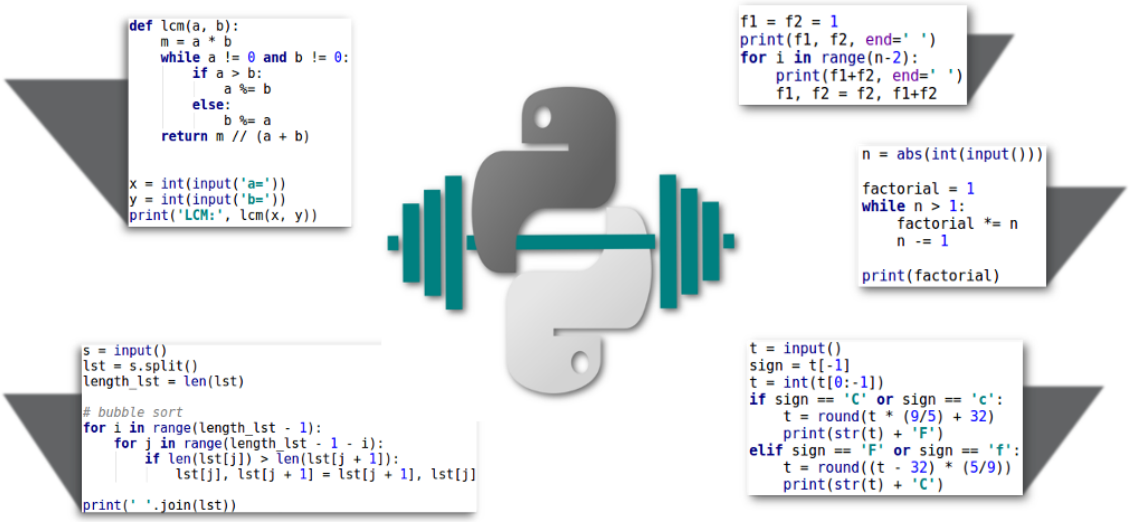


100 задач на Python

с решением

https://younglinux.info/python/task

Сборник подготовила: Светлана Шапошникова (plustilino)

Версия: июнь 2020 года

Содержание

Перечень задач

Линейные алгоритмы

Ветвления

№	*Формулировка задачи*	*Страница ответа*
1	Дана следующая функция *y* = f(*x*): *y* = *x* - 0.5, при *x* > 0; *y* = 0, при *x* = 0; *y* = |*x*|, при *x* < 0. Написать программу, определяющую значение *y* по переданному значению *x*.	26
2	Написать программу, которая вычисляет массу, плотность или объем, используя формулу *m = Vρ*. Пользователь выбирает, что он хочет вычислить и вводит два известных параметра.	27
3	В зависимости от того, что выберет пользователь, вычислить площадь либо прямоугольника (S = ab), либо треугольника ($S=\sqrt{p(p-a)(p-b)(p-c)}$, где p – полупериметр), либо круга (S = πr^2). Если выбраны прямоугольник или треугольник, то надо запросить длины сторон, если круг, то его радиус.	28
4	По введенным длинам трех отрезков определить, можно ли из них составить треугольник при условии, что у треугольника сумма любых двух сторон должна быть не меньше третьей.	29
5	Определить, принадлежит ли точка с координатами (x; y) кругу радиуса R с центром в начале координат. Пользователь вводит координаты точки и радиус круга.	30
6	Написать программу, которая выполняет над двумя вещественными числами одну из четырех арифметических операций (сложение, вычитание, умножение или деление). Числа и операцию вводит пользователь.	31
7	По введенным координатам точки определить, в какой координатной четверти она находится.	32
8	Вводится год. Определить, является ли он високосным или обычным.	34
9	Пользователь вводит температуру в Цельсиях или Фаренгейтах. Написать программу, которая переводит Цельсии в Фаренгейты по формуле *C * (9/5) + 32*, а Фаренгейты в Цельсии по формуле *(F - 32) * (5/9)*.	35
10	Вычислить корни квадратного уравнения, коэффициенты *a*, *b* и *c* которого вводятся с клавиатуры.	36

Циклы (часть 1)

№	*Формулировка задачи*	*Страница ответа*
1	Напишите цикл, который считает количество разрядов введенного с клавиатуры числа.	37
2	Определить, сколько в числе четных цифр, а сколько нечетных. Число вводится с клавиатуры.	38
3	Вводится натуральное число. Найти сумму и произведение цифр, из которых состоит это число. Например, сумма цифр числа 253 равна 10-ти, так как 2 + 5 + 3 = 10. Произведение цифр числа 253 равно 30-ти, так как 2 * 5 * 3 = 30.	39
4	Вывести число обратное введенному по порядку составляющих его цифр. Например, введено 3425, надо вывести 5243.	40
5	Вывести степени натуральных чисел, не превосходящие данного числа *n*. Пользователь задает показатель степени и число *n*.	42
6	Напишите программу вычисления факториала натурального числа. Число вводят с клавиатуры. Факториалом числа называют произведение всех натуральных чисел до него включительно. Например, факториал числа 5 равен произведению 1 * 2 * 3 * 4 * 5 = 120.	43
7	Вычислите *n*-ный элемент ряда Фибоначчи. Номер элемента (*n*) вводится с клавиатуры. Числа Фибоначчи – это ряд чисел, в котором каждое следующее число равно сумме двух предыдущих: 1, 1, 2, 3, 5, 8, 13, ...	44
8	Выведите на экран ряд Фибоначчи до *n*-ного элемента включительно. Номер последнего элемента (*n*) вводится с клавиатуры. Числа Фибоначчи – это ряд чисел, в котором каждое следующее число равно сумме двух предыдущих: 1, 1, 2, 3, 5, 8, 13, ...	45
9	Выведите на экран в табличном виде символы с номерами от 32 до 126-ти включительно.	46
10	Выведите на экран таблицу умножения.	47

Циклы (часть 2)

№	*Формулировка задачи*	*Страница ответа*
1	Компьютер "загадывает" случайное число от 1 до 100. Человек должен его угадать, сделав как можно меньшее число попыток. Перед каждой попыткой сообщается ее номер. Программа завершается, когда пользователь угадывает число.	48
2	Вывести на экран "прямоугольник", образованный из двух видов символов. Контур прямоугольника должен состоять из одного символа, а "заливка" - из другого.	49
3	Дана функция $y = -3x^2 - 4x + 20$. Вывести таблицу значений данной функции на отрезке и с шагом, которые вводит пользователь.	50
4	Дано действительное число a и натуральное число n. Написать программу, которая вычисляет произведение $a(a+1)(a+2)...(a+n-1)$.	51
5	Дан ряд чисел 1, -0.5, 0.25, -0.125, ... Требуется найти сумму стольких его элементов, сколько указал пользователь. Например, если n = 3, то надо сложить 1, -0.5, 0.25, что в результате даст 0.75.	52
6	Дана последовательность 1/2 - 2/4 + 3/8 - 4/16 + ... - … . Найти сумму и количество элементов последовательности, которые по модулю больше 0.001.	53
7	Напишите цикл, который переводит число из десятичной системы счисления в двоичную.	54
8	Вводится десятичное число и желаемая система счисления с основанием до 9. Перевести десятичное число в заданную систему счисления.	55
9	Проверьте, является ли введенное с клавиатуры целое число простым. Используйте метод перебора делителей.	56
10	Напишите цикл, который вычисляет наибольший общий делитель двух чисел, используя любую из вариаций алгоритма Евклида.	57

Списки (часть 1)

Списки (часть 2)

Матрицы

Строки

№	*Формулировка задачи*	*Страница ответа*
1	Вводится строка. Определить является ли она палиндромом и вывести соответствующее сообщение. Палиндромом называется слово или фраза, которые читаются одинаково справа налево, и слева направо.	93
2	Вводится имя файла. Требуется проверить, что его расширение входит в список допустимых.	94
3	Вводится строка, состоящая из слов, разделенных пробелами. Требуется сформировать новую строку из тех же слов, но так, чтобы слова в этой строке были отсортированы по возрастанию их длины.	95
4	Вводится строка. Необходимо определить в ней проценты прописных, то есть больших, и строчных, то есть малых, букв.	96
5	Определить длину самого короткого слова в строке.	97
6	Вводится строка, состоящая из нескольких слов. Выведите на экран строку с обратным порядком слов.	98
7	В строке найти и заменить одну подстроку на другую. Если одинаковых подстрок несколько, заменить все.	99
8	В строке заменить пробелы звездочкой. Если встречается подряд несколько пробелов, то их следует заменить одним знаком "*", пробелы в начале и конце строки удалить.	100
9	Дана строка, содержащая натуральные числа и слова. Необходимо сформировать список из чисел, содержащихся в этой строке. Например, задана строка "abc83 cde7 1 b 24". На выходе должен получиться список [83, 7, 1, 24].	101
10	Вводится строка. Требуется удалить из нее повторяющиеся символы и все пробелы. Например, если было введено "abc cde def", то должно быть выведено "abcdef".	103

Функции

№	*Формулировка задачи*	*Страница ответа*
1	Напишите функцию, которая вычисляет среднее арифметическое двух переданных ей числовых аргументов.	104
2	Напишите функцию, которая вычисляет среднее арифметическое произвольного числа аргументов.	105
3	Напишите функцию, которая заполняет список случайными числами. В качестве аргументов функция должна принимать список, количество элементов, минимальное и максимальное значение, то есть границы диапазона.	106
4	Напишите функцию, которая выполняет бинарный поиск элемента в списке. В качестве аргументов функция принимает список и значение, возвращает – индекс элемента с заданным значением.	107
5	Напишите функцию, которая выводит на экран ряд Фибоначчи в количестве элементов, соответствующим переданному в нее аргументу.	108
6	Напишите функцию, которая вычисляет факториал числа с помощью рекурсии.	109
7	Напишите функцию, которая проверяет число на простоту.	110
8	Напишите функцию, вычисляющую наименьшее общее кратное двух целых чисел.	111
9	Напишите функцию, выполняющую циклический сдвиг в списке целых чисел на указанное число шагов. Сдвиг должен быть кольцевым, то есть элемент, вышедший за пределы списка, должен появляться с другого его конца.	112
10	Список заполняется случайными числами. Требуется посчитать сколько раз в нем встречается каждое число. Заполнение списка и его анализ реализовать в виде отдельных функций.	113

Словари и файлы

№	*Формулировка задачи*	*Страница ответа*
1	Даны два списка. Создайте словарь, ключами которого являются элементы первого списка, а значениями – элементы второго, находящиеся в соответствующих позициях.	115
2	Дан словарь. Напишите программу, которая удаляет случайный элемент словаря.	116
3	Дан словарь, ключами которого являются строки, а значениями – числа. Выведите его на экран сначала в отсортированном по ключам виде (в алфавитном порядке), затем – отсортированным по возрастанию значений.	117
4	Дан словарь, в котором ключами являются названия товаров, а значениями – их цены. Напишите программу, которая выводит на экран товары и цены, запрашивает у пользователя, что он хочет купить, в каком количестве, и считает общую цену покупки.	118
5	Напишите программу, которая читает текстовый файл и помещает его содержимое в список строк.	119
6	Напишите программу, которая считает сколько раз в файле встречается заданное слово.	120
7	Определить, сколько в текстовом файле строк, слов и символов.	121
8	Дан текстовый файл, каждая строка которого содержит название товара, его цену и количество. Напишите программу, которая помещает содержимое этого файла в словарь так, что каждая запись словаря соответствует одной строке файла. Ключом записи является название товара, значением – список, первый элемент которого цена, второй – количество товара.	123
9	Имеется словарь, ключами которого являются товары, а значениями – их цена. Запишите данные словаря в файл так, чтобы в каждой строке находилась одна запись словаря.	124
10	В текстовом файле заменить все символы табуляции четырьмя пробелами.	125

Задачи с решением

Линейные алгоритмы

Школьники делят яблоки

> Школьники делят яблоки поровну так, чтобы каждому достались только целые яблоки, остальные яблоки остаются в корзине. Определить, сколько яблок достанется каждому школьнику и сколько яблок останется в корзине.

Данная задача сводится к двум операциям:

1. Делению нацело. Так находится количество целых яблок, приходящихся на каждого школьника.
2. Нахождению остатка от деления нацело. Так находится количество яблок, оставшихся в корзине.

В языке Python первая операция обозначается двумя знаками слэша `//`, вторая – знаком процента %. Обратите внимание, что при формировании строки (в скобках функции print) знак процента имеет совершенно другое значение – формат вывода.

```
pupils = input("Количество школьников: ")
apples = input("Количество яблок: ")

pupils = int(pupils)
apples = int(apples)

apples_on_pupil = apples // pupils
apples_in_basket = apples % pupils

print("У каждого школьника будет по %d яблок" % apples_on_pupil)
print("В корзине останется %d яблок" % apples_in_basket)
```

Пример выполнения:

```
Количество школьников: 10
Количество яблок: 25
У каждого школьника будет по 2 яблок
В корзине останется 5 яблок
```

Можно не использовать операцию нахождения остатка от деления. Вместо этого вычислить оставшиеся в корзине яблоки, вычтя из их исходного количества произведение школьников на яблоки, которые получил каждый из них.

```
…
apples_in_basket = apples - pupils * apples_on_pupil
…
```

Получить букву по ее номеру в алфавите

С клавиатуры вводится номер (позиция) буквы в английском алфавите. Вывести на экран запрашиваемый символ.

Для решения данной задачи потребуются две встроенные в Python функции – ord() и chr(). Функция ord() принимает строку, состоящую из одного символа, и возвращает его числовой код по таблице символов. Например, ord('a') вернет число 97.

Если к полученному числовому коду первой буквы алфавита прибавить порядковый номер необходимой нам буквы в алфавите и вычесть единицу, получим числовой код искомой буквы.

С помощью функции chr() получаем по нему требуемый символ.

```
number = input("Какую по счету букву алфавита вывести: ")
number = int(number)

first_letter_code = ord('A')
your_letter_code = first_letter_code + number - 1
your_letter = chr(your_letter_code)

print('Это буква "{}"'.format(your_letter))
```

Пример выполнения:

```
Какую по счету букву алфавите вывести: 10
Это буква "J"
```

Площадь и периметр прямоугольного треугольника

С клавиатуры вводятся длины двух катетов прямоугольного треугольника. Программа должна вычислять площадь и периметр этого треугольника и выводить полученные значения на экран.

Площадь прямоугольного треугольника равна половине площади прямоугольника, стороны которого равны длинам катетов. Это значит, чтобы найти площадь прямоугольного треугольника, надо перемножить катеты и разделить полученное произведение на два.

Периметр находится путем сложения длин всех сторон треугольника. Поскольку известны только катеты, необходимо вычислить гипотенузу по теореме Пифагора: $c^2 = a^2 + b^2$. Откуда длина гипотенузы равна:

$$c=\sqrt{a^2+b^2}$$

Чтобы извлечь квадратный корень в Python, можно воспользоваться функцией sqrt() из модуля math.

Пример кода:

```
import math

AB = input("Длина первого катета: ")
AC = input("Длина второго катета: ")

AB = float(AB)
AC = float(AC)

BC = math.sqrt(AB**2 + AC**2)

S = (AB * AC) / 2
P = AB + AC + BC

print("Площадь треугольника: %.2f" % S)
print("Периметр треугольника: %.2f" % P)
```

Пример выполнения:

```
Длина первого катета: 5.4
Длина второго катета: 2.1
Площадь треугольника: 5.67
Периметр треугольника: 13.29
```

Случайное целое в заданных границах

С клавиатуры вводятся границы числового диапазона. Получите случайное целое число в его пределах и выведите его на экран.

В модуле random есть функция randint(), которая генерирует случайное целое число. В качестве аргументов ей передаются границы диапазона (минимум и максимум).

```
from random import randint

a = input("Нижняя граница: ")
b = input("Верхняя граница: ")

a = int(a)
b = int(b)

n = randint(a, b)

print(n)
```

Пример выполнения:

```
Нижняя граница: 10
Верхняя граница: 20
13
```

Передаваемые в функцию randint() числа входя в диапазон. На языке математики это описывается как [a, b]. Если надо, чтобы верхняя граница не входила – [a, b), то из b надо вычесть 1 или воспользоваться функций randrange() из того же модуля.

```
from random import randrange

...

n = randrange(a, b)
```

Также данную задачу можно решить более сложным, но более универсальным способом, так как не в каждом языке есть несколько специализированных функции для получения тех или иных случайных значений. В модуле random есть одноименная функция random(), генерирующая случайное вещественное число от 0 до 1. Единица в диапазон не входит.

Если это число умножить на длину диапазона (b - a), результат привести к целому и прибавить нижнюю границу, мы получим число в диапазоне [a, b).

```
from random import random

...

n = int(random() * (b - a)) + a
```

Случайное вещественное в заданных границах

Получите случайное вещественное число в диапазоне от 0. 10 до 0. 50. Выведите его на экран с двумя знаками после запятой (точки) .

Функция random() из модуля random генерирует случайное вещественное число от 0 до 1. Если его умножить на длину диапазона, в данном случае равную 0.4 = 0.5 - 0.1, получим случайное от 0 до 0.4. Если теперь добавить нижнюю границу, равную 0.1, получим случайное от 0.1 до 0.5.

Вывести вещественное число с двумя знаками после запятой можно несколькими способами. Во-первых, воспользоваться функцией round(). Если ей передать второй аргумент, то она округлит не до целого, а до указанного количества знаков после запятой.

Во-вторых, можно воспользоваться возможностью форматирования строки в старом стиле. Запись `%.2f` означает, что мы выводим вещественное число с двумя знаками после запятой. Здесь также происходит округление, а не простое отбрасывание лишних знаков.

```
from random import random

a = 0.1
b = 0.5

n = random() * (b - a) + a

print(round(n, 2))  # 1-й вариант
print("%.2f" % n)  # 2-й вариант
```

Пример выполнения:

```
0.17
0.17
```

Сумма цифр трехзначного числа

Написать программу, которая генерирует случайное трехзначное число и вычисляет сумму его цифр.

