Программирование

на языке

Python

Блок 1:

самое необходимое.

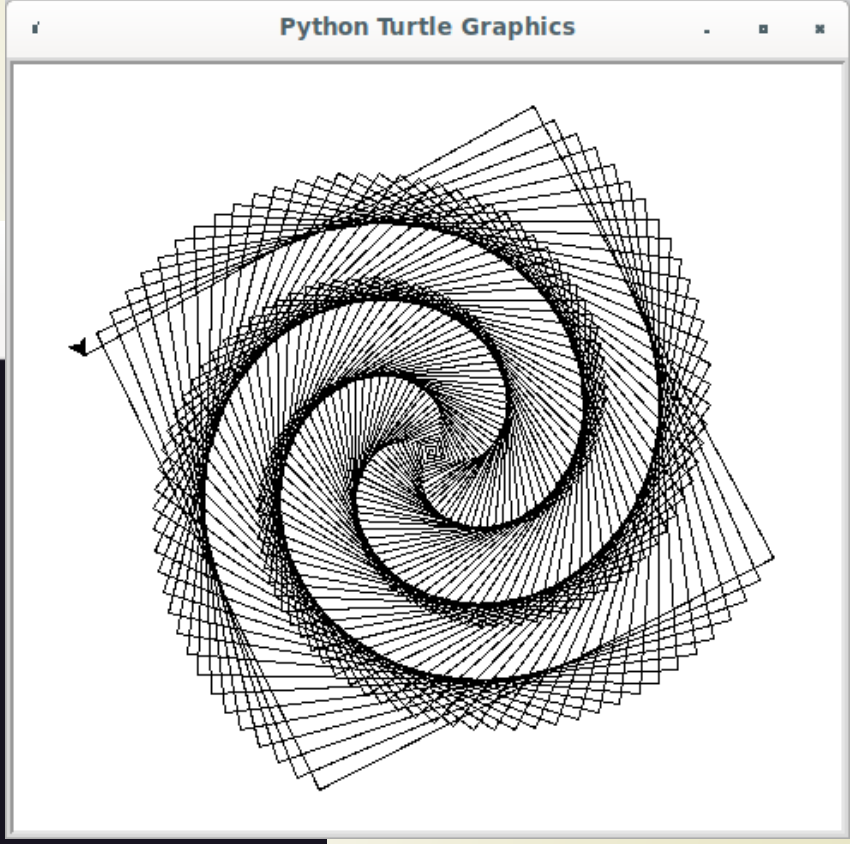
В. Б. Пикулев,
доцент КФТТ ПетрГУ.

Scilink.ru, 2024

Элементарно, Python

```
# перевести число из двоичной в десятичную систему счисления
s = input("Число в двоичной системе счисления: ")
print("Число в десятичной системе счисления: ", int(s, 2))
```

```
pikulev@VA-Space:~$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more
>>> import math
>>> A = math.pi
>>> print(A)
3.141592653589793
>>> print(A*2)
6.283185307179586
>>> print(str(A)*2)
3.1415926535897933.141592653589793
>>> print('Число Пи приблизительно равно ', A, ' - это все знают!')
Число Пи приблизительно равно  3.141592653589793  - это все знают!
>>> print(f'Число Пи приблизительно равно {A} - это все знают!')
Число Пи приблизительно равно 3.141592653589793 - это все знают!
>>> print(f'Число Пи приблизительно равно {A:4.3} - это все знают!')
Число Пи приблизительно равно 3.14 - это все знают!
>>>
>>> from turtle import *
>>> for i in range(300):
...     forward(i)
...     left(91)
```



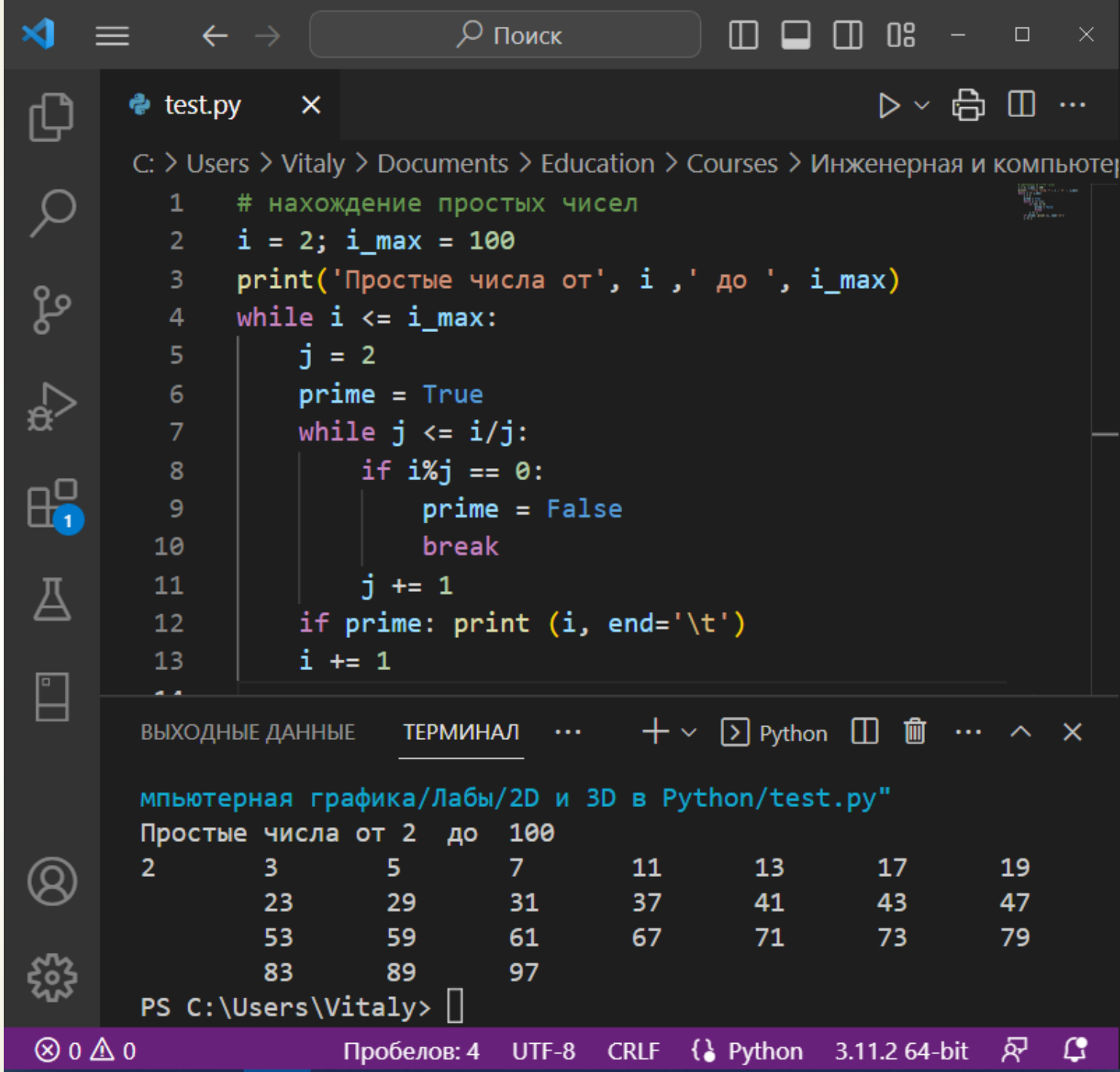
О красоте кода

3

```
# нахождение простых чисел - v2
for i in range(2, 101):
    j = 2
    while j <= i/j:
        if not(i % j): break
        j += 1
    else: print (i)
```

Если в теле цикла встречается **break**, то цикл завершается, при этом блок **else** не выполняется.

```
# нахождение простых чисел - v3
import math
print('2', end='\t')
for i in range(3,101,2):
    if all(i % j for j in
    range(2, 1+int(math.sqrt(i)))):
        print (i, end='\t')
```



Типы данных

`type(A)`

Название	Имя	Фикс.	Примеры
Булево значение	bool	да	True, False
Целое число	int	да	123, -2, 100_500
Вещественное число (с пл. точкой)	float	да	3.1415, 1.6e-19
Комплексное число	complex	да	3j, 0.12+2.5j
Текстовая строка	str	да	'пример', "ещё", '"и даже так..."'
Строка байтов	bytes	да	b'\x01\x02\x03\xff'
Массив байтов	bytearray	нет	bytearray(b'\x01\x02\x03\xff')
Список	list	нет	['Яблоко', 'Груша', 'Слива', 49]
Кортеж	tuple	да	(2.5, False, -1)
Множество	set	нет	set([1, 4, 7, 'строка', 7])
Фиксированное множество	frozenset	да	frozenset(['Elsa', 'Otto'])
Словарь	dict	нет	{'Фрукт': 'Яблоко', 'Год': 2024}
Ничто	None	да	A = None

Списки, кортежи, множества и словари

- **Список** наиболее близок к обычному типизированному массиву и является изменяемой последовательностью, т.е. можно различным образом модифицировать элементы списка. Оформляется в **[квадратные]** скобки.
- В отличие от списка, **кортеж** является неизменяемой последовательностью - нельзя менять его структуру (добавлять, удалять и изменять значения элементов). Оформляется в **(круглые)** скобки.
- **Словарь** очень напоминает *ассоциативный массив* в других языках программирования и представляет собой неупорядоченную коллекцию значений произвольных типов. Словарь состоит из набора пар «ключ: значение». В качестве ключа может выступать любой неизменяемый тип данных, в том числе и кортеж. Можно изменить значение для ключа, можно добавить и удалить любую пару "ключ: значение". Для задания словаря используются **{фигурные}** скобки.
- Тип **множество** в Python содержит неповторяющийся уникальный набор данных. Для задания множества используется функция **set()** либо **frozenset()**.

