

*Основы Python: функции,
исключения, файлы, матрицы*

Макар Басалаев

Логическая структура программы. Подпрограммы.

Любой язык программирования содержит в своем арсенале средства для **упрощения** написания и **понимания** программ.

Например, если один и тот же функционал необходимо использовать в программе несколько раз, то, конечно, его не придется описывать дважды.

Чтобы исключить такие ситуации, в язык вводятся **подпрограммы**, то есть такие участки кода, которые можно в дальнейшем использовать как некоторую маленькую часть программы.

Функции в языке Python

В языке Python такие подпрограммы называются **функциями**

Каждая функция, используемая в программе, должна иметь:

1. Имя
2. Входные данные. Их может быть сколько угодно
3. Функция **может** возвращать значение

Что такое функция?



Признаки:

1. Она похожа на мини-программу
2. Она что-то делает
3. Она что-то принимает на вход
4. Она может что-то вернуть

Виды функций

Все функции языка Python можно **условно** разделить на 2 группы:

1. Встроенные, то есть созданные **НЕ ВАМИ** (фактически, все то, что мы раньше называли словом **команда**, на самом деле - функции):
 - *input*
 - *int*
 - *abs*
 - *print*
2. Создаваемые вами

Использование функций

Использование функций состоит из **двух** этапов:

1. Создание (описание) функции
2. Вызов функции

Описание функции должно располагаться **раньше** ее вызова, иначе Python **не узнает** функцию.

Первая функция на Python

В качестве первого примера рассмотрим функцию вычисления факториала.

```
def fact(n):  
    res = 1  
    while n>=1:  
        res *= n  
        n -= 1  
    return res
```

```
n = int(input())  
f = fact(n)  
print(f)
```


Пояснения...

1. **def** - ключевое слово, означающее, что мы начинаем **описывать** (создавать) функцию.
2. **fact** - имя функции (мы его сами придумали)
3. После имени функции в скобках через запятую перечисляются **параметры** - данные, которые мы **передаем** в функцию
4. Затем ставим **двоеточие**
5. Внутри **функция** полностью напоминает программу
6. **return** - ключевое слово, означающее, что мы собираемся **возвратить** значение. В данном случае `res`

Пояснения (часть вторая)...

`f = fact(n)` - означает, что мы:

1. **Вызываем** функцию с именем `fact`
2. **Записываем** результат в переменную `f`

Алгоритм Евклида

Наибольший Общий Делитель (**НОД**) двух чисел - это самое большое число, на которое можно без остатка разделить эти два числа.

НОД на английский переводится как

GCD - **G**reatest **C**ommon **D**ivider

Алгоритм:

1. Ищем $\text{НОД}(a, b)$
2. Если $b=0$, то $\text{НОД}(a, b)$ - это число a
3. Заменяем a на b , а b - на $a \% b$
4. Перейдем к шагу №2

Алгоритм Евклида

$$\text{НОД}(24, 16) = \text{НОД}(16, 8) = \text{НОД}(8, 0)$$

Суть функции GCD

1. Передаем в функцию два параметра: a , b
2. Организуем цикл, пока b не станет равным 0
3. Возвращаем значение переменной a
4. В основной программе мы вводим x , y и передаем их в функцию `gcd`, при этом $x \rightarrow a$, $y \rightarrow b$
5. Печатаем результат

```
def gcd(a, b):  
    while b!=0:  
        a, b = b, a%b  
    return a  
  
x, y = int(input()), int(input())  
print(gcd(x, y))
```


Еще пример

Вычислить сумму $\sum_{i=1}^n \frac{1}{i!}$ с помощью функции

```
def fact(n):  
    res = 1  
    while n >= 1:  
        res *= n  
        n -= 1  
    return res  
  
def sum(n):  
    res = 0  
    for i in range(1, n+1):  
        res += 1/fact(i)  
    return res  
  
n = int(input())  
print(sum(n))
```

Как добавить «математику»

Python по умолчанию содержит немного функций.

