**Цикл for**

Переменная цикла

Решение задач

Аннотация. Урок посвящен циклу for. Что такое цикл for и как создавать программы, повторяющие определенные действия? Понятие переменной цикла.

Цикл for

Одно из преимуществ компьютеров перед людьми - способность повторять одни и те же действия многократно, быстро и совсем не утомляясь

В Python существует две основных разновидности циклов:

циклы, повторяющиеся определенное количество раз (**for**, счетные циклы, counting loops);

циклы, повторяющиеся до наступления определенного события (**while**, условные циклы, conditional loops).

Цикл for замечательно работает, если мы заранее знаем, сколько повторений (итераций) нам требуется сделать.

Рассмотрим код, который распечатает 10 раз слово "Привет":

for i in range(10): print('Привет')

Структура цикла for в Python выглядит так:

for название\_переменной\_цикла in range(количество повторений): блок кода

Двоеточие (:) в конце строки с инструкцией for сообщает интерпретатору Python, что дальше находится блок команд. В блок команд входят все строки, расположенные с отступом от строки с инструкцией for, вплоть до следующей строки без отступа.

￼   Блок команд, который выполняется в цикле for называется телом цикла.

В предыдущих уроках мы считывали несколько чисел, при помощи нескольких команд input. С помощью цикла for можно считывать и обрабатывать сколько угодно чисел.

Рассмотрим следующий программный код:

for i in range(5): num = int(input()) print('Квадрат вашего числа равен:', num \* num) print('Цикл завершен')

Такая программа считывает 5 чисел и выводит на экран их квадраты вместе с поясняющей надписью. Поскольку вторая и третья строки выделены отступом, Python считает, что это тело цикла, которое выполняется 5 раз. Четвертая строка не содержит отступа, поэтому не является частью цикла и будет выполнена всего один раз, после того как цикл завершится.

Примеры использования цикла for

Рассмотрим следующий программный код:

print('A') print('B') for i in range(5): print('C') print('D') print('E')

Результатом выполнения такой программы будут строки

A B C D C D C D C D C D E

То есть сначала программа распечатает символы А и В, затем символы C и D пять раз, а затем распечатает символ Е один раз. Тело цикла состоит из двух строк: четвертой и пятой и именно они будут повторяться.

В программе может быть сколько угодно циклов. Например, если мы хотим, чтобы сначала 5 раз был распечатан символ С, а затем 5 раз символ D, мы можем использовать 2 цикла:

print('A') print('B') for i in range(5): print('C') for i in range(5): print('D') print('E')

Результатом выполнения такой программы будут строки:

A B C C C C C D D D D D E

Примечания

Примечание 1. Однократное выполнение тела цикла называется итерацией цикла.

Примечание 2. Графическое представление цикла for имеет вид:

Примечание 3. Напомним, что блоком кода называют объединённые друг с другом строки. Они всегда связаны с определённой частью программы (например, с инструкцией if или for). В Python блоки кода формируются при помощи отступов.

Примечание 4. Слово for пишется маленькими буквами, первая строка должна заканчиваться двоеточием, и тело цикла должно быть выделено отступом.

Python is awesome

Напишите программу, которая выводит слова «Python is awesome!» (без кавычек) 10 раз.

Формат входных данных

Формат выходных данных  
Программа должна вывести 10 раз текст «Python is awesome!», каждый на отдельной строке.

s = "Python is awesome!"  
for i in range(10):  
    print(s)

**Переменная цикла**

Давайте еще раз взглянем на базовую структуру цикла for:

for название\_переменной\_цикла in range(количество повторений): блок кода

Не совсем понятно, для чего нужна и как работает переменная цикла.

Рассмотрим следующий код:

for i in range(10): print(i)

Результатом выполнения такого кода будет:

0 1 2 3 4 5 6 7 8 9

Когда цикл впервые начинает работу Python устанавливает значение переменной цикла i = 0. Каждый раз когда мы повторяем тело цикла Python увеличивает значение переменной на 1.