Преобразования типов


```
a = int(3.1415) # 3
b = int("10101", 2) # 21
c = int(True) # 1

a = float(3) # 3.0
b = float("-10000000000000000") # -1e+16
c = float(False) # 0.0
print(type(c))

p = math.pi
a = str(p)
for i in a : print(i, end=" ")
print()
# 3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3
q = [2, 4, 6, 8, 10]
b = str(q)
print(f'Пусть список {b =}’)
# Пусть список b ='[2, 4, 6, 8, 10]'
```

Функция	Описание
int(s, d)	Преобразует значение s в целое число в системе счисления d
float(s)	Преобразует значение s в число с плавающей точкой
str(n)	Преобразует значение n в строку
bin(n)	Преобразование числа n в строку символов двоичного числа
set(a)	Преобразование аргумента a во множество

```
age = 18
message = f"Возраст - {age}"
# либо
message = "Возраст - " + str(age)
print(message)

print(set((9,4,1,5,1,8,1,4,10)))
# {1, 4, 5, 8, 9, 10}
```

Строки

Строка – это последовательность (символов).

```
>>> def myFunc():
...     """ Так можно документировать программу
...     причём комментарий может быть многострочным."""
...     pass
...
>>> print(myFunc.__doc__)
 Так можно документировать программу
     причём комментарий может быть многострочным.
>>> _
```

```
строка = 'Python'
print(строка[1], end='*')
print(строка[-1])
```

Что будет выведено на терминал?

```
Python = 'строка'
print(Python[8])
```

А в этом случае?

```
>>> S = 'Вводим специальные символы: \n \' \\ \u02ac'
>>> print(S)
Вводим специальные символы:
 ' \ ¬
>>> path = 'C:\\windows\users\nataly'
  File "<stdin>", line 1
    path = 'C:\\windows\users\nataly'
           ^^^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 11-12: truncated \uXXXX escape
>>> path = r'C:\\windows\users\nataly'
>>> print(path)
C:\\windows\users\nataly
>>> _
```

Срезы для строк

Срез – это фрагмент последовательности.

```
s = "Пример строки"
print(s[:]) # выведет ту же строку
print(s[::-1]) # выведет строку в обратном порядке (икортс ремирП)
print(s[1:-1]) # выведет строку без первого и последнего символа (ример строк)
print(s[2:5]) # выведет "име"
print(s[1:8:2]) # что выведет?
# как вывести только слово "строки"?
```

```
s = "Пример строки"
print(s.upper())
print(s.lower())
print(s.title())
for i in s:
    print(i, end=" ")
print(f"\n Длина строки: {len(s)}")
print("-" * 25)
print(s.center(25, '-'))
words = s.split() # пробел - разделитель по умолчанию
for i in words:
    print(i)
```

```
ПРИМЕР СТРОКИ
пример строки
Пример Строки
П р и м е р   с т р о к и
 Длина строки: 13
-------------------------
------Пример строки------
Пример
строки
```

Преобразования типов вместе с map и join


```
# из строки в список - первый способ
L = list("1 2 3 4 5 6 7 8 9 10".split())
F = []
for i in range(len(L)):
    F.append(float(L[i]))
print(F)

# из строки в список - второй способ
L = list("1 2 3 4 5 6 7 8 9 10".split())
F = [float(i) for i in L]
print(F)

# из строки в список - третий способ
L = list("1 2 3 4 5 6 7 8 9 10".split())
F = list(map(float, L))
print(F)
```

```
# создать строку, содержащую ряд цифр от 0 до 9
S = ''
for i in range(10): S += str(i) + ' '
print(S)

# создать строку, содержащую ряд цифр
# от 0 до 9 с шагом 0.5
    #способ 1
import numpy as np
L = list(np.arange(0, 9.1, 0.5))
S = ' '.join(map(str, L))
print(S)
    #способ 2
i = 0.0
S = ''
while i < 9.1:
    S += str(i) + ' '
    i += 0.5
print(S)
    #способ 3
S = ' '.join([str(i) for i in np.arange(0, 9.1, 0.5)])
print(S)
```

0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0

Списки

Методы:

.append()
.insert()
.remove()
.pop()
.sort()
.reverse()

Функции:

len()
sorted()
del()

```
Оттенки = ['red', 'very red', 'must red', 'great red', 'dead red']
print(Оттенки[1])
# very red
for i in Оттенки: print(i, end=' -> ')
# red -> very red -> must red -> great red -> dead red
print(Оттенки[-1])
# dead red
Оттенки.append('like hell')
print(Оттенки)
# ['red', 'very red', 'must red', 'great red', 'dead red', 'like hell']
Оттенки.insert(1, 'bright red')
print(Оттенки[:4])
# ['red', 'bright red', 'very red', 'must red']
Оттенки.remove('like hell')
print(Оттенки[:])
# ['red', 'bright red', 'very red', 'must red', 'great red', 'dead red']
del(Оттенки[0])
print(Оттенки)
# ['bright red', 'very red', 'must red', 'great red', 'dead red']
Y = Оттенки.pop(3) # если не указывать индекс, удалится последний элемент
print(Y)
# great red
print(Оттенки)
# ['bright red', 'very red', 'must red', 'dead red']
```

Пример 1 работы со списком

```
# создать список из N элементов, заполненный случайными целыми числами от 0 до 100
# найти в списке максимальное и минимальное число, а также рассчитать среднее значение
N = int(input('Введите целое положительное число N: '))
import random
myList = [random.randint(0, 100) for i in range(N)]
print(myList)
print(f'Максимальное значение: {max(myList)}; Минимальное значение: {min(myList)}')
print(f'Среднее значение: {sum(myList) / len(myList) :.4f}')

# а так выглядит эта же программа без использования встроенных функций агрегирования
left = 0; right = 100; myList = []
for i in range(N):
    myList.append(random.randint(left, right))
print(myList)
maximum = left - 1
minimum = right + 1
average = 0
for i in range(N):
    if myList[i] > maximum:
        maximum = myList[i]
    if myList[i] < minimum:
        minimum = myList[i]
    average += myList[i]
print(f'Максимальное значение: {maximum}; Минимальное значение: {minimum}')
print(f'Среднее значение: {average / N :.4f}')
```

Пример 2: матрица как список списков

```
# вводим квадратную матрицу чисел на чистом python
# и найдём сумму элементов главной и побочной диагоналей
M = []
# N - размерность матрицы
N = int(input("Введите размер матрицы: "))
for i in range(N):
    M.append([])
    L = list(map(float, input("Введите элементы " + str(i + 1) + " строки: ").split()))
    if len(L) != N:
        print("Ошибка! Матрица должна быть квадратной!")
        quit()
    M[i] = L
# выводим матрицу на экран
for i in range(N):
    for j in range(N):
        print(M[i][j], end="\t")
    print()
# считаем сумму элементов
# главной и побочной диагоналей
S1 = S2 = 0
for i in range(N):
    S1 += M[i][i]
    S2 += M[i][-i]
print("Сумма элементов главной диагонали равна " + str(S1))
print("Сумма элементов побочной диагонали равна " + str(S2))
```

```
Введите размер матрицы: 3
Введите элементы 1 строки: -1 1.3 3.5
Введите элементы 2 строки: 8.2 2 -0.3
Введите элементы 3 строки: 5.5 0 -1
-1.0    1.3     3.5
8.2     2.0     -0.3
5.5     0.0     -1.0
Сумма элементов главной диагонали равна 0.0
Сумма элементов побочной диагонали равна 11.0
```