Однако, существует большое (**огромное**) количество библиотек, созданных для Python

Для их подключения служит директива **import**

```
import math  
print(math.sqrt(2))
```

В данном примере мы импортировали (**подключили**) библиотеку **math**

Для доступа к функциям библиотеки необходимо писать **имя_библиотеки.имя_функции**

Важные функции библиотеки math

Функция	Описание
floor	Округление числа «вниз» (пол)
ceil	Округление числа «вверх» (потолок)
trunc	Округление в сторону нуля
fabs	Модуль вещественного числа
sqrt	Квадратный корень
pi	Число пи

Основное про функции

1. Функции позволяют упростить структуру программы.
2. Функция - мини-программа, поэтому она имеет очень похожую на программу структуру.
3. В функцию можно передать параметры - данные, которыми эта функция может оперировать
4. Функция может вернуть значение (а может и не возвращать)

Рекомендации по написанию функций

1. Всегда представляйте функцию как **черный ящик**, в который что-то кладется и из которого что-то может быть получено.
2. Продумываем, как правильно поделить программу на функции. Обычно, функция - это:
 - или часто используемый фрагмент программы
 - или четко выраженная логическая единица программы
3. Главное, не переборщить. 😊
4. Называйте функцию **по смыслу**, а не абы как.

Задача №1

- Напишите функцию, в которую поступает число N , и которая возвращает количество цифр в этом числе

Задача №1

- Напишите функцию, в которую поступает число N, и которая возвращает количество цифр в этом числе

```
def NumOfDigits(n):  
    result = 0  
    while n>0:  
        result += 1  
        n //= 10  
    return result  
  
num = int(input("Введите число N: "))  
print(NumOfDigits(num))
```

Задача №2

- Вводятся натуральные числа N и K . Напишите функцию, которая возвращает количество цифр K в числе N .

Задача №2. Решение

- Вводятся натуральные числа N и K. Напишите функцию, которая возвращает количество цифр K в числе N.

```
def NumOfDigitsK(n, k):  
    result = 0  
    while n > 0:  
        if n % 10 == k:  
            result += 1  
        n //= 10  
    return result  
  
num = int(input("Введите число N: "))  
k = int(input("Введите цифру K: "))  
print(NumOfDigitsK(num, k))
```


Дополнительные задачи

- a. Вводится натуральное число N . Функция возвращает сумму цифр N .
- b. Вводится натуральное число $N > 99$. Функция возвращает число, получающееся отбрасыванием первой и последней цифры N .
- c. Вводится натуральное число $N > 9$. Функция возвращает число, получающееся заменой первой цифры N на 9, а последней на 0.
- d. Вводится натуральное число N . Функция возвращает сумму делителей числа N .
- e. Вводится натуральное число N . Функция возвращает перевернутую запись числа N .
- f. Вводится натуральное число N . Функция возвращает:
 - 1. 1, если N простое число;
 - 2. 0, если N составное число;
 - 3. -1, если $N=1$.
- g. Вводятся натуральные числа A и B . ($A < B$). Функция печатает через пробел все простые на $[A, B]$.
- h. Вводятся натуральные числа A и B . ($A < B$). Функция печатает через запятую все совершенные на $[A, B]$.

Области видимости переменных

Область видимости - место где определяются переменные и выполняется их поиск.

Всегда, когда в программе используется какое то имя, интерпретатор создает, изменяет или отыскивает это имя в **пространстве имен** - в области, где находятся имена.

Имена, определяемые внутри функции, видны **только** программному коду внутри этой функции. К этим именам нельзя обратиться за пределами функции.

Области видимости переменных

Имена, определяемые внутри функции, **не вступают в конфликт** с именами, находящимися за пределами функции, даже если и там и там присутствуют одинаковые имена.

```
x = 99  
  
def func():  
    x = 88
```

В этом примере описаны два имени «X».
Но это - разные X

Функция внутри функции

Функция может быть описана **внутри** другой функции.

```
def f1(a,b,c):  
    def f2(x,y,z):  
        return x+y+z  
    d = f2(a,b,c)  
    return a*b*c-d  
  
print(f1(1,2,3))
```

В этом примере функция `f2` описана внутри `f1`. Это означает, что вызвать `f2` мы можем **только** из `f1`

Локальные и глобальные переменные

Три основные области видимости:

- Если присваивание (определение) переменной выполняется внутри функции (инструкции `def`), то переменная является **локальной** для этой функции.
- Если присваивание производится в пределах родительской инструкции `def`, переменная является **нелокальной** для этой функции.
- Если присваивание производится за пределами всех инструкций `def`, она является **глобальной** для всего файла.

