**Лекция 9. Языки программирования**

1. **Сущность, понятия**
2. **Язык программирования Python**
3. **Арифметические операторы**

**4. Условия**

# 5. Циклы

# Алгоритмы

# 7.Списки

**8. Строки и функции**

1. **Сущность, понятия**

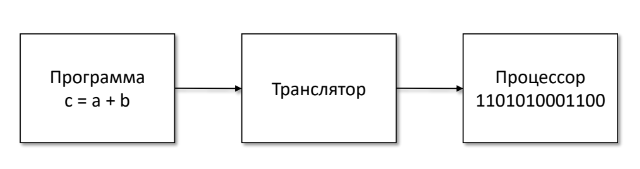
Все языки программирования делятся на языки **низкого** и **высокого** уровня.

К**языкам низкого уровня** относятся языки Ассемблера, они представляют собой символьную интерпретацию машинного кода. Мы знаем, что процессор компьютера обрабатывает информацию в двоичном виде. Все инструкции, как и данные, кодируются с помощью единиц и нулей и любая программа в конечном счете превращается в машинный код. Однако, для человека такой способ записи крайне неудобен, поэтому для удобства восприятия все машинные команды получили символьные обозначения, понятные человеку: например, ADD – это сложить, а SUB – вычесть, CMP – сравнить. Данные извлекаются из регистров процессора или ячеек оперативной памяти по значению адреса.

**Языки высокого уровня** отличаются большей наглядностью и понятностью. В них используются целые слова – операторы и привычные нам знаки арифметических операций. В итоге программа больше похожа на текст, чем на код. Часто можно понять суть программы, просто прочитав её.

В языках низкого уровня нам нужно знать систему команд процессора, структуру его регистров и оперативной памяти, знать принципы адресации и взаимодействовать с компьютером на его языке. Языки высокого уровня по уровню абстракции ближе к человеческому восприятию, нежели к компьютерному. Вы просто пишете команду, но уже не задумываетесь, как она будет выполняться на уровне процессора и памяти.

Для того, чтобы компьютер понял ваш высокоуровневый код, ему потребуется переводчик. Специальные программы – **трансляторы** – переводят исходный код программы на высокоуровневом языке в машинный код, понятный процессору.



Трансляторы делятся на два типа – это **компиляторы** и **интерпретаторы**.

**Компилятор**, подобно переводчику текста, статьи или книги, выполняет чтение и проверку всего кода программы от начала до конца. При обнаружении ошибок в коде компиляция считается неуспешной и программу не поучится запустить. **Интерпретаторы** можно сравнить с синхронным переводом: проверив одну строчку кода, интерпретатор тут же её выполняет. Выполнение программы прервётся, если в какой-либо строчке будет обнаружена ошибка.

Также языки программирования классифицируются по **парадигмам**. Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ. Выделяют следующие парадигмы программирования:

* Императивное программирование
* Процедурное программирование
* Объектно-ориентированное программирование
* Декларативное программирование
* Функциональное программирование
* Логическое программирование

Для **императивного программирования** характерна запись программы в виде последовательно выполняемых инструкций (команд). К нему обычно относят процедурные и объектно-ориентированные языки. В **процедурном программировании** программа состоит из шагов, которые могут объединяться в более крупные структурные единицы - подпрограммы. К таким языкам относятся Basic, C, Pascal. В **объектно-ориентированной** парадигме программы представляются в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы могут наследовать друг друга. Считается, что объектно-ориентированное программирование наиболее абстрактное и приближенное к мышлению человека.

В **декларативном программировании** задаётся спецификация решения задачи, то есть описывается, что представляют собой проблема и ожидаемый результат. К подвидам декларативного программирования также зачастую относят **функциональное** (процесс вычисления трактуется как вычисление значений функций в математическом понимании) и **логическое программирование**(основано на теории и аппарате математической логики с использованием математических принципов резолюций).

**2. Язык программирования Python**

Одним их ярких представителей объектно-ориентированных языков является язык Python.

Python стремится к более простому, менее громоздкому синтаксису и грамматике, предоставляя разработчикам выбор в их методологии кодирования. Python придерживается философии «должен существовать один — и, желательно, только один — очевидный способ сделать это».

Вместо того, чтобы встроить в ядро Python всю функциональность языка, он был спроектирован таким образом, чтобы быть легко расширяемым. Это сделало язык популярным средством добавления программируемых интерфейсов к существующим приложениям. В том числе поэтому язык Python широко применяется в самых различных сферах:

* Web-разработка (YouTube, Instagram, Netflix)
* Создание приложений и игр (GIMP, BitTorrent, World of Tanks)
* Анимация (Disney, Pixar)
* Системное администрирование (Linux)
* Анализ данных (Amazon, Spotify)
* Машинное обучение
* Нейронные сети
* Обработка данных в научных исследованиях (NASA)
* Чат-боты

**Алфавит языка Python** включает в себя набор допустимых символов, которые можно использовать для записи программы:

* латинские прописные и строчные буквы (А, В, С, ..., X, Y, Z, а, b, с, ..., х, у, z);
* арабские цифры (0, 1, 2, ..., 7, 8, 9);
* специальные символы (знак подчёркивания; знаки препинания; круглые, квадратные скобки; знаки арифметических операций, # - знак комментария и др.).

Всего в языке Python более 30 **ключевых слова**:

False - ложь.  
True - правда.  
None - "пустой" объект.  
and - логическое И.  
with / as - менеджер контекста.  
assert условие - вызывает исключение, если условие ложно.  
break - выход из цикла.  
class - пользовательский тип, состоящий из методов и атрибутов.  
continue - переход на следующую итерацию цикла.  
def - определение функции.  
del - удаление объекта.  
elif - в противном случае, если.  
else - иначе  
except - перехватить исключение.  
finally - вкупе с инструкцией try, выполняет инструкции независимо от того, было ли исключение или нет.  
for - цикл for.  
from - импорт нескольких функций из модуля.  
global - позволяет сделать значение переменной, присвоенное ей внутри функции, доступным и за пределами этой функции.  
if - если.  
import - импорт модуля.  
in - проверка на вхождение.  
is - ссылаются ли 2 объекта на одно и то же место в памяти.  
lambda - определение анонимной функции.  
nonlocal - позволяет сделать значение переменной, присвоенное ей внутри функции, доступным в объемлющей инструкции.  
not - логическое НЕ.  
or - логическое ИЛИ.  
pass - ничего не делающая конструкция.  
raise - вызвать исключение.  
return - вернуть результат.  
try - перехватить исключение.  
while - цикл while.  
yield - определение функции-генератора.

Одной из интересных синтаксических особенностей языка является выделение блоков кода с помощью отступов (пробелов или табуляций), поэтому в Python отсутствуют операторные скобки begin/end, как в языке Pascal, или фигурные скобки, как в С. Такой «трюк» позволяет сократить количество строк и символов в программе и приучает к «хорошему» стилю программирования.

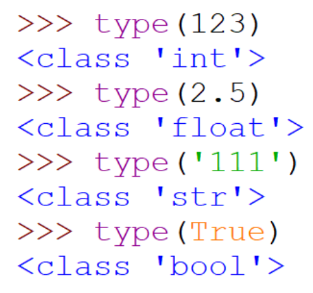
**Типы данных**

Вся информация, обрабатываемая компьютером, хранится в памяти в двоичном виде. Компьютер не различает, чем является та или иная последовательность единиц и нулей. Однако, для нас эта информация может являться числом, текстом, изображением или чем-то другим. При написании программ нам важно понимать, с каким типом информации мы работаем и явно указывать это в коде.

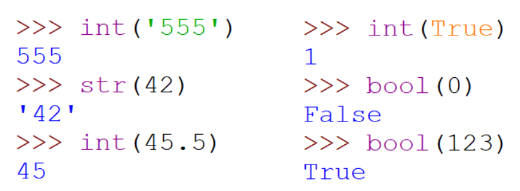
В языке Python мы будем пользоваться четырьмя базовыми типами данных:

* **int** – целое число (может быть как положительным, так и отрицательным) 0 250 -42
* **float** – вещественное число (имеет дробную часть, отделяемую точкой) 5.0 -78.31
* **str** – символьная строка (заключается в одинарные или двойные кавычки, может содержать цифры, буквы, пробелы, различные знаки препинания в формате Unicode, также может быть пустой)  
  ‘hello!’ ‘123’ “”    
  “““many  
  strings”””
* **bool** – логический тип (истина/ложь)True False

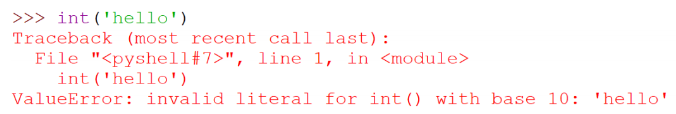
С помощью функции **type** мы можем определить, к какому типу относится то или иное значение. Результатом выполнения функции будет строка, начинающаяся со слова class и затем название типа данных.



Объекты одного типа можно преобразовать в другой, воспользовавшись функциями преобразования. Их легко запомнить, так как названия совпадают с названиями типов. Например, строку, состоящую из цифр, можно преобразовать в целое число с помощью функции int или наоборот, число преобразовать в строку с помощью функции str. При преобразовании вещественного числа в целое, его дробная часть отбрасывается. Также в целый тип легко превратить логические значения, True эквивалентно 1, а False – 0. В логический тип можно преобразовать любое число. Если это не 0, то получим значение True. Аналогично работает преобразование строк в логический тип: пустая строка – это False, не пустая – True.

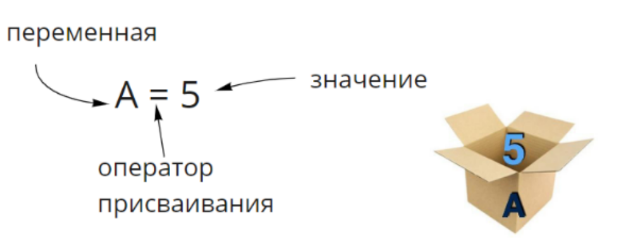


Если преобразовать в числовой тип некорректные данные, например, буквы, результатом выполнения функции будет ошибка ValueError.

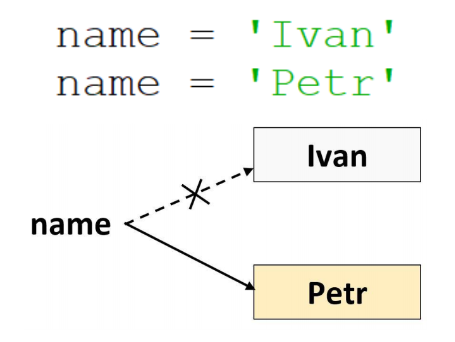


**Переменные**

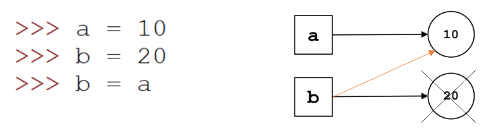
Для хранения значений в оперативной памяти им дают имена. Значение, которому присвоено имя, является **переменной**. Ячейку памяти можно представить себе как коробку, содержимое которой – это её значение, а имя – это уникальная надпись на коробке, позволяющая нам безошибочно найти её среди множества других.



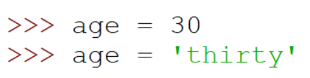
Присваивание имён переменным в языке Python называют связыванием. Имя переменной указывает на значение объекта в памяти. Если связать имя с новым значением, то старое значение будет удалено из памяти.



Аналогично, если в памяти хранились две переменные с именами a и b и мы выполнили действие b = a, то переменная b будет указывать на то же значение, что и а, при этом старое значение b будет удалено.