Пример 3: матрица как 2D массив

```python
# вводим квадратную матрицу чисел с использованием numpy
# и найдём сумму элементов главной и побочной диагоналей
import numpy as np
# N - размерность матрицы
N = int(input("Введите размер матрицы: "))
M = np.zeros((N, N))
for i in range(N):
    for j in range(N):
        M[i, j] = float(input(f"Введите элемент {i + 1} строки, {j + 1} столбца: "))
for i in range(N):
    for j in range(N):
        print(M[i, j], end="\t")
    print()
# считаем сумму элементов главной и побочной диагоналей
print('Сумма элементов главной диагонали равна', sum(np.diag(M)))
print('Сумма элементов побочной диагонали равна', sum(np.diag(np.fliplr(M))))
```

```python
# Чем хорош модуль numpy?
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]], float)
print('Вывод матрицы на экран\n', A)
print('Размер матрицы: ', A.shape)
print('Вывод строки матрицы: ', A[0])
print('Вывод столбца матрицы:\n', A[:, 2])
print('Сумма матриц (исходной и транспонированной)\n', A + A.T)
print('Матричное произведение\n', A * A)
```

```
Вывод матрицы на экран
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Размер матрицы:  (3, 3)
Вывод строки матрицы:  [[1. 2. 3.]]
Вывод столбца матрицы:
 [[3.]
 [6.]
 [9.]]
```

Кортежи, списки, множества, словари


```
our_info = [1,2,3,5,7,[2024,'Пример']]
print(our_info[2:4]) # [3, 5]
print(our_info[3:])
print(our_info[-1][0])
```

```
days = ('ПН','ВТ','СР','ЧТ','ПТ','СБ','ВС')
print(days[1])

my_info = ('Понедельник',)
print(my_info)
```

```
my_d = {}
my_d['color'] = 'red'
my_d['cat'] = 'Simon'
my_d[1000] = 'thousand'
print(my_d) # выводит весь словарь
print(my_d.keys()) # выводит все значения ключей
print(my_d['color']) # выводит значение по ключу
```

```
my_list1 = [1, 2, 3]
my_list2 = ["четыре", "пять", "шесть"]

combo_list = my_list1 + my_list2
print(combo_list)
# [1, 2, 3, 'четыре', 'пять', 'шесть']
```

```
my_list = [1, 17, 25, 1, 31, 17, 19, 25, 1]
my_set = set(my_list)
print(my_set)
```

{1, 17, 19, 25, 31}

```
my_dict1 = {"name": "Виталик", "job":"преподаватель"}
my_dict2 = {"name": "Дима", "job":"политик"}
my_list = [my_dict1, my_dict2]
print(my_list[1]["name"])
```

Использование функций (1)


```
# функция перевода из градусов Цельсия в шкалу Фаренгейта
def c_to_f(celsius):
    return celsius * 9 / 5 + 32

C = input("Введите температуру в градусах Цельсия: ")
print("Температура в шкале Фаренгейта: ", c_to_f(float(C)))
```

Функции упрощают работу с кодом, делая программу нагляднее, универсальнее и обеспечивая возможность повторного или многократного использования кода.

```
# функция "по требованию"

def обработать(вид_функции, аргументы):
    return вид_функции(аргументы)

def умножить(аргументы):
    M = 1
    for a in аргументы:
        M *= a
    return M

def сложить(аргументы):
    return sum(аргументы)

какое_то_условие = False
if какое_то_условие:
    Z = умножить; данные = (-1.1, 2.5, 11.7)
else:
    Z = сложить; данные = (1, 2, 3, 4)

print(обработать(Z, данные))
```

Рекурсивная – функция, вызывающая саму себя.

```
# функция, вычисляющая факториал числа
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

N = int(input("Введите целое положительное число N: "))
print("Факториал числа N равен: ", factorial(N))
```

Каждая функция определяет собственное пространство имён. Если определить переменную с именем **x** в основной программе, а другую переменную **x** в теле функции – они будут ссылаться на разные значения!

Использование функций (2)


```
# функция с любым количеством аргументов, которая возвращает
# сумму всех переданных аргументов и их количество

def sum_and_count(*args):
    return sum(args), len(args)

S, L = sum_and_count(10, 20, 30, 40, 50)
print(f"Сумма: {S}; Количество: {L}")
# Сумма: 150; Количество: 5
```

Можно также использовать символ ** перед аргументами, чтобы сгруппировать их в словарь, где имена аргументов станут ключами, а их значения — соответствующими значениями в словаре.

Функция может принимать любое количество аргументов и возвращать любое количество результатов любого типа. Если функция не вызывает **return** явно, вызывающая сторона всё равно получит результат: **None** (проверять: **if result is None**).

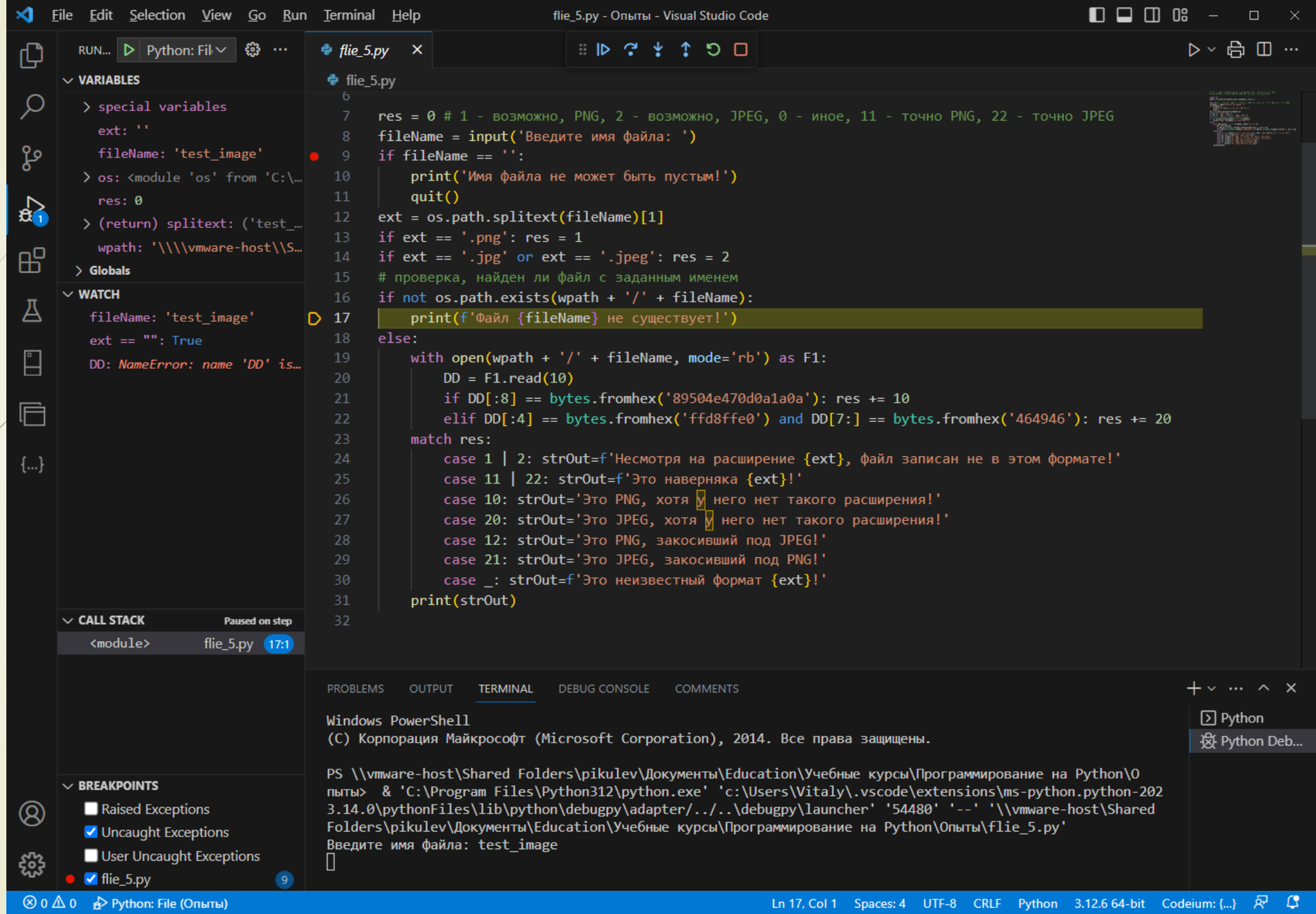
```
# особенность использования списка
# в качестве аргумента функции
def add_to_list(arg, local_list=[]):
    local_list.append(arg)
    return local_list

print(add_to_list(1))           # [1]
print(add_to_list('двойка'))    # [1, 'двойка']
print(add_to_list(3.1415))      # [1, 'двойка', 3.1415]
```

```
# расчёт факториала в виде вложенной функции
def factorial(n):
    def inner(n):
        if n == 0: return 1
        else: return n * inner(n - 1)

    if not isinstance(n, int):
        return (0, "N должно быть целым числом")
    if n < 0:
        return (0, "N должно быть положительным числом")
    return (inner(n), "OK!")

N = int(input("Введите число N: ")) # 19
print("Факториал числа N равен: ", factorial(N))
# (121645100408832000, 'OK!')
```

Отладка

17

1. На одной из первых строк мышкой поставить **breakpoint.**
2. Нажать на **F5** (старт с отладкой).
3. В окошке watch ввести искомое.
4. Нажимать **F11** (или **F10**) для выполнения каждой новой строки.
5. Нажать **Shift-F5** для остановки отладки.