Проще всего случайное трехзначное число сгенерировать функцией randint() модуля random.

Чтобы извлечь отдельные цифры числа, следует воспользоваться операциями деления нацело (//) и нахождения остатка от деления (%) на 10.

Если найти остаток от деления на 10 исходного трехзначного числа *n*, то получим последнюю цифру этого числа (*d1*). Если разделить нацело на 100 трехзначное число, то получим первую его цифру (*d3*).

Чтобы получить вторую цифру (*d2*) трехзначного числа, сначала найдем остаток от деления на 100. Получим двузначное число – две последние цифры трехзначного числа. Потом разделим нацело на 10 и получим первую цифру этого двузначного числа.

```
from random import randint

n = randint(100, 999)
print("Случайное число:", n)

d1 = n % 10
d2 = n % 100 // 10
d3 = n // 100

print("Сумма его цифр =", d1 + d2 + d3)
```

Пример выполнения:

```
Случайное число: 457
Сумма его цифр = 16
```

Есть другой, не математический, способ решения данной задачи – путем преобразования числа к строковому представлению с последующим извлечением каждого символа по индексу и его обратного перевода в число.

```
from random import randint

n = str(randint(100, 999))
print("Случайное число:", n)

d1 = int(n[0])
d2 = int(n[1])
d3 = int(n[2])

print("Сумма его цифр =", d1 + d2 + d3)
```

Однако в подобных задачах обычно имеют в виду первый способ, так как тип данных "строки" может быть еще не изучен.

Длина окружности и площадь круга

По введенному с клавиатуры радиусу вычислить длину соответствующей окружности и площадь круга. Формулы: $L = 2\pi r$, $S = \pi r^2$.

```
import math

r = input("Radius = ")
r = float(r)

ln = 2 * math.pi * r
area = math.pi * math.pow(r, 2)

print("Length = %.2f" % ln)
print("Area = %.2f" % area)
```

Пример выполнения:

```
Radius = 3.8
Length = 23.88
Area = 45.36
```

Данную задачу можно решить без использования "константы" и функции модуля math. Так в самом Питоне есть оператор возведения в степень – два знака звездочки `**`. Вместо "константы" `math.pi` можно просто записать число 3.14. Однако при этом пострадает точность вычисления.

```
r = input("Radius = ")
r = float(r)

ln = 2 * 3.14 * r
area = 3.14 * r**2

print("Length = %.2f" % ln)
print("Area = %.2f" % area)
```

Пример выполнения:

```
Radius = 3.8
Length = 23.86
Area = 45.34
```

Поскольку значение 3.14 играет роль константы и используется несколько раз, мы можем ввести свою "константу".

```
pi = 3.14
ln = 2 * pi * r
area = pi * r**2
```

Обратим внимание, в языке Python нет настоящих констант – переменных, навсегда привязанных к одному значению.

Площадь поверхности цилиндра

Напишите программу, которая вычисляет площадь поверхности цилиндра по заданным с клавиатуры радиусу и высоте цилиндра. Формула площади одного основания цилиндра: $S = \pi r^2$. Формула площади боковой поверхности цилиндра: $S = 2\pi rh$.

Чтобы найти общую площадь поверхности цилиндра, надо сложить площадь его боковой поверхности с площадями двух оснований.

Значение числа π можно ввести вручную (3.14), однако для более точных расчетов можно воспользоваться переменной `pi` из модуля math.

```
from math import pi

h = float(input('h = '))
r = float(input('r = '))

circles = 2 * (pi * r**2)
side = 2 * pi * r * h
area = circles + side

print('A =', round(area, 2))
```

Пример выполнения:

```
h = 2.3
r = 1
A = 20.73
```

Уравнение прямой

Напишите программу, которая выводит уравнение прямой y = kx + b, проходящей через заданные точки. Координаты точек вводятся с клавиатуры.

Уравнение прямой на координатной плоскости имеет следующий вид:

$$y = kx + b$$

Если известны координаты двух лежащих на этой прямой точек, можно определить значения коэффициентов k и b, решив систему уравнений. Таким образом выводится уравнение конкретной прямой, например, `y = 3x - 1`.

Система уравнений:

$$\begin{cases} y_1 = kx_1 + b \\ y_2 = kx_2 + b \end{cases}$$

Вывод формул для коэффициентов k и b:

$b = y_2 - kx_2$

$y_1 = kx_1 + y_2 - kx_2$

$k = (y_1 - y_2) / (x_1 - x_2)$

```
print("Координаты точки A(x1; y1):")
x1 = float(input("\tx1 = "))
y1 = float(input("\ty1 = "))

print("Координаты точки B(x2; y2):")
x2 = float(input("\tx2 = "))
y2 = float(input("\ty2 = "))

print("Уравнение прямой, проходящей через эти точки:")
k = (y1 - y2) / (x1 - x2)
b = y2 - k*x2
print(" y = %.2f*x + %.2f" % (k, b))
```

Примеры выполнения программы:

```
Координаты точки A(x1; y1):
      x1 = 4.3
      y1 = -1.2
Координаты точки B(x2; y2):
      x2 = -8.5
      y2 = 4
Уравнение прямой, проходящей через эти точки:
 y = -0.41*x + 0.55
```

Побитовые операции над двоичными числами

С клавиатуры вводятся два двоичных числа. Требуется вывести на экран в двоичном виде результаты побитовых операций И (&), ИЛИ (|), ИСКЛЮЧАЮЩЕЕ ИЛИ (^) над этими числами.

Побитовые операции выполняются над разрядами двоичного числа. Так если имеем два числа 101 и 100, над которыми выполняется побитовая операция ИЛИ, то получим двоичное число 101:

```
ИЛИ: 101    И: 101    ИСКЛЮЧАЮЩЕЕ ИЛИ: 101
     100        100                     100
     101        100                     001
```

Знаки побитовых операций в Python: | - ИЛИ, & - И, ^ - исключающее ИЛИ. Проблема в том, что в Python побитовые операции применяются к двоичным числам, но представленным в десятичной системе счисления, и в десятичной системе счисления они возвращают результат.

Таким образом, введенные пользователем двоичные числа надо преобразовать к десятичному виду, выполнить побитовые операции, результат которых преобразовать обратно к двоичному представлению.

Функция int() всегда преобразует переданную ей строку в десятичную систему счисления. При этом вторым аргументом можно указать, в какой системе счисления содержатся данные в этой строке (двоичной, восьмеричной, шестнадцатеричной).

Функция bin(), наоборот, принимает десятичное число и возвращает строку, являющуюся представлением переданного числа в двоичном виде.

```
n1 = input("Введите первое двоичное число: ")
n2 = input("Введите второе двоичное число: ")

n1 = int(n1,2)
n2 = int(n2,2)

bit_or = n1 | n2
bit_and = n1 & n2
bit_xor = n1 ^ n2

print("Результат побитового OR: %10s" % bin(bit_or))
print("Результат побитового AND: %10s" % bin(bit_and))
print("Результат побитового XOR: %10s" % bin(bit_xor))
```

Пример выполнения:

```
Введите первое двоичное число: 10100
Введите второе двоичное число: 10001
Результат побитового  OR:    0b10101
Результат побитового AND:    0b10000
Результат побитового XOR:      0b101
```

Ветвления

Функция y = f(x)

Дана следующая функция $y = f(x)$:

$y = x - 0.5$, при $x > 0$;

$y = 0$, при $x = 0$;

$y = |x|$, при $x < 0$.

Написать программу, определяющую значение y по переданному значению x.

```
x = float(input())

if x > 0:
    y = x - 0.5
elif x < 0:
    y = abs(x)
elif x == 0:
    y = 0

print(y)
```

Пример выполнения:

```
3.43
2.93
```

Последнюю ветку `elif` можно заменить на `else`, так как если `x` не больше и не меньше нуля, то равен нулю. Однако с `elif` код выглядит яснее.

Обратите внимание, что в Python нет необходимости определять переменную `y` до условного оператора. Не смотря на то, что первый раз она используется в теле условного оператора, она не является локальной, а видна в глобальной области видимости.

Однако в большинстве случаев такую переменную первый раз лучше определять за пределами условного оператора. Например, если в программе выше не было бы ветки `elif x == 0`, а пользователь ввел бы значение равное нулю, то переменная `y` оказалась бы неопределенной, что привело бы к ошибке. Поэтому программу можно написать еще и так:

```
x = float(input())
y = 0

if x > 0:
    y = x - 0.5
elif x < 0:
    y = abs(x)

print(y)
```

Вычислить массу, плотность или объем

Написать программу, которая вычисляет массу, плотность или объем, используя формулу $m = V\rho$. Пользователь выбирает, что он хочет вычислить и вводит два известных параметра.

```
flag = input("Что вычислить? (m, d, v): ")

result = 0

if flag == 'm':
    d = float(input("Плотность: "))
    v = float(input("Объем: "))
    result = d * v  # масса
elif flag == 'd':
    m = float(input("Масса: "))
    v = float(input("Объем: "))
    result = m / v  # плотность
elif flag == 'v':
    m = float(input("Масса: "))
    d = float(input("Плотность: "))
    result = m / d  # объем

print("%.2f" % result)
```

Пример выполнения:

```
Что вычислить? (m, d, v): d
Масса: 3.4
Объем: 2
1.70
```

Площадь прямоугольника, треугольника или круга

В зависимости от того, что выберет пользователь, вычислить площадь либо прямоугольника ($S = ab$), либо треугольника ($S = \sqrt{p(p-a)(p-b)(p-c)}$, где p – полупериметр), либо круга ($S = \pi r^2$). Если выбраны прямоугольник или треугольник, то надо запросить длины сторон, если круг, то его радиус.

```
figure = input("Выберите фигуру (1-прямоугольник, 2-треугольник, 3-круг): ")

if figure == '1':
    print("Длины сторон прямоугольника:")
    a = float(input("a = "))
    b = float(input("b = "))
    print("Площадь: %.2f" % (a * b))
elif figure == '2':
    print("Длины сторон треугольника:")
    a = float(input("a = "))
    b = float(input("b = "))
    c = float(input("c = "))
    p = (a + b + c) / 2
    import math
    s = math.sqrt(p * (p - a) * (p - b) * (p - c))
    print("Площадь: %.2f" % s)
elif figure == '3':
    r = float(input("Радиус круга R = "))
    import math
    print("Площадь: %.2f" % (math.pi * r ** 2))
else:
    print("Ошибка ввода")
```

Пример выполнения:

```
Выберите фигуру (1-прямоугольник, 2-треугольник, 3-круг): 2
Длины сторон треугольника:
a = 4.5
b = 2.8
c = 6.1
Площадь: 5.87
```

Существует ли треугольник

По введенным длинам трех отрезков определить, можно ли из них составить треугольник при условии, что у треугольника сумма любых двух сторон должна быть не меньше третьей.

Поскольку всего три стороны, то можно составить три варианта сложения двух сторон: `a + b`, `b + c`, `a + c`. Первую сумму сравниваем с оставшейся стороной `c`, вторую - с `a` и третью - с `b`. Если хотя бы в одном случае сумма окажется меньше третьей стороны, то делается вывод, что треугольник не существует.

```
print("Введите длины сторон предполагаемого треугольника:")
a = float(input("a = "))
b = float(input("b = "))
c = float(input("c = "))

if a + b >= c and a + c >= b and b + c >= a:
    print("Треугольник существует")
else:
    print("Треугольник не существует")
```

Можно решить задачу сложнее. Если требуется также определить, какая из сторон больше суммы двух других, то решение может быть таким:

```
print("Введите длины сторон предполагаемого треугольника:")
a = float(input("a = "))
b = float(input("b = "))
c = float(input("c = "))

flag = ''
if a + b > c:
    if a + c > b:
        if b + c > a:
            print("Треугольник существует")
        else:
            flag = 'a'
    else:
        flag = 'b'
else:
    flag = 'c'

if flag != '':
    print("Треугольник НЕ существует. ", end='')
    print("Сторона '%s' длиннее или равна сумме двух других." % flag)
```

```
Введите длины сторон предполагаемого треугольника:
a = 10
b = 2
c = 3
Треугольник НЕ существует. Сторона 'a' длиннее или равна сумме двух других.
```

Принадлежит ли точка кругу?

Определить, принадлежит ли точка с координатами $(x; y)$ кругу радиуса R с центром в начале координат. Пользователь вводит координаты точки и радиус круга.

Если выбрать точку на координатной плоскости, то можно увидеть, что проекции ее координат на оси *x* и *y* являются катетами прямоугольного треугольника. А гипотенуза этого прямоугольного треугольника как раз показывает расстояние от начала координат до точки. Таким образом, если длина гипотенузы будет меньше радиуса круга, то точка будет принадлежать кругу; иначе она будет находится за его пределами.

Длину гипотенузы вычисляется по теореме Пифагора: квадрат гипотенузы равен сумме квадратов катетов. Откуда гипотенуза равна квадратному корню из суммы квадратов катетов.

```
import math

print("Введите координаты точки и радиус круга")
x_point = float(input("x = "))
y_point = float(input("y = "))
r_circle = float(input("R = "))

hypotenuse = math.sqrt(x_point ** 2 + y_point ** 2)

if hypotenuse <= r_circle:
    print("Точка принадлежит кругу")
else:
    print("Точка НЕ принадлежит кругу")
```

Пример выполнения программы:

```
x = 1
y = -1
R = 3
Точка принадлежит кругу
```

Калькулятор

Написать программу, которая выполняет над двумя вещественными числами одну из четырех арифметических операций (сложение, вычитание, умножение или деление). Числа и операцию вводит пользователь.

При делении следует обработать случай, когда делитель является нулем.

```
s = input("Знак (+,-,*,/): ")
if s in ('+', '-', '*', '/'):
    x = float(input("x="))
    y = float(input("y="))
    if s == '+':
        print("%.2f" % (x+y))
    elif s == '-':
        print("%.2f" % (x-y))
    elif s == '*':
        print("%.2f" % (x*y))
    elif s == '/':
        if y != 0:
            print("%.2f" % (x/y))
        else:
            print("Деление на ноль!")
else:
    print("Неверный знак операции!")
```

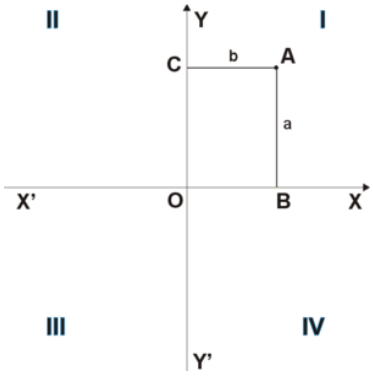
Пример выполнения:

```
Знак (+,-,*,/): /
x=67
y=34
1.97
```

В какой координатной четверти находится точка?

По введенным координатам точки определить, в какой координатной четверти она находится.

В прямоугольной системе координат на плоскости выделяют четверти:



I-й четветри соответствуют точки, имеющие обе (x и y) положительные координаты.

II-ая четверть: $x < 0$, $y > 0$.

III-ая четверть: $x < 0$, $y < 0$.

IV-ая четверть: $x > 0$, $y < 0$.

Один вариант решения:

```
print("Введите координаты точки:")
x = float(input("x = "))
y = float(input("y = "))

if x > 0 and y > 0:
    print("Точка находится в I четверти")
elif x < 0 and y > 0:
    print("Точка находится в II четверти")
elif x < 0 and y < 0:
    print("Точка находится в III четверти")
elif x > 0 and y < 0:
    print("Точка находится в IV четверти")
elif x == 0 and y == 0:
    print("Точка находится в начале координат")
elif x == 0:
    print("Точка находится на оси 'x'")
elif y == 0:
    print("Точка находится на оси 'y'")
```

Пример выполнения:

```
Введите координаты точки:
x = 6
y = -3
Точка находится в IV четверти
```

Другой вариант решения задачи "В какой координатной четверти находится точка?":

```
print("Введите координаты точки:")
x = float(input("x = "))
y = float(input("y = "))

if x == 0 and y == 0:
    print("Точка находится в начале координат")
elif x == 0:
    print("Точка находится на оси 'x'")
elif y == 0:
    print("Точка находится на оси 'y'")
elif x > 0:
    if y > 0:
        print("Точка находится в I четверти")
    elif y < 0:
        print("Точка находится в IV четверти")
elif x < 0:
    if y > 0:
        print("Точка находится в II четверти")
    elif y < 0:
        print("Точка находится в III четверти")
```

Високосный год или нет?

Вводится год. Определить, является ли он високосным или обычным.

Високосными являются года, которые делятся на 4, за исключением столетий, которые не делятся на 400.

Если год не делится нацело на 4, значит сразу можно сделать вывод, что он не является високосным.

Если все же делится, надо исключить столетия за исключением тех, которые нацело делятся на 400.

```
year = int(input())
if year % 4 != 0:
    print("Обычный")
elif year % 100 == 0:
    if year % 400 == 0:
        print("Високосный")
    else:
        print("Обычный")
else:
    print("Високосный")
```

Короткое решение:

```
year = int(input())
if year % 4 != 0 or (year % 100 == 0 and year % 400 != 0):
    print("Обычный")
else:
    print("Високосный")
```

Цельсии в Фаренгейты или наоборот

Пользователь вводит температуру в Цельсиях или Фаренгейтах. Написать программу, которая переводит Цельсии в Фаренгейты по формуле *C * (9/5) + 32*, а Фаренгейты в Цельсии по формуле *(F - 32) * (5/9)*.

```
t = float(input("Введите температуру: "))
sign = input("Это цельсии(1) или фаренгейты(2)? ")

if sign == '1':
    t = t * (9/5) + 32
    print("%dF" % round(t))
elif sign == '2':
    t = (t - 32) * (5/9)
    print("%dC" % round(t))
```

Пример выполнения:

```
Введите температуру: 100
Это цельсии(1) или фаренгейты(2)? 2
38C
```

Квадратное уравнение

Вычислить корни квадратного уравнения, коэффициенты *a*, *b* и *c* которого вводятся с клавиатуры.