Также в языке Python переменные не имеют фиксированного типа. Например, изначально переменная age указывала на целочисленное значение. Затем ей присвоили строковое значение. Тип переменной изменился, а имя осталось прежним.



Иногда нам может потребоваться присвоить нескольким переменным одно и то же значение. В этом случае мы может выполнить цепочку присваиваний и указать значение только один раз:

a = b = c = 0

Если нам нужно одновременно присвоить нескольким переменным разные значения, то сначала мы перечисляем через запятую все переменные, затем ставим оператор присваивания, а после в нужном порядке так же через запятую перечисляем значения:

a, b, c = 1, 2, 3

При объявлении переменных очень важно давать им правильные имена. Имя переменной может состоять из латинских букв, цифр и символа нижнего подчёркивания. При этом первый символ не должен быть цифрой, только буквой или нижним подчёркиванием. Имя переменной не может совпадать с ключевым словом, иначе интерпретатор подумает, что перед ним оператор. Используя в названии строчные и прописные буквы, учитывайте, что интерпретатор воспринимает их как разные символы. Если вы назвали переменную маленькой буквой а, то нельзя обращаться к ней, используя заглавную букву A, это разные имена. Также рекомендуется давать переменным такие имена, которые передают смысл хранимых в них значений. Если переменная хранит сумму, то так её и назовите. Или хотя бы буквой с. Это существенно облегчает читаемость кода.

**Функции ввода-вывода**

Кроме присваивания переменным конкретных значений, в программах нам потребуется возможность задавать их значения с клавиатуры и выводить значения на экран. Для этого мы будем пользоваться функциями ввода и вывода.

Функция ввода в языке Python называется **input** и в качестве результата возвращает прочитанное из консоли значение. По умолчанию все прочитанные значения имеют строковый тип. Если же нам нужно прочитать с клавиатуры число, то необходимо добавить функцию int или float.

a = input() #ввод строки

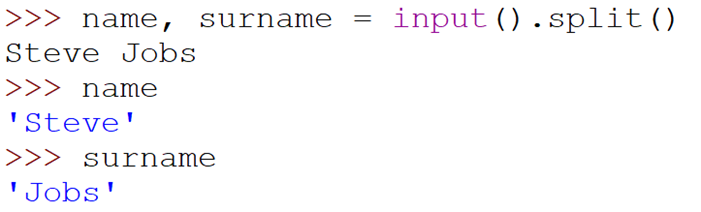
b = int(input()) #ввод целого числа

c = float(input()) #ввод вещественного числа

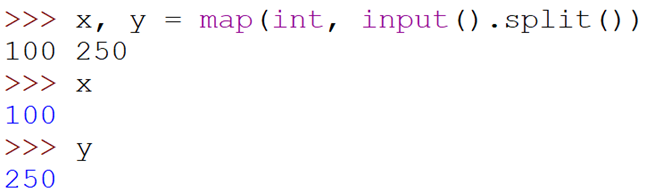
В качестве параметров функции input мы можем по желанию указать текстовую подсказку, которая будет выводиться слева от курсора (подсказку необходимо указывать в кавычках).

x = int(input('Введите число: '))

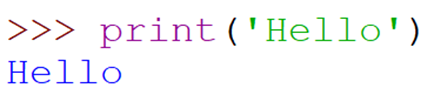
Часто несколько значений вводятся в одной строке и разделяются пробелом. Для того, чтобы присвоить каждое значение отдельной переменной, воспользуйтесь методом **split**. Он вызывается через точку после вызова функции input. Слева от оператора присваивания нужно перечислить через запятую имена переменных в том порядке, в котором необходимо присвоить прочитанные значения.



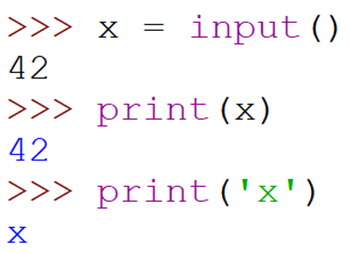
Если значения имеют числовой тип, то для преобразования нам понадобится функция **map**. Первым её параметром мы укажем, в какой тип нужно преобразовать каждое из считанных значений, а вторым параметром – саму считанную строку, разделенную по пробелам.



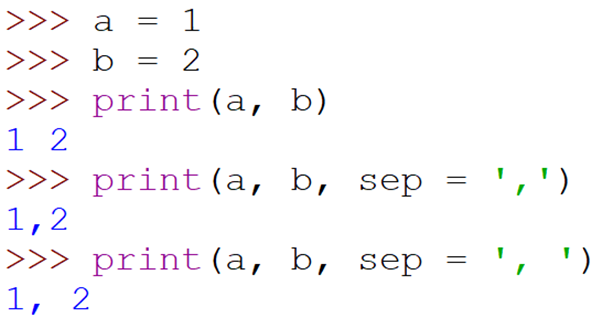
Для вывода в консоль будем использовать функцию **print**. Она позволяет выводить как константы, так и значения переменных. Чтобы вывести любой текст, поместите его в кавычки.



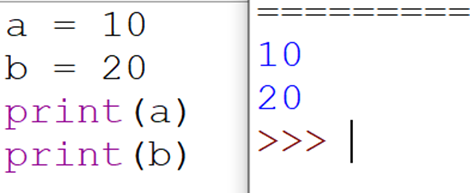
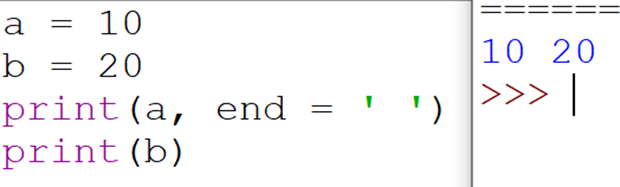
Для вывода значения переменной, укажите её имя без кавычек. Обратите внимание, что если указать имя переменной в кавычках, то будет выведена буква, а не её значение.



С помощью одной функции print можно выводить сразу несколько значений. Если перечислить их через запятую, то они будут выведены в указанном порядке и разделены пробелами. Если в качестве разделителя требуется другой символ, то он указывается с помощью именованного параметра **sep**.



Если же выводить несколько значений отдельными вызовами функции print, то по умолчанию каждое из них будет выведено с новой строки. Убрать перенос можно с помощью параметра **end**. Если задать ему значение, отличное от \n, то выводимые переменные будут расположены иначе, например, в одной строке через пробел.

### Примеры задач

**Пример 1.** В строке вводятся фамилия, имя и возраст через пробел. Прочитать эти данные и вывести каждое значение в отдельной строке с пояснением.

**Входные данные:**  
Александр Иванов 33

**Выходные данные:**  
Имя: Александр  
Фамилия: Иванов  
Возраст: 33

Решение:

Сначала прочитаем строку текста с помощью функции input(), а затем разделим её на части с помощью метода split(). Результат разделения запишем в 3 переменные: name (имя), surname (фамилия) и age (возраст).

name, surname, age = input().split()

Наконец, выведем каждую переменную в отдельной строке с пояснением:

print('Имя:', name)

print('Фамилия:', surname)

print('Возраст:', age)

**Пример 2.**Вывести на экран квадрат размером 4х4, состоящий из определённого символа, введённого с клавиатуры.

**Входные данные:**  
s

**Выходные данные:**  
ssss  
s  s  
s  s  
ssss

Решение:

Сначала прочитаем символ и запишем его в переменную с.

c = input()

Затем выведем 4 отдельных строки: первая и последняя должны состоять из 4 символов c, а вторая и третья - из символа с, двух пробелов и снова символа с. В каждой функции вывода параметр sep должен равняться пустой строке, чтобы между символами не было лишних пробелов. Получим:

print(c,c,c,c,sep='')

print(c,' ',' ',c,sep='')

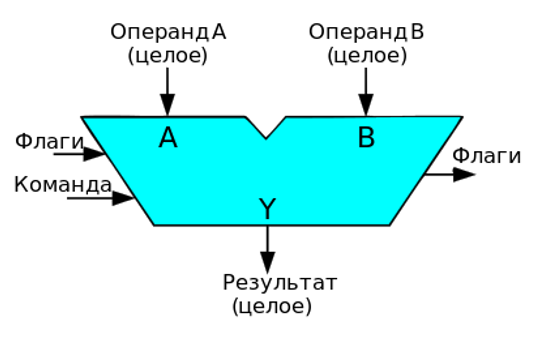
print(c,' ',' ',c,sep='')

print(c,c,c,c,sep='')

**3. Арифметические операторы**

Выполнение арифметических действий – это основное назначение процессора компьютера. Изначально компьютер предназначен именно для выполнения вычислений. И даже если вы редактируете изображение на мониторе, для процессора это превращается в операции сложения, вычитания, умножения или деления, применяемые к двоичным числам.

Основной вычислительный блок процессора – это АЛУ, арифметико-логическое устройство. Оно принимает на вход до двух значений операндов, а также код команды, которую необходимо выполнить. После выполнения команды, мы получаем на выходе результат вычислений.



В языке Python используются следующие арифметические операторы:

|  |  |
| --- | --- |
| + | сложение |
| - | вычитание |
| \* | умножение |
| / | деление |
| // | деление нацело |
| % | остаток от деления |
| \*\* | возведение в степень |

Арифметические операции имеют следующий приоритет выполнения:

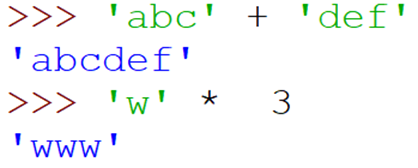
1. Скобки
2. Степень
3. Унарный минус
4. Умножение, деление
5. Сложение, вычитание

Часто в арифметических выражениях встречаются круглые скобки, они определяют, какое действие следует выполнить в первую очередь. Далее по приоритету идёт операция возведения в степень, затем оператор минус, меняющий знак числа, в том случае, если он стоит не между двух операндов, а перед одним. Все операции деления и умножения выполняются раньше, чем сложение и вычитание. Операции одного порядка при отсутствии скобок выполняются по порядку слева направо.

В тех случаях, когда к исходной переменной мы добавляем или вычитаем или выполняем другую арифметическую операцию со вторым числом и результат вычислений при этом записывается с первую переменную, возникает дублирование одной и той же переменной в левой и в правой частях выражения. Этого дублирования можно избежать, если вынести знак арифметической операции вперёд и поставить его перед оператором присваивания. Такие операторы называются **сокращенными** и делают код более компактным.

|  |  |
| --- | --- |
| a = a + b | a += b |
| a = a – b | a -= b |
| a = a \* b | a \*= b |
| a = a / b | a /= b |
| a = a // b | a //= b |
| a = a % b | a %= b |

Несмотря на то, что арифметические операторы в основном предназначены для работы с числами, некоторые из них можно использовать и со строками. Оператор сложения выполняет конкатенацию строк, то есть соединяет их в одну строчку. Порядок сложения строк имеет значение, если записать слагаемые наоборот, то и результат получится другой. Также строки можно умножать на целые числа. При умножении на положительное число происходит дублирование строки указанное количество раз. При умножении на 0 или на отрицательное число, всегда получаем пустую строку.