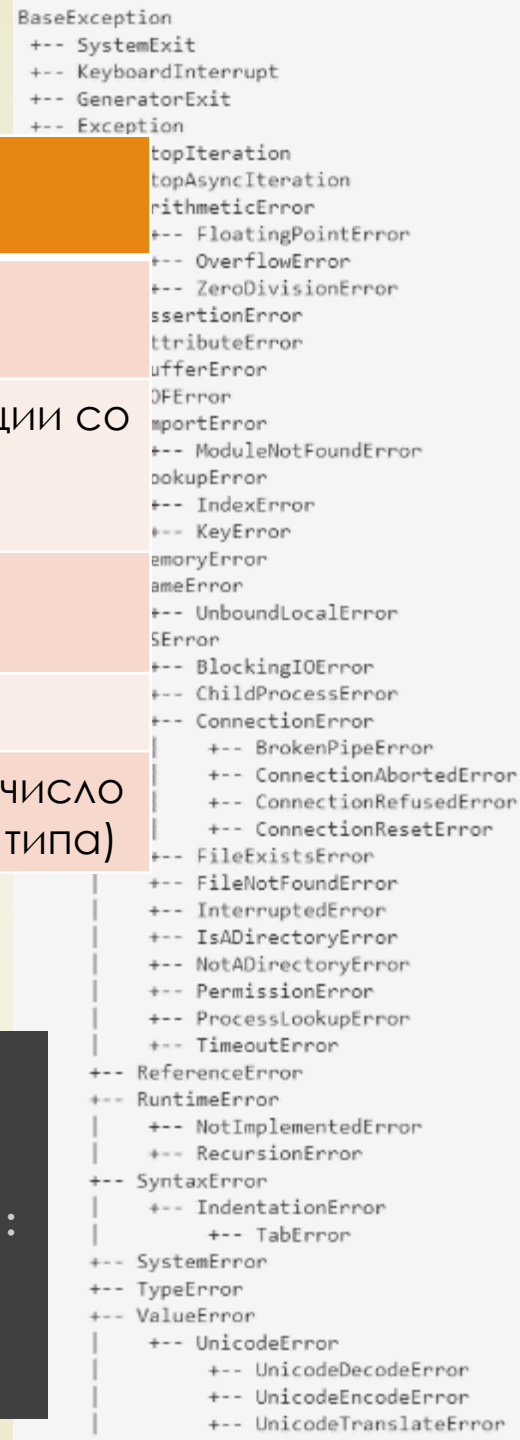
Обработка ошибок: try и except


```
res = True
while res:
    try:
        A = int(input('введите целое число A: '))
        B = int(input('введите целое число B: '))
        for i in range(A, B+1):
            print(1/i)
    except ZeroDivisionError:
        print('Ошибка: деление на ноль!')
    except ValueError:
        print('Ошибка: введено не целое число!')
    except:
        print('Что-то вообще пошло не так!')
    else:
        res = False
    finally:
        pass # этот блок здесь лишний
print('Программа завершена.')
```

Название ошибки	Описание ошибки
ValueError	Ошибка в значении переменной
TypeError	Выполнение операции со значением неправильного типа
NameError	Неизвестное имя переменной
ZeroDivisionError	Деление на нуль
OverflowError	Слишком большое число (для используемого типа)

```
def div(num1 = 0, num2 = 1):
    try:
        res = str(num1 / num2)
    except (TypeError, ZeroDivisionError):
        res = 'Ошибка!'
    finally:
        return res
```

Перехват всех исключений – скорее зло, нежели благо!



Чтение и запись файлов

Работаем с текстовыми файлами

Если python-программа аварийно завершится после того, как файл будет открыт, но до его закрытия командой **close()**, то системные ресурсы, задействованные для работы с файлом, так и останутся зарезервированными в памяти. Чтобы гарантировать, что ресурсы файловой системы будут освобождены даже в случае сбоя программы, операции с файлом следует проводить внутри конструкции **with**. Команда close в этом случае не нужна.

```
# из текстового файла 1 переписать
# всё содержимое в текстовый файл 2,
# удалив символы перехода на новую строку

wpath = '.'
try:
    with open(wpath + '/text_1.txt', mode='r') as F1:
        with open(wpath + '/text_2.txt', mode='w') as F2:
            for line in F1:
                line = line.replace('\r', '')
                line  = line.replace('\n', '')
                F2.write(line)
except(FileNotFoundError, PermissionError):
    print('Ошибка открытия файла!')
else:
    print(f'Файл {wpath + "/text_2.txt"} создан.')
```

```
>>>
>>> ord('\r')
13
>>> ord('\n')
10
>>>
```

Каждая строка текстового файла завершается одним или двумя символами (\r, \n), которые интерпретируются как метка конца строки. Эти символы обычно не отображаются в текстовом редакторе, но существуют в данных файла в виде байтов.

Вызов .open() с mode="**w**" приводит к тому, что содержимое исходного файла всегда будет перезаписано, а ранее существовавшие данные в файле с этим именем – уничтожены.

Основы работы с файлами

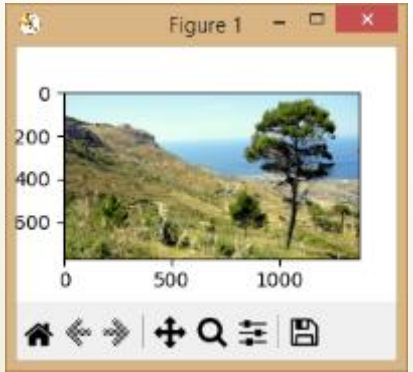
20 Работаем с текстовыми файлами

```
# запись в файл
F = open('file_3.txt','w')
F.write('Первая строка\n')
F.write('Вторая строка\n')
F.write('Третья строка\n')
F.close()

# чтение файла целиком
F = open('file_3.txt','r')
s = F.read()
print(s)
F.close()
# чтение файла построчно
F = open('file_3.txt','r')
while True:
    s = F.readline()
    if s == '':
        break
    print(s, end='')
F.close()
```

Метод **.read()** читает текст из файла в одну-единственную строку.
Метод **.readline()** читает файл по строкам.
Метод **.readlines()** возвращает итерируемый набор текстовых строк.

```
# запись списка в текстовый файл
F = open('file_4.txt', 'w')
L = ['Товарищ, верь!\n',
     'Взойдёт она -\n',
     'Звезда пленительного счастья!\n',
    ]
F.writelines(L)
F.close()

# чтение списка из текстового файла
F = open('file_4.txt', 'r')
L = F.readlines()
print(L)
F.close()
```

Делаем вид, что работаем с изображениями

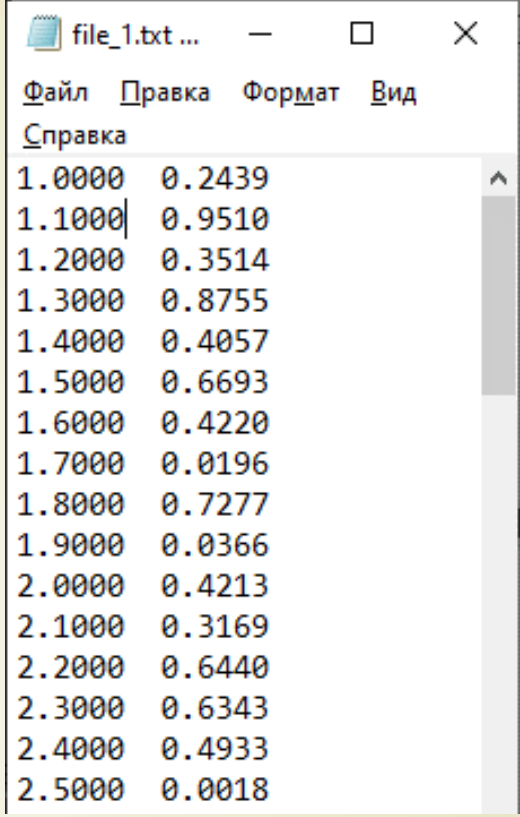



```
# загрузить изображение из файла и показать его
import matplotlib.image as mp
import matplotlib.pyplot as plt
img = mp.imread('image_1.jpg')
plt.imshow(img)
plt.show()
```