Почему большинство программистов начинают цикл с 0, а не с 1? Раньше некоторые начинали цикл с 1, а некоторые с 0. Те и другие приводили весьма изощренные аргументы, споря о том, какой способ лучше. Но в конце концов победили сторонники второго варианта. С тех пор большинство начинает циклы с 0. В частности, в Python цикл for начинается с 0, однако в будущих уроках вы узнаете как это изменить.

Поскольку переменная цикла i увеличивается на 1 каждый раз, то ее можно использовать для отслеживания номера итерации, на которой мы находимся в циклическом процессе.

  Рассмотрим следующий код:

for i in range(10): print(i, '-- Привет')

Результатом выполнения такого кода будет:

0 -- Привет 1 -- Привет 2 -- Привет 3 -- Привет 4 -- Привет 5 -- Привет 6 -- Привет 7 -- Привет 8 -- Привет 9 -- Привет

Если мы хотим начать с 1, то можем написать код:

for i in range(10): print(i + 1, '-- Привет')

Результатом выполнения такого кода будет:

1 -- Привет 2 -- Привет 3 -- Привет 4 -- Привет 5 -- Привет 6 -- Привет 7 -- Привет 8 -- Привет 9 -- Привет 10 -- Привет

Обратите внимание, за счет выражения i + 1, мы начинаем вывод с 1, а не с 0.

**Имена переменных цикла**

Ранее говорилось, что имена переменных должны носить осмысленный характер и описывать их назначение. Однако для переменных цикла иногда делается исключения. В программировании для переменных цикла обычно используют буквы i, j, k.

Следующие две программы абсолютно одинаковые: в первой программе переменная цикла имеет название i, во второй программе number:

for i in range(10): for number in range(10): print(i) print(number)

Результатом выполнения обеих программ будет:

0 1 2 3 4 5 6 7 8 9

￼Почему для переменной циклов зарезервированы буквы i, j, k? Дело в том, что раньше программы использовались для математических расчетов, а в математике буквы a, b, c и x, y, z уже зарезервированы для других целей. Поэтому программисты выбрали для этой цели переменные i, j, k и это стало общепринятой практикой.

Бывают ситуации когда переменная цикла не используется в теле цикла. В таком случае, вместо того, чтобы давать ей имя, мы можем указать символ нижнего подчеркивания \_:

for \_ in range(5): print('Python - awesome!')

Результатом выполнения такого кода будет:

Python - awesome! Python - awesome! Python - awesome! Python - awesome! Python - awesome!

￼   Если переменная цикла не используется в теле цикла, то указывайте вместо нее символ нижнего подчеркивания \_.

Примечания

Примечание. Следует помнить, что правая граница цикла в Python всегда не включительна. Таким образом следующий код:

for i in range(5): print(i)

распечатает числа от 0 до 4:

0 1 2 3 4

Если требуется распечатать числа от 1 до 5, то мы пишем код:

for i in range(5): print(i + 1)

**Повторяй за мной 2**

Напишите программу, которая считывает одну строку текста и выводит 10 строк, пронумерованных от 0 до 9, каждая с указанной строкой текста.

Формат входных данных  
На вход программе подается одна строка текста.

Формат выходных данных  
Программа должна вывести десять строк в соответствии с условием задачи.

s = input()  
for i in range(10):  
  print(i, s)

**Тема урока: цикл for**

Цикл for

Функции range() с одним параметром

Функции range() с двумя параметрами

Функции range() с тремя параметрами

Решение задач

Аннотация. Урок посвящен циклу for, в частности функции range(), которая позволяет генерировать последовательность чисел. Изучим две дополнительные перегрузки функции range(), которые позволяют настраивать элементы последовательности.

Функция range() с одним параметром

Рассмотрим программный код:

for i in range(10): print('Привет', i)

Значение, которое мы указываем в скобках у функции range() обозначает количество итераций цикла, при этом переменная i принимает последовательно значения: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Если быть более точным, то мы говорим, что функция range(n) генерирует последовательность чисел от 0 до n-1, а цикл for последовательно перебирает эту последовательность.

Перегрузка range() с двумя параметрами

Если мы хотим начинать последовательность не с 0, а с какого-то другого числа, то мы можем использовать перегрузку функции range() принимающую два параметра. Например, вызов функции range(1, 5) сгенерирует последовательность чисел 1, 2, 3, 4  (будьте внимательны, правая граница не включительна). Если нам нужны числа от 1 до 5 включительно, то мы используем range(1, 6).

