



Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

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

Выполнили студенты:

Цыпков И.В и Гусев.Н.С.

_____ (подпись)

Гусятинер Л. Б.

_____ (подпись)

_____ (оценка)

Содержание отчёта

Раздел 1. Техника решения задач с использованием структурного программирования.	3
1.1 Установка интерпретатора Python 3 и настройка окружения	3
1.2 Техника работы в командной строке и среде IDLE	6
1.3 Техника работы с линейными и разветвляющимися программами	8
1.4 Техника работы с циклическими программами, цикл while	12
1.5 Техника работы с числами	17
1.6 Техника работы со строками	20
1.7 Техника работы со списками	24
1.8 Техника работы с циклом for и генераторами списков	26
1.9 Техника работы с функциями	27
1.10 Техника работы с словарями	27
1.11 Техника работы с множествами	28
1.12 Техника работы с кортежами	29
1.13 Техника работы с файлами	32
1.14 Техника работы с модулями	33
1.15 Техника работы с классами	40

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/>

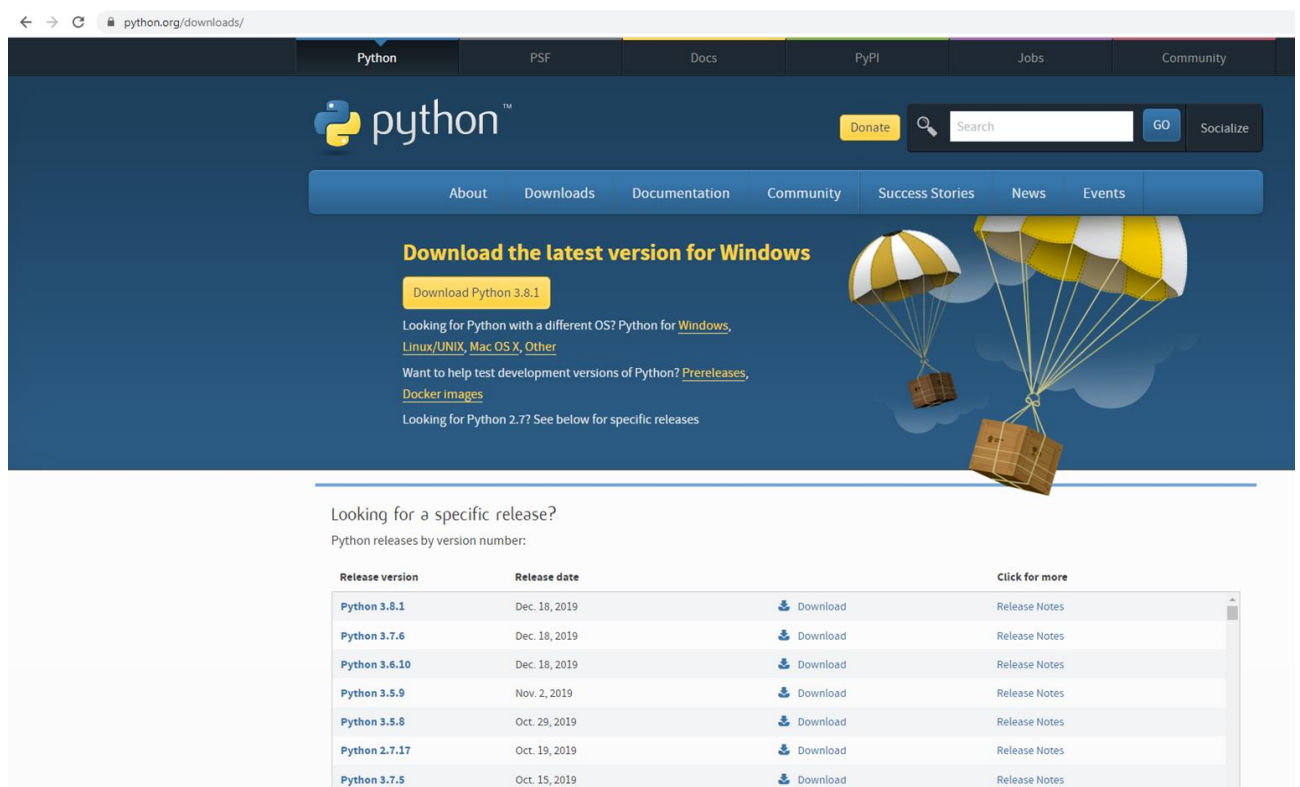


Рисунок 1. Официальный сайт Python

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)