Квадратное уравнение имеет вид $ax^2 + bx + c = 0$.

При его решении сначала вычисляют дискриминант по формуле $D = b^2 - 4ac$.

Если $D > 0$, то квадратное уравнение имеет два корня; если $D = 0$, то 1 корень; и если $D < 0$, то делают вывод, что корней нет.

Таким образом, программа для нахождения корней квадратного уравнения может иметь три ветви условного оператора.

```
import math

print("Введите коэффициенты для квадратного уравнения (ax^2 + bx + c = 0):")
a = float(input("a = "))
b = float(input("b = "))
c = float(input("c = "))

discr = b ** 2 - 4 * a * c
print("Дискриминант D = %.2f" % discr)

if discr > 0:
    x1 = (-b + math.sqrt(discr)) / (2 * a)
    x2 = (-b - math.sqrt(discr)) / (2 * a)
    print("x1 = %.2f \nx2 = %.2f" % (x1, x2))
elif discr == 0:
    x = -b / (2 * a)
    print("x = %.2f" % x)
else:
    print("Корней нет")
```

Примеры выполнения кода:

```
Введите коэффициенты для квадратного уравнения (ax^2 + bx + c = 0):
a = 3.2
b = -7.8
c = 1
Дискриминант D = 48.04
x1 = 2.30
x2 = 0.14
```

Обратим внимание, что для данной программы коэффициент *a* не должен быть равен нулю. Иначе в первой ветке условного оператора будет происходить попытка деления на 0. Если $a = 0$, то квадратное уравнение превращается в линейное, которое решается иным способом. Оно всегда имеет один корень.

Циклы (часть 1)

Количество разрядов числа

Напишите цикл, который считает количество разрядов введенного с клавиатуры числа.

В данном случае следует использовать математический способ решения задачи – путем последовательного избавления от разрядов числа делением нацело на 10 в цикле.

Первое деление выносится за цикл потому, что если число равно нулю, то цикл while его не обработает, и счетчик цифр `count` останется равным нулю.

```
n = int(input())
n = abs(n)

count = 1
n //= 10

while n > 0:
    n //= 10
    count += 1

print(count)
```

Пример выполнения:

```
-123
3
```

Однако случай введения числа 0 можно обработать и через if:

```
if n == 0:
    count = 1
```

Обратим внимание, что если число оканчивается на 0, например 150, программа работает правильно, так как от разрядов числа избавляются с конца, а не с начала. Первое деление нацело учтет цифру 0 в счетчике `count`.

Если же перед числом поставить 0, например 015, программа насчитает 2 разряда, так как проверка `n > 0` вернет ложь, когда от числа останется одна первая цифра.

Четные и нечетные цифры числа

Определить, сколько в числе четных цифр, а сколько нечетных. Число вводится с клавиатуры.

Если число делится без остатка на 2, его последняя цифра четная. Увеличиваем на 1 счетчик четных цифр `even`. Иначе последняя цифра числа нечетная, увеличиваем счетчик нечетных цифр `odd`.

В Python операцию нахождения остатка от деления выполняет знак `%`.

Чтобы избавиться от младшего уже учтенного разряда, число следует разделить нацело на 10. Деление нацело обозначается двумя слэшами `//`.

```
a = input()
a = int(a)

even = 0
odd = 0

while a > 0:
    if a % 2 == 0:
        even += 1
    else:
        odd += 1
    a = a // 10

print("Even: %d, odd: %d" % (even, odd))
```

Пример выполнения:

```
65439
Even: 2, odd: 3
```

Кроме чисто математического подхода в Python можно решить задачу "через строку". Мы не будем переводить введенное строковое представление числа к целочисленному типу, вместо этого переберем символы строки в цикле for. Каждый символ преобразуем к числу и проверим на четность.

```
a = input()

even = 0
odd = 0

for i in a:
    if int(i) % 2 == 0:
        even += 1
    else:
        odd += 1

print("Even: %d, odd: %d" % (even, odd))
```

Сумма и произведение цифр числа

> Вводится натуральное число. Найти сумму и произведение цифр, из которых состоит это число. Например, сумма цифр числа 253 равна 10-ти, так как 2 + 5 + 3 = 10.
>
> Произведение цифр числа 253 равно 30-ти, так как 2 * 5 * 3 = 30.

Если число разделить нацело на 10, произойдет "потеря" последней цифры числа. Например, 253 ÷ 10 = 25 (остаток 3). С другой стороны, эта потерянная цифра есть остаток от деления. Получив эту цифру, мы можем добавить ее к сумме цифр и умножить на нее произведение цифр числа.

Пусть *n* – само число, *suma* – сумма его цифр, а *mult* – произведение. Тогда алгоритм нахождения суммы и произведения цифр можно словесно описать так:

1. Переменной *suma* присвоить ноль.
2. Переменной *mult* присвоить единицу. Присваивать 0 нельзя, так как при умножении на ноль результат будет нулевым.
3. Пока значение переменной *n* больше нуля повторять следующие действия:
   a. Найти остаток от деления значения *n* на 10, то есть извлечь последнюю цифру числа.
   b. Добавить извлеченную цифру к сумме и увеличить на эту цифру произведение.
   c. Избавиться от последнего разряда числа *n* путем деления нацело на 10.

В языке Python операция нахождения остатка от деления обозначается знаком процента - %. Деление нацело - двумя слэшами - `//`.

```
n = int(input())

suma = 0
mult = 1

while n > 0:
    digit = n % 10
    suma = suma + digit
    mult = mult * digit
    n = n // 10

print("Сумма:", suma)
print("Произведение:", mult)
```

Пример выполнения:

```
253
Сумма: 10
Произведение: 30
```

Перевернуть число

Вывести число обратное введенному по порядку составляющих его цифр. Например, введено 3425, надо вывести 5243.

Алгоритм:

1. Найдем остаток от деления на 10 исходного числа. Тем самым получим последнюю его цифру.
2. Добавим эту цифру к новому числу.
3. Разделим нацело на 10 исходное число. Тем самым избавимся от последней цифры в нем.
4. Снова найдем остаток от деления на 10 того, что осталось от первого числа. Запомним эту цифру.
5. Умножим на 10 второе число. Тем самым увеличим его разрядность до двух и сдвинем первую цифру в разряд десятков.
6. Добавим к полученному второму числу запомненную ранее цифру из первого числа.
7. Будем повторять действия п. 3-6 пока исходное число не уменьшится до нуля, т. е. пока не избавимся от всех его разрядов.

```
n1 = int(input("Введите целое число: "))
n2 = 0

while n1 > 0:
    digit = n1 % 10  # находим остаток - последнюю цифру числа
    n1 = n1 // 10    # делим нацело - убираем из числа последнюю цифру
    n2 = n2 * 10     # увеличиваем разрядность второго числа
    n2 = n2 + digit  # добавляем очередную цифру

print('"Обратное" ему число:', n2)
```

Пример выполнения:

```
Введите целое число: 32809
"Обратное" ему число: 90823
```

Приведенный алгоритм решения является математическим, он универсален для большинства языков. Однако средства Python позволяют решить подобную задачу более практично. Так у списков есть метод reverse(), позволяющий изменять порядок элементов на обратный. Мы можем получить из исходной строки список символов, выполнить его реверс, после чего с помощью строкового метода join() опять собрать в единую строку.

```
n1 = input("Введите целое число: ")
n_list = list(n1)
n_list.reverse()
n2 = "".join(n_list)
print('"Обратное" ему число:', n2)
```

Также можно воспользоваться взятием среза из исходной строки с первого до последнего символа с обратным шагом:

```
n1 = input("Введите целое число: ")
n2 = n1[::-1]
print('"Обратное" ему число:', n2)
```

Возведение чисел в степень

Вывести степени натуральных чисел, не превосходящие данного числа *n*.

Пользователь задает показатель степени и число *n*.

Алгоритм решения задачи:

1. Ввести показатель степени и присвоить его переменной *p*.
2. Ввести число *n*.
3. Пока натуральное число *i* возведенное в степень *p* меньше или равно *n*, то
   a. выводить на экран *i* в степени *p* и
   b. увеличивать *i* на 1 (т. е. переходить к следующему натуральному числу).

```
p = int(input("Показатель степени: "))
n = int(input("Предел: "))

i = 1
while i ** p <= n:
    print(i ** p, end=' ')
    i += 1

print("\nПоследнее число, возведенное в степень:", i - 1)
```

Примеры выполнения кода:

```
Показатель степени: 2
Максимальный предел степени: 100
1 4 9 16 25 36 49 64 81 100
Последнее число, возведенное в степень: 10
```

Факториал числа

Напишите программу вычисления факториала натурального числа. Число вводят с клавиатуры. Факториалом числа называют произведение всех натуральных чисел до него включительно. Например, факториал числа 5 равен произведению 1 * 2 * 3 * 4 * 5 = 120.

Формула нахождения факториала:

$$n! = 1 * 2 * \ldots * n,$$

где *n* – это число, а *n!* – факториал этого числа.

Формулу можно представить в таком виде:

$$n! = 1 * \ldots * (n-2) * (n-1) * n,$$

т. е. каждый предыдущий множитель меньше на единицу, чем последующий.

С помощью цикла можно найти факториал как по первой, так и второй формуле. Для вычисления факториала с помощью рекурсии используется вторая формула.

```
n = int(input())

factorial = 1
while n > 1:
    factorial *= n
    n -= 1

print(factorial)
```

Вычисление факториала с помощью цикла for:

```
n = int(input())

factorial = 1

for i in range(2, n+1):
    factorial *= i

print(factorial)
```

Следует отметить, модуль math языка программирования Python содержит функцию factorial(), принимающую в качестве аргумента неотрицательное целое число и возвращающую факториал этого числа:

```
>>> import math
>>> math.factorial(4)
24
```

N-ный элемент ряда Фибоначчи

Вычислите n- ный элемент ряда Фибоначчи. Номер элемента (n) вводится с клавиатуры. Числа Фибоначчи – это ряд чисел, в котором каждое следующее число равно сумме двух предыдущих: 1, 1, 2, 3, 5, 8, 13, ...

Вычисление n-го числа ряда Фибоначчи с помощью цикла while:

1. Присвоить переменным `fib1` и `fib2` значения двух первых элементов ряда, то есть присвоить переменным единицы.
2. Запросить у пользователя номер элемента, значение которого он хочет получить. Присвоить номер переменной `n`.
3. Выполнять следующие действия `n - 2` раз, так как первые два элемента уже учтены:
   a. Сложить `fib1` и `fib2`, присвоив результат переменной для временного хранения данных, например, `fib_sum`.
   b. Переменной `fib1` присвоить значение `fib2`.
   c. Переменной `fib2` присвоить значение `fib_sum`.
4. Вывести на экран значение `fib2`.

```
fib1 = 1
fib2 = 1

n = input("Номер элемента ряда Фибоначчи: ")
n = int(n)

i = 0
while i < n - 2:
    fib_sum = fib1 + fib2
    fib1 = fib2
    fib2 = fib_sum
    i = i + 1

print(fib2)
```

Компактный вариант кода:

```
fib1 = fib2 = 1

n = int(input("Номер элемента ряда Фибоначчи: ")) - 2

while n > 0:
    fib1, fib2 = fib2, fib1 + fib2
    n -= 1

print(fib2)
```

Вывод ряда Фибоначчи

Выведите на экран ряд Фибоначчи до *n*-ного элемента включительно. Номер последнего элемента (*n*) вводится с клавиатуры. Числа Фибоначчи – это ряд чисел, в котором каждое следующее число равно сумме двух предыдущих: 1, 1, 2, 3, 5, 8, 13, ...

В данном случае выводится не только значение искомого элемента ряда Фибоначчи как в предыдущей задаче, но и все числа до него включительно. Для этого вывод значения fib2 помещен в цикл.

```
fib1 = fib2 = 1

n = int(input())

if n < 2:
    quit()

print(fib1, end=' ')
print(fib2, end=' ')

for i in range(2, n):
    fib1, fib2 = fib2, fib1 + fib2
    print(fib2, end=' ')

print()
```

Пример выполнения:

```
10
1 1 2 3 5 8 13 21 34 55
```

Таблица символов

Выведите на экран в табличном виде символы с номерами от 32 до 126- ти включительно.

В Python v.3 для строк используется кодировка Unicode. Следует помнить, в Python, в отличие от многих других языков программирования, нет такого типа данных как одиночный символ. В Python любой символ - это строка, длина которой равна единице.

Первые 127 символов по таблице Unicode такие же как и в таблице символов ASCII. Выведем их, начиная с пробела, кодовый номер которого 32.

Чтобы привести вывод к табличной форме будем переходить на новую строку после каждого десятого выведенного на экран символа. Для этого в коде ниже используется оператор if.

Функция chr() возвращает символ из таблицы Unicode, соответствующий переданному коду-числу.

```
for i in range(32, 127):
    print(chr(i), end=' ')
    if (i - 1) % 10 == 0:
        print()

print()
```

Пример выполнения:

```
  ! " # $ % & ' ( )
* + , - . / 0 1 2 3
4 5 6 7 8 9 : ; < =
> ? @ A B C D E F G
H I J K L M N O P Q
R S T U V W X Y Z [
\ ] ^ _ ` a b c d e
f g h i j k l m n o
p q r s t u v w x y
z { | } ~
```

Таблица умножения

Выведите на экран таблицу умножения.

Особенностью данной задачи является необходимость использования вложенного цикла. Во внешнем цикле меняется один множитель, во внутреннем – второй. Один оборот внешнего цикла формирует целую строку. Один оборот внутреннего – вычисляет один элемент строки.

Решение задачи с помощью циклов while:

```
i = 1
while i < 10:
    j = 1
    while j < 10:
        print("%4d" % (i * j), end="")
        j += 1
    print()
    i += 1
```

Циклом for:

```
for i in range(1, 10):
    for j in range(1, 10):
        print("%4d" % (i*j), end='')
    print()
```

Выполнение:

```
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81
```

Циклы (часть 2)

Программа "Угадай число"

Компьютер "загадывает" случайное число от 1 до 100. Человек должен его угадать, сделав как можно меньшее число попыток. Перед каждой попыткой сообщается ее номер. Программа завершается, когда пользователь угадывает число.

```
from random import randint

secret_num = randint(1, 100)

user_num = -1

try_count = 1

while secret_num != user_num:
    print("%d попытка: " % try_count, end='')
    user_num = int(input())
    if user_num < secret_num:
        print("Слишком мало")
    elif user_num > secret_num:
        print("Слишком много")
    else:
        print("Вы угадали число!")
    try_count += 1
```

Пример выполнения:

```
1 попытка: 50
Слишком мало
2 попытка: 75
Слишком мало
3 попытка: 88
Слишком много
4 попытка: 81
Слишком много
5 попытка: 78
Слишком много
6 попытка: 77
Вы угадали число!
```

Подсказка: чтобы угадать число за меньшее число попыток, диапазон в который оно входит надо всегда делить на два.

Прямоугольник из символов

Вывести на экран "прямоугольник", образованный из двух видов символов. Контур прямоугольника должен состоять из одного символа, а "заливка" - из другого.

Пусть прямоугольник будет состоять из 20 символов по горизонтали и 5 по вертикали.

Во внешнем цикле перебираем строки:

1. Если строка первая или последняя, то всю ее выводим "контурными" символами.
2. Иначе выводим один "контурный" символ, затем 18 символов "заливки", после чего снова выводим "контурный".
3. Переходим на новую строку.

```
for i in range(5):
    if i == 0 or i == 4:
        for j in range(20):
            print('%', end='')
    else:
        print('%', end='')
        for j in range(1, 19):
            print('$', end='')
        print('%', end='')
    print()
```

Результат:

```
%%%%%%%%%%%%%%%%%%%%
%$$$$$$$$$$$$$$$$$$%
%$$$$$$$$$$$$$$$$$$%
%$$$$$$$$$$$$$$$$$$%
%%%%%%%%%%%%%%%%%%%%
```

Таблица значений функции

Дана функция $y = -3x^2 - 4x + 20$. Вывести таблицу значений данной функции на отрезке и с шагом, которые вводит пользователь.

Алгоритм решения задачи:

1. Запросить у пользователя точки начала и конца отрезка, а также шаг.
2. Если значение точки начала отрезка больше значения точки конца, то поменять значения.
3. Пока значение первой точки не достигнет второй
   a. вычислить значение функции,
   b. вывести на экран текущие значения *x* и *y*,
   c. увеличить значение первой точки на шаг.

```
x1 = float(input("Точка начала отрезка: "))
x2 = float(input("Точка конца отрезка: "))
step = float(input("Шаг: "))

if x1 > x2:
    x1, x2 = x2, x1

print("Функция: y = -3x**2 - 4x + 20")
print("   x        y")
while x1 <= x2:
    y = -3 * x1 ** 2 - 4 * x1 + 20
    print('%5.2f | %7.2f' % (x1, y))
    x1 += step
```

Пример выполнения:

```
Точка начала отрезка: -1
Точка конца отрезка: 2.9
Шаг: 0.7
Функция: y = -3x**2 - 4x + 20
   x        y
-1.00 |   21.00
-0.30 |   20.93
 0.40 |   17.92
 1.10 |   11.97
 1.80 |    3.08
 2.50 |   -8.75
```

Произведение a(a+1)(a+2)...(a+n-1)

Дано действительное число a и натуральное число n. Написать программу, которая вычисляет произведение a(a+1) (a+2) . . . (a+n- 1) .

```
a = float(input("a = "))
n = int(input("n = "))

product = a

for i in range(1, n):
    product *= (a + i)

print("Результат:", product)
```

Пример выполнения:

```
a = 0.8
n = 4
Результат: 15.3216
```

Обратим внимание, в функцию range() передается *n*, а не *n - 1*, так как последнее число не входит в диапазон.