То же, но по-другому

Для функции **open**:
r – открыть для чтения, **w** – открыть на запись (обнуление),
x – открыть на запись, только если файл не существует,
a – продолжение записи, **rb** – чтение бинарного файла, **wb** – запись бинарного файла, **r+** - открытие на чтение/запись.

```
# записать текстовый файл, содержащий табулированный
# набор данных (x, y), где x изменяется от 1 до 10
# с шагом 0.1, y - случайное число в диапазоне от 0 до 1

from pathlib import Path
from random import random
import numpy as np

# создаем путь к файлу (в текущей директории)
myFilePath = Path.cwd() / 'file_1.txt'
# открываем файл для записи
with open(myFilePath, mode='w', encoding='utf-8') as F1:
    for x in np.arange(1, 10.1, 0.1):
        y = random()
        print(f'{x:6.3f}\t{y:6.3f}', file=F1)

print(f'Файл {myFilePath} создан')
```



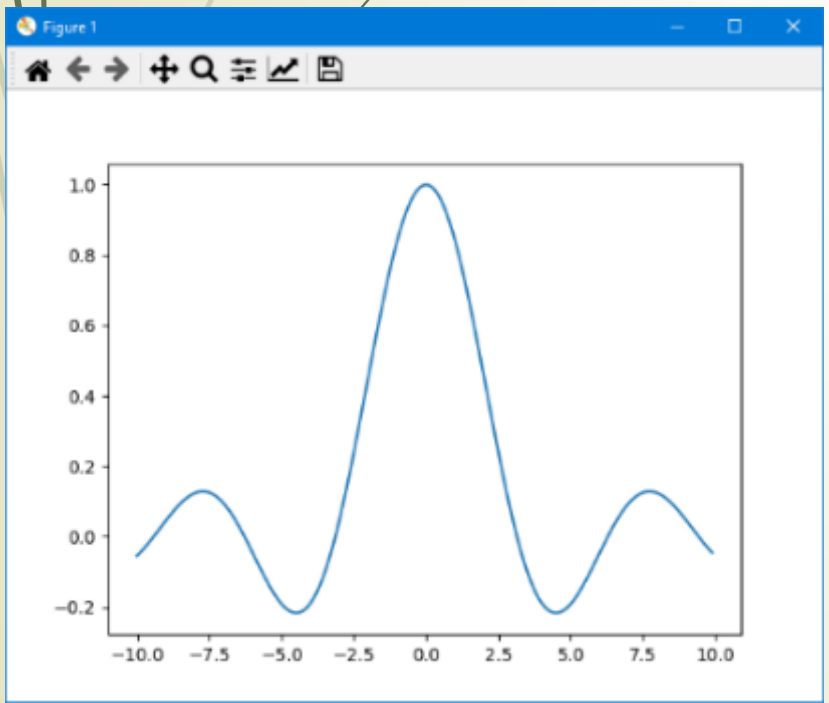

```
# чтение файла
import os

wpath = os.path.dirname(
os.path.realpath(__file__))

with open(wpath + '/file_1.txt',
mode='r', encoding='utf-8') as F1:
    for line in F1:
        print(line, end='')
```

Как построить график?

- `pip install matplotlib`
- `pip install numpy`

Построение разнообразных графиков на языке Python в большинстве случаев требует весьма небольшого кода, если обратиться за помощью к пакету matplotlib. Эта библиотека воспроизводит визуализацию данных в стиле пакета MATLAB (SciLab).

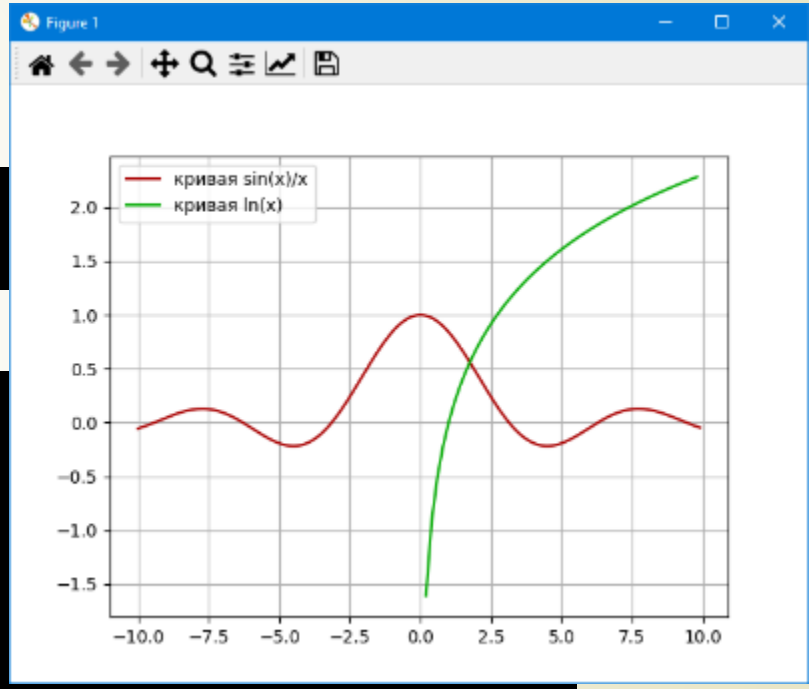
```
import matplotlib.pyplot as plt
import numpy as np
X = np.arange(-10, 10, 0.1)
Y = np.sin(X)/X
plt.plot(X, Y)
plt.show()
```




```
plt.plot(X, Y, 'r+')
```

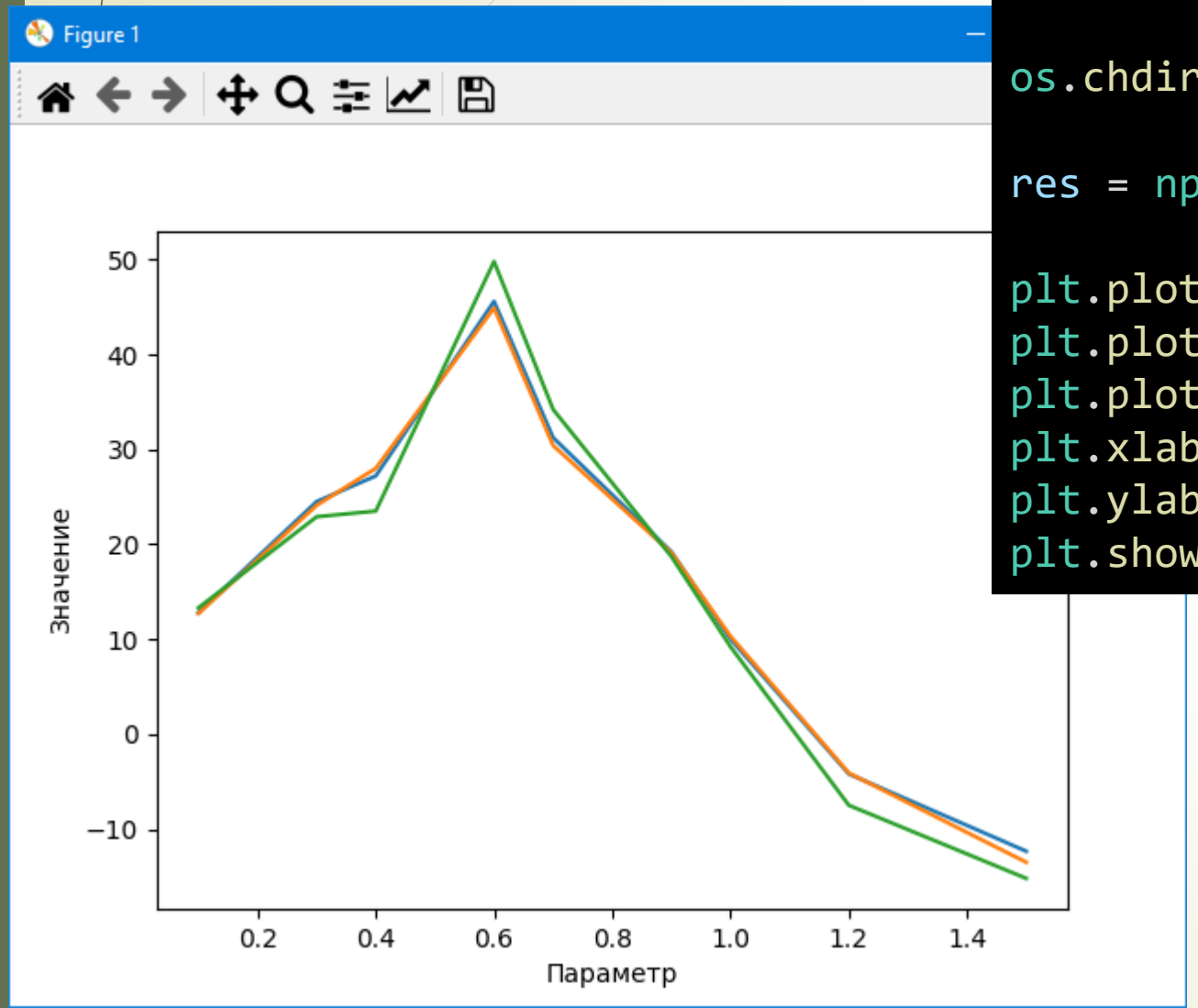
```
plt.plot(X, Y, color='black',
marker='o', linestyle='dashed')
```

```
import matplotlib.pyplot as plt
import numpy as np

X1 = np.arange(-10, 10, 0.1)
Y1 = np.sin(X1)/X1
X2 = np.arange(0, 10, 0.2)
Y2 = np.log(X2) # вычисляется натуральный логарифм
plt.grid() # включаем сетку графика
plt.plot(X1, Y1, '#AA0000', label='кривая sin(x)/x')
plt.plot(X2, Y2, '#00AA00', label='кривая ln(x)')
plt.legend(loc='upper left') # отображаем легенду
plt.show() # показываем результат
```