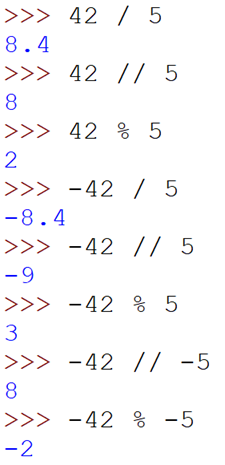
 Рассмотрим более подробно операторы деления в языке Python. Из математики нам известно следующее:



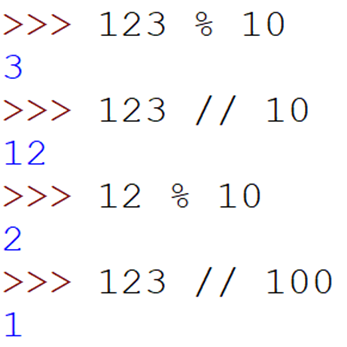
Для любого целого числа справедливо соотношение:

частное \* делитель + остаток = делимоечастное∗делитель+остаток=делимое

Поэтому операторы /, // и % для различных целых чисел (как положительных, так и отрицательных) будут давать следующие результаты:



Одним из основных применений целочисленного деления в алгоритмах можно назвать выделение разрядов числа. Например, для любого десятичного числа справедливо утверждение, что остаток от деления на 10 – это его младший разряд, а частное – это все цифры без младшего разряда. Последовательное применение операций деления и деления с остатком позволит выделить все разряды из числа.



### Примеры задач

**Пример 1.** С клавиатуры вводится трёхзначное число. Вывести сумму его цифр.

**Входные данные:**  
512

**Выходные данные:**  
8

**Решение:**

Выделим каждый разряд числа с помощью операторов целочисленного деления и остатка, затем сложим полученные разряды и выведем на экран:

x = int(input())

sot = x // 100

des = x // 10 % 10

ed = x % 10

print(sot + des + ed)

**Пример 2.** С клавиатуры вводится три цифры, каждая в отдельной строке. Составить из этих цифр трёхзначное число и вывести на экран квадрат этого числа.

**Входные данные:**  
1  
0  
5

**Выходные данные:**  
11025

Решение:

Составим из трёх разрядов число х: сотни умножим на 100, десятки - на 10 и прибавим к ним количество единиц. Полученное число возведем во вторую степень и выведем на экран.

a = int(input())

b = int(input())

c = int(input())

x = a \* 100 + b \* 10 + c

print(x \*\* 2)

**Пример 3.** Продолжительность фильма t минут. Выведите продолжительность фильма в формате часы:минуты. Количество минут должно быть двухразрядным.

**Входные данные:**  
125

**Выходные данные:**  
2:05

**Решение:**

Т.к. в 1 часе 60 минут, найдем количество часов с помощью целочисленного деления на 60, а количество минут как остаток от деления на 60.

t = int(input())

h = t // 60

m = t % 60

print(h, ':', m // 10, m % 10, sep = '') #выводим минуты порязрядно: сначала кол-во десятков,

#потом кол-во единиц, чтобы получить двухразрядное число

**Пример 4.** С клавиатуры вводятся 3 числа в следующем порядке: в первой строке количество пирожков, затем во второй строке стоимость пирожка (сначала рубли, потом копейки через пробел). Вывести общую стоимость всех пирожков.

**Входные данные:**  
5  
12 80

**Выходные данные:**  
64 руб. 0 коп.

**Решение:**

Найти общую стоимость всех пирожков будет удобнее, если перевести стоимость одного пирожка в копейки, умножить на их количество и затем перевести обратно в рубли и копейки с помощью целочисленного деления и остатка от деления на 100.

k = int(input())

rub, kop = map(int, input().split())

itog = (rub \* 100 + kop) \* k #итоговая стоимость k пирожков в копейках

print(itog // 100, 'руб.', itog % 100, 'коп.')

Практическое задание

* 1. **С клавиатуры вводятся два целых числа (в строку через пробел). Найти их сумму и вывести на экран выражение, как показано в примере.**

**Sample Input:**

12 36

**Sample Output:**

12 + 36 = 48

* 1. **С клавиатуры вводится четырёхзначное натуральное число. Вывести на экран произведение его цифр.**

**Sample Input:**

1984

**Sample Output:**

288

* 1. **С клавиатуры вводится трёхзначное число. Вывести это число задом-наперёд (единицы должны стать сотнями, а сотни - единицами)**

**Sample Input:**

451

**Sample Output:**

154

4. **Вводятся два трёхзначных числа. Составить из них три двузначных числа по следующему принципу: первое число состоит из цифр сотен первого и второго числа, второе - из цифр десятков, третье - из единиц. Затем найти сумму полученных чисел. На экран вывести квадратный корень из вычисленной суммы.**

Например, даны числа 123 и 456. Из них составляем числа 14, 25 и 36. Сумма этих чисел равна 14 + 25 + 36 = 75, квадратный корень из этого числа 8.660254037844387

**Sample Input:**

123

456

**Sample Output:**

8.660254037844387

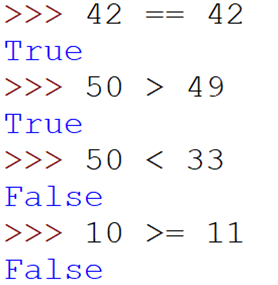
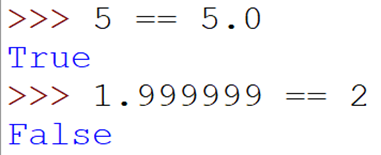
**4. Условия**

Условия в программировании связаны с логикой. Мы уже знаем, что любое логическое высказывание может быть истинным или ложным. В зависимости от этого можно управлять ходом программы и выбирать, какое действие выполнится в каком случае. Высказывания в языке Python имеют тип bool и могут принимать значения True или False.

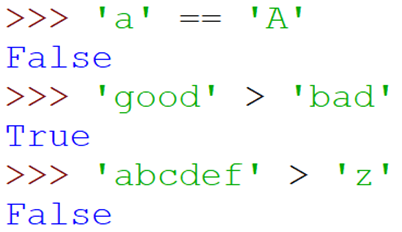
Простые высказывания в программе могут состоять из переменных, констант, арифметических операторов и обязательно содержат **операторы сравнения**. Они обозначаются следующим образом:

|  |  |
| --- | --- |
| == | равно |
| != | не равно |
| > | больше |
| < | меньше |
| >= | больше или равно |
| <= | меньше или равно |

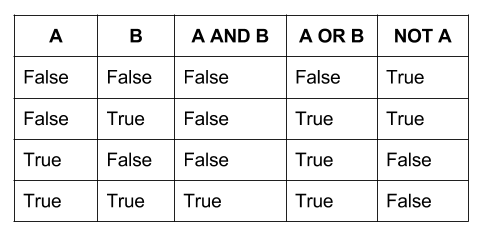
Результатом сравнения будет истина или ложь. При сравнении целых и вещественных чисел нужно учесть, что целое значение 5 и вещественное 5.0 – это одно и то же значение, результат их сравнения – истина, но число 1.99999 не равно числу 2, хоть и близко к нему.

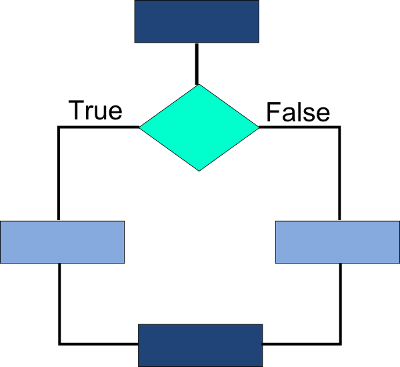
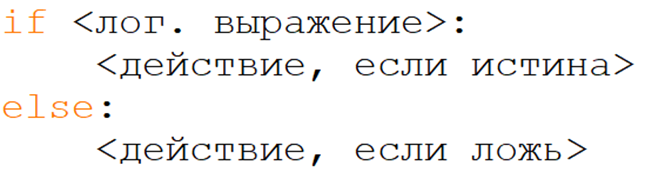
Сравнение строк осуществляется по кодам символов. У заглавной и строчной букв а разные коды, поэтому они не равны друг другу. В сроках 'good' и 'bad' первые буквы соотносятся так, что g больше чем b, т.к. расположена дальше по алфавиту, значит её код больше и вся строка будет больше второй. Это никак не связано с длинами строк. Например, строка 'abcdef' не будет больше одной буквы 'z' только из-за того, что буква a меньше буквы z, длина строки здесь не играет роли.



Кроме простых высказываний мы можем составлять и сложные с использованием **логических операторов**. В Python они обозначаются ключевыми словами and, or и not.



Сами по себе логические выражения редко используются в программах, как правило они являются частью **условного оператора**. После проверки истинности логического выражения мы можем сделать выбор. Если выражение оказалось истинным, то выполним одно действие, а в противном случае – другое.

Условный оператор начинается с ключевого слова if после которого следует логическое выражение. Логическое выражение может состоять из переменных, констант, арифметических операций, операторов сравнения и логических операторов. При необходимости для обеспечения нужного приоритета выполнения операторов ставятся круглые скобки. В конце логического выражения обязательно ставится двоеточие. Следующий блок операторов пишется на следующей строке, причем правее, чем остальной код. Такой отступ необходим, чтобы показать вложенность оператора в условный. Стандартный размер такого отступа составляет 4 пробела. Если действий выполняется несколько, то все они идут с одинаковым отступом. Действие, выполняемое при ложном значении выражения, записывается после оператора else. За ним также следует двоеточие и вложенные операторы идут с отступом в 4 пробела.

Оператор else не является обязательным и в некоторых случаях условие может быть записано и без него. Такое условие называют **неполным ветвлением**. Данные фрагменты кода выводят наибольшее из двух чисел.

#полное ветвление

if a > b:

print(a)

else:

print(b)

#неполное ветвление

m = b

if a > m:

m = a

print(m)

Также условия можно размещать внутри других условий, т.е. делать их **вложенными**. Данный фрагмент кода находит  наибольшее из трёх чисел и запоминает его значение в переменной m.

#вложенные условия

if a > b:

if a > c:

m = a

else:

m = c

else:

if b > c:

m = b

else:

m = c

В некоторых случаях в каждом операторе else может находиться следующий if. Например, если мы хотим по номеру дня определить, какой сейчас день недели и вывести его название. Начинаем проверять с единицы, если это не 1, то сравниваем с двойкой и так далее. Недостатком такой организации вложенных условий является то, что каждый следующий if идёт с отступом правее предыдущего и при большом количество условий такая конструкция сильно смещается вправо.

if day == 1:

print('Monday')

else:

if day == 2:

print('Tuesday')

else:

if day == 3:

print('Wednesday')

else:

if day == 4:

...

Разработчики языка учли этот момент и специально для таких случаев, когда внутри else идёт следующий if, можно объединить два оператора в один, получится elif. Таких операторов в условии может быть сколь угодно много и дополнительные отступы не требуются. Называется эта конструкция **каскадным условием**. Начинается она с оператора if, затем нужное количество elif, в конце может присутствовать единственный оператор else.

#каскадные условия

if day == 1:

print('Monday')

elif day == 2:

print('Tuesday')

elif day == 3:

print('Wednesday')

elif day == 4:

...

else:

print('There are only 7 days in a week')

Возможна также последовательная проверка нескольких сложных условий. Например, вы вводите натуральное число и хотите определить его разрядность. Это можно реализовать следующим образом:

if a >= 10 and a <= 99:

print('Число двухзначное')

if a >= 100 and a <= 999:

print('Число трёхзначное')

if a >= 1000 and a <= 9999:

print('Число четырёхзначное')

В этом случае гарантированно выполнится только один из условных операторов или ни один из них. Невозможно выполнение двух и более условий одновременно.

Итак, в этом уроке уроке мы научились использовать условный оператор if, познакомились с его разновидностями и синтаксическими особенностями.