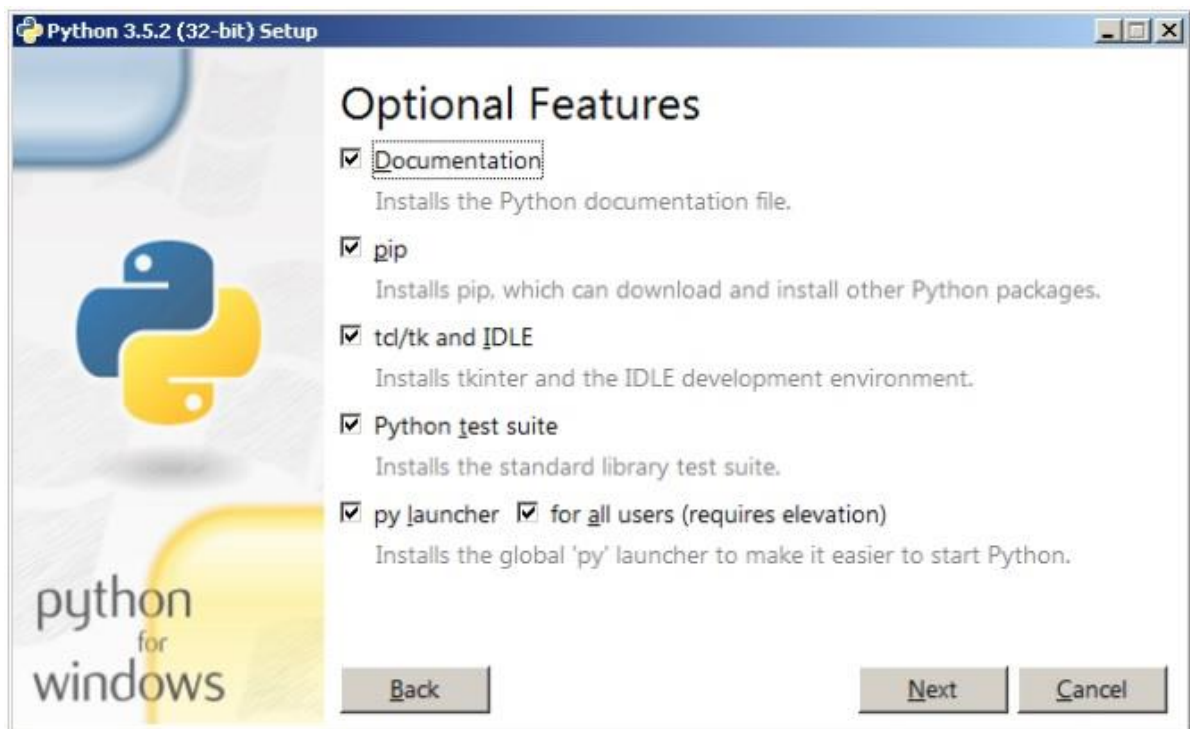


Рисунок 3. Опции установки

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)