Ряд 1, -0.5, 0.25, -0.125, ...

Дан ряд чисел 1, -0.5, 0.25, -0.125, ... Требуется найти сумму стольких его элементов, сколько указал пользователь. Например, если n = 3, то надо сложить 1, -0.5, 0.25, что в результате даст 0.75.

При решении таких задач сначала необходимо определить, в чем заключается особенность предложенного ряда. В данном случае видим, что каждый последующий элемент в 2 раза меньше предыдущего по модулю и взят с противоположным знаком. Далее следует найти арифметическое действие, которое из предыдущего элемента дает последующий. Здесь, например, надо предыдущий элемент делить на -2.

Алгоритм решения задачи:

1. Присвоим переменной *a* первое число ряда.
2. Создадим счетчик суммированных элементов ряда *i*.
3. Создадим переменную для накопления суммы элементов ряда.
4. Пока счетчик не отсчитает заданного количества элементов ряда (пока $i < n$) будем выполнять нижеследующие действия.
    a. К сумме добавим значение текущего элемента.
    b. Изменим значение текущего элемента на то, которое должно быть следующим. В данном случае разделим на -2.
    c. Увеличим значение счетчика *i* на 1.

```
n = input("Количество элементов ряда: ")
n = int(n)
a = 1
i = 0
summa = 0
while i < n:
    summa += a
    a = a / -2
    i += 1

print(summa)
```

Пример выполнения:

```
Количество элементов ряда: 5
0.6875
```

Последовательность 1/2 - 2/4 + 3/8 - 4/16 + ...

Дана последовательность 1/ 2 - 2/ 4 + 3/ 8 - 4/ 16 + . . . - … . Найти сумму и количество элементов последовательности, которые по модулю больше 0. 001.

В данной последовательности каждый последующий элемент отличается от предыдущего:

- знаком,
- числитель увеличен на 1,
- знаменатель увеличен в 2 раза.

Значит каждый следующий элемент ряда вычисляется из текущего по формуле *-(a+1)/(b*2)*, где *a* и *b* - числитель и знаменатель дроби числа ряда.

```
a = 1  # числитель
b = 2  # знаменатель
sign = 1  # будет умножаться на 1 или -1, чтобы поменять знак числа
n = 0  # количество чисел ряда
summa = 0  # сумма n-элементов ряда чисел
while a / b > 0.001:
    summa += sign * a / b
    n += 1
    a = a + 1
    b = b * 2
    sign = -sign

print(summa)
print(n)
```

Пример выполнения:

```
0.2227783203125
13
```

Из десятичной системы счисления в двоичную

Напишите цикл, который переводит число из десятичной системы счисления в двоичную.

Обычно десятичное число преобразуют к двоичному представлению путем нахождения остатков от деления на 2. При этом полученное на предыдущем шаге частное выступает в качестве делимого на следующем шаге. Деление заканчивается, когда делимое обращается в ноль. Остатки собираются в двоичное число начиная с конца, то есть последний остаток будет первой цифрой двоичного числа. Например, надо перевести число 8 в двоичную систему:

8 / 2 = 4, остаток 0
4 / 2 = 2, остаток 0
2 / 2 = 1, остаток 0
1 / 2 = 0, остаток 1
0 - конец деления
Сборка: 1000_2

При реализации данного алгоритма с помощью языка программирования надо организовать хранение остатков. Сделать это можно в переменной строкового типа или в списке. В случае строки каждый новый остаток следует добавлять в начало.

```
n = int(input())

b = ''

while n > 0:
    b = str(n % 2) + b
    n = n // 2

print(b)
```

Пример выполнения:

```
8
1000
```

Заметим, что в языке Python есть встроенная функция bin(), которая переводит десятичное число в двоичную систему счисления:

```
>>> bin(5)
'0b101'
>>> bin(10)
'0b1010'
```

Из десятичной системы в любую с основанием до 9-ти

Вводится десятичное число и желаемая система счисления с основанием до 9.

Перевести десятичное число в заданную систему счисления.

```
# Ввод числа и преобразование к целому
num = int(input())
# Ввод системы счисления
base = int(input("Base (2-9): "))

# Проверка корректности ввода системы счисления.
# Если основание не принадлежит указанному диапазону,
# то происходит выход из программы
if not(2 <= base <= 9):
    quit()

# Переменная для хранения строкового представления
# числа в заданной системе счисления
newNum = ''

# Пока исходное число больше 0,
while num > 0:
    # находится остаток от его деления на основание,
    # остаток преобразовывается к строковому типу и
    # добавляется в начало строкового представления нового числа
    newNum = str(num % base) + newNum
    # Само десятичное число делится нацело
    # на основание заданной системы счисления
    num //= base

# Вывод строкового представления числа
# в системе счисления с основанием base
print(newNum)
```

Примеры выполнения:

```
255
Base (2-9): 2
11111111
```

```
34
Base (2-9): 7
46
```

Является ли число простым

Проверьте, является ли введенное с клавиатуры целое число простым. Используйте метод перебора делителей.

Простые числа - это натуральные числа больше единицы, которые делятся нацело только на единицу и на себя. Например, число 3 простое, так как нацело делится только на 1 и 3. Число 4 сложное, так как нацело делится не только на 1 и 4, но также на число 2.

Перебор делителей – это алгоритм, применяемый для определения, является ли натуральное число простым, или оно является сложным, то есть составным. Алгоритм перебора делителей заключается в последовательном делении заданного натурального числа на все целые числа, начиная с двойки и заканчивая значением меньшим или равным квадратному корню из тестируемого числа.

Если хотя бы один делитель делит исследуемое число без остатка, то это число является составным. Если ни одного такого делителя не находится, то число признается простым.

Число 2 является простым, но при этом оно делится на 2. Цикл while определил бы двойку как составное число, что неправильно. Поэтому для числа 2 необходима отдельная проверка.

Программа дойдет до последней строчки кода, только если не был найден ни один делитель. Если это так, значит число простое.

```
import math

n = int(input())

if n < 2:
    print("Число должно быть больше 1")
    quit()
elif n == 2:
    print("Это простое число")
    quit()

i = 2 # первый делитель

limit = int(math.sqrt(n))

while i <= limit:
    if n % i == 0:
        print("Это сложное число")
        quit() # выход из программы
    i += 1 # переход к следующему делителю

print("Это простое число")
```

Пример выполнения:

```
37
Это простое число
```

Алгоритм Евклида

Напишите цикл, который вычисляет наибольший общий делитель двух чисел, используя любую из вариаций алгоритма Евклида.

Наибольший общий делитель (НОД) – это число, которое делит без остатка два числа и делится само без остатка на любой другой делитель данных двух чисел. Проще говоря, это самое большое число, на которое можно без остатка разделить два числа, для которых ищется НОД.

Алгоритм нахождения НОД делением:

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

Пример:

Найти НОД для 30 и 18.

30 / 18 = 1 (остаток 12)
18 / 12 = 1 (остаток 6)
12 / 6 = 2 (остаток 0)

Конец: НОД – это делитель 6.

НОД (30, 18) = 6

```
a = 50
b = 130

while a != 0 and b != 0:
    if a > b:
        a = a % b
    else:
        b = b % a

print(a + b)
```

В цикле в переменную *a* или *b* записывается остаток от деления. Цикл завершается, когда хотя бы одна из переменных равна нулю. Это значит, что другая содержит НОД. Однако какая именно, мы не знаем. Поэтому для НОД находим сумму этих переменных. Поскольку в одной из переменных ноль, он не оказывает влияние на результат.

Алгоритм нахождения НОД вычитанием:

1. Из большего числа вычитаем меньшее.
2. Если получается 0, то значит, что числа равны друг другу и являются НОД (следует выйти из цикла).
3. Если результат вычитания не равен 0, то большее число заменяем на результат вычитания.
4. Переходим к пункту 1.

Пример:

Найти НОД для 30 и 18.

30 - 18 = 12
18 - 12 = 6
12 - 6 = 6
6 - 6 = 0

Конец: НОД – это уменьшаемое или вычитаемое.

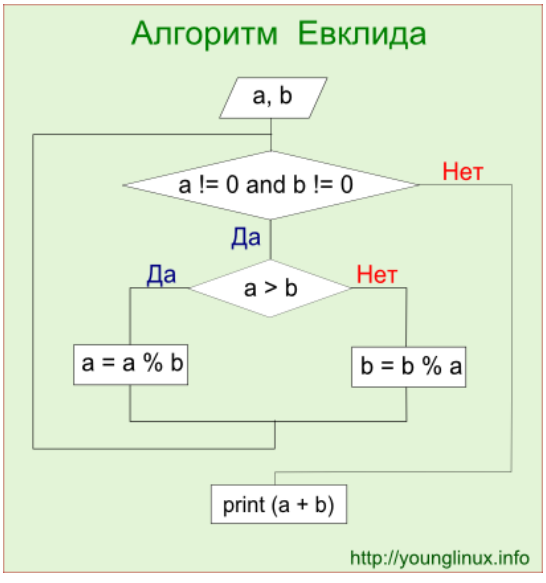
НОД (30, 18) = 6

```
a = 50
b = 130

while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a

print(a)
```

Блок-схема алгоритма Евклида:



Списки (часть 1)

Отрицательные – на -1, положительные – на 1

Дан список целых чисел. Заменить отрицательные числа на число -1, положительные - на число 1, ноль оставить без изменений.

```
num_list = [1, -5, 3, 8, 0, 9, -6, 3, -1, 5]
print(num_list)

for i in range(len(num_list)):
    if num_list[i] > 0:
        num_list[i] = 1
    elif num_list[i] < 0:
        num_list[i] = -1

print(num_list)
```

Результат:

```
[1, -5, 3, 8, 0, 9, -6, 3, -1, 5]
[1, -1, 1, 1, 0, 1, -1, 1, -1, 1]
```

Решение с использованием функции enumerate():

```
num_list = [1, -5, 3, 8, 0, 9, -6, 3, -1, 5]
print(num_list)

for index, value in enumerate(num_list):
    if value > 0:
        num_list[index] = 1
    elif value < 0:
        num_list[index] = -1

print(num_list)
```

Количество положительных, отрицательных и равных нулю

Дан список целых чисел. Посчитать, сколько в списке содержится положительных, отрицательных и равных нулю элементов.

```
num_list = [1, -5, 3, 8, 0, 9, -6, 3, -1, 5]
print(num_list)

neg = pos = zero = 0

for i in num_list:
    if i > 0:
        pos += 1
    elif i < 0:
        neg += 1
    else:
        zero += 1

print("Положительных: ", pos)
print("Отрицательных: ", neg)
print("Равных нулю: ", zero)
```

Результат выполнения:

```
[1, -5, 3, 8, 0, 9, -6, 3, -1, 5]
Положительных:  6
Отрицательных:  3
Равных нулю:  1
```

Удалить элементы, входящие в диапазон от 35 до 65

Заполнить список случайными числами от 1 до 100. Из списка удалить элементы, значения которых больше 35 и меньше 65. При этом удаляемые числа сохранить в другом списке.

В Python с помощью инструкции del можно удалить элемент списка, указав сам список и индекс удаляемого элемента.

Алгоритм решения задачи выглядит простым. Достаточно перебрать элементы списка и удалить те, которые удовлетворяют условию. Однако при удалении элемента на его место становится следующий, но поскольку мы переходим к следующему элементу, то пропускаем проверку того, что стал на место удаленного. Цикл for использовать нельзя, т. к. меняется количество элементов списка.

Можно использовать цикл while, измеряя на каждой его итерации длину списка, индекс же увеличивать только в том случае, если удаления элемента не произошло.

```
import random

a = []
for i in range(20):
    n = round(random.random() * 100)
    a.append(n)
print("A =", a)

b = []
i = 0
while i < len(a):
    if 35 < a[i] < 65:
        b.append(a[i])
        del a[i]
    else:
        i += 1

print("A =", a)
print("B =", b)
```

Пример выполнения программы:

```
A = [68, 88, 64, 39, 13, 99, 6, 35, 50, 76, 47, 82, 76, 87, 44, 86, 75, 46, 8, 32]
A = [68, 88, 13, 99, 6, 35, 76, 82, 76, 87, 86, 75, 8, 32]
B = [64, 39, 50, 47, 44, 46]
```

Положительные – в один список, отрицательные – в другой

Программа генерирует случайные числа в диапазоне от -5 до 5 включительно.

Требуется положительные числа помещать в один список, а отрицательные – в другой.

Числа, равные нулю, игнорировать. Вывести на экран получившиеся списки.

```
from random import randint

pos = []
neg = []
for i in range(20):
    n = randint(-5, 5)
    if n > 0:
        pos.append(n)
    elif n < 0:
        neg.append(n)

print(pos)
print(neg)
```

Пример выполнения:

```
[4, 5, 1, 4, 2, 4, 2, 4, 4, 5, 2]
[-1, -3, -1, -3, -3, -3]
```

Вариант решения с использованием генератора списка и функции filter():

```
origin_list = [randint(-5, 5) for i in range(20)]
pos_list = list(filter(lambda x: x > 0, origin_list))
neg_list = list(filter(lambda x: x < 0, origin_list))

print(pos_list)
print(neg_list)
```

Пороговое значение

Заполнить список случайными числами и вывести на экран. Заменить все элементы меньше порогового значения, которое вводит пользователь, этим значением. Вывести на экран измененный список.

```
from random import randint

num_list = [randint(1, 99) for i in range(10)]
print(num_list)

limit = int(input("Порог: "))

for index, value in enumerate(num_list):
    if value < limit:
        num_list[index] = limit

print(num_list)
```

Пример выполнения:

```
[69, 33, 51, 31, 13, 90, 30, 48, 15, 48]
Порог: 50
[69, 50, 51, 50, 50, 90, 50, 50, 50, 50]
```

Минимальный по модулю

Найти минимальный по модулю элемент списка. Вывести его индекс и значение.

Например, в массиве [10, - 3, - 5, 2, 5] минимальным по модулю элементом является число 2. Его индексный номер равен 3.

```
from random import randint
N = 10
arr = []
for i in range(N):
    arr.append(randint(-50, 49))
print(arr)

num = 0
for i in range(1, N):
    if abs(arr[i]) < abs(arr[num]):
        num = i

print("Минимальный по модулю элемент:")
print("arr[{}] = {}".format(num, arr[num]))
```

Пример выполнения:

```
[-12, 6, 3, -44, -22, 41, -18, 28, -10, -14]
Минимальный по модулю элемент:
arr[2] = 3
```

Другой вариант решения:

```
from random import randint
N = 10

arr = [randint(-50, 49) for i in range(N)]
print(arr)

arr2 = [abs(i) for i in arr]
index_min = arr2.index(min(arr2))

print("Минимальный по модулю элемент:")
print("arr[{}] = {}".format(index_min, arr[index_min]))
```

В данном случае создается второй список, где все элементы первого списка приведены к абсолютному значению. Среди них с помощью функции min() находится минимальный, а с помощью функции index() берется его индекс. Этот же индекс будет указывать на минимальный по модулю элемент первого списка.

Пример выполнения:

```
[-9, -10, -1, -45, -14, 39, 17, -49, 47, 24]
Минимальный по модулю элемент:
arr[2] = -1
```

Два наибольших значения

В списке целых чисел найти два наибольших значения. Они могут быть как равны между собой (оба являться максимальными), так и различаться.

Универсальный, не привязанный к Python, алгоритм заключается в поиске сначала первого максимума, его исключении из поиска, поиске второго максимума.

```
from random import randint
N = 10

a = [randint(0, 30) for i in range(N)]
print(a)

max1_index = 0
for i in range(1, N):
    if a[i] > a[max1_index]:
        max1_index = i

max2_index = 0
if max2_index == max1_index:
    max2_index = 1
for i in range(1, N):
    if i == max1_index:
        continue
    if a[i] > a[max2_index]:
        max2_index = i

print("Первый максимум =", a[max1_index])
print("Второй максимум =", a[max2_index])
```

Пример выполнения:

```
[18, 2, 10, 15, 8, 1, 15, 7, 0, 4]
Первый максимум = 18
Второй максимум = 15
```

Пользуясь встроенными в Python функциями и методами списка, задачу можно решить иначе:

```
from random import randint
N = 10

a = [randint(0, 30) for i in range(N)]
print(a)

b = sorted(a)
max1_value = b[-1]
max2_value = b[-2]
max1_index = a.index(max1_value)
max2_index = a.index(max2_value)

print("Первый максимум =", a[max1_index])
print("Второй максимум =", a[max2_index])
```

Больше среднего арифметического

Вывести элементы, которые больше среднего арифметического от значений всех элементов списка.

Чтобы найти среднее арифметическое, надо сложить все элементы массива и разделить сумму на их количество.

Далее следует перебрать список, сравнивая каждый элемент со средним арифметическим.

```
from random import randint

N = 10
a = []
suma = 0
for i in range(N):
    a.append(randint(0, 9))
    print(a[i], end=' ')
    suma += a[i]
print()

average = suma / N
print("Среднее арифметическое: %.2f" % average)

for i in a:
    if i > average:
        print(i, end=' ')
print()
```

Пример выполнения:

```
8 6 5 4 6 2 7 4 5 0
Среднее арифметическое: 4.70
8 6 5 6 7 5
```

Первый, третий и шестой положительные

В списке, состоящем из положительных и отрицательных целых чисел, найти первый, третий и шестой положительные элементы и вычислить их произведение.

Задача усложнена тем, что, например, третий по счету положительный элемент может быть далеко не третьим в списке, а каким угодно.