**Примеры задач**

**Пример 1.**C клавиатуры вводится целое число. Определить, является ли оно чётным или нечётным

**Входные данные:**  
42

**Выходные данные:**  
Чётное

**Решение:**

Число считается чётным, если не даёт остатка при делении на 2. Найдём этот остаток и проверим, равен ли он нулю. В противном случае число является нечётным.

x = int(input())

if x % 2 == 0:

print('Чётное')

else:

print('Нечётное')

**Пример 2.**С клавиатуры вводятся 2 числа в строчку через пробел. Сравнить их и вывести сообщение. Если числа равны, то написать, что они равны.

**Входные данные:**  
33 25

**Выходные данные:**  
33 больше 25

Решение:

Для сравнения чисел воспользуемся каскадными условиями и последовательно проверим все варианты.

a, b = map(int, input().split())

if a > b:

print(a, 'больше', b)

elif a < b:

print(a, 'меньше', b)

else:

print(a, 'равно', b)

**Пример 3.**С клавиатуры вводится два числа и арифметический оператор (каждое значение с новой строки). Вычислить значение выражения.

**Входные данные:**  
50  
+  
30

**Выходные данные:**  
80

**Решение:**

Прочитаем с клавиатуры входные данные в указанном порядке, затем в зависимости от значения оператора выполним нужное арифметическое действие. Предусмотрим также ситуацию, когда выполняется деление и второе число равно 0: в этом случае необходимо вывести сообщение об ошибке. Также выводим сообщение об ошибке, если введённый оператор не является ни одним из известных.

a = int(input())

op = input()

b = int(input())

if op == '+':

print(a + b)

elif op == '-':

print(a - b)

elif op == '\*':

print(a \* b)

elif op == '/':

if b != 0:

print(a / b)

else:

print('На ноль делить нельзя')

else:

print('Некорректный оператор')

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. **Смоделируем процесс регистрации пользователя в системе. С клавиатуры вводятся логин пользователя. Затем в следующей строке вводится пароль. В третьей строке пользователь вводит пароль повторно. Необходимо убедиться, что пароли совпадают. Если пароли одинаковые, то вывести "Вы зарегистрированы", в противном случае: "Пароли не совпадают".**

**Sample Input 1:**

admin

8765asdf

8765asdf

**Sample Output 1:**

Вы зарегистрированы

**Sample Input 2:**

user

p@$$w0rD

paSSw0rD

**Sample Output 2:**

Пароли не совпадают

### Напишите программу.

1. **Теперь выполним авторизацию пользователя. Добавьте к имеющейся программе код, позволяющий пользователю войти в систему, используя указанный логин и пароль. В зависимости от введённых значений, выведите сообщения: "Пользователь с таким именем не зарегистрирован", "Неверный пароль" или "Вы вошли в систему".**

**Sample Input 1:**

admin

p@$$w0rD

**Sample Output 1:**

Вы вошли в систему

**Sample Input 2:**

admin

8765fdsa

**Sample Output 2:**

Неверный пароль

**Sample Input 3:**

user

8765asdf

**Sample Output 3:**

Пользователь с таким именем не зарегистрирован

### Напишите программу.

1. **С клавиатуры вводится номер месяца. Вывести на экран название времени года, к которому относится этот месяц. Если номер месяца меньше 1 или больше 12, то вывести сообщение об ошибке "Неверный номер месяца".**

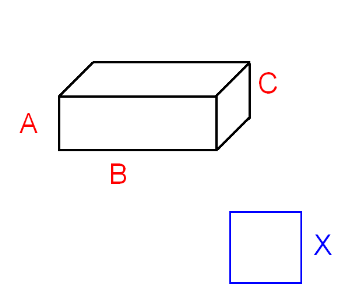
**Sample Input:**

3

**Sample Output:**

Весна

1. **С клавиатуры вводятся размеры кирпича A, B, C и сторона квадратного отверстия X (A, B, C, X - целые числа). Определить, пройдёт ли кирпич в отверстие. Вывести ответ в формате Да/Нет.**



**Sample Input 1:**

3

2

5

4

**Sample Output 1:**

Да

**Sample Input 2:**

8

5

5

5

**Sample Output 2:**

Нет

### Напишите программу.

1. **Дано трёхзначное положительное число. Определить, содержится ли в этом числе хотя бы одна цифра 1. Вывести ответ в формате Да/Нет.**

**Sample Input:**

256

**Sample Output:**

Нет

### Напишите программу.

# 5. Циклы

### Цикл while

Если мы хотим реализовать алгоритм обратного отсчета, который выводит на экран числа 10, 9, 8 и так далее до 1, то нам придется выводить каждое число с помощью отдельной функции print.

n = 10

print(n)

print(n - 1)

print(n - 2)

print(n - 3)

print(n - 4)

print(n - 5)

print(n - 6)

print(n - 7)

print(n - 8)

print(n - 9)

Но если нам понадобится увеличить количество чисел, то размер программы существенно возрастёт. Чтобы избежать такого громоздкого кода с повторяющимися командами, можно повторять любое действие в программе, пока выполняется условие цикла. Это записывается с помощью оператора цикла **while**:

n = 10

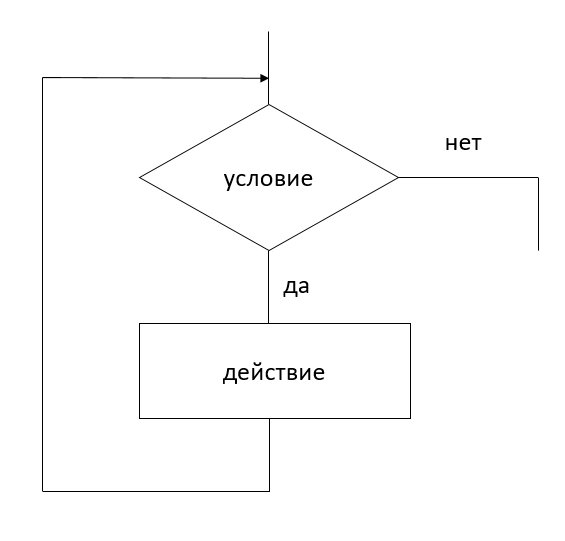
while n > 0:

print(n)

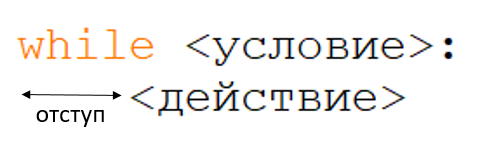
n = n - 1

До тех пор, пока наше число n больше нуля, мы будем выводить его текущее значение, а затем уменьшать на единицу. После того, как мы выведем число 1, значение n после вычитания станет равно 0 и цикл остановится. На экране мы увидим все числа от 10 до 1.

Цикл представляет собой ветвление с обратной связью. Если условие выполнилось, то мы заходим в цикл и выполняем действие, после чего возвращаемся на проверку условия. До тех пор, пока логическое выражении в условии истинно, тело цикла будет выполняться. Как только условие станет ложным, выполнение цикла прекратится.

блок-схема цикла while

На языке Python такая конструкция начинается с ключевого слова while. После него записывается условие, затем ставится двоеточие и на следующей строке с отступом в 4 пробела записываются действия, повторяемые в цикле.



Иногда нам может понадобиться прервать цикл до того, как он завершится сам. Описать это можно с помощью оператора **break**. Он помещается внутрь условного оператора и выполняется в момент достижения условия. Например, в данном случае цикл не досчитает до нуля, а остановится на значении 13.

a = 100

while a > 0:

a = a - 1

if a == 13:

break

print(a)

Второй оператор, который действует в обход основного условия цикла – это оператор **continue**. Он позволяет пропустить текущую итерацию и перейти сразу к следующей. Оператор continue также помещается внутрь условия. В этом примере цикл пропустит значение 13 и не выведет его на экран. Все остальные числа будут выведены в порядке убывания, начиная со ста и до нуля.

a = 100

while a > 0:

a = a - 1

if a == 13:

continue

print(a)

Также в циклах может использоваться оператор else. Он выполняется в том случае, когда цикл завершился естественным образом, в тот момент, когда условие цикла стало ложным. Часто имеет смысл использовать else совместно с оператором break.

В данном случае отсчёт начинался с 12-ти и условие выхода из цикла по оператору break не выполнилось. Цикл завершился в тот момент, когда х стал равен нулю. После завершения цикла мы попадаем в блок оператора else и выводим сообщение The end:

x = 12

while x > 0:

print(x)

x = x - 1

if x == 13:

break

else:

print('The end')

А здесь программа распечатает числа 15 и 14, а на значении 13 мы выйдем из цикла. В этом случае блок оператора else не выполняется, так как из цикла мы вышли преждевременно:

x = 15

while x > 0:

print(x)

x = x - 1

if x == 13:

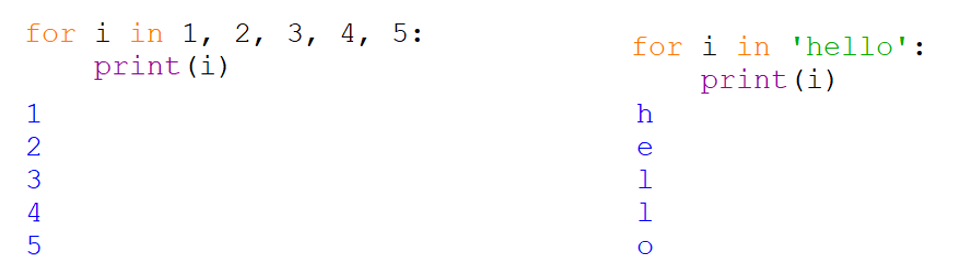
break

else:

print('The end')

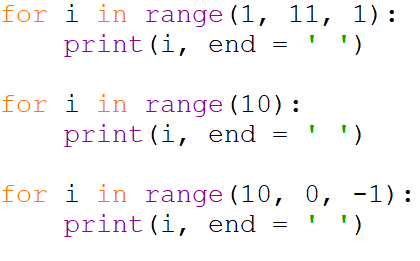
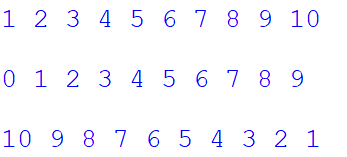
### Цикл for

Когда цикл должен отсчитывать повторения по заранее определённой последовательности, мы можем явно перечислить все значения и порядок, в котором они должны меняться. Для этого используется оператор **for**. Мы выбираем переменную и значения, которые она будет принимать. Например, переменная i принимает значения от 1 до 5. Или каждый раз принимает значение определённого символа.



Для того, чтобы задать определённое количество шагов для цикла for удобно использовать функцию **range**. Это функция, генерирующая арифметические прогрессии с заданным первым и последним элементом, а также шагом прогрессии. Все параметры должны быть целыми числами. Конец диапазона по умолчанию не включается в генерируемую последовательность. Если шаг положительный, то start < stop. При отрицательном шаге start > stop. Если не указать стартовое значение, то по умолчанию оно будет равно 0. Если не указать шаг, то он принимается за 1.

Если задать диапазон от 1 до 11 не включительно с шагом 1, то получим числа от 1 до 10. Если указать в параметрах функции range только одно число, то это будет значение stop. Начинаться диапазон будет с 0 и идти с шагом 1. Получим числа от 0 до 9, 10 не включительно. Диапазон от 10 до нуля с шагом -1 даст нам ряд чисел 10, 9, 8 и так далее до 1.

Итак, в этом уроке мы рассмотрели принцип работы циклов while и for, также узнали об операторах break и continue, научились определять количество итераций цикла и как распознать бесконечный цикл.

