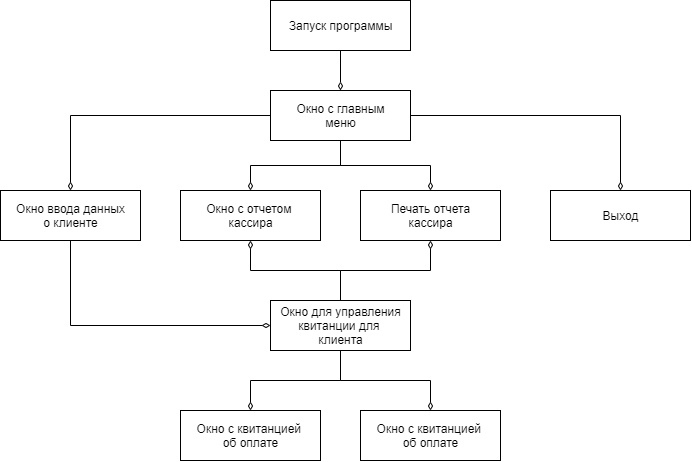
В данном разделе приведен сценарий использования программы пользователем .

Рисунок 25. Сценарий проекта

* 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, как и с приложением в целом, было решено написать отдельный класс backend.

Листинг 57: [backend.py](https://github.com/prog-kkkmt/p1-18/blob/Gymrasimov/%D0%A3%D0%9F/project_w_bd/sqlighter.py)

import sqlite3, time, os

from pprint import pprint

def connect():

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS book (id INTEGER PRIMARY KEY, title TEXT , author TEXT, year INTEGER, price INTEGER)")

conn.commit()

conn.close()

def view():

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("SELECT \* FROM book")

rows = cur.fetchall()

conn.close()

return rows

def insert(title, author, year, price):

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("INSERT INTO book VALUES (NULL,?,?,?,?)",(title, author, year, price))

conn.commit()

conn.close()

def delete(id):

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("DELETE FROM book WHERE id = ?", (id,))

conn.commit()

conn.close()

def search(title = " ", author = " ", year = " ", price = " "):

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("SELECT \* FROM book WHERE title = ? OR author = ? OR year = ? OR price = ?",(title, author, year, price))

rows = cur.fetchall()

conn.close()

return rows

def update(id, title, author, year, price):

conn = sqlite3.connect("books.db")

cur = conn.cursor()

cur.execute("UPDATE book SET title = ?, author = ?, year = ?, price = ? WHERE id = ?",(title, author, year, price, id))

conn.commit()

conn.close()

connect()

* 1. Разработка главного модуля

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

Листинг 58. Главный модуль main.py

#Импортируем библиотеки и подключаем второй файл

from tkinter import \*

import backend

main\_win = Tk() #Создание главного окна

main\_win.title("Менеджер книжного магазина") #Меняем название

main\_win["bg"] = "#555D50"

#При нажатии на строку в списке автоматически заполняем поля ввода данных(помогает если вы удалили что-то случайно из списка и хотите вернуть)

def get\_selected\_row(event):

global selected\_tuple

index = list\_box.curselection()[0]

selected\_tuple = list\_box.get(index)

e\_title.delete(0, END)

e\_title.insert(END, selected\_tuple[1])

e\_author.delete(0, END)

e\_author.insert(END, selected\_tuple[2])

e\_year.delete(0, END)

e\_year.insert(END, selected\_tuple[3])

e\_price.delete(0, END)

e\_price.insert(END, selected\_tuple[4])

#Настройка кнопок

def view\_command(): #Вывести строки из базы данных на экран

list\_box.delete(0, END)

for row in backend.view():

list\_box.insert(END, row)

def search\_command(): #Поиск по тегам

list\_box.delete(0, 100)

for row in backend.search(title\_text.get(), author\_text.get(), year\_text.get(), price\_text.get()):

list\_box.insert(END, row)

e\_title.delete(0, END)

e\_author.delete(0, END)

e\_year.delete(0, END)

e\_price.delete(0, END)

def add\_command(): #Добавить строку в базу данных

backend.insert(title\_text.get(), author\_text.get(), year\_text.get(), price\_text.get())

list\_box.delete(0, END)

list\_box.insert(END,(title\_text.get(), author\_text.get(), year\_text.get(), price\_text.get()))

e\_title.delete(0, END)

e\_author.delete(0, END)

e\_year.delete(0, END)

e\_price.delete(0, END)

def update\_command(): #Изменить строку в базе данных

backend.update(selected\_tuple[0], title\_text.get(), author\_text.get(), year\_text.get(), price\_text.get())

view\_command()

def delete\_command(): #Удалить строку из базы данных

backend.delete(selected\_tuple[0])

view\_command()

#Названия и вводы данных

l\_title = Label(main\_win, text = "Название", bg = "#C2B280")

l\_title.grid(row = 0, column =0)

title\_text = StringVar()

e\_title = Entry(main\_win, textvariable = title\_text)

e\_title.grid(row = 0, column = 1)

l\_author = Label(main\_win, text = "Автор",bg = "#C2B280")

l\_author.grid(row = 0, column = 2)

author\_text = StringVar()

e\_author = Entry(main\_win, textvariable = author\_text)

e\_author.grid(row = 0, column = 3)

l\_year = Label(main\_win, text = "Год выпуска",bg = "#C2B280")

l\_year.grid(row = 1, column = 0)

year\_text = StringVar()

e\_year = Entry(main\_win, textvariable = year\_text)

e\_year.grid(row = 1, column = 1)

l\_price = Label(main\_win, text = "Цена",bg = "#C2B280")

l\_price.grid(row = 1, column = 2)

price\_text = StringVar()

e\_price = Entry(main\_win, textvariable = price\_text)

e\_price.grid(row = 1, column = 3)

list\_box = Listbox(main\_win, height = 8 , width = 55)

list\_box.grid(row = 2, column = 0, rowspan = 6, columnspan = 2)

list\_box.bind("<<ListboxSelect>>", get\_selected\_row)

#Кнопки

b\_view = Button(main\_win, text = "Список", width = 10, bg = "#EFDFBB",command = view\_command)

b\_view.grid(row = 2, column =3)

b\_search = Button(main\_win, text = "Найти", width = 10, bg = "#EFDFBB",command = search\_command)

b\_search.grid(row = 3, column = 3)

b\_add = Button(main\_win, text = "Добавить", width = 10, bg = "#EFDFBB",command = add\_command)

b\_add.grid(row = 4, column =3)

b\_update = Button(main\_win, text = "Обновить", width = 10, bg = "#EFDFBB",command = update\_command)

b\_update.grid(row = 5, column = 3)

b\_delete = Button(main\_win, text = "Удалить", width = 10, bg = "#EFDFBB",command = delete\_command)

b\_delete.grid(row = 6, column = 3)

b\_exit = Button(main\_win, text = "Выход", width = 10, bg = "#EFDFBB",command = main\_win.destroy)

b\_exit.grid(row = 7, column = 3)

main\_win.mainloop()#Вывод приложения на экран

* 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. Разработка документации. |  |
| 23.04.2021 | Разработка документации. Защита проекта. |  |
| 26.04.2021 | Защита проекта. Сдача зачёта по практике. |  |