Как Python определяет, что переменная - локальная

1. Если в функции переменная используется в **левой** стороне операции присваивания, то она - локальная.
2. Причем, **неважно**, в каком месте функции это присваивание находится.

Следующий пример содержит ошибку

```
def test():  
    print(a)  
    a = 2  
  
a = 5  
test()
```


Как Python определяет, что переменная - локальная

Исправим

```
def test():  
    a = 2  
    print(a)  
  
a = 5  
test()
```

Как Python определяет, что переменная - локальная

1. Если в функции переменная используется в **левой** стороне операции присваивания, то она - локальная.
2. Причем, **неважно**, в каком месте функции это присваивание находится.

Следующий пример содержит ошибку

```
def test():  
    print(a)  
    a = 2  
  
a = 5  
test()
```

Локальные и глобальные переменные

```
x = 1
def func1():
    x = 2
    print('Func1 #1', x)
    def func2():
        x = 3
        print('Func2 #1', x)
    func2()
    print('Func1 #2', x)

print('Main #1', x)
func1()
print('Main #2', x)
```

Что будет выведено в процессе работы программы?

Локальные и глобальные переменные

```
x = 1
def func1():
    x = 2
    print('Func1 #1', x)
    def func2():
        x = 3
        print('Func2 #1', x)
    func2()
    print('Func1 #2', x)

print('Main #1', x)
func1()
print('Main #2', x)
```

Что будет выведено в процессе работы программы?

Main #1 1
Func1 #1 2
Func2 #1 3
Func1 #2 2
Main #2 1

Как Python определяет, что переменная - НЕлокальная

Если в функции переменная используется в **правой** стороне операции присваивания, и она не является локальной, то она - **нелокальная**

```
def test():  
    print(a)  
    b = a + 7  
    print(b)  
  
a = 5  
test()
```

Глобальные переменные

Остался не рассмотренным один вопрос:

```
x = 5
def f1():
    x = 10

print(x) # >>> 5
f1()
print(x) # >>> 5
```

КАК ИЗМЕНИТЬ В ФУНКЦИИ ГЛОБАЛЬНУЮ ПЕРЕМЕННУЮ?

Глобальные переменные

- Ключевое слово **global** говорит о том, что под именем `x` будет пониматься **глобальная** переменная, а не локальная.
- Поэтому значение `x` изменяется

```
x = 5
def f1():
    global x
    x = 10
print(x) # >>> 5
f1()
print(x) # >>> 10
```

Что будет выведено на экран?

```
a,b,c = 2,3,0

def f1(a,b):
    global c
    b = b * 2
    c = a - b
    a = c - b
    return a

def f2(a):
    global b,c
    a = c - 5
    b = c * 3
    c = a + b - c
    return a*b

print(a,b,c)
a = f1(a, b)
print(a,b,c)
b = f2(a)
print(a,b,c)
```

Что будет выведено на экран?

```
a,b,c = 2,3,0

def f1(a,b):
    global c
    b = b * 2
    c = a - b
    a = c - b
    return a

def f2(a):
    global b,c
    a = c - 5
    b = c * 3
    c = a + b - c
    return a*b

print(a,b,c) # 2 3 0
a = f1(a, b)
print(a,b,c) # -10 3 -4
b = f2(a)
print(a,b,c) # -10 108 -17
```


Возврат нескольких значений из функции

Мы говорили, что любая функция может возвращать только одно значение, или не возвращать ничего.

Откуда тогда такой странный заголовок?

Дело в том, что функция может вернуть **кортеж**, а в кортеже может находиться несколько значений.

```
def minmax(a):  
    return (min(a), max(a))  
  
res = minmax((1, 7, 2, 4, 5, 3, -2))  
print(res, type(res))  
  
>>> (-2, 7) <class 'tuple'>
```

Передача произвольного количества параметров в функцию

Оказывается, в функцию можно передать **произвольное** количество параметров.