### Примеры задач

**Пример 1.** Вывести на экран все чётные числа в диапазоне от 2 до 100 в одну строку через пробел.

**Выходные данные:**  
2 4 6 8 10 12 14 16 ... 92 94 96 98 100

Решение:

Воспользуемся оператором for. В качестве начального значения в функции range укажем 2, в качестве конечного - 101, т.к. нам необходимо вывести числа до 100 включительно, и в качестве шага - 2. Для того, чтобы числа выводились в строку через пробел, укажем значение параметра end в функции print.

for i in range(2, 101, 2):

print(i, end = ' ')

**Пример 2.** Вывести на экран все числа, кратные 5, в диапазоне от a до b (вводятся с клавиатуры)

**Входные данные:**  
2 40

**Выходные данные:**  
5 10 15 20 25 30 35 40

Решение:

Одним из вариантов решения является цикл for с проверкой каждого числа на кратность пяти.

a, b = map(int, input().split())

for i in range(a, b + 1):

if i % 5 == 0:

print(i, end = ' ')

Возможно также решение данной задачи без использования оператора if. Если мы начнём цикл с ближайшего к а числа, кратного 5, то сможем реализовать цикл for с шагом 5 и вывести все подходящие числа. Предположим, а = 2. Ближайшее кратное 5 - это число 5. Мы получим 5, если вычтем из 5 остаток от деления а на 5, а затем прибавим эту разницу к числу а. Если же а уже кратно 5, то необходимо оставить это число без изменений. Для этого снова возьмем остаток от деления на 5 для полученной разницы, чтобы в случае, когда мы получили 5 - 0 = 5, превратить 5 в 0.

a, b = map(int, input().split())

for i in range(a + ((5 - a % 5) % 5), b + 1, 5):

print(i, end = ' ')

**Пример 3.** В первой строке вводится размер числовой последовательности N, а затем N целых чисел (каждое число с новой строки). Найти сумму всех этих чисел.

**Входные данные:**  
5  
56  
1  
23  
409  
52

**Выходные данные:**  
541

Решение:

Обозначим переменной s сумму чисел последовательности, n - количество чисел в последовательности и а - текущее число. Обнулим значение суммы и прочитаем с клавиатуры длину последовательности n. Далее n раз повторим ввод числа а и будем прибавлять каждое новое число к общей сумме с помощью оператора +=. После завершения цикла выведем накопленную сумму на экран.

s = 0

n = int(input())

for i in range(n):

a = int(input())

s += a

print(s)

**Пример 4.** С клавиатуры вводится последовательность целых чисел (каждое число с новой строки), до тех пор пока не будет введён 0 (признак окончания последовательности, количество  
чисел заранее неизвестно). Вывести наибольшее число в этой последовательности.

**Входные данные:**  
5  
21  
89  
456  
76  
512  
54  
511  
0

**Выходные данные:**  
512

Решение:

Обозначим переменной а текущее число последовательности, а переменной m - наибольшее из чисел. Прочитаем с клавиатуры первое число и примем его за максимальное. Затем будем сравнивать текущее число с максимальным и обновлять значение а до тех пор, пока оно не станет равным 0. Каждое новое значение, превышающее прежнее значение максимума, становится новым максимумом. При вводе нуля цикл остановится и мы выведем найденный результат.

a = int(input())

m = a

while a != 0:

if a > m:

m = a

a = int(input())

print(m)

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. Напишите программу, которая выводит на экран последовательность чисел 3 7 11 15 19 ... 63 в строчку через пробел.

**Sample Input:**

**Sample Output:**

3 7 11 15 19 23 27 31 35 39 43 47 51 55 59 63

1. С клавиатуры вводится целое число N (от 1 до 9). Вывести на экран таблицу умножения числа N на все числа от 1 до 10.

**Sample Input:**

2

**Sample Output:**

2\*1 = 2

2\*2 = 4

2\*3 = 6

2\*4 = 8

2\*5 = 10

2\*6 = 12

2\*7 = 14

2\*8 = 16

2\*9 = 18

2\*10 = 20

### Напишите программу.

1. С клавиатуры вводятся числа a и b. Вывести на экран все нечётные числа, принадлежащие отрезку [a, b], причём, если a < b, то вывести числа по возрастанию, если a > b, то по убыванию. Гарантируется, что a не будет равно b.

**Sample Input 1:**

2

11

**Sample Output 1:**

3 5 7 9 11

**Sample Input 2:**

31

20

**Sample Output 2:**

31 29 27 25 23 21

### Напишите программу.

1. С клавиатуры вводится сначала количество чисел в последовательности, а затем сами числа (все числа целые, каждое число в отдельной строке). Найти среднее арифметическое всех чётных чисел (сумму всех чётных поделить на количество всех чётных). Гарантируется, что в последовательности будет хотя бы одно чётное число.

Например, для входной последовательности из 5 чисел

12  
17  
94  
116  
911

чётными являются 12, 94 и 116. Их сумма равна 222, а количество равно 3. 222 / 3 = 74

**Sample Input:**

5

12

17

94

116

911

**Sample Output:**

74.0

### Напишите программу.

1. С клавиатуры вводится последовательность целых чисел неизвестного размера, признаком окончания последовательности является число 0 (в последовательность не входит). Найти наименьшее трёхзначное число в этой последовательности. Если трёхзначных чисел нет, то вывести NO.

**Sample Input:**

-120

31

456

9

-250

-180

0

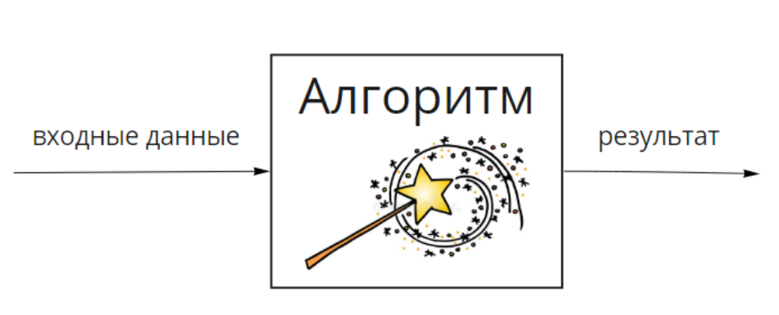
**Sample Output:**

-250

### Напишите программу.

# Алгоритмы

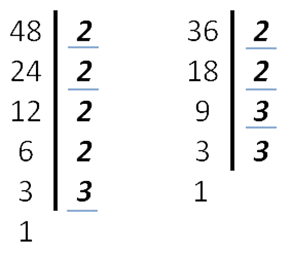
Алгоритмом называют некоторую последовательность действий, которая преобразует входные данные в результат. В большинстве программ мы сначала вводим данные, затем происходят определенные преобразования, после которых мы получаем нужный нам результат и выводим его.



Мы рассмотрим реализацию таких алгоритмов, как поиск наибольшего общего делителя двух натуральных чисел, алгоритмы обработки цифр натурального числа и алгоритмы обработки числовых последовательностей.

### Алгоритм поиска наибольшего общего делителя

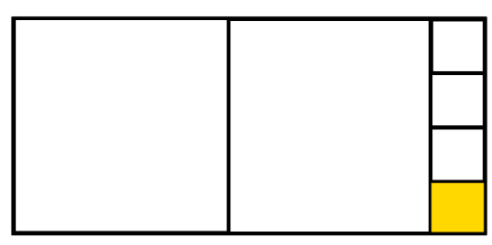
В математике наибольшим общим делителем (НОД) называют наибольшее натуральное число, на которое делятся одновременно два заданных числа. Чтобы его вычислить, мы раскладываем каждое число на простые множители и находим все совпадающие. Произведение этих множителей и даёт нам в итоге значение наибольшего общего делителя.



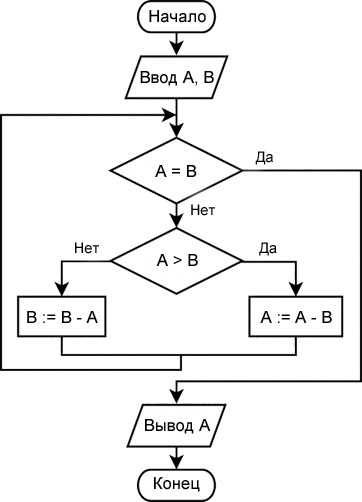
НОД(48, 36) = 2 \cdot 2 \cdot 3 = 12НОД(48,36)=2⋅2⋅3=12

Можно также сформулировать задачу следующим образом: как покрыть минимальным количеством квадратных плиток прямоугольную поверхность, чтобы сторона квадрата была наибольшей? Мы подбираем число, на которое одновременно делятся как длина, так и ширина прямоугольника, чтобы все плитки остались целыми.

Реализацию алгоритма поиска наибольшего общего делителя предложил греческий математик Евклид. Смысл алгоритма заключается в следующем: мы будем последовательно отрезать прямоугольника квадраты, сторона которых равна меньшей из сторон. Если после отрезания таких квадратов у нас получился прямоугольник, то повторяем те же действия для него, пока не останется квадрат. Сторона наименьшего квадрата и будет наибольшим общим делителем.



Записывается такой алгоритм как последовательное вычитание из большего числа меньшего до тех пор, пока числа не сравняются. Реализован он будет с помощью цикла while и условия, проверяющего, какое из двух чисел больше. Из большего мы всегда вычитаем меньшее и запоминаем это значение вместо большего числа.



Приведенный программный код считывает строку из двух чисел, разделённых пробелом, находит их НОД и выводит его на экран.

a, b = map(int, input().split())

while a != b:

if a > b:

a = a - b

else:

b = b - a

print(a)

Недостатком предыдущего алгоритма является то, что при большой разнице между исходными числами, количество вычитаний может быть очень велико. Если попытаться найти НОД(100000000000, 2), то прежде чем они уравняются, вам придётся выполнить вычитание 5 миллиардов раз. Даже у современного компьютера на выполнение такого количества итераций уйдёт порядка 15 минут. Ускорить работу алгоритма можно, если заменить операцию вычитания на деление с остатком.

a, b = map(int, input().split())

while a != 0 and b != 0;

if a > b:

a = a % b

else:

b = b % a

print(a + b)

### Алгоритм обработки цифр натурального числа

Процесс последовательного нахождения цифр числа можно реализовать с помощью цикла while и операций деления. Возьмем произвольное натуральное число a. До тех пор, пока оно больше нуля, будем находить очередную цифру, двигаясь справа налево, и запоминать её в переменную d. После того, как текущая цифра обработана, делим исходное число нацело на 10 и повторяем цикл. Когда переменная а станет одноразрядным числом, после деления на 10 получим 0 и цикл завершится.

a = int(input())

while a > 0:

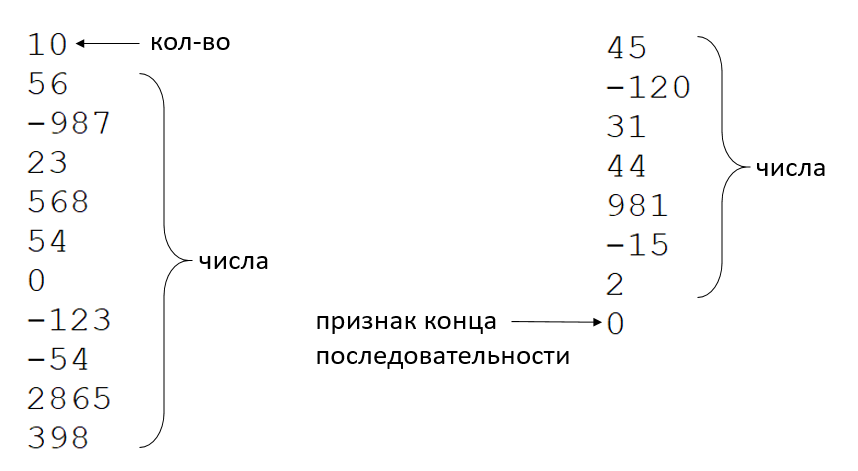
d = a % 10

#обработка текущей цифры

a = a // 10

### Алгоритмы обработки числовых последовательностей

Существует два способа ввода числовой последовательности: первый – это сначала указать точное количество чисел, которое будет вводиться, а затем произвести ввод самих чисел. Второй способ – это вводить числа до тех пор, пока не достигнем признака окончания последовательности. Получается, что заранее мы не знаем, сколько будет вводиться чисел, но как только они закончатся, мы введём особое значение, обозначающее конец последовательности. Часто в задачах признаком конца последовательности служит число 0.