Таким образом:

range(n): создает последовательность чисел 0, 1, 2, 3, ..., n - 1;

range(n, m): создает последовательность чисел n, n + 1, n + 2, ..., m - 2, m - 1.

Напишем программу, которая выводит те числа из промежутка [100;999][100;999], которые оканчиваются на 7.

Используя функцию range() с двумя параметрами, получаем:

for i in range(100, 1000): # перебираем числа от 100 до 999 if i % 10 == 7: # используем остаток от деления на 10, для получения последней цифры print(i)

Обратите внимание, в качестве второго параметра мы передали число 1000.

￼Если первый параметр больше второго, то функция range() генерирует пустую последовательность. Например, вызов функции range(10, 1) приводит к генерации пустой последовательности.

Перегрузка range() с 3 параметрами

Передавая два параметра в функцию range() мы можем генерировать любую последовательность целых чисел с шагом 1. Но, что делать если нужно поменять шаг? Как быть если мы хотим сгенерировать последовательность чисел 5, 10, 15, 20, 25? В этом случае существует еще одна перегрузка функции range(), принимающая три параметра: range(n, m, k). Первый параметр задает старт последовательности, второй параметр задает стоп последовательности и третий – шаг генерации чисел.

Например, вызов функции range(1, 10, 2) создаст последовательность чисел 1, 3, 5, 7, 9, а вызов функции range(5, 30, 5) сгенерирует последовательность 5, 10, 15, 20, 25.

Напишем программу, которая выводит все четные числа из промежутка [56;170][56;170].

Используя функцию range() с тремя параметрами, получаем:

for i in range(56, 171, 2): print(i)

Обратите внимание, мы можем использовать функцию range() с двумя параметрами:

for i in range(56, 171): if i % 2 == 0: print(i)

однако такой код получается менее эффективным.

Отрицательный шаг генерации

Если шаг генерации является положительным числом, то генерируемая последовательность будет возрастать. Мы можем указать отрицательный шаг генерации (третий параметр), что приведет к генерированию убывающей последовательности.

В случае отрицательного шага, мы должны гарантировать, что старт последовательности (первый параметр) больше чем конец последовательности (второй параметр).

Например, вызов функции range(20, 16, -1) создаст последовательность чисел 20, 19, 18, 17, а вызов функции range(20, 10, -3) сгенерирует последовательность 20, 17, 14, 11.

Напишем программу, которая отсчитывает от 5 до 1, а затем выводит текст Взлетаем!!!:

for i in range(5, 0, -1): print(i, end=' ') print('Взлетаем!!!')

Результатом будет:

5 4 3 2 1 Взлетаем!!!

￼Если величина шага отрицательна и первый параметр меньше второго, то функция range() генерирует пустую последовательность. Например, вызов функции range(1, 10, -1) приводит к генерации пустой последовательности.

Примеры использования функции range()

  Вызов функции   Последовательность чисел   range(10)0, 1, 2, 3, 4, 5, 6, 7, 8, 9range(1, 10)1, 2, 3, 4, 5, 6, 7, 8, 9range(3, 7)3, 4, 5, 6range(7, 3)пустая последовательностьrange(2, 15, 3)2, 5, 8, 11, 14range(9, 2, -1)9, 8, 7, 6, 5, 4, 3range(3, 10, -2)пустая последовательность

Примечания

Примечание 1. Функция range() может принимать от одного до трех параметров: range(n), range(n, m), range(n, m, k)

первый параметр – это старт последовательности (включительно);

второй параметр – это стоп последовательности (не включительно);

третий параметр – это величина шага.

Примечание 2. Функция range() может генерировать только целые числа, включая отрицательные.

Примечание 3. Величина шага не может равняться нулю. Следующий код:

for i in range(1, 10, 0): print(i)

приведет к ошибке ValueError: range() arg 3 must not be zero.

**Последовательность чисел 1**

Даны два целых числа �m и �n ( �≤�m≤n). Напишите программу, которая выводит все числа от �m до �n включительно.