Мы уже с этим встречались. Посмотрите внимательно на функцию `sum`.

Давайте сделаем свою такую функцию. У нее будет только один параметр, но со **звездочкой**. И этот параметр - кортеж.

```
def sum(*args):  
    res = 0  
    for i in args:  
        res += i  
    return res
```

```
print(sum(1,2))
```

Задание на практику

1. Написать функцию, которая возвращает минимум и максимум из вводимой до нуля последовательности. Число 0 не учитывать, последовательность вводить **внутри** функции.
2. Написать функцию, которая получает произвольное число параметров целых чисел и возвращает их среднее арифметическое.

Исключения

Что будет, если мы напишем вот такую инструкцию?

```
>>> 2/0
```

Исключения

Что будет, если мы напишем вот такую инструкцию?

```
>>> 2/0
Traceback (most recent call last):
  Python Shell, prompt 3, line 1
builtins.ZeroDivisionError: division by zero
```

Это - ошибка. Специально смоделированная, но **ошибка**.

При этом программа, в которой произошла ошибка **будет остановлена**

Исключения

1. Ошибки есть всегда! 😊
2. Ошибки могут быть хорошими и плохими
 1. Хорошие - это те, которые не зависят от программиста, например, при отправке данных по сети.
 2. Плохие - это те, которые были сделаны программистом (что-то упустил, забыл, поделил на ноль и т.д.)
3. Вообще, программисты заменяют понятие «ошибка» на понятие «**исключительная ситуация**» или «**исключение**».
4. Исключения можно **обрабатывать**.

Еще виды исключений

```
>>> 2 + '1'
Traceback (most recent call last):
  Python Shell, prompt 4, line 1
builtins.TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

```
>>> int('vasya')
Traceback (most recent call last):
  Python Shell, prompt 5, line 1
builtins.ValueError: invalid literal for int()
with base 10: 'vasya'
```

Обработка исключений

1. Мы ввели два целых числа
2. Давайте попробуем (try) выполнить команду **print**
3. Если она выполнится, то все замечательно
4. А если возникнет исключение, то мы попадем в блок **except**

```
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))

try:
    print('Результат:', a / b)
except ZeroDivisionError:
    print('На ноль делить нельзя')
```

Обработка исключений. Продолжение

1. Введем вместо числа слово «vasya»
2. Опять исключение, только другое - **ValueError**.
3. Давайте и его обработаем

```
try:
    a = int(input("Введите первое число: "))
    b = int(input("Введите второе число: "))
    print('Результат:', a / b)
except ZeroDivisionError:
    print('На ноль делить нельзя')
-----
Traceback (most recent call last):
  Python Shell, prompt 1, line 2
builtins.ValueError: invalid literal for int() with
base 10: 'vasya'
```


Обработка исключений. Продолжение

```
try:
    a = int(input("Введите первое число: "))
    b = int(input("Введите второе число: "))
    print('Результат:', a / b)
except ZeroDivisionError:
    print('На ноль делить нельзя')
except ValueError:
    print('Вводите, пожалуйста, правильно')
```

И еще...

Если нам совсем лень обрабатывать конкретное исключение, то надо писать так:

```
try:
    a = int(input("Введите первое число: "))
    b = int(input("Введите второе число: "))
    print('Результат:', a / b)
except Exception:
    print('Случилась ошибка')
```

Если мы хотим обработать любую арифметическую ошибку:

```
try:
    ...
except ArithmeticError:
    print('Случилась арифметическая ошибка')
```

И в итоге...

Вообще, обработка исключений выглядит так:

```
try:
    a = int(input("Введите первое число: "))
    b = int(input("Введите второе число: "))
    print('Результат:', a / b)
except ZeroDivisionError:
    print('На ноль делить нельзя')
except ValueError:
    print('Вводите, пожалуйста, правильно')
else:
    print('Супер. Никаких ошибок не было')
finally:
    print('Мы закончили')
```

- Ветка **else** срабатывает, если не сработал никакой **except**
- Ветка **finally** работает в любом случае

Задание на практику

1. Напишите программу, которая запрашивает ввод двух значений. Если хотя бы одно из них не является числом, то должна выполняться конкатенация, т. е. соединение, строк. В остальных случаях введенные числа суммируются.

Файлы

Файл - это структурированные данные, сохраненные на носителе информации.

Для удобства организации информации файлы группируются в каталоги (папки).

В операционных системах семейства Windows существует понятие **Диска** - еще один логический способ организации.

В операционных системах семейства UNIX все есть файл. Это означает, что работа с любыми устройствами устроена через работу с файлами.

Имена файлов

Обычно имя файла состоит из двух частей:

1. Собственно, **имени**.
2. и **расширения** - короткого названия, по которому можно определить область применения данного файла. Например, расширение `jrg` говорит о том, что в файле хранится *картинка* в формате JPEG
3. Обычно расширение дописывается к имени файла через точку, например, `бланк.docx` - имя файла.

Наличие расширения является не обязательным требованием

Полное имя файла

Полным именем файла называется склейка имени файла с полным путем по дереву каталогов до места хранения этого файла.

Примеры:

1. `C:\windows\system32\drivers\etc\hosts`
2. `D:\documents\film.avi`
3. `/usr/local/bin/ls`

Типы файлов

Файлы бывают:

1. Бинарными, когда хранимые в них данные представляют собой последовательность чисел, обычно заданных в двоичной (шестнадцатеричной) системе счисления.
2. Текстовыми, когда хранимые в них данные представляют собой последовательность строк в различных кодировках.

Кодировки текстовых файлов

Кодировка (кодировочная таблица) - это принцип сопоставления символу числового номера.

Исторически первая кодировка - **ASCII** - содержала английские буквы, цифры и другие символы. Всего - 128 штук.

В нашей стране были популярны кодировки **cp1251**, **KOI8-R**

Самая популярная сейчас кодировка - **UTF-8**

Работа с файлами в Python

С точки зрения программиста файл - это длинный непрерывный объект, по которому перемещается некоторый маркер-указатель.

Работа с файлами в языке Python **всегда** состоит из трех блоков:

1. Открытие файла на запись или чтение
2. Работа с файлом
3. Заккрытие файла

Открытие файла

Для того, чтобы открыть файл необходимо выполнить следующий код:

```
f = open("filename", "mode", encoding="utf-8")
```

filename - это строка, содержащая имя файла (как полное, так и сокращенное)

mode - строка, указывающая на тип файла и те действия, которые необходимо с ним произвести. Состоит из 1 или 2-х букв.

После выполнения **open** создается переменная-объект с именем **file**

Варианты параметра mode

1 буква mode:

r	Чтение файла
w	Запись файла. Если файла не существует, он будет создан. Если файл существует, он будет перезаписан
x	Запись файла, но только если файла еще не существует
a	Добавление данных в конец файла, если он существует.

2 буква mode:

t	Файл текстовый (может не указываться)
b	Файл бинарный

Заккрытие файла

Файл закрывается функцией-методом `close()`

```
file.close()
```

Запись данных в текстовый файл

Чтобы записать данные в текстовый файл, можно воспользоваться двумя механизмами:

1. Использовать стандартную функцию `print()`

```
fobj = open("text1.txt", "wt", encoding="utf-8")  
print("Первый вариант", file=fobj)  
fobj.close()
```

2. Использовать функцию-метод `write()`, но только для строк

```
file = open("text1.txt", "wt", encoding="utf-8")  
file.write("Первый вариант")  
file.close()
```

Чтение текстового файла целиком

Если надо «скушать» файл сразу целиком, то для этого есть функция-метод **read**

```
file = open("burns.txt", "rt")
data = file.read()
print(type(data))
print(len(data))
file.close()
```

data - это строка

Единственное, что можно сделать, - это ограничить **read** в аппетите и указать в скобках это ограничение.

Чтение текстового файла целиком, ну или почти...

Единственное, что можно сделать, - это ограничить `read` в аппетите и указать в скобках это ограничение.