Организовать ввод и обработку последовательности заданной длины можно с помощью цикла for. Сначала вводим n – это размер последовательности (количество вводимых чисел). Затем в цикле n раз мы читаем с клавиатуры в переменную а очередное число и обрабатываем его. При такой организации цикла мы не запоминаем все введённые числа, а помним только одно текущее значение на каждом шаге.

n = int(input())

for i in range(n):

a = int(input())

#обработка числа

Последовательность, оканчивающаяся нулём, может быть считана с помощью цикла while: сначала введём первое число и убедимся, что это не ноль. До тех пор, пока это не ноль, будем обрабатывать введённые числа и затем считывать следующее. Когда будет введён 0, цикл завершится.

a = int(input())

while a != 0;

#обработка числа

a = int(input())

Итак, в этом уроке мы разобрали реализацию основных вычислительных алгоритмов из школьного курса информатики, а именно алгоритм Евклида, алгоритм поразрядной обработки и алгоритмы обработки числовых последовательностей.

### Примеры задач

**Пример 1.** С клавиатуры вводится натуральное число. Найти:  
а) количество цифр  
б) сумму чётных цифр   
в) произведение цифр, которые больше 2  
г) наибольшую цифру  
д) наименьшую цифру

**Входные данные:**  
61325

**Выходные данные:**  
а) 5  
б) 8  
в) 90  
г) 6  
д) 1

Решение:

Объявим переменные для всех искомых значений. Количество и сумма изначально равны 0, произведение равно 1, т.к. при первом умножении мы должны получить первое число. Наибольшая цифра изначально принимает наименьшее возможное значение, а наименьшая, наоборот, - наибольшее. По мере выделения цифр наибольшее значение будет расти, а наименьшее - уменьшаться.

k = 0 #количество цифр

s = 0 #сумма чётных цифр

p = 1 #произведение цифр, больше 2

mx = 0 #наибольшая цифра

mn = 9 #наименьшая цифра

x = int(input())

while x > 0:

d = x % 10

k += 1

if d % 2 == 0:

s += d

if d > 2:

p \*= d

if d > mx:

mx = d

if d < mn:

mn = d

x //= 10

print('а)', k)

print('б)', s)

print('в)', p)

print('г)', mx)

print('д)', mn)

**Пример 2.** С клавиатуры вводится сначала число N, обозначающее размер числовой последовательности, а затем ровно N целых чисел. Найти:  
а) существует ли в последовательности хотя бы одно, кратное 5 (если да, вывести его номер, иначе ‘NO’)  
  
б) сумму чисел, оканчивающихся цифрой 8  
в) наибольшее трёхзначное число (если таких нет, то вывести ‘NO’)

**Входные данные:**  
5  
107  
28  
-45  
339  
88

**Выходные данные:**  
а) 3  
б) 116  
в) 339

Решение:

k5 = 0 #количество чисел, кратных 5

s8 = 0 #сумма чисел, оканчивающихся цифрой 8

mx3 = -1000 #наибольшее трёхзначное число

n = int(input())

for i in range(n):

a = int(input())

if a % 5 == 0:

k5 += 1

if k5 == 1:

i5 = i + 1

if a % 10 == 8:

s8 += a

if abs(a) >= 100 and abs(a) <= 999: #берём абсолютное значение числа, т.к. в последовательности

#встречаются как положительные, так и отрицательные числа

if a > mx3:

mx3 = a

if k5 > 0:

print('a)', i5)

else:

print('a) NO')

print('б)', s8)

if mx3 != -1000:

print('в)', mx3)

else:

print('в) NO')

**Пример 3.** С клавиатуры вводится последовательность целых чисел, оканчивающаяся нулём. Найти:  
а) количество отрицательных чисел  
  
б) произведение всех положительных чисел меньших 15 (если таких нет, то вывести 'NO')  
в) наименьшее чётное число (если таких нет, то вывести ‘NO’)

**Входные данные:**  
6  
-20  
7  
3  
-125  
0

**Выходные данные:**  
а) 2  
б) 126  
в) -20

Решение:

k = 0 #кол-во отрицательных чисел

k15 = 0 #количество положительных чисел меньше 15

p = 1 #произведение положительных меньше 15

mn = 1000000000 #наименьшее чётное

a = int(input())

while a != 0:

if a < 0:

k += 1

if a > 0 and a < 15:

p \*= a

k15 += 1

if a % 2 == 0 and a < mn:

mn = a

a = int(input())

print('a)', k)

if k15 > 0:

print('б)', p)

else:

print('б) NO')

if mn != 1000000000:

print('в)', mn)

else:

print('в) NO')

1. Дано натуральное число. Определить:  
   а) сколько раз в нем встречается последняя цифра;  
   б) сумму его цифр, больших пяти (если таковых нет, то -1);  
   в) произведение его цифр, меньших семи (если таковых нет, то -1);

**Sample Input:**

552845

**Sample Output:**

а) 3

б) 8

в) 1000

### Напишите программу.

1. Дано натуральное число, в котором все цифры различны. Определить порядковые номера его максимальной и минимальной цифр (цифры нумеруются слева направо, начиная с 1).

Например, в числе 36129 наибольшая цифра равна 9 и она стоит на 5 месте, наименьшая цифра - это 1 и она стоит на 3 месте

**Sample Input:**

36129

**Sample Output:**

Максимальная: 5

Минимальная: 3

1. Известно, что наименьшее общее кратное двух чисел (НОК) можно найти по формуле: (a \* b) / НОД(a,b). Введите с клавиатуры два натуральных числа a и b и выведите НОК(a, b).

**Sample Input:**

21

14

**Sample Output:**

42

### Напишите программу.

1. Дана последовательность из N целых чисел (N вводится с клавиатуры). Определить:   
   а) есть ли в последовательности хотя бы одно число, оканчивающееся цифрой 7? В случае положительного ответа вывести порядковый номер первого из них, иначе вывести 'NO'.  
   б) наименьшее из чисел  
   в) сумму всех чисел, кратных 3, но не кратных 4 (если таковых нет, то 0)

**Sample Input:**

5

30

-27

47

-911

125

**Sample Output:**

а) 2

б) -911

в) 3

5. Дана последовательность целых чисел, оканчивающаяся нулём. Найти:  
а) среднее арифметическое всех положительных двузначных чисел (если таких чисел нет, то вывести 0)  
б) наибольшее из чисел  
в) количество нечётных чисел в последовательности

**Sample Input:**

45

12

-9

33

115

-240

0

**Sample Output:**

а) 30.0

б) 115

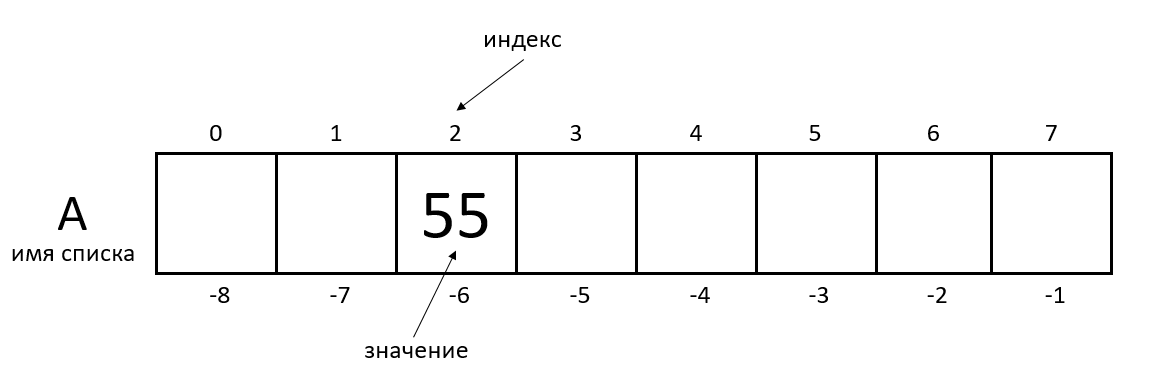
в) 4

### Напишите программу.

# Списки

В прошлом уроке мы рассмотрели алгоритмы обработки числовых последовательностей, но при этом мы не сохраняли всю последовательность в памяти компьютера. Однако, во многих задачах нужно именно сохранять всю последовательность, например, если бы нам требовалось изменить порядок элементов, вставить значения или удалить.

Структура данных, используемая для хранения массива данных, называется **списком.**В списке элементы расположены по порядку, каждый элемент пронумерован либо начиная с нуля слева направо, либо начиная с минус единицы справа налево. Порядковые номера элементов называются **индексами**. Список целиком имеет общее имя. Обращение к конкретному элементу осуществляется по индексу: для этого пишем сначала имя списка, а затем в квадратных скобках индекс. Каждый элемент имеет своё значение. Так мы можем последовательно хранить и обрабатывать большой объём данных.



Чтобы создать список в программе, мы сначала указываем его имя (оно составляется по тем же правилам, что и имя переменной), затем оператор присваивания и после пустые квадратные скобки. Также можно создать список с помощью функции list.

A = [] #пустой список

B = list() #тоже пустой список

Список может хранить любые элементы по порядку. Если вы хотите заранее заполнить список значениями, просто перечислите их внутри квадратных скобок через запятую. Можно создать список чисел или список строк. У элементов списка в языке Python не обязательно должен быть одинаковый тип, можно хранить вместе и разнотипные данные, но при решении задач мы чаще будем работать с однотипными данными.

L = [1, 2, 3, 4, 5]

M = ['here', 'are', 'some', 'words', '...']

Также функция list может преобразовать строку в список символов.

C = list('abcdefg') #C = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

Списки, как и строки, можно складывать друг с другом и дублировать, умножая их на целые числа.

A = [1, 2, 3] + [4, 5, 6] #A = [1, 2, 3, 4, 5, 6]

B = [0] \* 5 #B = [0, 0, 0, 0, 0]

Для работы со списками можно использовать встроенные функции (возвращают число):

A = [10, 20, 30, 40, 50]

len(A) #длина списка = 5

sum(A) #сумма элементов списка = 150

max(a) #наибольшее значение = 50

min(a) #наименьшее значение = 10

В отличие от функций, которые могут работать с объектами разных типов, методы принадлежат определённому классу и вызывается через точку. Сначала вы указываете имя списка, затем точку и после имя метода, который хотите вызвать.