Формат входных данных  
На вход программе подаются два целых числа �m и �n, каждое на отдельной строке.

Формат выходных данных  
Программа должна вывести числа в соответствии с условием задачи.

m=int(input())  
n=int(input())  
for i in range(m, n+1):  
    print(i)

**Таблица умножения**

Дано натуральное число �n. Напишите программу, которая выводит таблицу умножения на �n.

Формат входных данных  
На вход программе подается натуральное число.

Формат выходных данных  
Программа должна вывести таблицу умножения на введенное число.

Примечание. В качестве знака умножения используйте английскую букву x.

n = int(input())  
for i in range(1, 11):  
    print(n, "x", i, "=", n \* i)

**Тема урока: частые сценарии**

Подсчет количества

Вычисление суммы и произведения

Обмен значений переменных

Сигнальные метки

Определение максимума и минимума

Расширенные операторы присваивания (+=, -=, //= и т.д.)

Аннотация. Рассмотрим частые сценарии при написании циклов.

Подсчет количества

Нередко нужно, чтобы наши программы подсчитывали сколько раз что-либо произошло. К примеру видео игра может подсчитывать количество поворотов персонажа или математическая программа может считать как много чисел обладают некоторым свойством. Ключ к подсчету - использование переменной счетчика.

Напишем программу, которая считывает 10 чисел и определяет сколько из них больше 10.

counter = 0 for \_ in range(10): num = int(input()) if num > 10: counter = counter + 1 print('Было введено', counter, 'чисел, больших 10.')

Каждый раз когда мы считываем число большее 10, мы добавляем 1 к нашему текущему значению переменной counter. В программе это реализовано в строке counter = counter + 1. Обратите внимание на начальное значение переменной счетчика counter = 0. Без начального значения мы получили бы ошибку, поскольку дойдя до строки counter = counter + 1 Python ничего не знал бы о переменной counter . Строка кода counter = counter + 1 означает: возьми старое значение переменной counter, прибавь к нему 1 и переприсвой переменной это значение. Если не придать переменной начальное значение, то непонятно, к чему прибавлять 1 в самый первый раз.

Подсчет количества – это очень частый сценарий. Он состоит из двух шагов:

Создание переменной счетчика и придание ей первоначального значения: counter = 0;

Увеличение переменной счетчика на 1: counter = counter + 1.

Часто при написании программ требуется использовать несколько счетчиков. Модифицируем предыдущую программу: посчитаем еще и количество нулей среди введенных чисел.

counter1 = 0 counter2 = 0 for \_ in range(10): num = int(input()) if num > 10: counter1 = counter1 + 1 if num == 0: counter2 = counter2 + 1 print('Было введено', counter1, 'чисел, больших 10.') print('Было введено', counter2, 'нулей.' )

Рассмотрим еще один пример: подсчитать количество чисел из диапазона [1;100][1;100], квадрат которых оканчивается на 4.

counter = 0 for i in range(1, 101): if i\*\*2 % 10 == 4: counter = counter + 1 print(counter)

Мы используем функцию range() с двумя параметрами для генерации последовательности чисел от 1 до 100. Переменная i последовательно принимает значения от 1 до 100, мы проверяем, условие: квадрат числа i оканчивается на 4 с помощью условия i\*\*2 % 10 == 4.

￼   Для переменной счетчика удобно использовать имя counter.

Вычисление суммы и произведения

Наравне с подсчетом количества по частоте стоит задача вычисления суммы. К примеру видео игра должна считать сумму очков. В таком случае начальное значение переменной будет равно 0, а далее оно будет увеличиваться на некоторое количество заработанных очков, скажем на 10. Мы пишем следующий код:

score = 0 ... score = score + 10

Напишем программу, которая считывает 10 чисел и определяет сумму тех из них, которые больше 10.

total = 0 for \_ in range(10): num = int(input()) if num > 10: total = total + num print('Сумма чисел больших 10 равна', total)

Каждый раз когда программа считывает число большее 10, она добавляет его к текущему значению переменной total. Это реализовано в строке total = total + num. Обратите внимание на начальное значение переменной сумматора total = 0. Без начального значения, мы получили бы ошибку, поскольку дойдя до строки total = total + num Python ничего не знал бы о переменной total . Строка кода total = total + num означает возьми старое значение переменной total, прибавь к нему num и переприсвой переменной это значение. Если не придать переменной начальное значение, то не к чему прибавлять num в самый первый раз.