1. Ввести в программу счетчик положительных элементов *j*. Присвоить ему значение 1, то есть он указывает на первый положительный элемент.
2. Перебирать элементы списка. Если очередной элемент положительный и счетчик положительных элементов равен 1 или 3 или 6, то сохранить в соответствующей переменной значение данного элемента. Кроме того, если шестой положительный элемент найден, то нет смысла продолжать цикл дальше и можно его завершить (оператор break).

```
from random import randint
N = 15

a = [randint(-10, 10) for i in range(N)]
print(a)

j = 1
v1 = v3 = v6 = 0

for value in a:
    if value > 0:
        if j == 1:
            v1 = value
        elif j == 3:
            v3 = value
        elif j == 6:
            v6 = value
            break
        j += 1

mult = v1 * v3 * v6
print("%d * %d * %d = %d" % (v1, v3, v6, mult))
```

Пример выполнения:

```
[0, 2, 4, 8, -10, -7, 8, -9, -9, 0, 4, -3, -8, 6, 4]
2 * 8 * 6 = 96
```

Решение с использованием встроенных функций языка Python:

```
...
b = list(filter(lambda x: x > 0, a))
mult = b[0] * b[2] * b[5]
print("%d * %d * %d = %d" % (b[0], b[2], b[5], mult))
```

Пересечение списков

Найти область пересечения двух списков, то есть элементы, которые встречаются в обоих списках.

Код ниже подходит для списков, содержащих неповторяющиеся значения в самих себе. Иначе в результирующем списке могут появиться одинаковые элементы.

```
a = [5, [1, 2], 2, 'r', 4, 'ee']
b = [4, 'we', 'ee', 3, [1, 2]]
c = []
for i in a:
    for j in b:
        if i == j:
            c.append(i)
            break

print(c)
```

Берется первый элемент первого списка (внешний цикл for) и последовательно сравнивается с каждым элементом второго списка (вложенный цикл for). В случае совпадения значений элемент добавляется в третий список `c`. Команда break служит для выхода из внутреннего цикла, так как в случае совпадения дальнейший поиск при данном значении `i` бессмыслен.

Если в самом списке могут встречаться одинаковые элементы, чтобы они не повторялись в результирующем списке, следует добавить проверку наличия такого элемента в третьем списке:

```
a = [5, [1, 2], 2, 'r', 4, 'ee', 'ee']
b = [4, 'we', 'ee', 3, [1, 2]]
c = []
for i in a:
    if i in c:
        continue
    for j in b:
        if i == j:
            c.append(i)
            break

print(c)
```

Результат выполнения программы:

```
[[1, 2], 4, 'ee']
```

Другой способ – это использование множеств. Подходит только для списков, которые не содержат вложенных списков, так как функция set() выдает ошибку, если ей передать список со вложенными изменяемыми объектами.

Множества не могут содержать одинаковых элементов. Результатом операции пересечения двух множеств является множество, содержащее значения элементов, которые были и в первом и во втором операндах-множествах. Например, если первое множество было {1, 4, 9, 12}, а второе – {4, 6, 9, 14, 18}, то результатом операции пересечения будет {4, 9}.

Списки преобразуются во множества с помощью функции set.

```
a = [5, 2, 'r', 4, 'ee']
b = [4, 1, 'we', 'ee', 2, 'r']
c = list(set(a) & set(b))
print(c)
```

При этом если в списках встречались одинаковые значения, то во множествах они будут уже представлены единственным вариантом. Поэтому в данном случае списки могут содержать одинаковые значения.

После того, как списки были преобразованы во множества, над ними можно выполнить операцию пересечения.

С помощью знака & выполняется пересечение множеств. Кроме того, над множествами можно выполнять ряд других операций: | (объединение), - (разность), ^ (исключающее ИЛИ).

Списки (часть 2)

Сумма элементов между минимальным и максимальным

В одномерном массиве найти сумму элементов, находящихся между минимальным и максимальным элементами. Сами минимальный и максимальный элементы в сумму не включать.

Универсальный алгоритм:

```
from random import random

N = 10
a = [0]*N
for i in range(N):
    a[i] = int(random()*50)
    print('%3d' % a[i], end='')
print()

min_id = 0
max_id = 0
for i in range(1, N):
    if a[i] < a[min_id]:
        min_id = i
    elif a[i] > a[max_id]:
        max_id = i
print(a[min_id], a[max_id])

if min_id > max_id:
    min_id, max_id = max_id, min_id

summa = 0
for i in range(min_id+1, max_id):
    summa += a[i]
print(summa)
```

```
 17  5 15 22 20 49  9 11  9 20
5 49
57
```

С помощью функций и методов Python:

```
min_id = a.index(min(a))
max_id = a.index(max(a))

print(a[min_id], a[max_id])

if min_id > max_id:
    min_id, max_id = max_id, min_id

summa = sum(a[min_id+1:max_id])
print(summa)
```

Больше, чем у соседей

Сколько элементов списка имеют значения больше, чем у их соседних элементов?

Чтобы ответить на вопрос, надо каждый элемент списка сравнить с предстоящим и послестоящим. Если он больше них, увеличить счетчик.

Первый и последний элемент списка имеют только одного "соседа". Поэтому, если первый и последний учитывать, их следует обработать отдельно.

```
from random import randint
N = 10

a = [randint(1, 50) for i in range(N)]
print(a)

qty_big = 0

for i in range(1, N-1):
    if a[i-1] < a[i] > a[i+1]:
        qty_big += 1

if a[0] > a[1]:
    qty_big += 1
if a[N-1] > a[N-2]:
    qty_big += 1

print(qty_big)
```

```
[44, 42, 15, 23, 43, 4, 16, 44, 3, 19]
4
```

Заметим, что если какой-то элемент больше своих соседей, то следующий за ним сосед уже точно не будет больше своих соседей. Значит, его можно пропустить, выполнив меньшее число сравнений.

```
...
i = 1
if a[i-1] > a[i]:
    qty_big += 1
    i = 2
if a[N-1] > a[N-2]:
    qty_big += 1
    N -= 1

while i < N-1:
    if a[i-1] < a[i] > a[i+1]:
        qty_big += 1
        i += 2
    else:
        i += 1

print(qty_big)
```

Анализ выборки

Дан список. Напишите программу, которая считает количество значений, попавших в разные числовые диапазоны.

Бывает надо проанализировать данные-выборку на предмет распределения значений по диапазонам. Полученный таким образом результат используют для построения графиков и диаграмм частот встречаемости значений.

Пусть у нас будет список чисел от 1 до 9 включительно. Требуется выяснить, сколько значений попало в диапазон от 1 до 3, от 4 до 6, от 7 до 9.

Для этого введем три соответствующих счетчика. Далее будем извлекать каждый элемент из списка и проверять его на попадание в тот или иной диапазон. При этом счетчик элементов этого диапазона должен увеличиваться.

```
a = [3, 5, 7, 3, 8, 1, 8, 1, 7, 3, 2,
     4, 6, 8, 5, 4, 3, 3, 6, 5, 7, 8,
     9, 5, 3, 2, 3]

count_1_3 = 0
count_4_6 = 0
count_7_9 = 0

for i in a:
    if 1 <= i <= 3:
        count_1_3 += 1
    elif 4 <= i <= 6:
        count_4_6 += 1
    elif 7 <= i <= 9:
        count_7_9 += 1

print("Range 1-3:", count_1_3, "items")
print("Range 4-6:", count_4_6, "items")
print("Range 7-9:", count_7_9, "items")
```

Результат:

```
Range 1-3: 11 items
Range 4-6: 8 items
Range 7-9: 8 items
```

С использованием функции filter():

```
...
count_1_3 = len(list(filter(lambda i: 1 <= i <= 3, a)))
count_4_6 = len(list(filter(lambda i: 4 <= i <= 6, a)))
count_7_9 = len(list(filter(lambda i: 7 <= i <= 9, a)))
...
```

Проверка уникальности элементов

Напишите программу, которая проверяет, все ли элементы списка являются уникальными, т. е. каждое значение встречается только один раз.

Решить данную задачу на языке Python можно несколькими способами. Классический вариант - брать по очереди элементы списка и сравнить каждый со стоящими за ним. При первом же совпадении элементов делается вывод, что в списке есть одинаковы элементы и работа программы завершается.

Еще одним способом решения может быть использование типа данных "множества" (set). Как известно, в множествах не может быть одинаковых элементов. При преобразовании списка во множество в нем одинаковые элементы будут представлены единожды, то есть дубли удалятся. Если после этого сравнить длину исходного списка и множества, то станет ясно, есть ли в списке одинаковые элементы. Если длины совпадают, значит все элементы списка уникальны. Если нет, значит, были одинаковые элементы.

Допустим, исходный список генерируется таким кодом:

```
from random import random
N = 10
arr = [0] * N
for i in range(N):
    arr[i] = int(random() * 50)
print(arr)
```

Пример решения классическим способом:

```
for i in range(N-1):
    for j in range(i+1, N):
        if arr[i] == arr[j]:
            print("Есть одинаковые")
            quit()
print("Все элементы уникальны")
```

```
[2, 4, 1, 2, 45, 38, 26, 11, 49, 25]
Есть одинаковые
```

Здесь `j` принимает значения от следующего элемента за тем, для которого ищется совпадение, до последнего в списке. Сравнивать элемент с индексом `i` с элементами, стоящими впереди него, не надо, т. к. эти сравнения уже выполнялись на предыдущих итерациях внешнего цикла.

Решение задачи с помощью множества:

```
setarr = set(arr)
if len(arr) == len(setarr):
    print("Все элементы уникальны")
else:
    print("Есть одинаковые")
```

Вывести только уникальные

В одномерном массиве найти элементы, которые в нем встречаются только один раз, и вывести их на экран.

Проще всего данную задачу решить с помощью преобразования исходного списка во множество. Во множестве будут исключены повторы. Далее, перебирая множество, с помощью метода count() списка будем выяснять количество каждого значения в исходном списке. Если это количество равно 1, значит элемент уникальный.

```
from random import randint
N = 10

a = [randint(10, 20) for i in range(N)]
print(a)

unic_set = set(a)

for i in unic_set:
    if a.count(i) == 1:
        print(i)
```

Пример выполнения:

```
[10, 14, 11, 10, 10, 18, 18, 17, 18, 11]
14
17
```

Решение без использования функций предполагает, что мы будем сравнивать каждый элемент массива со всеми остальными. Если совпадения значений не будет найдено, значит элемент уникальный и выводится на экран.

```
from random import random
N = 20
a = [0] * N
for i in range(N):
    a[i] = int(random()*15)
    print(a[i],end=' ')
print()
for i in range(N):
    f = True
    for j in range(N):
        if a[i] == a[j] and i != j:
            f = False
            break
    if f == True:
        print(a[i],end=' ')
print()
```

```
1 3 2 8 12 6 10 6 13 11 6 4 10 9 2 8 7 13 1 9
3 12 11 4 7
```

Наиболее встречаемое значение

Дан список чисел. Определить, какое число в списке встречается чаще всего.

В программах ниже ищется только первое самое встречаемое число. Если есть другое число, которое встречается с такой же частотой как первое, то оно не определяется.

Будем записывать в переменную *num* найденный самый встречаемый элемент, а в переменную *max_frq* - количество раз, которое он встречается.

Будем по-очереди (в цикле) брать элементы массива с первого до предпоследнего и сравнивать его с элементами, стоящими после него. С элементами, стоящими впереди, сравнивать не надо, так как если там есть равный ему, то текущий элемент уже был учтен, и количество раз, какое он встречается в массиве, уже находилось. Последний элемент не с чем сравнивать, поэтому цикл идет до предпоследнего элемента.

В теле цикла переменной *frq* присваивается значение 1, т.е. вначале предполагается, что текущий элемент встречается 1 раз. Далее перебираются элементы, стоящие после текущего. Если значения совпадают, то переменная *frq* увеличивается на 1. После того, как посчитано количество раз, какое встречается элемент, переменная *frq* сравнивается с *max_frq*. Если *frq* больше, то перезаписываются значения *max_frq* и *num*.

```
from random import random
N = 15
arr = [0] * N
for i in range(N):
    arr[i] = int(random() * 20)
print(arr)

num = arr[0]
max_frq = 1
for i in range(N-1):
    frq = 1
    for k in range(i+1,N):
        if arr[i] == arr[k]:
            frq += 1
    if frq > max_frq:
        max_frq = frq
        num = arr[i]

if max_frq > 1:
    print(max_frq, 'раз(а) встречается число', num)
else:
    print('Все элементы уникальны')
```

```
[7, 18, 16, 18, 1, 11, 2, 16, 2, 4, 7, 7, 10, 17, 2]
3 раз(а) встречается число 7
```

Также задачу можно решить, получив из списка множество. Его следует перебрать в цикле и с помощью метода count() списка посчитать число вхождений в него каждого элемента.

Двоичный поиск

Напишите программу, которая выполняет двоичный (бинарный) поиск элемента в отсортированном списке.

Двоичный, или бинарный, поиск значения в списке или массиве используется только для упорядоченных последовательностей, то есть отсортированных по возрастанию или убыванию. Заключается в определении, содержит ли массив искомое значение и где

1. Находится средний элемент последовательности. Для этого первый и последний индексы связываются с переменными, а индекс среднего элемента вычисляется.
2. Значение среднего элемента сравнивается с искомым значение. В зависимости от того, больше оно или меньше значения среднего элемента, дальнейший поиск будет происходить только в левой или только в правой половинах массива. Если значение среднего элемента оказывается равным искомому, поиск завершается.
3. Иначе одна из границ исследуемой последовательности сдвигается. Если искомое значение больше значения среднего элемента, то нижняя граница сдвигается за средний элемент на один элемент справа. Если искомое значение меньше значения среднего элемента, то верхняя граница сдвигается на элемент перед средним.
4. Снова находится средний элемент теперь уже в выбранной половине. Описанный выше алгоритм повторяется для данного среза.

```
from random import randint

a = []
for i in range(15):
    a.append(randint(1, 50))
a.sort()
print(a)

value = int(input())

mid = len(a) // 2
low = 0
high = len(a) - 1

while a[mid] != value and low <= high:
    if value > a[mid]:
        low = mid + 1
    else:
        high = mid - 1
    mid = (low + high) // 2

if low > high:
    print("No value")
else:
    print("ID =", mid)
```

Сортировка выбором

Выполните сортировку списка целых чисел. Используйте метод сортировки выбором.

Алгоритм сортировки выбором заключается в поиске на необработанном срезе массива или списка минимального значения и в дальнейшем обмене этого значения с первым элементом необработанного среза. На следующем шаге необработанный срез уменьшается на один элемент.

1. Найти наименьшее значение в списке.
2. Записать его в начало списка, а первый элемент - на место, где раньше стоял наименьший.
3. Снова найти наименьший элемент в списке. При этом в поиске не участвует первый элемент.
4. Второй минимум поместить на второе место списка. Второй элемент при этом перемещается на освободившееся место.
5. Продолжать выполнять поиск и обмен, пока не будет достигнут конец списка.

Код программы см. на следующей странице!

Пример выполнения:

```
[77, 46, 11, 89, 48, 14, 67, 73, 22, 26]
[11, 14, 22, 26, 46, 48, 67, 73, 77, 89]
```

```
# Заполняем список из 10 элементов
# случайными числами от 1 до 99 и
# выводим неотсортированный список на экран.
from random import randint
N = 10
arr = []
for i in range(N):
    arr.append(randint(1, 99))
print(arr)


# В цикле переменная i хранит индекс ячейки,
# в которую записывается минимальный элемент.
# Сначала это будет первая ячейка.
i = 0

# N - 1, так как последний элемент
# обменивать уже не надо.
while i < N - 1:

    # ПОИСК МИНИМУМА
    # Сначала надо найти минимальное значение
    # на срезе от i до конца списка.
    # Переменная m будет хранить индекс ячейки
    # с минимальным значением.
    # Сначала предполагаем, что
    # в ячейке i содержится минимальный элемент.
    m = i
    # Поиск начинаем с ячейки следующей за i.
    j = i + 1
    # Пока не дойдем конца списка,
    while j < N:
        # будем сравнивать значение ячейки j,
        # со значением ячейки m.
        if arr[j] < arr[m]:
            # Если в j значение меньше, чем в m,
            # сохраним в m номер найденного
            # на данный момент минимума.
            m = j
        # Перейдем к следующей ячейке.
        j += 1

    # ОБМЕН ЗНАЧЕНИЙ
    # В ячейку i записывается найденный минимум,
    # а значение из ячейки i переносится
    # на старое место минимума.
    arr[i], arr[m] = arr[m], arr[i]

    # ПЕРЕХОД К СЛЕДУЮЩЕЙ НЕОБРАБОТАННОЙ ЯЧЕЙКЕ
    i += 1


# Вывод отсортированного списка
print(arr)
```

Сортировка пузырьком

Выполните сортировку списка целых чисел. Используйте метод сортировки пузырьком.

Сортировка пузырьком - это метод сортировки массивов и списков путем последовательного сравнения и обмена соседних элементов, если предшествующий оказывается больше последующего. В процессе выполнения данного алгоритма элементы с большими значениями оказываются в конце списка, а элементы с меньшими значениями постепенно перемещаются по направлению к началу списка. Образно говоря, тяжелые элементы падают на дно, а легкие медленно всплывают подобно пузырькам воздуха.

В сортировке методом пузырька количество итераций внешнего цикла определяется длинной списка минус единица, так как когда второй элемент становится на свое место, то первый уже однозначно минимальный и находится на своем месте.

Количество итераций внутреннего цикла зависит от номера итерации внешнего цикла, так как конец списка уже отсортирован, и выполнять проход по этим элементам смысла нет.

Пусть имеется список [6, 12, 4, 3, 8].