Как построить график по данным из файла?


```
import matplotlib.pyplot as plt
import numpy as np
import os

os.chdir(os.path.dirname(os.path.realpath(__file__)))

res = np.genfromtxt('results.txt')

plt.plot(res[:,0], res[:,1])
plt.plot(res[:,0], res[:,2])
plt.plot(res[:,0], res[:,3])
plt.xlabel('Параметр')
plt.ylabel('Значение')
plt.show()
```



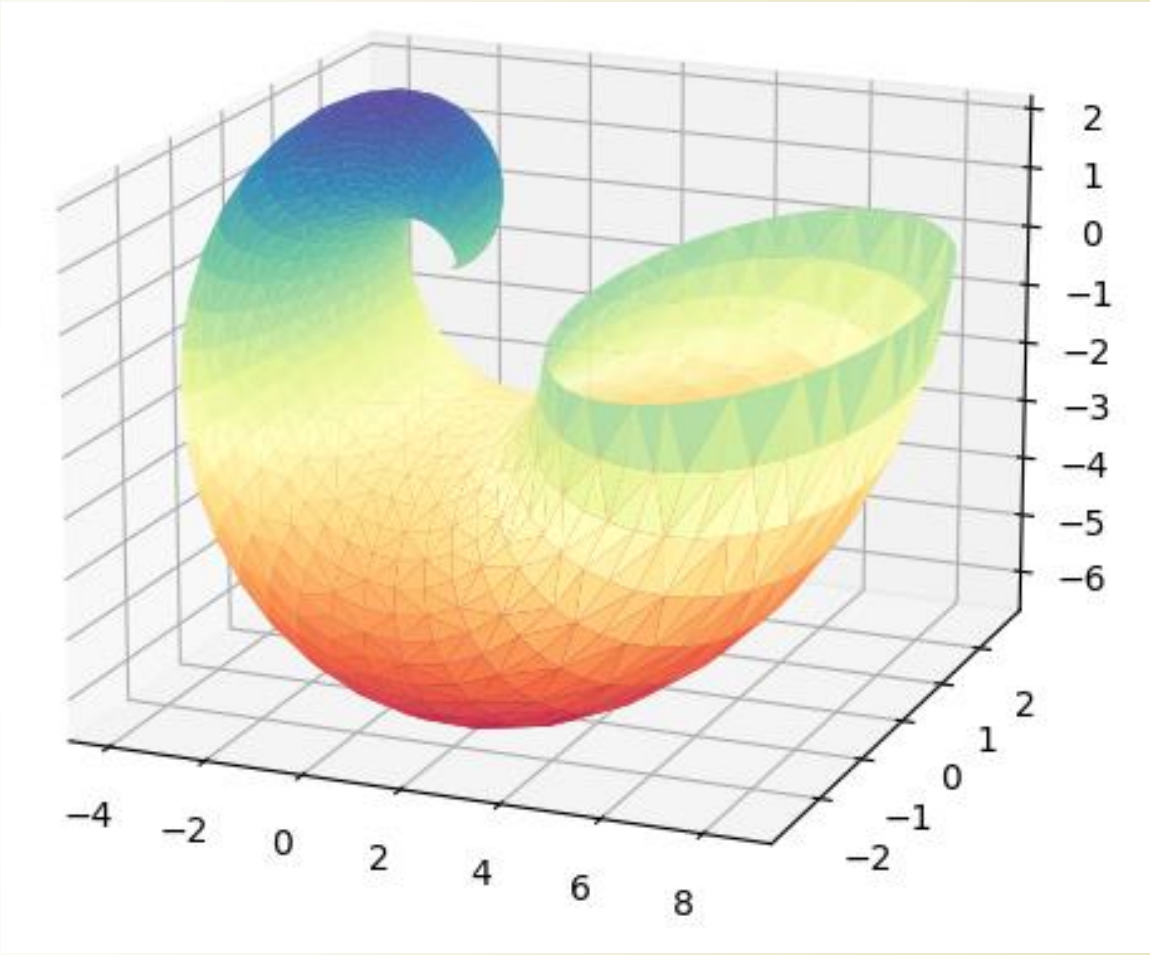
results.txt — Блокнот

Файл	Правка	Формат	Вид	Справка
0.1	12.8	12.7	13.3	
0.3	24.5	24.1	22.9	
0.4	27.2	28.0	23.5	
0.6	45.6	44.9	49.8	
0.7	31.23	30.4	34.2	
0.9	19.2	19.1	18.7	
1.0	10.0	10.3	9.2	
1.2	-4.2	-4.1	-7.5	
1.5	-12.34	-13.5	-15.2	

Разграничитель столбцов – табуляция (по умолчанию)

Пример топологической красоты…


```
import matplotlib.pyplot as plt
import matplotlib.tri as mtri
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.set_title('График параметрически заданной поверхности')
# Подготовка данных
u = np.linspace(0, 2*np.pi, 40)
v = u.copy()
u, v = np.meshgrid(u, v)
u, v = u.flatten(), v.flatten()

X = u*np.cos(u)*(1+np.cos(v)/2)
Y = 0.5*u*np.sin(v)
Z = u*np.sin(u)*(1+np.cos(v)/2)
# Построение графика
tri = mtri.Triangulation(u, v)
ax.plot_trisurf(X, Y, Z, triangles=tri.triangles,
cmap=plt.cm.Spectral)
plt.show()
```



Метод *flatten()* превращает матрицу в одномерный массив (по умолчанию – строка за строкой).

Работа с датой и временем

25 Модуль datetime

```
# работа с датой и временем
import datetime as dt

# получение текущей даты
today = dt.date.today()
print(today.isoformat()) # 2024-10-03

# получение текущей даты и времени
just_now = dt.datetime.now()
print(just_now.isoformat()) # 2024-10-03T17:14:45.209295
print(just_now.strftime("%d.%m.%Y, %H:%M:%S")) # 03.10.2024, 17:14:45
print(just_now.timestamp()) # 1727965089.45299
print(f'''Часы: {just_now.time().hour}, минуты: {just_now.time().minute},
секунды: {just_now.time().second},
микросекунды: {just_now.time().microsecond}.''')

# смещение по датам
delta = dt.timedelta(days=7)
print((today - delta).strftime("%d.%m.%Y")) # 26.09.24

# сколько дней осталось до Нового Года
print((dt.date(today.year + 1, 1, 1) - today).days) # 90
```

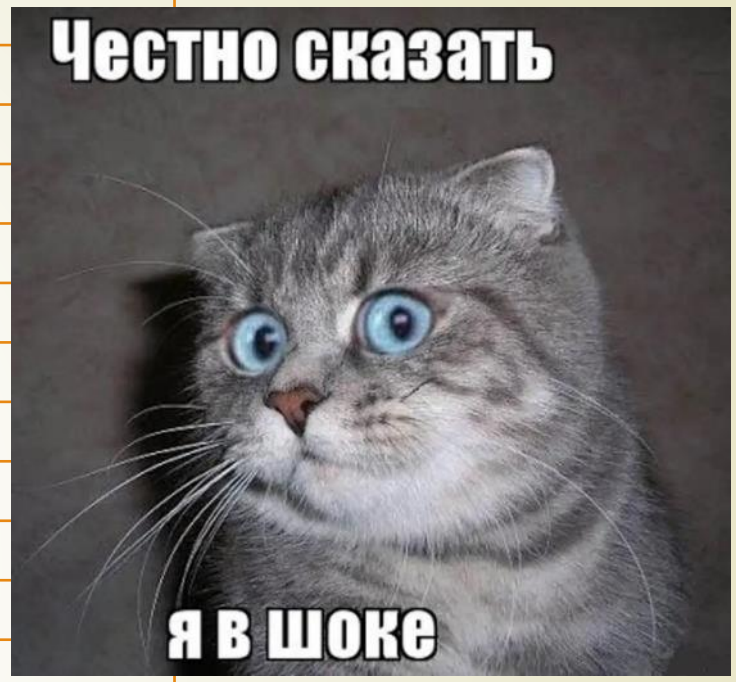
```
# актуальное время в разных часовых поясах
now_utc = dt.datetime.now(dt.timezone.utc)
print('В Лондоне ', now_utc)
now_ptz = dt.datetime.now()
print('В Петрозаводске ', now_ptz)
# смещение на восток
now_novosib = dt.datetime.now(dt.timezone(dt.timedelta(hours=7)))
print('В Новосибирске ', now_novosib)
```

```
# ввести конкретную дату
d = dt.date(2008, 10, 31)
print(d.strftime("%d.%m.%Y"))

# конкретное время
t = dt.time(11, 55, 0)
print(t.strftime("%H:%M:%S"))

# преобр. в тип даты и времени
d = dt.datetime.strptime("2008
10 31", "%Y %m %d")
print(d.strftime("%d.%m.%Y"))
```