Методы добавления и удаления элементов (изменяют текущий список):

A = [10, 20, 30, 40, 50]

A.append(60) #добавление элемент в конец списка

#A = [10, 20, 30, 40, 50, 60]

A.pop(0) #удаление элемента по индексу

#A = [20, 30, 40, 50, 60]

A.insert(3, 45) #вставить на позицию номер 3 элемент со значением 45

#A = [20, 30, 40, 45, 50, 60]

Методы подсчёта элементов и нахождения индекса (возвращают число):

A = [10, 20, 10, 30, 10, 40, 50]

A.count(10) #количество вхождений элемента в список = 3

A.index(50) #индекс первого вхождения элемента = 6

Методы сортировки и изменения порядка следования элементов (изменяют текущий список):

A = [30, 50, 10, 20, 40]

A.sort() #упорядочить элементы по возрастанию

#A = [10, 20, 30, 40, 50]

A.reverse() #расположить элементы списка в обратном порядке

#A = [50, 40, 30, 20, 10]

Вывести список на экран можно с помощью известной нам функции print. Если просто указать имя списка в качестве параметра, то на экран будет выведен список в квадратных скобках, а элементы будут перечислены через запятую, так же, как и указывается в программе. Если вы хотите вывести список без лишних символов, то поставьте перед его именем звёздочку, тогда элементы будут выведены в строчку через пробел.

A = [10, 20, 30, 40, 50]

print(A) #[10, 20, 30, 40, 50]

print(\*A) #10, 20, 30, 40, 50

Ввод списка можно осуществить двумя способами. Первый вариант – это вводить каждый элемент с новой строки. Сначала введём размер списка и создадим пустой список, а затем в цикле прочитаем ровно такое количество чисел и каждое прочитанное значение будем добавлять в конец списка.

n = int(input()) #длина списка

a = [] #пустой список

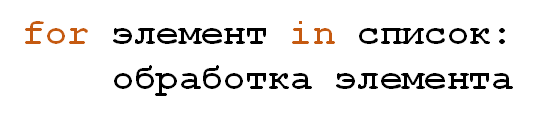
for i in range(n):

a.append(int(input())) #каждый элемент вводится с новой строки

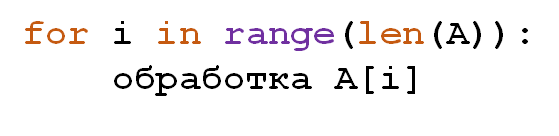
Второй способ – вводить элементы в одной строке через пробел. Такой способ подойдёт для небольших списков. Достаточно ввести элементы как строку, затем разбить её на отдельные строки с помощью метода split и преобразовать каждое значение в нужный тип с помощью функции map. В данном примере мы читаем список целых чисел, поэтому преобразуем значения в int. Вся эта конструкция подставляется в функцию list, чтобы получился именно список, и присваивается списковой переменной.

a = list(map(int, input().split())) #элементы списка вводятся в строку через пробел

Заполненный список можно обрабатывать в цикле for. В данном случае цикл пройдёт по всем элементам от начала до конца. Переменная el будет изменять своё значение и на каждом шаге цикла будет равна очередному элементу списка а. Такой цикл не подойдет, если вы хотите обрабатывать не все элементы списка.



Более универсальный способ – это сделать цикл по индексам элементов.



Если нужно обработать не все элементы списка, а некоторые, то удобно воспользоваться **срезом**. Срез создаёт новый список на основе исходного, из которого выбраны элементы от начального до конечного не включительно с заданным шагом. Параметры среза напоминают параметры функции range, только здесь они разделяются символом двоеточия. Срез, в котором не указаны параметры start и stop по умолчанию идёт от начала до конца списка. Шаг среза, если не указан, принимается за 1.



A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

A[3] #4

A[0:10:2] #[1, 3, 5, 7, 9]

A[-1:-10:-2] #[10, 8, 6, 4, 2]

A[::-1] #[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

### Примеры задач

**Пример 1.** Ввести с клавиатуры список целых чисел в одной строке через пробел. Скопировать все двузначные числа в другой список и вывести его, расположив числа в порядке убывания.

**Входные данные:**  
5 99 -32 567 8 21 48 100

**Выходные данные:**  
99 48 21 -32

Решение:

a = list(map(int, input().split()))

b = []

for el in a:

if abs(el) >= 10 and abs(el) <= 99: #берём абсолютное значение числа, т.к. элементы могут быть

#как положительные, так и отрицательные

b.append(el)

b.reverse()

print(\*b)

**Пример 2.** Ввести с клавиатуры список целых чисел в одной строке через пробел. Сдвинуть все элементы списка на 1 шаг вправо (так, чтобы 1-ый элемент оказался на месте 2-го, 2-ой на месте 3-го, а последний элемент на месте 1-го).

**Входные данные:**  
1 2 3 4 5 6 7

**Выходные данные:**  
7 1 2 3 4 5 6

Решение:

Задачу можно решить двумя способами. Первый способ: двигаясь по списку справа налево будем каждому i-ому элементу присваивать значение элемента, который находится слева от него. Предварительно запишем последний элемент в переменную t, чтобы не потерять это значение. После цикла поставим последний элемент в начало списка.

a = list(map(int, input().split()))

t = a[-1]

for i in range(len(a) - 1, 0, -1):

a[i] = a[i - 1]

a[0] = t

print(\*a)

Второй способ: воспользоваться срезами и поставить последний элемент в начало списка, не меняя порядка всех остальных элементов.

a = a[-1] + a[0:len(a) - 1]

print(\*a)

**Пример 3.** Ввести с клавиатуры список целых чисел в одной строке через пробел. Если все элементы списка расположены строго по возрастанию, что вывести +, если по убыванию, то -, если элементы не упорядочены ни тем, ни иным образом, то вывести 0.

**Входные данные:**  
5 4 3 2 1

**Выходные данные:**  
-

Решение:

Объявим 2 логические переменные: asc будет признаком того, что числа в списке возрастают, а desc - признаком убывания. Изначально присвоим им истинное значение. Затем в цикле будем проверять каждую пару соседних элементов. Если найдётся хотя бы одна пара, в которой i-ый элемент больше (i+1)-го, значит последовательность точно не возрастающая. Аналогично, если i-ый элемент меньше (i+1)-го, значит последовательность точно не убывающая. Если же два соседних элемента равны, то последовательность не может быть ни строго возрастающей, ни строго убывающей: оба признака примут значение False. После окончания цикла проверим, какой из признаков остался истинным и выведем + или -. Если же оба признака ложны, то выведем 0 (последовательность не возрастает и не убывает).

a = list(map(int, input().split()))

asc = desc = True

for i in range(len(a) - 1): #т.к. мы сравниваем пары элементов, количество шагов цикла на 1 меньше,

#чем длина списка (последняя пара - это предпоследний и последний элементы

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

asc = False

elif a[i] < a[i + 1]:

desc = False

else:

asc = desc = False

if asc:

print('+')

elif desc:

print('-')

else:

print('0')

**Пример 4.** Ввести с клавиатуры список целых чисел в одной строке через пробел. Удалить из этого списка все отрицательные элементы. Измененный список вывести на экран.

**Входные данные:**  
5 -4 3 -2 9 -1

**Выходные данные:**  
5 3 9

Решение:

Для решения этой задачи создадим пустой список b и будем копировать в него только те элементы, которые не удовлетворяют условию. Пользоваться функцией удаления элементов внутри цикла обработки списка нельзя, т.к. это нарушит его выполнение.

a = list(map(int, input().split()))

b = []

for el in a:

if el >= 0:

b.append(el)

print(\*b)

1. **Строки и функции**

Строки представляют собой символьные последовательности, заключенные в кавычки. В языке Python можно использовать как одинарные, так и двойные кавычки для обозначения строк и даже ставить три кавычки подряд в начале и в конце строки, если она содержит переносы. Последний вариант также часто используют для создания многострочного комментария в коде.

s = 'Это строка'

ss = "И это тоже строка"

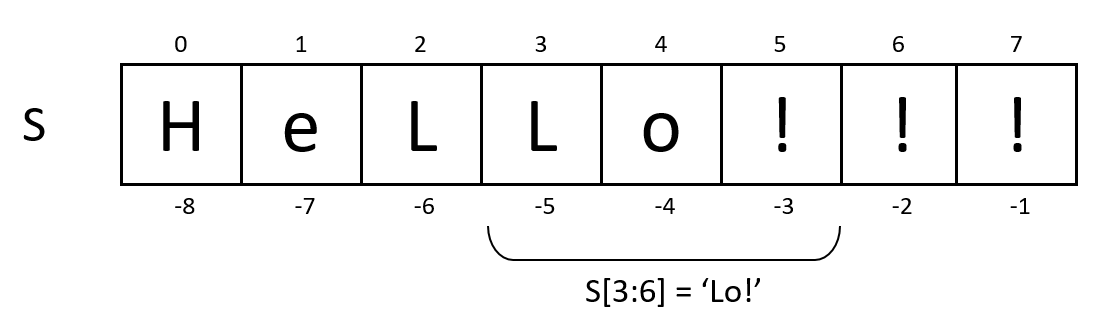
sss = """И

это

тоже

строка"""

Нумерация символов и срезы полностью аналогичны спискам, которые мы рассмотрели в прошлом уроке. Символы нумеруются с 0, если двигаться слева направо, и начиная с -1, если двигаться справа налево. Срезы позволяют выделить определённую часть строки, указав начальный индекс, конечный и шаг.



|  |  |
| --- | --- |
| if a in s: | содержится ли подстрока (символ) а в строке s |

В отличие от списков, при обращении к конкретному символу строки по индексу, вы не можете изменить его значение с помощью оператора присваивания. Если вы предполагаете изменение символов в процессе обработки строки, то необходимо преобразовать её в список с помощью функции list.

s = 'hello, world!'

s = 'hello, world!'  
~~s[0] = 'H'~~

#### Строки - это неизменяемый тип данных

Если вы предполагаете изменение символов в процессе обработки строки, то необходимо преобразовать её в список с помощью функции list.

#### Строки - это неизменяемый тип данных

Если вы предполагаете изменение символов в процессе обработки строки, то необходимо преобразовать её в список с помощью функции list.

t = list(s)  
t[0] = 'H'

|  |  |
| --- | --- |
| s1 + s2 | сложение строк |
| s \* 5 | дублирование строки |
| str(123) | преобразование числа в строку |
| s = input() | ввод строки с клавиатуры |
| print(s) | вывод строки на экран |
| len(s) | длина строки |
| s.islower() | является ли символ или строка строчной буквой |
| s.isupper() | является ли символ или строка заглавной буквой |
| s.isdigit() | является ли символ или строка цифрой |
| s.split() | преобразование строки в список строк (разбиение по пробелам или другим символам) |
| ' '.join(s) | объединение списка строк в одну строку |
| s.lower() | преобразовать все символы строки в строчные |
| s.upper() | преобразовать все символы строки в заглавные |
| s.count(a) | количество вхождений символа или подстроки а в строку s |
| s.replace(a, b, n) | заменить n первых вхождений подстроки (символа) a на подстроку (символ) b |
| if a in s: | содержится ли подстрока (символ) а в строке s |

### Функции, методы и операторы для обработки строк

Если для решения задачи вам нужно обработать каждый символ строки, воспользуйтесь циклом for. Можно обращаться к каждому символу через параметр или же по индексу так же, как и со списками.

s = input()

for sym in s:

#обработка символа sym

s = input()

for i in range(s):

#обработка символа s[i]

Если предполагается обработка отдельных слов, то предварительно разбейте строку с помощью метода split и затем организуйте цикл.

s = input().split()

for word in s:

#обработка слова s

Возможен вариант с вложенными циклами: сначала мы идём циклом по словам, а каждое слово разбираем по символам.

s = input().split()

for word in s:

for sym in word:

#обработка символов слова

Итак, в этом уроке мы узнали о возможностях строкового типа данных, познакомились с новыми функциями и методами, которые сможем применять для решения задач.

### Примеры задач

**Пример 1.** С клавиатуры вводится строка, состоящая из слов, разделённых пробелами. Определить количество слов в строке.