За первую итерацию внешнего цикла число 12 переместится в конец. Для этого потребуется 4 сравнения во внутреннем цикле:

6 > 12? Нет
12 > 4? Да. Меняем местами
12 > 3? Да. Меняем местами
12 > 8? Да. Меняем местами
Результат: [6, 4, 3, 8, 12]

За вторую итерацию внешнего цикла число 8 переместиться на предпоследнее место. Для этого потребуется 3 сравнения:

6 > 4? Да. Меняем местами
6 > 3? Да. Меняем местами
6 > 8? Нет
Результат: [4, 3, 6, 8, 12]

На третьей итерации внешнего цикла исключаются два последних элемента. Количество итераций внутреннего цикла равно двум:

4 > 3? Да. Меняем местами
4 > 6? Нет
Результат: [3, 4, 6, 8, 12]

На четвертой итерации внешнего цикла осталось сравнить только первые два элемента, поэтому количество итераций внутреннего равно единице:

3 > 4? Нет

Результат: [3, 4, 6, 8, 12]

Реализация сортировки пузырьком с помощью циклов for:

```
from random import randint

N = 10
a = []
for i in range(N):
    a.append(randint(1, 99))
print(a)


for i in range(N-1):
    for j in range(N-i-1):
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]

print(a)
```

Пример выполнения кода:

```
[63, 80, 62, 69, 71, 37, 12, 90, 19, 67]
[12, 19, 37, 62, 63, 67, 69, 71, 80, 90]
```

С помощью циклов while:

```
from random import randint

N = 10
a = []
for i in range(N):
    a.append(randint(1, 99))
print(a)

i = 0
while i < N - 1:
    j = 0
    while j < N - 1 - i:
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
        j += 1
    i += 1

print(a)
```

Решето Эратосфена

Найдите все простые числа до заданного натурального числа N. Используйте алгоритм "решето Эратосфена".

Решето Эратосфена – это алгоритм нахождения простых чисел до заданного натурального числа путем постепенного отсеивания составных чисел. Образно говоря, через решето Эратосфена в процессе его тряски проскакивают составные числа, а простые остаются в решете.

Чтобы понять данный алгоритм, вспомним, что числа являются простыми, если делятся только на единицу и самих себя. Первое простое число - это 2, второе простое число - это 3. Теперь начнем рассуждать:

1. Все четные числа, кроме двойки, - составные, т. е. не являются простыми, так как делятся не только на себя и единицу, а также еще на 2.
2. Все числа кратные трем, кроме самой тройки, - составные, так как делятся не только на самих себя и единицу, а также еще на 3.
3. Число 4 уже выбыло из игры, так как делится на 2.
4. Число 5 простое, так как его не делит ни один простой делитель, стоящий до него.
5. Если число не делится ни на одно простое число, стоящее до него, значит оно не будет делиться ни на одно сложное число, стоящее до него.

Последний пункт вытекает из того, что сложные числа всегда можно представить как произведение простых. Поэтому если одно сложное число делится на другое сложное, то первое должно делиться на делители второго. Например, 12 делится на 6, делителями которого являются 2 и 3. Число 12 делится и на 2, и на 3.

Алгоритм Эратосфена как раз заключается в последовательной проверке делимости чисел на предстоящие простые числа. Сначала берется первое простое и из ряда натуральных чисел высеиваются все кратные ему. Затем берется следующее простое и отсеиваются все кратные ему и так далее.

При реализации алгоритма Эратосфена на языке программирования есть некоторая сложность. Допустим, мы помещаем натуральные числа до заданного числа n в массив. Далее в процессе выполнения алгоритма будем заменять обнаруженные сложные числа нулями. После выполнения алгоритма те ячейки массива, которые не содержат нули, содержат простые числа, которые выводятся на экран.

Однако индексация массива начинается с нуля, а простые числа начинаются с двойки. Эта проблема решаема, но добавляет сложности в код. Поскольку алгоритм Эратосфена не такой уж простой, легче пренебречь началом и взять массив от 0 до *n*. Здесь важнее индексы, чем значения элементов. Значениями может быть True, обозначающее простое число, и False, обозначающее сложное число.

В данном примере реализации алгоритма Эратосфена список заполняется числами от 0 до *n* включительно так, что индексы элементов совпадают с их значениями. Далее все непростые числа заменяются нулями:

```
n = int(input())

# список заполняется значениями от 0 до n
a = []
for i in range(n + 1):
    a.append(i)

# Вторым элементом является единица,
# которую не считают простым числом
# забиваем ее нулем.
a[1] = 0

# начинаем с 3-го элемента
i = 2
while i <= n:
    # Если значение ячейки до этого не было обнулено,
    # в этой ячейке содержится простое число.
    if a[i] != 0:
        # первое кратное ему будет в два раза больше
        j = i + i
        while j <= n:
            # это число составное, поэтому заменяем его нулем
            a[j] = 0
            # переходим к следующему числу, которое кратно i (оно на i больше)
            j = j + i
    i += 1

# Превращая список во множество,
# избавляемся от всех нулей кроме одного.
a = set(a)
# удаляем ноль
a.remove(0)
print(a)
```

Пример выполнения:

```
23
{2, 3, 5, 7, 11, 13, 17, 19, 23}
```

Матрицы

Количество элементов, принадлежащих диапазону

Определите, сколько элементов в матрице принадлежат заданному диапазону.

Эта задача на простой перебор матрицы. В цикле надо пройтись по ее строкам. Во внутреннем цикле пройтись по строке и проверить каждый элемент на его вхождение в диапазон. Если он в него входит, следует увеличить счетчик таких элементов на 1.

```
from random import randint
N = 5
M = 7

matrix = []
for i in range(N):  # заполнение матрицы
    row = []
    for j in range(M):
        row.append(randint(0, 20))
    matrix.append(row)

for row in matrix:  # вывод матрицы
    for item in row:
        print("%4d" % item, end='')
    print()

low = int(input("Нижняя граница диапазона: "))
high = int(input("Верхняя граница диапазона: "))
count = 0

for i in range(N):
    for j in range(M):
        if low <= matrix[i][j] <= high:
            count += 1

print("Количество элементов, принадлежащих диапазону:", count)
```

```
   1   3   1   3   1   9  17
  17   3  18   4   9  17  14
  14  10  10   6   9  14   5
  11  19  13   4   8   9  12
   2   5  20   8   7   3   5
Нижняя граница диапазона: 10
Верхняя граница диапазона: 15
Количество элементов, принадлежащих диапазону: 8
```

Поскольку в данной задаче не важны индексы найденных элементов, то цикл поиска можно написать не через индексы, а перебирая строки и элементы матрицы:

```
for row in matrix:
    for item in row:
        if low <= item <= high:
            count += 1
```

Индексы ячеек с заданным числом

Дана матрица целых чисел. Вводится число. Вывести на экран индексы ячеек матрицы, в которых содержится данное число.

Данная задача отличается от предыдущей в основном только тем, что требуется выводить индексы найденных элементов матрицы.

```
from random import randint
N = 5
M = 7

matrix = []
for i in range(N):  # заполнение матрицы
    row = []
    for j in range(M):
        row.append(randint(0, 20))
    matrix.append(row)

for row in matrix:  # вывод матрицы
    for item in row:
        print("%4d" % item, end='')
    print()

num = int(input("Искомое число: "))

for i in range(N):
    for j in range(M):
        if num == matrix[i][j]:
            print("[{}, {}]".format(i, j))
```

Пример выполнения:

```
  17  19  14  14  19  13  11
  17  15  10   6  20  14  20
   6   8  15  11  17  18   9
   3  10  10   0  17  19  17
   5  10  13   0  17  12   4
Искомое число: 10
[1, 2]
[3, 1]
[3, 2]
[4, 1]
```

Обмен столбцов

В числовой матрице поменять местами два столбца, то есть все элементы одного столбца поставить на соответствующие им позиции другого и наоборот.

Столбец матрицы определяется вторым индексом. Поэтому в цикле следует перебирать строки матрицы, то есть переменная-счетчик будет использоваться в качестве первого индекса двумерного массива.

В каждой строке менять местами элементы с номерами указанных столбцов. Обмен можно производить через третью переменную.

Пусть номера столбцов, которые следует поменять местами, задаются путем ввода значений в момент выполнения программы.

Если в языке программирования индексация массивов начинается с нуля, а столбцы указываются, исходя из их фактического положения в матрице (начиная с 1), то следует из переменных, в которых записаны номера столбцов, вычесть единицу перед их использованием в цикле.

```
from random import random
M = 10
N = 5
a = []
for i in range(N):
    b = []
    for j in range(M):
        b.append(round(random()*2))
    a.append(b)
    print(b)

c1 = int(input("Один столбец: ")) - 1
c2 = int(input("Другой столбец: ")) - 1

for i in range(N):
    a[i][c1], a[i][c2] = a[i][c2], a[i][c1]
    print(a[i])
```

Пример выполнения:

```
[2, 1, 2, 2, 1, 0, 2, 1, 0, 1]
[1, 1, 1, 1, 1, 1, 2, 2, 2, 1]
[1, 1, 0, 1, 1, 1, 2, 0, 0, 1]
[0, 0, 0, 2, 0, 0, 1, 2, 1, 2]
[1, 0, 0, 1, 2, 0, 2, 1, 1, 0]
Один столбец: 3
Другой столбец: 10
[2, 1, 1, 2, 1, 0, 2, 1, 0, 2]
[1, 1, 1, 1, 1, 1, 2, 2, 2, 1]
[1, 1, 1, 1, 1, 1, 2, 0, 0, 0]
[0, 0, 2, 2, 0, 0, 1, 2, 1, 0]
[1, 0, 0, 1, 2, 0, 2, 1, 1, 0]
```

Суммы строк и столбцов

Посчитать суммы каждой строки и каждого столбца матрицы. Вывести суммы строк в конце каждой строки, а суммы столбцов под соответствующими столбцами.

Поскольку двумерный массив обычно перебирается построчно, то сумму строк считать проще. Можно, заполняя строку матрицы и выводя ее элементы на экран, накапливать в переменной сумму элементов строки и выводить ее в конце строки.

В Python суммы строк матрицы можно вычислить с помощью функции sum(), которой передается текущий список-строка.

Суммы столбцов вычисляются путем прохода по каждому столбу матрицы. Обратите внимание, что здесь наоборот: внешний цикл - проход по столбцам, внутренний - по строкам.

```
from random import random
M = 10
N = 5
a = []
for i in range(N):
    b = []
    for j in range(M):
        b.append(int(random()*11))
        print("%3d" % b[j], end='')
    a.append(b)
    print('   |', sum(b))

for i in range(M):
    print(" --", end='')
print()

for i in range(M):
    s = 0
    for j in range(N):
        s += a[j][i]
    print("%3d" % s, end='')
print()
```

Пример выполнения:

```
  6  7  3 10 10 10  4  2  6  5   | 63
  2  8  0  9  0  4  9  3  6  3   | 44
  5  3  1 10  5  6  5  2  0  9   | 46
 10  9 10  8  7  8  5  2 10  9   | 78
  3  3  6  0  4  1  6 10 10  3   | 46
 -- -- -- -- -- -- -- -- -- --
 26 30 20 37 26 29 29 19 32 29
```

Записать сумму в последнюю ячейку строки

В матрице заменить последний элемент каждой строки на сумму предыдущих элементов той же строки.

Алгоритм решения основной части задачи (только сложение и замена):

1. Для каждой строки присвоить переменной `summa` значение 0, индексу элемента `i` также значение 0.
2. Пока индекс элемента не дошел до последнего элемента строки, увеличивать значение `summa` на значение очередного элемента строки матрицы.
3. Иначе (когда индекс элемента указывает на последний элемент строки) изменить значение последнего элемента строки на значение переменной `summa`.

```
import random

n = 4
m = 6

matrix = []
for i in range(n):  # формируем исходную матрицу
    line = []
    for j in range(m):
        line.append(random.randint(0, 5))
    matrix.append(line)

for line in matrix:  # вывод исходной матрицы
    for i in line:
        print('%4d' % i, end='')
    print()

print()

for line in matrix:  # изменение матрицы
    summa = 0
    i = 0
    while i < m - 1:
        summa += line[i]
        i += 1
    else:
        line[m - 1] = summa

for line in matrix:  # вывод измененной матрицы
    for i in line:
        print('%4d' % i, end='')
    print()
```

Столбец с максимальной суммой

Задана матрица неотрицательных чисел. Посчитать сумму элементов в каждом столбце. Определить, какой столбец содержит максимальную сумму.

Вычислив сумму элементов очередного столбца, сравним ее со значением переменной, предназначенной для хранения максимальной суммы. Если текущая окажется больше, то запишем ее в указанную переменную. Кроме того, запомним в отдельной переменной номер текущего столбца.

Если в матрице содержится несколько столбцов с одинаковой максимальной суммой и все их надо определить, то задача решается немного по-другому. В этом случае следует запоминать суммы столбцов для их последующего сравнения с ранее найденным максимумом.

```
from random import random
M = 10
N = 5
a = []
for i in range(N):
    b = []
    for j in range(M):
        b.append(int(random()*11))
        print("%3d" % b[j], end='')
    a.append(b)
    print()

for i in range(M):
    print(" --", end='')
print()

max_sum = 0
col = 0
for i in range(M):
    s = 0
    for j in range(N):
        s += a[j][i]
    print("%3d" % s, end='')
    if s > max_sum:
        max_sum = s
        col = i
print()
print(col+1)
```

```
  5  3  4  6  0  6  6 10  1  8
  6  5 10  5  3  5  7  7 10  2
  5 10  2  4  7  5  3  4  2  2
  1  3  0  9  0  8  4  5  5  9
  9  8  5  3  7  8  4  1  5  4
 -- -- -- -- -- -- -- -- -- --
 26 29 21 27 17 32 24 27 23 25
6
```

Сумма элементов диагонали

Найти сумму элементов главной диагонали матрицы и сумму элементов ее побочной диагонали.

Диагонали есть только у квадратных матриц. Квадратная матрица – это матрица, у которой количество строк равно количеству столбцов. Главная диагональ идет из верхнего левого угла в правый нижний, побочная - из верхнего правого угла в левый нижний.

У элементов главной диагонали совпадают оба индекса. У элементов побочной диагонали второй индекс противоположен первому.

```
from random import random
N = 5
matrix = []
for i in range(N):
    row = []
    for j in range(N):
        row.append(int(random()*10))
    matrix.append(row)

for row in matrix:
    print(row)

sumMain = 0
sumSecondary = 0
for i in range(N):
    sumMain += matrix[i][i]
    sumSecondary += matrix[i][N-i-1]

print(sumMain)
print(sumSecondary)
```

Пример выполнения:

```
[8, 7, 9, 4, 2]
[3, 4, 1, 4, 2]
[3, 4, 4, 2, 1]
[1, 6, 9, 8, 3]
[1, 9, 4, 7, 9]
33
17
```

Обмен значений диагоналей

В квадратной матрице в каждой строке обменять значения элементов, которые расположены на главной и побочной диагоналях. То есть произвести обмен значений главной и побочной диагоналей.

Диагонали можно выделить лишь в квадратной матрице (когда количество строк равно количеству столбцов). Главная диагональ идет из верхнего левого угла в правый нижний, а побочная - из правого верхнего в левый нижний. Таким образом элементы главной диагонали имеют равные индексы строки и столбца, а у побочной - противоположные.

Поскольку обмен происходит в одной строке, то значения подлежащие обмену имеют следующие индексы:

- у элемента главной диагонали - первый и второй индексы равны друг другу (`i` = `j`),
- у элемента побочной диагонали - первый индекс равен `i`, а второй `N-1-j` (при индексации с нуля).

При проходе по матрице обмен значений происходит лишь в том случае, когда `i = j`.

```
from random import random
N = 10
a = []
for i in range(N):
    z = []
    for j in range(N):
        n = int(random() * 100)
        z.append(n)
        print("%4d" % n, end='')
    print()
    a.append(z)
print()

for i in range(N):
    for j in range(N):
        if i==j:
            b = a[i][j]
            a[i][j] = a[i][N-1-j]
            a[i][N-1-j] = b

for i in range(N):
    for j in range(N):
        print("%4d" % a[i][j], end='')
    print()
```

Максимальный среди минимальных элементов столбцов

Найдите максимальный элемент среди минимальных элементов столбцов матрицы.

Данная задача состоит из двух подзадач: 1) найти свой минимальный элемент в каждом столбце матрицы; 2) сравнить найденные минимумы и выбрать из них максимум.

В одной из переменных будем сохранять минимальный элемент текущего столбца. После того, как столбец просмотрен, будем сравнивать его значение со значением переменной, предусмотренной для хранения максимального элемента среди минимальных.

Начальными значениями этих переменных должны быть: для хранения текущего минимума - большее значение, чем максимально возможное; для хранения максимума среди минимумов - меньшее значение, чем минимально возможное. Например, если диапазон матрицы от 0 до 199, то одной переменной надо присвоить 200, а другой -1.

Поскольку перебираются столбцы, то во внутреннем цикле меняется значение переменной, отвечающей за строки. При выводе же матрицы на экран осуществляем обычный построчный вывод.

```
from random import random
M = 10
N = 5
a = []
for i in range(N):
    b = []
    for j in range(M):
        n = int(random()*200)
        b.append(n)
        print('%4d' % n,end='')
    a.append(b)
    print()

mx = -1
for j in range(M):
    mn = 200
    for i in range(N):
        if a[i][j] < mn:
            mn = a[i][j]
    if mn > mx:
        mx = mn
print("Максимальный среди минимальных: ", mx)
```

```
 104  60  85 107 189 194   2 143  65 117
 109  74 186 102 166 133 169 117  94 190
 152  20  63  98   1  93  81 109  23  86
  89 186 191 176  67   7   3 109 120  66
  89 185  86 167  16   3 163  62  24  84
Максимальный среди минимальных:  98
```

В четвертом столбце минимальный элемент равен 98. Во всех других столбцах минимальные элементы меньше этого значения.