Справка по формату вывода даты-времени

.strftime

%a	Сокращенное название дня недели
%A	Полное название дня недели
%b	Сокращенное название месяца
%B	Полное название месяца
%c	Дата и время
%d	День месяца [01,31]
%H	24-часовой формат часа [00,23]
%I	12-часовой формат часа [01,12]
%j	День года. Цифровой формат [001,366]
%m	Номер месяца. Цифровой формат [01,12]
%M	Минута. Цифровой формат [00,59]
%p	До полудня или после (AM или PM)
%S	Секунда. Цифровой формат [00,61]
%U	Номер недели в году. Цифровой формат [00,53] (с воскресенья)
%w	День недели. Цифровой формат [0(воскресенье), 6]
%W	Номер недели в году. Цифровой формат [00,53] (с понедельника)
%x	Дата
%X	Время
%y	Год без века. Цифровой формат [00,99]
%Y	Год с веком. Цифровой формат
%Z	Временная зона



Альтернативные модули:

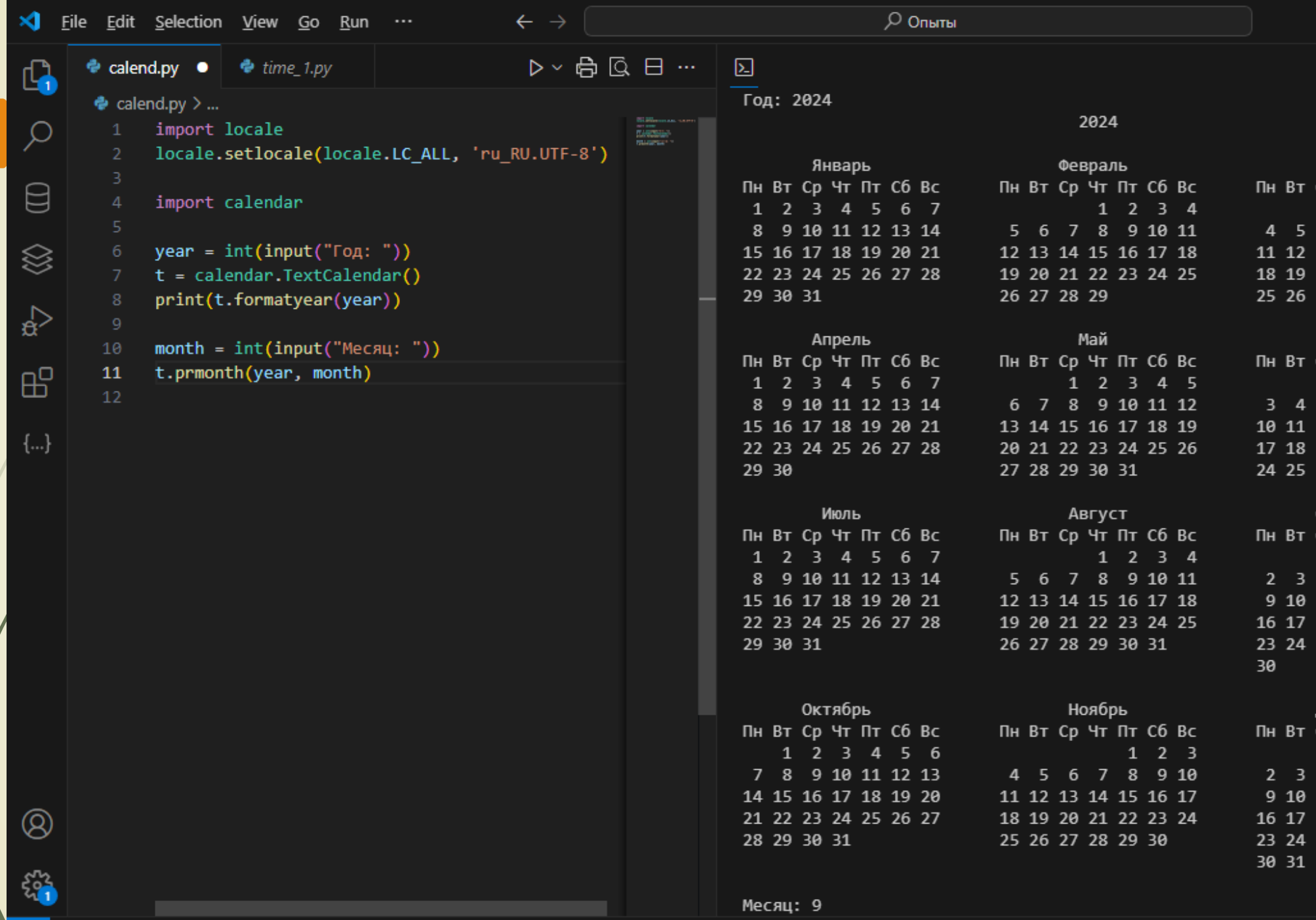
- time
- arrow
- dateutil
- fleming
- calendar
- timeit

Работа с датой и временем

27 Модуль time

```
import locale
locale.setlocale(locale.LC_ALL, 'ru_RU.UTF-8')
import time

print(time.time())  # 1727981677.1631289
print(time.ctime())  # Tue Oct  3 17:14:45 2024
format = "%d %B %Y %H:%M:%S"
print(time.strftime(format, time.localtime()))
# 03 Октябрь 2024 17:14:45

year = time.localtime().tm_year
print(year)  # 2024
day_of_year = time.localtime().tm_yday
print(day_of_year)  # 277

# введём дату в заданном формате
d = input("введите дату в формате dd.mm.yyyy: ")
dd = time.strptime(d, "%d.%m.%Y")
print(time.strftime("%d %B %Y года", dd))
```

```
# различные варианты задержек
import time
print("Старт!")
time.sleep(2.0)
print("Стоп! Прошло 2 секунды.")

wait = input("Нажмите Enter...")
print("Старт!")
start = time.time()
while time.time() - start < 2:
    # можно что-нибудь простое делать
    pass
print("Стоп! Прошло 2 секунды.")

wait = input("Нажмите Enter...")
print("Старт!")
start = time.monotonic()
while time.monotonic() - start < 2:
    # можно что-нибудь простое делать
    pass
print("Стоп! Прошло 2 секунды.")
```


File Edit Selection View Go Run
Опыты
calend.py
time_1.py
calend.py > ...
1 import locale
2 locale.setlocale(locale.LC_ALL, 'ru_RU.UTF-8')
3
4 import calendar
5
6 year = int(input("Год: "))
7 t = calendar.TextCalendar()
8 print(t.formatyear(year))
9
10 month = int(input("Месяц: "))
11 t.prmonth(year, month)
12
Год: 2024
2024
Январь
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
Февраль
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
Март
Пн Вт Ср Чт Пт Сб Вс
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
Апрель
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Май
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
Июнь
Пн Вт Ср Чт Пт Сб Вс
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
Июль
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
Август
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
Сентябрь
Пн Вт Ср Чт Пт Сб Вс
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
Октябрь
Пн Вт Ср Чт Пт Сб Вс
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
Ноябрь
Пн Вт Ср Чт Пт Сб Вс
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
Декабрь
Пн Вт Ср Чт Пт Сб Вс
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
Месяц: 9
Ln 11, Col 1 Spaces: 4 UTF-8 CRLF {} Python

Специфические конструкции Python

списковые включения

```
# список, содержащий квадраты целых чисел
N = 10
myList = []
for i in range(N):
    myList.append(i**2)
print(myList)

# соответствующий однострочник
print([i**2 for i in range(N)])

# ещё один вариант
print(list(map(lambda x: x**2, range(N))))
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
# пример 5, все возможные сочетания
cross = [(a, b) for a in ['a', 'b', 'c'] for b in [1, 2, 3]]
print(cross)

# то же самое, но поиск наименьшего произведения двух чисел
myList = [1, 2, 3, 4]
cross = [a * b for a in myList for b in myList if a != b]
print(min(cross))
```

```
# поиск простых чисел
myList = list(filter(lambda x : all(x % y != 0 for y in range(2, x)), range(2, N)))
print(myList)
```

```
# находим последний член ряда Фибоначчи
def fib1(x):
    if x <= 2:
        return 1
    return fib1(x - 1) + fib1(x - 2)
print(fib1(N))

# соответствующий однострочник
fib2 = lambda x: 1 if x <= 2 else fib2(x - 1) + fib2(x - 2)
print(fib2(N))
```