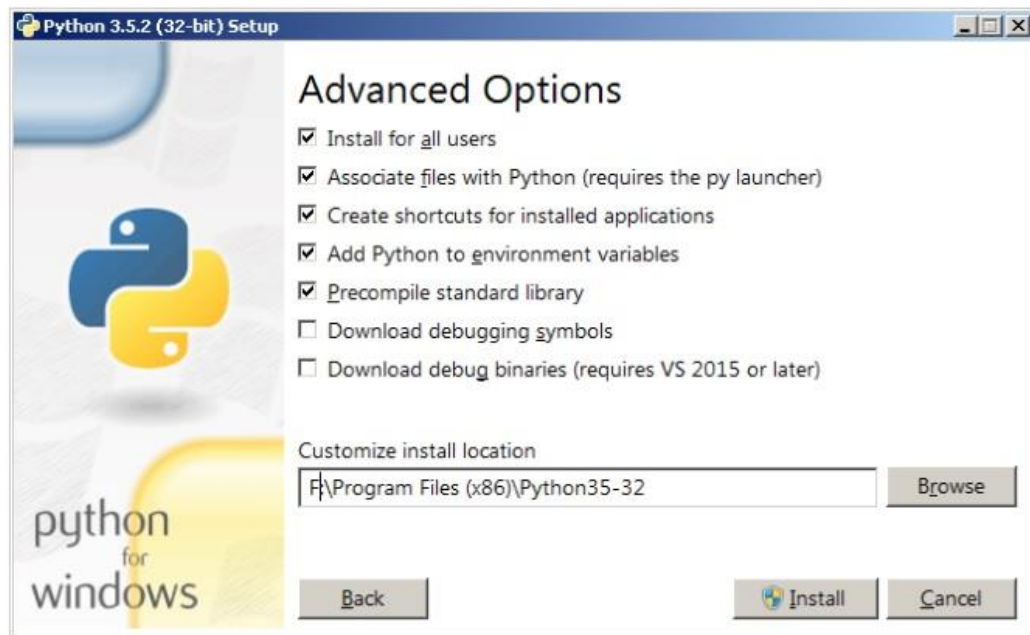


Рисунок 4. Продвинутые опции установки

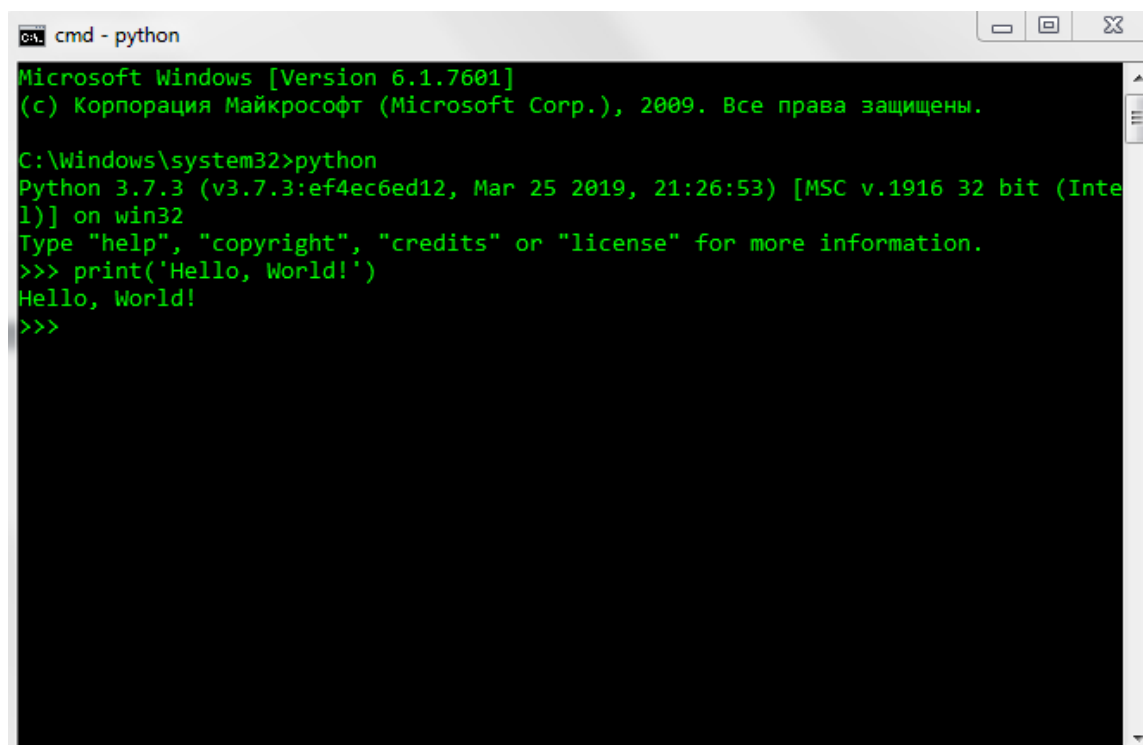
5. После успешной установки:



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.



```
cmd - python
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Windows\system32>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, World!')
Hello, World!
>>>
```

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 - 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
```

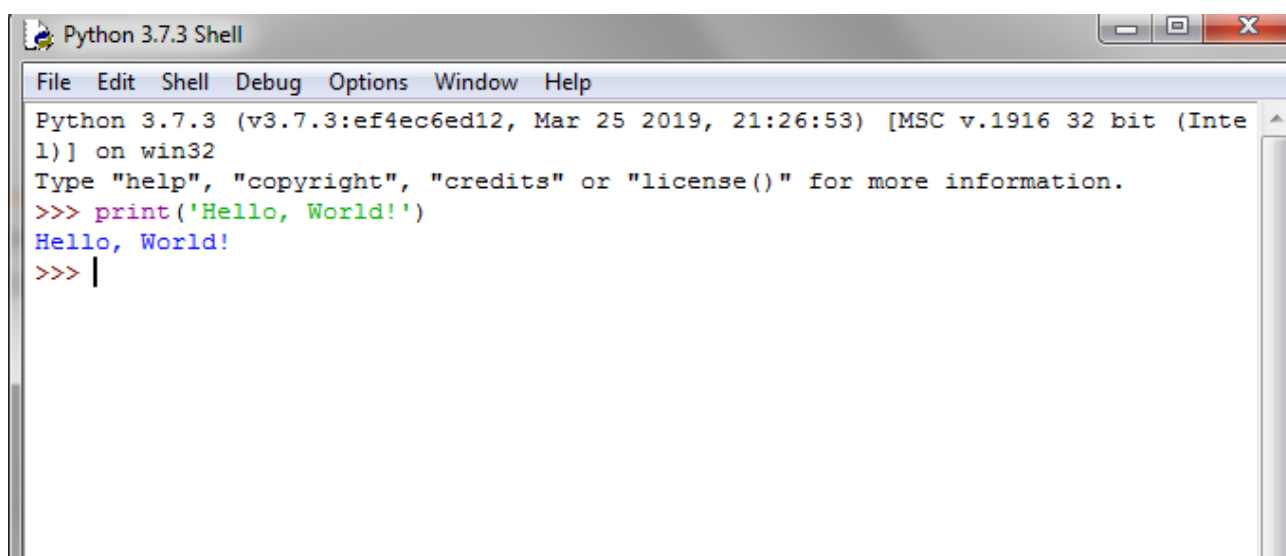


Рисунок 7. Первая программа

1.3 Техника работы с линейными и разветвляющимися программами

Листинг 1. K4_1.py data.txt text.txt

```
# Выполнили:Гусев Никита и Цыпков Илья
# Группа: П1-18

# Задание. Разработать программы по темам
# - input
# - print

name = str(input())

print("Hello,", name)
```

Листинг 1. К4_2_1.

```
#Выполнили: Цыпков И. и Гусев Н.
#Группа: П1-18
'''
#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', 'восемидесятого', 'девяностого')
y5_list = (
    'двадцать', 'тридцать', 'сорок', 'пятьдесят',
    'шестьдесят', 'семьдесят', 'восемьдесят', 'девяносто')
y6_list = (
```



```
                break
            k+=1
        g+=1
    j+=1
i+=1
```

1.4 Техника работы с циклическими программами, цикл while

Листинг 2. K5_1.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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.
#a) 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 inlside the ring:', counter)
print('Approximate number of pi:', Pi)
print('difference:', math.pi - 4*k/j)

```

Листинг 3. K5_1_2.