```
file = open("burns.txt", "rt")
data = file.read(10)
print(type(data))
print(len(data))
file.close()
```

Чтение текстового файла по строкам

Опять же, есть 2 способа читать файл по строкам:

1. Способ №1

```
file = open("burns.txt", "rt")
poem = ""
while True:
    line = file.readline()
    if not line:
        break
    poem += line
file.close()
```

Чтение текстового файла по строкам

2. Способ №2

```
file = open("burns.txt", "rt")  
poem = ""  
for line in file:  
    poem += line  
file.close()
```

Самое главное:

Каждая строка оканчивается символом «перевода строки» - '\n'

Получение списка строк текстового файла

Метод **readlines** возвращает список строк файла, причем каждая строка заканчивается на '\n'

```
file = open("burns.txt", "rt", encoding="utf-8")  
liststr = [x.strip() for x in file.readlines()]  
file.close()
```

В данном примере метод **strip()** удаляет сначала и из конца строки все «пробельные» символы

Что, если забыть закрыть файл...

В этом случае Python сам его закроет за вас, когда сочтет это необходимым, но до этого момента **НИКТО** не сможет пользоваться файлом.

Чтобы не забывать закрывать файл, можно пользоваться следующим синтаксисом:

```
with open("burns.txt", "rt") as file:  
    liststr = [x.strip() for x in file.readlines()]
```

Python сам закроет файл по окончании работы **with**

Задание на практику

1. Скачайте с сайта `lib.ru` любой текстовый документ под именем `doc.txt`. Проверьте кодировку. Напишите программу, которая сделает копию данного файла и сохранит ее в файл `doc1.txt`
2. Посчитайте сколько слов содержит скаченный файл.
3. В файле `input.txt` вразнобой записаны целые числа. Напишите программу, которая создаст файл `output.txt` и запишет в него сумму этих чисел.

Задачи (дополнительно)

- a. С клавиатуры через пробел вводится список целых чисел. Запишите его в файл с именем `list1.txt`. Затем перенесите из этого файла в файл `list2.txt` все числа, которые стоят на четных местах. Позиции нумеровать с единицы.
- b. Создайте текстовый файл с произвольным наполнением. Узнайте, сколько символов в нем содержится. Символы перевода строки `\n` учитывать не надо.
- c. Напишите программу, которая копирует данные из одного файла в другой, но в обратном порядке, то есть в новом файле сначала идет последняя строка из старого.
- d. Напишите функцию `delete_file(f)`, удаляющую из файла `f` все символы '+' и '-'.

Матрицы

- Матрицей или двумерным массивом называется прямоугольная таблица размерности n на m , где n - число строк, а m - число столбцов.
- Пример матрицы 2×3

A_{00}	A_{01}	A_{02}
A_{10}	A_{11}	A_{12}

- Положение элемента в матрице определяется номерами **строки** и **столбца**, в которых он находится. Например, a_{43} - 4 строка, 3 столбец (при начале от нуля)

Как описать матрицу на языке Python?

Если мы точно знаем все содержимое матрицы, то все просто:

```
a = [  
    [1, 2, 3],  
    [4, 5, 6]  
]
```

Мы видим, что физически матрица описывается как список, элементами которого являются другие списки.

В принципе, вложенные списки могут быть разной длины, но это мы пока не рассматриваем.

Как получить доступ к элементу матрицы?

Для доступа к элементу матрицы с номерами i и j необходимо написать следующую конструкцию:

`a[i][j]`

```
>>> a = [  
    [1,2,3],  
    [4,5,6]  
]  
>>> a[0][2]  
3  
>>> a[1][2] = 8  
>>> a  
[[1,2,3], [4,5,8]]
```

Напишем функцию, которая заполняет матрицу $n \times m$ последовательностью целых чисел

```
def fill(n, m):  
    a = []  
    c = 1  
    for i in range(n):  
        a.append([])  
        for j in range(m):  
            a[i].append(c)  
            c += 1  
    return a  
  
print(fill(3,4))
```

Напишем функцию, которая заполняет матрицу $n \times m$ последовательностью целых чисел

```
def fill(n, m):  
    a = []  
    c = 1  
    for i in range(n):  
        a.append([])  
        for j in range(m):  
            a[i].append(c)  
            c += 1  
    return a  
  
print(fill(3,4))
```

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```


Функция, заполняющая матрицу `nхm` последовательностью целых чисел. Пояснения.

1. Сначала мы создаем пустой список
2. Заполнение матрицы требует вложенного цикла
3. На внешнем уровне мы сначала добавляем новую строку в матрицу
4. А на внутреннем уровне - добавляем в эту строку элементы.

Как написать аналогичную функцию, вот с таким заголовком?