лямбда-функции

Any()
All()

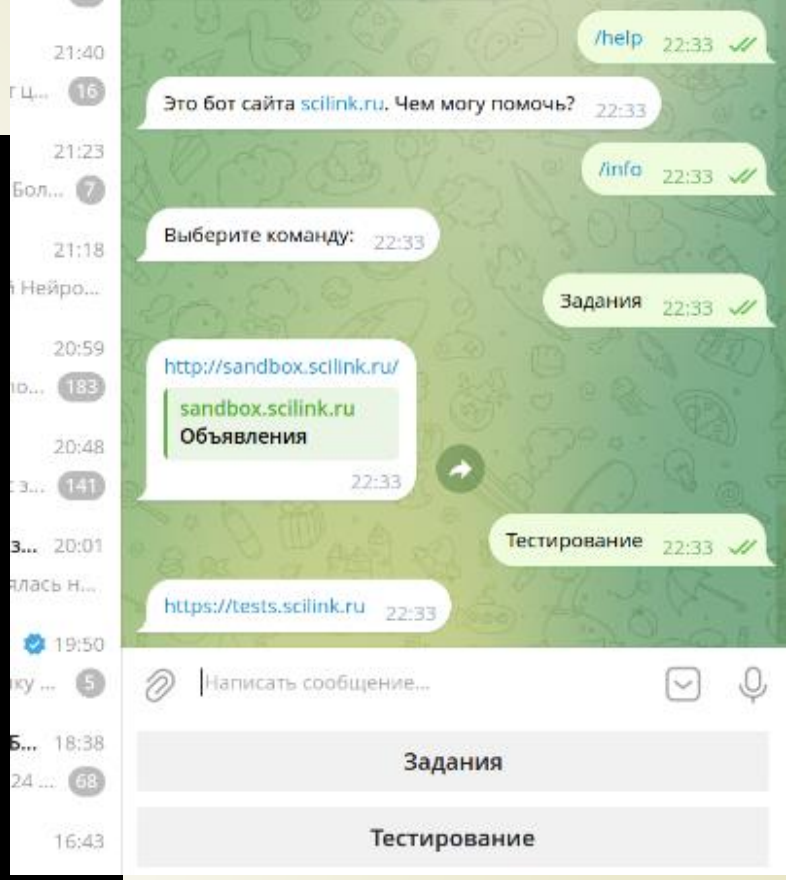
fliter()
map()
zip()

срезы

```
# список, содержащий квадраты чётных целых чисел
myList = []
for i in range(N):
    if i%2==0:
        myList.append(i**2)
print(myList)

# соответствующий однострочник
print([i**2 for i in range(N) if i % 2 == 0])

# ещё один вариант
print(list(map(lambda x: x**2, range(0, N, 2))))
# [0, 4, 16, 36, 64]
```

Боты для Telegram

```python
import telebot
from telebot import types

API_TOKEN = '65535230869:iTiSAmYrEalwErysEcrettOken'
bot = telebot.TeleBot(API_TOKEN)

@bot.message_handler(commands = ['start','help'])
def start_message(message):
    bot.send_message(message.chat.id,'Это бот сайта scilink.ru. Чем могу помочь?')
@bot.message_handler(commands = ['menu','info'])
def button_message(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard = True)
    item1 = types.KeyboardButton("Задания")
    markup.add(item1)
    item2 = types.KeyboardButton("Тестирование")
    markup.add(item2)
    bot.send_message(message.chat.id, 'Выберите команду:', reply_markup = markup)
@bot.message_handler(content_types='text')
def message_reply(message):
    if message.text=="Задания":
        bot.send_message(message.chat.id,'http://sandbox.scilink.ru/')
    elif message.text=="Тестирование":
        bot.send_message(message.chat.id,"https://tests.scilink.ru")
    else: bot.send_message(message.chat.id, f"Не понимаю, что значит {message.text} ?")
print('Бот запущен')
bot.polling(none_stop=True, interval=0)
```



- pip install pyTelegramBotAPI, aiogram

Использование виртуального окружения

Виртуальное окружение (virtual environment) – инструмент, позволяющий изолировать зависимости Python-проекта от глобального окружения и предотвращать возможные конфликты между библиотеками.

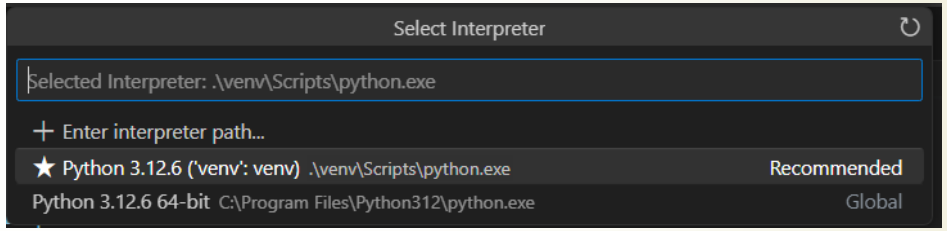
```
497             self.easyWait(0.1)
498             # запись данных в файл
499             fname = self.activeDirectory + os.sep + AG['serName']
500             with open(fname, 'w', encoding='cp1251') as f:
501                 f.write('MDR-2019 data file. '+
502                 datetime.today().strftime("%d-%m-%Y-%H:%M:%S") + '
```

PROBLEMS 9 OUTPUT TERMINAL DEBUG CONSOLE COMMENTS

```
Программа стартует... готово.
Программа корректно завершена.
PS C:\Users\Vitaly\Documents\PROJECTS\LumiScan>
* History restored

Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation), 2014. Все права защищены.

PS C:\Users\Vitaly\Documents\PROJECTS\LumiScan> activate
(venv) PS C:\Users\Vitaly\Documents\PROJECTS\LumiScan>
```

Виртуальное окружение создает "с нуля" окружающую среду для разрабатываемого проекта, независимую от других проектов и, в идеальном случае, от операционной системы. То есть, можно достаточно легко перенести свой проект на другой компьютер или дать другим пользователям работать с вашим проектом.

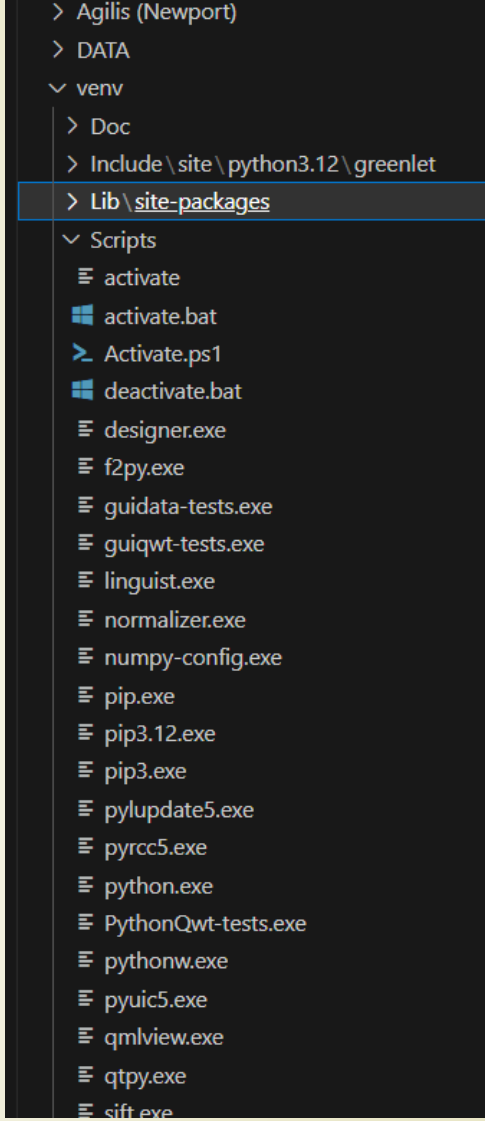
```
python -m venv myvenv
myvenv\Scripts\activate
pip install …
python …
…
deactivate
```



Для Visual Studio Code можно использовать встроенный терминал, однако следует не забыть после установки виртуального окружения вручную выбрать интерпретатор Python именно из виртуального окружения: из списка в строке состояния, либо воспользоваться командой редактора **Python: Select Interpreter**.

Если используется оболочка Windows PowerShell, очень часто следует разблокировать выполнение скриптов в этой оболочки изменением системной политики:

```
PS> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```



В программировании есть что-то сродни магии:
чем более сложные и непонятные заклинания вы используете –
тем более странным и непредсказуемым получается результат.

scilink.ru

Конец первого блока.