#Выполнили: Гусев Никита и Цыпков Илья

#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 4. K5_2_2

```
# Выполнили: Гусев Никита и Цыпков Илья

# 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 5. K5_2_3.

```
#Выполнили:Гусев Никита и Цыпков Илья

#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 6. K5_2_5.

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 7. К6 1.

```
#Выполнили: Цыпков Илья и Гусев Никита
#Группа: П1-18
'''
#К6_1. Техника работы с числами

#Задание 1.
#Подготовить инструкцию по использованию модулей fractions, decimal.
#С помощью модуля fractions выполним обычные математические функции.
#Подготовить инструкцию по использованию модулей fractions, decimal.
'''
```

Инструкция по использованию модуля fraction.

```
#В языке программирования Python для работы с рациональными числами
#предлагается класс Fraction. В классе соответствующим образом
#представлены числитель m и знаменатель n. В
#классе Fraction автоматически осуществляется упрощение дроби
#включить модуль fractions
#Пример:
#from fractions import Fraction
#Объект класса Fraction можно создать одним из двух способов.
#Способ 1. С помощью конструктора, который содержит целочисленные
значения #числителя и знаменателя.
#Пример.
#a = Fraction(5, 6) # a = 5/6 - рациональное число
#b = Fraction(8, 12) # b = 2/3 - рациональное число
#Способ 2. С помощью конструктора, который получает строку с
вещественным #значением.
#Пример.
a = Fraction('1.33') # a = 133/100
b = Fraction('3.719') # b = 3719/1000
c = Fraction('-1.5') # c = -3/2
d = Fraction('3.7') + Fraction('5.2') # d = 89/10

#Класс fraction может использоваться для вычисления легких
математических #функций.
```

Листинг 8. К6_1.

```
from fractions import Fraction

a = Fraction(3, 10)
b = Fraction(7, 8)
```

```
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

#Класс fraction может использоваться для вычисления НОД в дробях.

Листинг 9. К6_1.

```
from fractions import gcd

print(fractions.gcd(1000, 771))
print(fractions.gcd(0, 9))
print(fractions.gcd(-2, 9))
```

#Класс fraction может использоваться для преобразования в дробное число и #тип fraction.

Листинг 10. К6_1.

```
from fractions import Fraction

# преобразование в дробное число
a = (3.5).as_integer_ratio()
b = (11.7).as_integer_ratio()
print(a, b)

# преобразование в тип Fraction
a = 8.5
c = Fraction(*a.as_integer_ratio())
print(c)
```

Инструкция по использованию модуля decimal

```
# 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 11. K7_1_1.

#Выполнили: Гусев Никита и Цыпков Илья

#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 12. K7_1_2.py

#Выполнили: Гусев Никита и Цыпков Илья

#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 согдиться для логина.

#Если строка состоит из одного слова, менять ничего не нужно.

```
a = input()
b = a.replace(" ", "_")
print(b)
```

Листинг 13. K7_1_3.py

#Выполнили: Гусев Никита и Цыпков Илья

#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 14. K7_1_4.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 сообщение.
#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 15. K7_2.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 больше. 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']))
```

#Спецификация 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 16. K8_1_1.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 17. K8_1_2.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 18. K8_2_1.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 19. K8_2_3.py

```
#Выполнили: Гусев Никита и Цыпков Илья
```

```
#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 20. K9_2_1.py

```
#Выполнили:Гусев Никита и Цыпков Илья

#Array55. Дан целочисленный массив А размера N (<= 15). Переписать в
новый #целочисленный
#массив В все элементы с нечетными порядковыми номерами (1, 3, ...)
#ивывести размер
#полученного массива В и его содержимое. Условный оператор не
использовать.

import random
n = random.randrange(2, 15)
mass_a = [i for i in range(n)]
mass_b = a[1::2]
print(len(mass_b))
print(mass_b)
```