```
def fill(a, n, m):  
    ...  
  
a = []  
fill(a, 3, 4)
```

Как написать аналогичную функцию, вот с таким заголовком?

```
def fill(a, n, m):  
    c = 1  
    for i in range(n):  
        a.append([])  
        for j in range(m):  
            a[i].append(c)  
            c += 1  
  
    return  
  
a = []  
fill(a, 3, 4)  
print(a)
```


Создание матриц

```
a = []  
for i in range(n):  
    a.append([int(j) for j in input().split()])
```

```
a = []  
for i in range(n):  
    row = input().split()  
    for i in range(len(row)):  
        row[i] = int(row[i])  
    a.append(row)
```

```
a = [[int(j) for j in input().split()] for i in  
range(n)]
```

Функция, которая по матрице формирует массив, содержащий суммы строк

```
def sumstr(a):  
    res = []  
    for i in a:  
        s = 0  
        for j in i:  
            s += j  
        res.append(s)  
    return res  
  
a = [[1,2,3], [4,5,6], [7,8,9], [10,11,12]]  
print(*sumstr(a))
```

6 15 24 33

Задание на практику (Python)

1. Напишите программу, заполняющую матрицу следующим образом: строка слева направо, потом строка справа налево и т.д. С клавиатуры вводятся размеры матрицы. Необходимо вывести заполненную матрицу. При выводе определяйте для каждого числа 4 позиции и прижимайте его к левой стороне.

Пример входных данных	Пример выходных данных
3 5	1 2 3 4 5 10 9 8 7 6 11 12 13 14 15
4 2	1 2 4 3 5 6 8 7

Задание на практику (доп)

- а. Напишите программу, заполняющую матрицу по кругу. С клавиатуры вводятся размеры матрицы. Необходимо вывести заполненную матрицу. При выводе определяйте для каждого числа 4 позиции и прижимайте его к левой стороне.

Пример входных данных	Пример выходных данных
3 5	1 2 3 4 5 14 15 16 17 6 13 20 19 18 7 12 11 10 9 8
4 2	1 2 8 3 7 4 6 5

Тренировка

Напишите функцию, которая удаляет из списка все элементы, равные переданному `elem`

```
def delelems(L, elem):
```

Тренировка

Напишите функцию, которая удаляет из списка все элементы, равные переданному `elem`

```
def delelems(L, elem):  
    i = 0  
    while i < len(L):  
        if L[i] == elem:  
            L.pop(i)  
        else:  
            i += 1  
  
X = [1, 2, 1, 2, 3, 2, 1, 2, 3, 2]  
delelems(X, 2)  
print(*X)  
  
1 1 3 1 3
```


Тренировка

Напишите функцию, которая удаляет из списка все элементы, равные переданному `elem`

```
def delelems(L, elem):  
    while elem in L:  
        L.remove(elem)  
  
X = [1,2,1,2,3,2,1,2,3,2]  
delelems(X, 2)  
print(*X)
```

```
1 1 3 1 3
```

Как удалить строку из матрицы?

Как удалить строку из матрицы?

```
M = [[1, 2, 3], [4, 5, 6]]  
M.pop(1)
```


Ввод матриц по строкам

Допустим, мы хотим вводить матрицу по строкам, задавая заранее только количество ее строк.

Как нам это сделать?

Ввод матриц по строкам

Допустим, мы хотим вводить матрицу по строкам, задавая заранее только количество ее строк.

Как нам это сделать?