**Входные данные:**  
There are some words

**Выходные данные:**  
4

Решение:

Если между всеми словами стоит ровно один пробел, то можно вычислить количество пробелов с помощью метода count. Количество слов будет на 1 больше, чем это значение (в примере 4 слова и 3 пробела).

s = input()

print(s.count(' ') + 1)

Если не гарантируется, что в тексте нет лишних пробелов, то воспользуемся методом split для создания списка слов, разделённых пробелами (даже если их больше одного, лишние пробелы не повлияют на результат), а затем найдём длину этого списка.

s = input().split()

print(len(s))

**Пример 2.** В строке, состоящей из слов, разделённых пробелами, определите самое длинное слово.

**Входные данные:**  
Try to find the loooongest word

**Выходные данные:**  
loooongest

Решение:

Разобьём исходную строку на слова с помощью метода split и будем анализировать каждое слово в полученном списке. Если длина текущего слова больше максимального значения max\_len, то перезаписываем максимальную длину и запоминаем найденное слово в переменную max\_word. После окончания цикла выводим max\_word.

s = input().split()

max\_len = 0

max\_word = ''

for word in s:

if len(word) > max\_len:

max\_len = len(word)

max\_word = word

print(max\_word)

**Пример 3.** В первой строке вводится предложение, а во второй строке – слово. Определить, сколько раз это слово встречается в предложении.

**Входные данные:**  
I will you will we will they will  
will

**Выходные данные:**  
4

Решение:

Прочитаем строку s и искомое слово w. Затем воспользуемся методом count для подсчёта количества вхождений слова в строку.

s = input()

w = input()

print(s.count(w))

**Пример 4.** Строка состоит из слов, разделенных пробелами. Удалить все слова, начинающиеся с заглавной буквы.

**Входные данные:**  
London is the capital of Great Britain

**Выходные данные:**  
is the capital of

Решение:

Разобьём исходную строку на слова с помощью метода split и будем анализировать каждое слово в полученном списке. Если нулевой символ каждого слова не является заглавной буквой, то будем добавлять его во второй список (изначально пустой). Так мы сформируем список слов, из которого удалены все слова, начинающиеся с заглавной буквы. Наконец, преобразуем полученный список в строку, разделив слова пробелами, с помощью метода join и выведем измененную строку на экран.

s = input().split()

t = []

for word in s:

if not word[0].isupper():

t.append(word)

s = ' '.join(t)

print(s)

**Функции**

**Функцией** в программировании называется подпрограмма, самостоятельный фрагмент программного кода, к которому можно обратиться из любого места программы.

Чтобы сделать код программы более компактным и читабельным, избавиться от повторяющихся фрагментов и структурировать программу мы можем описывать свои собственные функции и использовать их многократно.

Описание функции начинается с ключевого слова **def**, затем записывается имя функции, состоящее из латинских букв, цифр и символа нижнего подчеркивания, далее в круглых скобках перечисляются имена параметров функции. Иногда они могут отсутствовать, тогда скобки остаются пустыми. После заголовка функции ставится двоеточие. Тело функции записывается с отступом относительно её заголовка. Также у функции может быть определённый результат работы, который она возвращает в основную программу. Для возврата значения в конце функции записывается оператор return и значение результата.

def имя\_функции(параметр1, параметр2, ...):

#тело фукнции

#...

#...

return результат

После того, как функция определена, её можно использовать в программе. Обращение к функции называется вызовом. Вызвать функцию можно в любом месте программы в тот момент, когда она вам понадобится. Чаще всего мы будем вызывать описанные функции в основной программе, то есть после определений.

Например, для вычисления количества сочетаний используется формула:

## **C\_n^m = \frac{n!}{m!(n-m)!}Cnm​=m!(n−m)!n!​**

Здесь нам потребуется трижды вычислить факториал для трёх разных чисел.

n = int(input())

m = int(input())

fn = 1

for i in range(1, n + 1):

fn \*= i

fn = 1

for i in range(1, n + 1):

fn \*= i

fm = 1

for i in range(1, m + 1):

fm \*= i

fnm = 1

for i in range(1, n - m + 1):

fnm \*= i

print(fn / (fm \* fnm))

Каждый из факториалов вычисляется в цикле от единицы до нужного значения включительно, поэтому в функции range мы записываем n + 1, m + 1 и так далее. Очевидно, что 3 фрагмента кода очень похожи и отличаются лишь одним из параметров цикла. Можно сократить код программы, если реализовать вычисление факториала в виде функции.

def fact(n):

f = 1

for i in range(1, n + 1):

f \*= i

return f

n = int(input())

m = int(input())

print(fact(n) / (fact(m) \* fact(n-m)))

Рассмотрим другие примеры функций. Функция square принимает один параметр и выводит на экран квадрат со стороной n, составленный из символов «собачка».

def square(n):

for i in range(n):

print('@' \* n)

square(5) #вызов функции с параметром n = 5

В результате на экран будут выведены следующие символы:

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

Функция p вычисляет периметр прямоугольника по двум сторонам:

def p(a, b):

return (a + b) \* 2

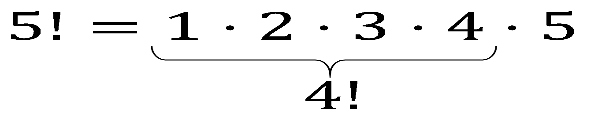
per = p(10, 15) #вызов фукнции с параметрами 10 и 15

print(per) #вывод результата

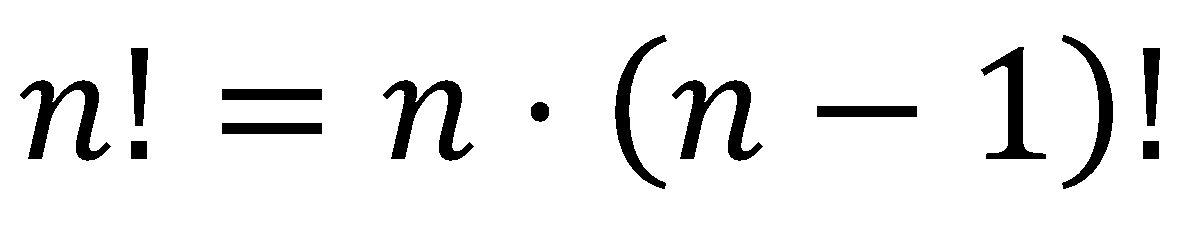
Она возвращает удвоенную сумму сторон a и b. Результат вызова функции необходимо записать в переменную.

### Рекурсия

Говоря о функциях, нельзя не упомянуть такое интересное явление, как рекурсия. **Рекурсией** называют определение чего-либо через себя самого. Например, если мы взглянем на то, как вычисляется факториал, то заметим, что аргумент умножается на произведение всех предыдущих чисел, то есть на факториал меньшего на единицу числа.



Это даёт нам следующее рекуррентное соотношение:



Программная реализация такой функции интересна тем, что функция fact вызывает сама себя внутри тела этой же функции. Такой вызов заменяет цикл. Но чтобы рекурсия не была бесконечной, мы ставим ограничение с помощью условия: вычислять факториал меньшего числа имеет смысл только при n>1, в остальных случаях факториал равен 1.

def fact(n):

if n > 1:

return n \* fact(n - 1)

else:

return 1

### Локальные и глобальные переменные

Любая функция создаёт собственную область видимости в программе. Считается, что все данные, описанные в теле функции, существуют и доступны только внутри этой функции. Это так называемые локальные переменные. Например, внутри функции объявлена переменная k со значением 1. В зависимости от параметра n это значение может увеличиться или уменьшиться в 10 раз.

def f(n):

k = 1 #локальная переменная

if n > 0:

k \*= 10

else:

k //= 10

f(12)

print(k)

Если вызвать данную функцию и попытаться вывести значение переменной k в основной программе, то это приведет к возникновению ошибки NameError. Переменная k в основной программе не существует. Вывести её можно только в теле функции f, но не за её пределами.

https://ucarecdn.com/f7872b57-0122-4586-914a-e28c9afc0064/

Однако, переменные из основной программы можно сделать доступными внутри функции, если указать для них ключевое слово **global**.

def f(n):

global k

if n > 0:

k \*= 10

else:

k //= 10

k = 1

f(12)

print(k) #k = 10

f(30)

print(k) #k = 100

Теперь переменная k существует в основной программе, а внутри функции f мы к ней обращаемся и изменяем её значение. После каждого вызова функции значение переменной будет меняться.

### Передача списков в функцию

Если мы передаём в функцию какое-либо значение из основной программы, все изменения также не выходят за пределы функции. Но это не относится к спискам, так как передача списков в функцию происходит **по ссылке**, а не по значению.

def f(a):

for i in range(len(a)):

if a[i] % 5 == 0:

a[i] = 555

a = [1, 2, 5, 6, 15, 31]

f(a)

print(a) #[1, 2, 555, 6, 555, 31]

В функцию f передали список а. Внутри этой функции элементы, кратные 5, заменяются на значение 555. После вызова функции числа 5 и 15 в исходном списке заменятся на 555. Изменение списка внутри функции повлияло на список в основной программе.

### Примеры задач

**Пример 1.** Реализуйте функцию вывода на экран треугольника из чисел.

**Входные данные:**  
4

**Выходные данные:**  
1  
22  
333  
4444

Решение:

Т.к. количество строк в треугольнике равно n, необходимо n раз повторить вывод текущего значения параметра цикла. Для получения строки, состоящей из нескольких одинаковых цифр, преобразуем число в строку и воспользуемся операцией умножения для дублирования числа нужное количество раз.

def triangle(n):

for i in range(1, n + 1):

print(str(i) \* i)

n = int(input())

triangle(n)

**Пример 2.**С клавиатуры вводится последовательность из N натуральных чисел. Определить число с наибольшим количеством чётных цифр. Если таких чисел несколько, вывести то, которое встретилось раньше. Если числа с чётными цифрами не встречаются в последовательности, вывести NO. Нахождение количества чётных цифр реализовать в виде функции.

**Входные данные:**  
5  
215  
144  
7023  
6  
68

**Выходные данные:**  
144

Решение:

def count\_even(n):

k = 0

while n > 0:

d = n % 10

if d % 2 == 0:

k += 1

n //= 10

return k

n = int(input())

mx = -1 #наибольшее количество чётных цифр

for i in range(n):

a = int(input())

k = count\_even(a) #выхов функции нахождения кол-ва чётных цифр

if k > mx:

mx = k #запоминаем новое наибольшее кол-во чётных цифр

ma = a #запоминаем число с таким количеством

if mx != -1: #если удалось найти хотя бы одно число с чётными цифрами

print(ma)

else:

print('NO')

**Пример 3.** Опишите функцию, вычисляющую n-ый элемент последовательности Фибоначчи, учитывая, что:  
F(0) = F(1) = 1  
F(n) = F(n - 1) + F(n - 2)

**Входные данные:**  
7

**Выходные данные:**  
21

Решение:

def F(n):

if n <= 1:

return 1

else:

return F(n-1) + F(n-2)

n = int(input())

print(F(n))

**Пример 4.** Реализуйте функцию, заменяющую каждый двузначный элемент списка на сумму его цифр. Элементы списка являются натуральными числами и вводятся в строку через пробел.

**Входные данные:**  
7 28 365 33 17 128 2 1000

**Выходные данные:**  
7 10 365 6 8 128 2 1000

Решение:

def f(a):

for i in range(len(a)):

if a[i] >= 10 and a[i] <= 99:

a[i] = a // 10 + a % 10

a = list(map(int, input().split()))

f(a)

print(\*a)