Сортировка столбцов

Отсортируйте столбцы матрицы по возрастанию их сумм.

При нахождении суммы элементов конкретного столбца меняется первый индекс, то есть номер строки.

В программе ниже найденные суммы элементов столбцов заносятся в одномерный список. Количество элементов этого списка равно количеству столбцов матрицы.

Далее происходит сортировка пузырьком списка сумм. В случае если происходит обмен значений в списке сумм, то выполняется обмен соответствующих столбцов матрицы. Индексы обменивающихся значений в списке сумм, совпадают с индексами обменивающихся столбцов. При обмене значений столбцов следует в цикле пройтись по всем строкам матрицы. Таким образом, вместе с сортировкой одномерного списка происходит сортировка столбцов матрицы.

В конце программы матрица и суммы столбцов снова выводятся на экран, но уже в отсортированном виде.

```
from random import random
N = 5
M = 7

matrix = []
for i in range(N):
    a = [0] * M
    for j in range(M):
        a[j] = int(random()*10)
        print('%3d' % a[j],end='')
    matrix.append(a)
    print()

suma = [0] * M
for j in range(M):
    for i in range(N):
        suma[j] += matrix[i][j]
    print('%3d' % suma[j],end='')
print('\n')

for j in range(M-1):
    for i in range(M-j-1):
        if suma[i] > suma[i+1]:
            suma[i],suma[i+1] = suma[i+1],suma[i]
            for k in range(N):
                matrix[k][i],matrix[k][i+1] = matrix[k][i+1],matrix[k][i]

for i in matrix:
    for j in i:
        print('%3d'%j,end='')
    print()
for i in suma:
    print('%3d'%i,end='')
print()
```

Строки

Палиндром

Вводится строка. Определить является ли она палиндромом и вывести соответствующее сообщение. Палиндромом называется слово или фраза, которые читаются одинаково справа налево, и слева направо.

```
# ввод строки
s = input()

# длинна строки
l = len(s)

# длина половины строки
# ( Находится делением нацело на 2.
# Если количество символов нечетно,
# то стоящий в середине не учитывается,
# т.к его сравниваемая пара - он сам. )
l = l//2

# количество итераций цикла равно длине половины строки
for i in range(l):
    # Если символ с индексом i не равен "симметричному"
    # символу с конца строки (который находится путем
    # индексации с конца),
    if s[i] != s[-1-i]:
        # то выводится сообщение, что строка не палиндром
        print("It's not palindrome")
        # выход из программы
        quit()

# До этого места кода программа дойдет, если не произойдет
# выход из программы в цикле выше.
# Если выхода не произошло, значит строка - палиндром.
print("It's palindrome")
```

Пример выполнения:

```
banana
It's not palindrome
```

Проверка расширения файла

Вводится имя файла. Требуется проверить, что его расширение входит в список допустимых.

```
# список допустимых расширений
extensions = ['png', 'jpg', 'jpeg', 'gif', 'svg']

# Введенная строка преобразуется в список.
# Разделение происходит по точке.
file = input().split('.')

# Если длина списка равна двум (или больше),
# то расширение было указано.
if len(file) >= 2:
    # Оно последнее в списке. Извлекаем его
    # и преобразуем к нижнему регистру.
    fileExtension = file[-1].lower()
    # Если расширение содержится в списке
    # допустимых, то выводится "Yes".
    if fileExtension in extensions:
        print("Yes")
    # Когда расширения нет в списке.
    else:
        print("No")
# Длина списка file меньше двух. Значит, у файла
# нет расширения.
else:
    print("The file doesn't have an extention.")
```

Пример выполнения программы:

```
house.JPEG
Yes
```

Сортировка по возрастанию длины слов

Вводится строка, состоящая из слов, разделенных пробелами. Требуется сформировать новую строку из тех же слов, но так, чтобы слова в этой строке были отсортированы по возрастанию их длины.

Для решения данной задачи проще воспользоваться встроенными в Python функциями и методами строк и списка.

1. С помощью строкового метода split() разделяем строку по пробелам.
2. С помощью функции sorted() или метода списка sort() сортирует по длине строки. При этом функция len() передается как значение аргумента key. Различие между sorted() и sort() заключается в том, что первый возвращает новый список, а второй сортирует на месте, т. е. изменяет имеющийся.
3. С помощью строкового метода join() формируем из списка новую строку.

```
s = input()
s = s.split()
s.sort(key=len)
s = " ".join(s)
print(s)
```

Пример работы программы:

```
blue red green yellow pink
red blue pink green yellow
```

Процент строчных и прописных букв

Вводится строка. Необходимо определить в ней проценты прописных, то есть больших, и строчных, то есть малых, букв.

Как известно в кодировках символы упорядочены, при этом код символа 'a' меньше символа 'b'. Поэтому если очередной символ принадлежит диапазону от 'a' до 'z', значит это строчная буква; если диапазону от 'A' до 'Z' - то прописная.

```
string = input("Введите текст: ")

length = len(string)
let_s = let_b = 0
for i in string:
    if 'a' <= i <= 'z':
        let_s += 1
    elif 'A' <= i <= 'Z':
        let_b += 1

print("%% строчных букв: %.2f" % (let_s / length * 100))
print("%% прописных букв: %.2f" % (let_b / length * 100))
```

Удвоенный знак процента в начале строки вывода стоит для того, чтобы вывести один знак процента. Иначе одиночный % интерпретируется как начало формата вывода.

```
Введите текст: Hello, G!
% строчных букв: 44.44
% прописных букв: 22.22
```

В языке Python есть строковые методы islower() и isupper(). Первый проверяет, являются ли все символы в строке маленькими, второй - большими. Эти методы могут работать и со строками длинной в один символ. Поэтому решение задачи может выглядеть так:

```
string = input()

length = len(string)

lower = upper = 0

for i in string:
    if i.islower():
        lower += 1
    elif i.isupper():
        upper += 1

per_lower = lower / length * 100
per_upper = upper / length * 100
print("Lower: %.2f%%" % per_lower)
print("Upper: %.2f%%" % per_upper)
```

Это более правильное решение, так как здесь проверяются и русские буквы.

Самое короткое слово

Определить длину самого короткого слова в строке.

В Python у строкового типа данных есть метод split(), который разделяет слова по пробелу (или другому символу, если он передан в качестве аргумента). Получается список слов.

Далее из этого списка можно извлекать отдельные слова, обращаясь к ним по индексам.

В программе ниже сначала предполагается, что первое слово в списке и есть самое короткое. Остальные слова перебираются в цикле, где их длина сравнивается со словом в переменной shortest. Если длина текущего слова меньше, то значение shortest заменяется.

```
string = input()

words = string.split()

shortest = words[0]

for i in words[1:]:
    if len(i) < len(shortest):
        shortest = i

print(shortest)
print(len(shortest))
```

Пример выполнения:

```
ones two three four five six
two
3
```

Если нужно вывести просто длину:

```
string = input()

words = string.split()
shortest = min([len(word) for word in words])

print(shortest)
```

Обратный порядок слов

Вводится строка, состоящая из нескольких слов. Выведите на экран строку с обратным порядком слов.

Для решения данной задачи следует воспользоваться строковыми методами split() и joit(), а также списковым reverse(). Метод split() преобразует исходную строку в список строк, reverse() поменяет список слов на обратный, а join() объединит слова списка снова в единую строку.

```
s = input()

word_list = s.split()

word_list.reverse()

s2 = " ".join(word_list)

print(s2)
```

Пример выполнения:

```
one two three four
four three two one
```

Замена подстроки

В строке найти и заменить одну подстроку на другую. Если одинаковых подстрок несколько, заменить все.

У строк есть метод find(), который возвращает индекс первого символа подстроки в строке или -1, если подстрока не найдена.

Алгоритм решения задачи может быть таким:

1. Находим первое вхождение подстроки в строке.
2. Берем срез от начала строки до начала старой подстроки.
3. Берем срез от конца старой подстроки до конца строки. Пунктами 2-3 мы как бы вырезаем старую подстроку.
4. Соединяем начало строки, новую подстроку, конец строки.
5. Снова проверяем строку на вхождение в нее старой подстроки.

```
things = "tree, box, chair, lamp, \n" \
         "desk, cat, dog, grass, \n" \
         "pig, box, lamp, shelf"
print(things)
print()

old_item = input("Old item: ")
new_item = input("New item: ")
len_old_item = len(old_item)

i = things.find(old_item)
while i > 0:
    before = things[:i]
    after = things[i+len_old_item:]
    things = before + new_item + after
    i = things.find(old_item)

print()
print(things)
```

Пример выполнения:

```
tree, box, chair, lamp,
desk, cat, dog, grass,
pig, box, lamp, shelf

Old item: box,
New item: cube!

tree, cube! chair, lamp,
desk, cat, dog, grass,
pig, cube! lamp, shelf
```

Заменить пробелы звездочкой

В строке заменить пробелы звездочкой. Если встречается подряд несколько пробелов, то их следует заменить одним знаком "*", пробелы в начале и конце строки удалить.

Решение данной задачи классическим способом, без использования продвинутых возможностей Python, может быть таким.

Сначала избавляемся от пробелов в начале и конце строки, если они имеются. Для этого перебираем строку посимвольно сначала или с конца. Как только встречается первый непробельный символ, берем срез от него до конца строки или до начала, в случае удаления пробелов с конца.

Далее снова посимвольно перебираем строку. Если очередной символ не является пробелом, то формировать новую строку добавлением к ней этого символа. В ветку elif попадают символы пробела, но здесь проверяется не был ли равен предыдущий символ пробелу. Если это не так (не был равен), то только тогда к новой строке добавляется "*". Лишние пробелы будут просто пропущены и не добавлены к новой строке.

```
s = input()
i = 0
while s[i] == ' ':
    i += 1
s = s[i:]
i = len(s)
while s[i - 1] == ' ':
    i -= 1
s = s[:i]

s_new = s[0]
i = 1
while i < len(s):
    if s[i] != ' ':
        s_new += s[i]
    elif s[i - 1] != ' ':
        s_new += '*'
    i += 1
print(s_new + '!')
```

```
   one    two three four f*ve    six seven
one*two*three*four*f*ve*six*seven!
```

Однако в языке программирования Python данную задачу проще решить, преобразовав строку в список. С помощью метода split() строка разделяется на слова по пробелам. При этом неважно сколько их. Далее остается только снова собрать слова в строку методом join():

```
s = input()
l = s.split()
s1 = '*'.join(l)
print(s1)
```

Изъять числа из строки

Дана строка, содержащая натуральные числа и слова. Необходимо сформировать список из чисел, содержащихся в этой строке. Например, задана строка "abc83 cde7 1b 24". На выходе должен получиться список [83, 7, 1, 24].

Следует посимвольно перебирать строку. Если очередной символ цифра, надо добавить ее в новую строку. Далее проверять символы за ней, и если они тоже цифры, то добавлять их в конец этой новой подстроки из цифр. Когда очередной символ окажется не цифрой, или будет достигнут конец строки, то надо преобразовать строку из цифр в число и добавить в список.

```
s = input()
l = len(s)
integ = []
i = 0
while i < l:
    s_int = ''
    a = s[i]
    while '0' <= a <= '9':
        s_int += a
        i += 1
        if i < l:
            a = s[i]
        else:
            break
    i += 1
    if s_int != '':
        integ.append(int(s_int))

print(integ)
```

```
5 dkfj78df 9 8 dkfj8
[5, 78, 9, 8, 8]
```

Решение через цикл for:

```
a = input()
num_list = []

num = ''
for char in a:
    if char.isdigit():
        num = num + char
    else:
        if num != '':
            num_list.append(int(num))
            num = ''
if num != '':
    num_list.append(int(num))

print(num_list)
```

Если в строке числа всегда отделены от слов пробелами, задача решается проще:

```
s = input()
word_list = s.split()
num_list = []

for word in word_list:
    if word.isnumeric():
        num_list.append(int(word))

print(num_list)
```

Здесь происходит разделение строки на слова по пробелам. В цикле с помощью метода isnumeric() каждое слово проверяется, является ли оно числом. Подобную задачу можно решить в одну строку, если использовать функцию filter().

```
s = input()
word_list = s.split()

num_list = [int(num) for num in filter(lambda num: num.isnumeric(), word_list)]

print(num_list)
```

В функцию filter() передается лямбда-выражение, проверяющее слова, и список слов. Функция возвращает список строк-чисел. Далее с помощью генератора списка строки преобразовываются в целочисленный тип.

На практике при решении подобных задач, когда надо найти и извлечь из строки что-либо, обычно пользуются регулярными выражениями. В примере ниже не обязательно, чтобы число было отделено пробелами.

```
import re

s = input()

nums = re.findall(r'\d+', s)

nums = [int(i) for i in nums]

print(nums)
```

Для поиска вещественных чисел:

```
import re

s = input()

nums = re.findall(r'\d*\.\d+|\d+', s)

nums = [float(i) for i in nums]

print(nums)
```

Удалить повторяющиеся символы

Вводится строка. Требуется удалить из нее повторяющиеся символы и все пробелы.

Например, если было введено "abc cde def", то должно быть выведено "abcdef".

Проще всего задачу решить, если формировать результирующую строку в другой переменной, а не изменять значение переменной, которой была присвоена введенная строка.

Извлекать каждый символ введенной строки. Если он не встречается в новой строке и не является пробелом, то добавлять его в конец новой строки.

```
s = input()
new_s = ''
for i in s:
    if i not in new_s and i != ' ':
        new_s += i

print(new_s)
```

Пример выполнения:

```
one two three
onetwhr
```

Также можно воспользоваться функцией find():

```
for i in s:
    if new_s.find(i) == -1 and i != ' ':
        new_s += i
```

Функции

Среднее арифметическое двух чисел

Напишите функцию, которая вычисляет среднее арифметическое двух переданных ей числовых аргументов.

```
def average(n1, n2):
    m = (n1 + n2) / 2
    return m


a = int(input("A = "))
b = int(input("B = "))

avrg = average(a, b)
print(round(avrg, 2))
```

Пример выполнения:

```
A = 40
B = 93
66.5
```

Среднее арифметическое произвольного количества чисел

Напишите функцию, которая вычисляет среднее арифметическое произвольного числа аргументов.

Для решения этой задачи необходимо знать о том, как на Python в функцию передавать произвольное число аргументов. Они передаются через кортеж, который присваивается переменной-параметру, перед которой стоит знак звездочки.

Кортеж – это почти то же самое что список, только кортеж нельзя изменять. Однако у кортежа, также как у списка, можно найти сумму его элементов и выяснить длину, следовательно, вычислить среднее арифметическое его элементов.

```
def average(*n):
    m = sum(n) / len(n)
    return m


avrg1 = average(10, 12)
avrg2 = average(1, 2, 3.4, 3, 8)
avrg3 = average(-0.5, 2.3, 1, 5.3, -2, 1.55)
print(round(avrg1, 2))
print(round(avrg2, 2))
print(round(avrg3, 2))
```

Пример выполнения:

```
11.0
3.48
1.27
```

Однако на практике в функцию обычно передают уже готовый список (точнее ссылку на него):

```
def list_avrg(lst):
    l = len(lst)
    suma = 0
    for i in lst:
        suma += i
    return suma / l


print("Input integers:")
a = input()
a = a.split()
for i in range(len(a)):
    a[i] = int(a[i])

avrg = list_avrg(a)

print("Average:")
print(round(avrg, 2))
```

Заполнение списка случайными числами

Напишите функцию, которая заполняет список случайными числами. В качестве аргументов функция должна принимать список, количество элементов, минимальное и максимальное значение, то есть границы диапазона.

```
from random import randint


def fill_list(lst, qty, low, high):
    """Функция заполняет список 'lst'
    случайными значениями количеством 'qty'
    в диапазоне от 'low' до 'high' включительно."""

    for i in range(qty):
        lst.append(randint(low, high))


# границы диапазона
minimum = int(input("Min: "))
maximum = int(input("Max: "))

# количество элементов списка
n = int(input("Quantity: "))

# заполняемый список
a = []

# В языке Python в функцию
# передается ссылка на список.
# Поэтому функция fill_list()
# ничего не возвращает,
# она будет заполнять внешний для нее список.
fill_list(a, n, minimum, maximum)

print(a)
```

Пример выполнения:

```
Min: 11
Max: 29
Quantity: 5
[11, 29, 17, 28, 21]
```

Функция бинарного поиска

Напишите функцию, которая выполняет бинарный поиск элемента в списке. В качестве аргументов функция принимает список и значение, возвращает – индекс элемента с заданным значением.