```
def fillmatr(A, n):  
    for i in range(n):  
        A.append(list(map(int, input().split())))  
  
X = []  
fillmatr(X, int(input('Введите кол-во строк: ')))  
print(X)
```


Задание на практику (Python)

1. Напишите программу, складывающую две матрицы. С клавиатуры вводятся размеры матриц, за ними - сами матрицы. Необходимо вывести итоговую матрицу. При выводе определяйте для каждого числа 4 позиции и прижимайте его к левой стороне. Решение оформить в виде функции.
2. Напишите программу, вычитающую матрицы. *Требования аналогичные предыдущей задаче*

Печать матрицы

```
A = [[1,2,3], [4,5,6]]  
  
for i in A:  
    for j in i:  
        print(j, end=' ')  
    print()
```

А как еще можно распечатать матрицу?

Печать матрицы

```
A = [[1,2,3], [4,5,6]]

for i in A:
    for j in i:
        print(j, end=' ')
    print()
```

```
A = [[1,2,3], [4,5,6]]

for i in A:
    print(' '.join(list(map(str, i))))
```

Прочитаем матрицу из файла

Пусть в файле записаны данные матрицы по строкам через пробелы. Прочитаем их, запишем в матрицу и выведем на экран.

```
def readmatrix(filename):  
    matr = []  
    file = open(filename, "rt")  
    for x in file:  
        matr.append(list(map(int, x.split())))  
    file.close()  
    return matr
```

```
def readmatrix1(filename):  
    with open(filename, "rt", encoding="utf-8") as file:  
        matr = [list(map(int, x.split())) \  
                 for x in file.readlines()]  
    return matr
```


Запишем матрицу в файл

```
def writematr(filename):  
    matr = [[10, -2, 4, 5], [1, 2, -1, 2], [5, 3, -1, 2]]  
    with open(filename, "wt") as f:  
        for line in matr:  
            print(" ".join(list(map(str, line))), file=f)  
  
writematr("matr2.txt")
```

Работа с матрицей

Заполнить матрицу

1 0 0 0

2 1 0 0

2 2 1 0

2 2 2 1

- 1) Главная диагональ заполняется единицами
- 2) Все, что выше главной диагонали - нулями
- 3) Все, что ниже - двойками

Вариант №1

```
n = int(input())
A = []
for i in range(n):
    A.append([0] * n)
for i in range(n):
    for j in range(n):
        if i > j:
            A[i][j] = 2
        elif i == j:
            A[i][j] = 1
for i in A:
    print(' '.join(list(map(str, i))))
```


Вариант №2

```
n = int(input())
A = []
for i in range(n):
    A.append([0] * n)

for i in range(n):
    for j in range(i):
        A[i][j] = 2
    A[i][i] = 1

for i in A:
    print(' '.join(list(map(str, i))))
```

Вариант №3

```
n = int(input())
A = []

for i in range(n):
    A.append([2] * i + [1] + [0] * (n - i - 1))

for i in A:
    print(' '.join(list(map(str, i))))
```

Вариант №4

```
n = int(input())

A = [[2] * i + [1] + [0] * (n-i-1) for i in
range(n)]

for i in A:
    print(' '.join(list(map(str, i))))
```


Копирование матриц

1. Матрицы (списки списков) хранятся, как ссылки, поэтому операция присваивания не порождает новую матрицу, а создает новую ссылку
2. У нас есть операция срез, которая может сделать копию списка. Что будет выведено и почему?

```
a = [[1, 2], [3, 4]]  
b = a[:]  
b.append([5, 6])  
b[0][0] = 100  
print(b)  
print(a)
```

Копирование матриц (продолжение)

Для удобства копирования списков и матриц и других объектов лучше применять следующий способ.

```
import copy

a = [[1, 2], [3, 4]]
b = copy.copy(a)      // поверхностное копирование
c = copy.deepcopy(a)  // глубокое копирование

b.append([5, 6])
b[0][0] = 100
c.append([5, 6])
c[0][0] = 100

print(b)
print(a)
print(c)
```

Задание на практику (Python)

1. Программа получает на вход размеры матрицы n и m , затем n строк по m целых чисел в каждой. Найдите индексы первого вхождения максимального элемента.
2. Даны два числа n и m . Создайте матрицу размером n на m и заполните ее символами "." и "*" в шахматном порядке. В левом верхнем углу должна стоять точка.

Вопросы?

8-800-201-01-50

brainskills.ru