Листинг 21. K9_2_2.py

```
#Выполнили:Гусев Никита и Цыпков Илья

#Array57. Дан целочисленный массив А размера N. Переписать в новый
#целочисленный массив В
#того же размера вначале все элементы исходного массива с четными
#номерами,
#a затем – с нечетными:
#A[0], A[2], A[4], A[6], ..., A[1], A[3], A[5], ... .
#Условный оператор не использовать.

n = int(input())
mass_a = [i for i in range(n)]
mass_b = mass_a[0::2] + mass_a[1::2]
print(mass_b)

print(b)
print(len(b))
```

Листинг 22. K9_2_4.py

```
#Выполнили: Гусев Никита и Цыпков Илья
#Matrix3. Даны целые положительные числа М, N и набор из М чисел.
#Сформировать матрицу размера М 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 23. K10_1_.py

```

#Выполнили:Гусев Никита и Цыпков Илья

#Задание 2. Func6. Описать функцию SumRange(A, B) целого типа,
находящую
#сумму всех целых чисел от A до B включительно (A и B – целые). Если A
> #B,
#то функция возвращает 0. С помощью этой функции найти суммы чисел от
A #до B
#и от B до C, если даны числа A, B, C.

def SumRange(a, b):
    if a > b:
        return 0
    else:
        sum = 0
        while a <= b:
            sum += a
            a += 1
        return sum
A = int(input())
B = int(input())
C = int(input())
print(SumRange(A,B))
print(SumRange(B,C))

```

.

1.10 Техника работы с словарями.

Листинг 24. K11_1_.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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())
i = 0
while a <= b:
    print(a**c, end = ' ')
    a += 1
```

Листинг 25. K11_2_.py

```
#Выполнили:Гусев Никита и Цыпков Илья

#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 26. K12_1_.py

```
#Выполнили:Гусев Никита и Цыпков Илья

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

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

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

Листинг 27. K12_2_.py

```
#Выполнили:Гусев Никита и Цыпков Илья

#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))
```

1.12 Техника работы с кортежами.

Листинг 28. K13_1_.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 29. 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 применяться 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 именowanego #кортежа #__module__ присваивается значение module.

#Экземпляры именowanych 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 именowanym 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 30. K14_1_1.py

#Выполнили: Гусев Никита и Цыпков Илья

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

```
a = open('text.txt', 'a')
a.write(input()+'\n')
a.close()
```

Листинг 31. K14_1_2.py

#Выполнили: Гусев Никита и Цыпков Илья

#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 32. K15_1_1.py

#Выполнили: Гусев никита

#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 использования
deq = deque()
x = input()
#добавлять в обратном порядке до 1-го вхождения ',' и в конце
#вывести содержимое
while x != '.':
    deq.appendleft(x)
    x= input()
print(deq)

```

Листинг 33. K15_1_2.py

```

#Выполнил: Цыпков Илья
#Подготовить инструкцию по использованию модулей Counter.

#класс collections.Counter() предназначен для удобных и быстрых
подсчетов
#количества появлений неизменяемых элементов в последовательностях.

#from 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,
#a 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 = 'aabbccccdeff'

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 34. 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 35. K16_1_.py

```
#Выполнили: Гусев Никита и Цыпков Илья

#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 36. K16_2.py

```

#Выполнили: Гусев Никита и Цыпков Илья

#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 переменной 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 37. K16_3.py

#Выполнили: Гусев Никита и Цыпков Илья

```
#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 isinstance(sub, sup) возвращает значение True,  
#если данный подкласс sub действительно является подклассом sup.  
#Логическая функция issubclass(sub, sup) возвращает True, если sub  
#является экземпляром класса sup 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 34. K16_8.py

#Выполнили: Гусев Никита и Цыпков Илья

#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_consumption):
        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_consumption = fuel_consumption
    #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_consumption))
        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(carl.gas_tank_volume) ошибка
```

```
#В подклассах можно изменять методы классов  
machine1.what_is_this()  
carl.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__)
```