Пояснения к алгоритму см. в задаче "Двоичный поиск", раздел "Списки (часть2)".

```
from random import randint


def search(lst, item):
    mid = len(lst) // 2
    low = 0
    high = len(lst) - 1
    while lst[mid] != item and low <= high:
        if item > lst[mid]:
            low = mid + 1
        else:
            high = mid - 1
        mid = (low + high) // 2
    if low > high:
        return None
    else:
        return mid


a = []
for i in range(10):
    a.append(randint(1, 20))
a.sort()
print(a)

value = int(input())

print(search(a, value))
```

Пример выполнения:

```
[1, 8, 8, 10, 12, 13, 15, 18, 19, 19]
18
7
```

Ряд Фибоначчи

Напишите функцию, которая выводит на экран ряд Фибоначчи в количестве элементов, соответствующим переданному в нее аргументу.

```
def fib_row(item):
    f1 = f2 = 1
    print(f1, f2, end=' ')
    while item > 2:
        buff = f2
        f2 = f1 + f2
        f1 = buff
        print(f2, end=' ')
        item -= 1
    print()


n = int(input())

m = fib_n(n)
print(m)

fib_row(n)
```

Пример выполнения:

```
9
1 1 2 3 5 8 13 21 34
```

Рекурсивное вычисление факториала

Напишите функцию, которая вычисляет факториал числа с помощью рекурсии.

```
def factorial(a):
    """Функция принимает натуральное число и
    возвращает его факториал.
    Факториал вычисляется с помощью
    рекурсивного вызова самой функции."""

    # факториал единицы равен единице
    if a == 1:
        return a

    # Пример потока выполнения, если a = 3.
    # 1) 3 * factorial(2)
    # 2) 2 * factorial(1)
    # 3) return 1
    # 2) return 2 * 1
    # 1) return 3 * 2
    return a * factorial(a-1)


n = int(input())
print(factorial(n))
```

Пример выполнения:

```
6
720
```

Функция проверки числа на простоту

Напишите функцию, которая проверяет число на простоту.

```
from math import sqrt


def is_prime(n):
    # Если число меньше двух, то оно ни простое, ни сложное.
    if n < 2:
        return False
    # Число 2 является простым.
    if n == 2:
        return True
    # Верхняя граница делителей.
    limit = sqrt(n)
    # Нижняя граница делителей.
    i = 2
    while i <= limit:
        # Если число делится на делитель, оно составное.
        if n % i == 0:
            return False
        i += 1
    # Если до этого не произошел выход из функции, значит число простое.
    return True


# Функция вызывается три раза.
# Аргумент для нее вводит пользователь.
for i in range(3):
    num = int(input())
    print(is_prime(num))
```

Пример выполнения:

```
17
True
57
False
103
True
```

Наименьшее общее кратное

Напишите функцию, вычисляющую наименьшее общее кратное двух целых чисел.

```
# Наименьшее общее кратное двух чисел -
# это наименьшее число,
# которое делится нацело на оба заданных числа.

def lcm(a, b):

    # Произведение всегда кратно
    # любому из своих множителей.
    m = a * b
    # Однако оно может быть
    # не наименьшим общим кратным.

    # поиск наибольшего общего делителя
    while a != 0 and b != 0:
        if a > b:
            a = a % b
        else:
            b = b % a

    # Наименьшее общее кратное вычиляется
    # с помощью деления произведения чисел
    # на их наибольший общий делитель.
    # Здесь используется целочисленное деление
    # только потому, что Python
    # при обычном делении (/)
    # всегда возвращает тип float.
    return m // (a + b)


x = int(input('a='))
y = int(input('b='))
print('LCM:', lcm(x, y))
```

Пример выполнения:

```
a=14
b=6
LCM: 42
```

Циклический сдвиг

Напишите функцию, выполняющую циклический сдвиг в списке целых чисел на указанное число шагов. Сдвиг должен быть кольцевым, то есть элемент, вышедший за пределы списка, должен появляться с другого его конца.

Для решения этой задачи можно воспользоваться следующими методами списка:

- append() - добавляет элемент в конец списка,
- insert() - вставляет элемент по указанному индексу,
- pop() - извлекает элемент с конца списка или, если был передан индекс, по индексу.

Пусть функция shift() в качестве аргументов принимает список и количество шагов сдвига. Если шаги представлены положительным числом, то сдвиг выполняется вправо, то есть надо извлечь последний элемент и добавить его в начало.

Если шаги - отрицательное число, то будем выполнять сдвиг справа налево, то есть надо извлечь первый элемент и добавить его в конец.

```
def shift(lst, steps):
    if steps < 0:
        steps = abs(steps)
        for i in range(steps):
            lst.append(lst.pop(0))
    else:
        for i in range(steps):
            lst.insert(0, lst.pop())


nums = [4, 5, 6, 7, 8, 9, 0]
print(nums)

shift(nums, -2)
print(nums)

shift(nums, 3)
print(nums)
```

Пример выполнения:

```
[4, 5, 6, 7, 8, 9, 0]
[6, 7, 8, 9, 0, 4, 5]
[0, 4, 5, 6, 7, 8, 9]
```

Как часто встречается каждый элемент списка

Список заполняется случайными числами. Требуется посчитать сколько раз в нем встречается каждое число. Заполнение списка и его анализ реализовать в виде отдельных функций.

Для хранения количества каждого встречающегося в списке значения создадим словарь. В нем ключами будут числа, которые встречаются в списке, а значениями - количества этих чисел в списке. Например, для списка [1, 1, 3, 2, 1, 3, 4] в итоге должен был бы получиться такой словарь: {1:3, 3:2, 2:1, 4:1}.

Пусть в программе будет функция, которая заполняет список случайными числами в диапазоне и количестве, указанными пользователем.

Другая функция будет считать количество каждого значения и заносить данные в словарь. Алгоритм подсчета заключается в следующем. Если очередной элемент списка уже есть в качестве ключа словаря, то следует увеличить значение этого ключа на единицу. Если очередного элемента списка нет в качестве ключа в словаре, то такой ключ следует добавить и присвоить ему значение, равное единице.

Для того, чтобы вывести содержимое словаря в отсортированном по возрастанию ключей виде, используется функция sorted(). Она сортирует ключи словаря и помещает их в список.

```
def fill_list(min_bound, max_bound, amount_items, filling_list):
    from random import randint
    for i in range(amount_items):
        filling_list.append(randint(min_bound, max_bound))


def analysis(your_list, your_dict):
    for i in your_list:
        if i in your_dict:
            your_dict[i] += 1
        else:
            your_dict[i] = 1


lst = []
dct = {}

mn = int(input('Минимум: '))
mx = int(input('Максимум: '))
qty = int(input('Количество элементов: '))

fill_list(mn, mx, qty, lst)
analysis(lst, dct)

for item in sorted(dct):
    print("'%d':%d" % (item, dct[item]))
```

Пример выполнения:

```
Минимум: 100
Максимум: 104
Количество элементов: 20
'100':4
'101':7
'102':2
'103':4
'104':3
```

Словари и файлы

Создать словарь из двух списков

Даны два списка. Создайте словарь, ключами которого являются элементы первого списка, а значениями – элементы второго, находящиеся в соответствующих позициях.

```
a = [1, 2, 3, 4, 5]
b = ['a', 'b', 'c', 'd', 'e']

c = {}

# Количество итераций
# не должно превышать длину
# более короткого списка,
# если списки разной длины.
for i in range(len(a)):

    # Элемент под индексом 'i'
    # одного списка будет ключом.
    key = b[i]

    # Элемент под индексом 'i'
    # другого списка будет значением.
    value = a[i]

    # В словарь добавляется пара
    # ключ-значение.
    c[key] = value

print(c)
```

Результат:

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Иное решение – использовать имеющуюся в Python функцию zip(), которая создает объект-итератор, из которого на каждой итерации цикла извлекается кортеж состоящий из двух элементов. Первый берется из первого аргумента-последовательности, второй – из второго. Функция dict() автоматически проитерирует zip-объект и создаст словарь.

```
c = dict(zip(b, a))
```

Удалить случайный элемент словаря

Дан словарь. Напишите программу, которая удаляет случайный элемент словаря.

```
import random

d = {"A": 4, "O": 6, "P": 10,
     "M": 7, "B": 3}
print(d)

# Метод keys() создает объект,
# содержащий ключи словаря.
# Этот объект преобразуется в
# список ключей с помощью функции list().
keys = list(d.keys())

# Функция choice() выбирает
# случайный элемент из последовательности.
# В переменную 'what_del'
# записывается случайный ключ.
what_del = random.choice(keys)

# Удаляем из словаря элемент
# по его ключу.
del d[what_del]

print(d)
```

Пример выполнения:

```
{'A': 4, 'O': 6, 'P': 10, 'M': 7, 'B': 3}
{'A': 4, 'O': 6, 'M': 7, 'B': 3}
```

Сортировка словаря

Дан словарь, ключами которого являются строки, а значениями – числа. Выведите его на экран сначала в отсортированном по ключам виде (в алфавитном порядке), затем – отсортированным по возрастанию значений.

На самом деле содержимое словаря отсортировать нельзя, так как словарь в Python - это неупорядоченная структура данных. Даже если вы будете добавлять в словарь элементы упорядоченно, например по алфавиту, при выводе они могут отобразится по-другому.

Однако при извлечении элементов из словаря можно сделать так, чтобы это происходило согласно определенному порядку. Для этого дополнительно используется упорядоченная структура, которую можно отсортировать. Например, список.

Проще всего выполнить сортировку словаря по ключам. Алгоритм вывода содержимого словаря:

1. Создать список ключей словаря.
2. Отсортировать его.
3. В цикле for перебрать элементы списка, используя элемент списка как ключ словаря.

Отсортировать словарь по значениям сложнее, так как обращаться к элементам словаря можно только по ключам. Однако можно создать список кортежей ("ключ", "значение") и отсортировать его по вторым элементам пар. Далее в программе используется именно данная упорядоченная структура, а не сам оригинальный словарь.

```
d = {'a': 10, 'b': 15, 'c': 4}

print("Сортировка по ключам:")
list_keys = list(d.keys())
list_keys.sort()
for i in list_keys:
    print(i, ':', d[i])
print()

print("Сортировка по значениям")
list_d = list(d.items())
list_d.sort(key=lambda i: i[1])
for i in list_d:
    print(i[0], ':', i[1])
```

```
Сортировка по ключам:
a : 10
b : 15
c : 4

Сортировка по значениям
c : 4
a : 10
b : 15
```

Поход в магазин

Дан словарь, в котором ключами являются названия товаров, а значениями – их цены. Напишите программу, которая выводит на экран товары и цены, запрашивает у пользователя, что он хочет купить, в каком количестве, и считает общую цену покупки.

```
goods = {"Apple": 4.5,
         "Orange": 6.2,
         "Pineapple": 10.0,
         "Mango": 7.5,
         "Banana": 3.8}
# Методо items() словаря возвращает объект,
# содержащий кортежи 'ключ-значение'.
# На каждой итерации цикла
# из этого объекта извлекается
# очередной кортеж.
# Ключ присваивается переменной 'good',
# значение - переменной 'price'.
for good, price in goods.items():
    print(good, " - ", price)

# общая стоимость покупки
cost = 0
while True:
    # Пользователь должен ввести
    # название товара или букву 'n',
    # чтобы завершить работу цикла.
    good = input("What? (n - nothing) ")
    if good == 'n':
        break

    # Пользователь должен ввести
    # количество указанного товара.
    qty = int(input("How many? "))

    # Вычисляется общая стоимость
    # указанного количества товара
    # и добавляется к общей стоимости.
    cost += goods[good] * qty

print("Price:", cost)
```

```
Apple  -  4.5
Orange  -  6.2
Pineapple  -  10.0
Mango  -  7.5
Banana  -  3.8
What? (n - nothing) Apple
How many? 2
What? (n - nothing) Mango
How many? 1
What? (n - nothing) n
Price: 16.5
```

Поместить содержимое файла в список

Напишите программу, которая читает текстовый файл и помещает его содержимое в список строк.

```
data = []

# Функция open() возвращает
# файловый объект, который
# поддерживает итерацию по нему.
f = open("text.txt")

# На каждой итерации из файла
# извлекается строка вместе
# с символом перехода на новую строку,
# присваивается переменной 'i'
# и добавляется в список.
for i in f:
    data.append(i)

print(data)

# Избавляемся от символа перехода
# на новую строку.
for i in range(len(data)):
    if data[i][-1] == '\n':
        data[i] = data[i][:-1]


print(data)
```

Пример выполнения:

```
['one two\n', 'town city\n', 'big small']
['one two', 'town city', 'big small']
```

Обрезать символ конца строки проще с помощью метода strip():

```
for i in range(len(data)):
    data[i] = data[i].strip()
```

Сколько раз встречается слово в файле

Напишите программу, которая считает сколько раз в файле встречается заданное слово.

Пример текстового файла:

```
apple banana potato apple
pineapple orange lemon
kiwi lemon orange
potato orange apple
```

Код программы:

```
f = open("text.txt")
word = input()

word_count = 0
for line in f:
    word_list = line.split()
    word_count += word_list.count(word)

print(word_count)
```

Пример выполнения:

```
lemon
2
```

С другой стороны, можно было бы не извлекать данные построчно из файлового объекта-итератора, а считать сразу все данные методом read(). Однако для больших файлов это нерационально, так как данные займут много памяти.

```
f = open("text.txt")
all_lines = f.read()

word = input()
word_count = all_lines.count(word)

print(word_count)
```

Обратим внимание, что мы считаем слова, а не количество вхождений подстроки в строку. В последнем случае, следовало воспользоваться методом find() строкового объекта, который возвращает индекс первого символа подстроки.

Количество строк, слов и символов в файле

Определить, сколько в текстовом файле строк, слов и символов.

Открытый файл читается построчно. Если очередная строка считывается, то увеличивается счетчик строк.

Длину строки, то есть количество в ней символов, определяют с помощью встроенной в Python функции len().

Для подсчета количества слов вводится переменная-флаг, с помощью которой определяется, находится ли очередной символ внутри слова или он - начало нового слова. Если символ не пробел и флаг сообщает, что он вне слова, то увеличивается счетчик слов, и флаг устанавливается в значение "внутри слова". Как только встречается пробел, флаг сбрасывается в позицию "вне слова".

```
import sys

fname = sys.argv[1]
lines = 0
words = 0
letters = 0

for line in open(fname):
    lines += 1
    letters += len(line)

    pos = 'out'
    for letter in line:
        if letter != ' ' and pos == 'out':
            words += 1
            pos = 'in'
        elif letter == ' ':
            pos = 'out'

print("Lines:", lines)
print("Words:", words)
print("Letters:", letters)
```

Пример выполнения:

```
user@comp:~$ python3 words.py text.txt
Lines: 3
Words: 6
Letters: 27
```

При выполнении программы words.py вторым аргументом (индекс 1) в командной строке передается имя файла text.txt, анализ которого выполняется. Он находится в том же каталоге, что и программа. Команда вызова интерпретатора – `python3` – к аргументам не относится.

С другой стороны, имя файла можно жестко вшить в программу или запрашивать с ввода и присваивать переменной `fname`.

Поскольку в Python есть строковый метод split(), то подсчет слов можно упростить, если преобразовывать каждую строку в список слов, измерять его длину, которую добавлять к общему количеству:

```
import sys

fname = sys.argv[1]
lines = 0
words = 0
letters = 0

for line in open(fname):
    lines += 1
    letters += len(line)
    words += len(line.split())

print("Lines:", lines)
print("Words:", words)
print("Letters:", letters)
```

Прочитать файл в словарь

Дан текстовый файл, каждая строка которого содержит название товара, его цену и количество. Напишите программу, которая помещает содержимое этого файла в словарь так, что каждая запись словаря соответствует одной строке файла. Ключом записи является название товара, значением – список, первый элемент которого цена, второй – количество товара.

Содержимое файла goods.txt:

```
Apple 4.5 10
Orange 6.2 5
Pineapple 10.0 1
Mango 7.5 2
Banana 3.8 10
```

Программа:

```
products = {}

# На каждой итерации цикла
# считывается одна строка файла.
for i in open("goods.txt"):

    # строка преобразуется в список слов
    row = i.split()

    # Второй элемент преобразуется
    # в вещественное число,
    row[1] = float(row[1])
    # третий элемент - в целое.
    row[2] = int(row[2])

    # В словарь добавляется пара ключ-значение.
    # Ключом является первый элемент списка.
    # Значением будет список, состоящий
    # из второго и третьего элементов.
    products[row[0]] = row[1:]

print(products)
```

Результат выполнения:

```
{'Apple': [4.5, 10], 'Orange': [6.2, 5], 'Pineapple': [10.0, 1], 'Mango': [7.5, 2],
'Banana': [3.8, 10]}
```

Записать словарь в файл

Имеется словарь, ключами которого являются товары, а значениями – их цена.

Запишите данные словаря в файл так, чтобы в каждой строке находилась одна запись словаря.

```
goods = {"Apple": 4.5,
         "Orange": 6.2,
         "Pineapple": 10.0,
         "Mango": 7.5,
         "Banana": 3.8}

f = open("write_goods.txt", 'w')

template = "{} {}\n"
for fruit, price in goods.items():
    line = template.format(fruit, price)
    f.write(line)

f.close()
```

Содержимое файла после выполнения программы:

```
Apple 4.5
Orange 6.2
Pineapple 10.0
Mango 7.5
Banana 3.8
```

Замена табуляции пробелами

В текстовом файле заменить все символы табуляции четырьмя пробелами.

Пример текстового файла tab.txt, содержащего символы табуляции (это не программа к задаче!):

```
if a > b:
	print(c)

while c > 0:
	n = c % 10
	if n > 5:
		print(n)
	c %= 10
```

Решение задачи:

```
# файл открывается на чтение
tab_file = open("tab.txt")
# читается все содержимое файла
tab_text = tab_file.read()
# файл закрывается
tab_file.close()

# Функция repr() используется,
# чтобы увидеть непечатыемые
# символы форматирования
# (табуляцию, переход на новую строку и др.)
print(repr(tab_text))

# Текст преобразуетс в список.
# Разделение происходит в местах
# знака табуляции.
list_text = tab_text.split('\t')

# Список объединяется в текст.
# Между элементами списка вставляется строка,
# состоящая из четырех пробелов.
space_text = '    '.join(list_text)

print(repr(space_text))

# файл открывается на запись
space_file = open("space.txt", 'w')
# текст записывается в файл
space_file.write(space_text)
# файл закрывается
space_file.close()
```

Вывод на экран:

```
'if a > b:\n\tprint(c)\n\nwhile c > 0:\n\tn = c % 10\n\tif n > 5:\n\t\tprint(n)\n\tc %= 10
\n'
'if a > b:\n    print(c)\n\nwhile c > 0:\n    n = c % 10\n    if n > 5:\n        print(n)\n
c %= 10    \n'
```