Подсчет суммы состоит из двух шагов:

Создание переменной сумматора и придание ей первоначального значения: total = 0;

Увеличение переменной сумматора на нужное число: total = total + num.

Напишем программу, которая считает сумму натуральных чисел от 1 до 100:

total = 0 for i in range(1, 101): total = total + i print('Сумма равна', total)

Рассмотрим еще один пример: напишем программу, которая запрашивает 10 целых чисел и находит их среднее значение:

total = 0 for \_ in range(10): num = int(input()) total = total + num average = total / 10 print('Среднее значение равно', average)

Аналогичным образом вычисляется произведение. При вычислении произведения, начальное значение переменной мультипликатора мы устанавливаем равным 1, в отличии от сумматора, где оно равно 0.

 ￼   Для переменной сумматора и мультипликатора удобно использовать имя total.

Обмен значений переменных

Очень часто нам требуется обменять значения двух переменных x и y. Начинающие программисты иногда пишут такой код:

x = y y = x

Однако он не работает. Предположим, что x = 3 и y = 5. Первая строка присвоит переменной x значение 5, что правильно, однако вторая строка установит значение переменной y в 5, поскольку значение x уже равно 5. Для решения задачи мы можем использовать временную переменную:

temp = x x = y y = temp

Такой код пишут почти во всех языках программирования. Однако в Python есть и более простой способ. Мы можем написать так:

x, y = y, x

В результате выполнения такого кода Python поменяет значения переменных x и y местами.

Сигнальные метки

Сигнальная метка (флажок) может использоваться, когда надо чтобы одна часть программы узнала, о происходящем в другой части программы.  
Напишем программу, определяющую, что натуральное число является простым:

num = int(input())

flag = True

for i in range(2, num):

if num % i == 0: # если исходное число делится на какое-либо отличное от 1 и самого себя

flag = False

if num == 1:

print('Это единица, она не простая и не составная')

elif flag == True:

print('Число простое')

else:

print('Число составное')

Напомним, что число является простым, если оно не имеет делителей, кроме 1 и самого себя. Вышеприведенная программа работает следующим образом: начальное значение переменной флага равно True, что говорит о том, что число является простым. Затем мы перебираем все числа от 2 до num - 1 (включительно). Если одно из этих значений оказывается делителем числа num, тогда число num является составным и мы устанавливаем значение флага False. Как только цикл завершен, мы проверяем, установлен флаг или нет. Если это так, мы знаем, что был делитель, и число не является простым. В противном случае число должно быть простым.

Флаговые переменные могут иметь более осмысленное название. Например, в случае с проверкой числа на простоту, название флаговой переменной могло бы быть is\_prime.

Максимум и минимум

Поиск наибольшего или наименьшего значения в некоторой последовательности чисел, также частая задача в программировании. Напишем программу, которая считывает 10 положительных чисел и находит среди них наибольшее число.

largest = -1 for \_ in range(10): num = int(input()) if num > largest: largest = num print('Наибольшее число равно', largest)

Мы устанавливаем начальное значение переменной largest  в -1. Далее программа считывает 10 чисел, и если какое-то из них оказывается больше текущего значения largest, переприсваивает его. В качестве начального значения взято число -1, поскольку мы знаем, что все числа положительны, таким образом уже первое сравнение приведет к переприсваиванию.

Распространен подход, когда в качестве начального значения переменной, сразу принимается первый элемент последовательности. Напишем программу, которая считывает 10 чисел (необязательно положительных) и находит среди них наибольшее:

largest = int(input()) # принимаем первое число за максимальное for \_ in range(9): num = int(input()) if num > largest: largest = num print('Наибольшее число равно', largest)

Для нахождения наименьшего значения последовательности следует поменять знак неравенства (>) на противоположный (<). В таком случае название переменной largest стоит заменить на smallest.

 ￼   Для переменных, хранящих наибольшее и наименьшее значения, подходят имена largest и smallest.

Расширенные операторы присваивания

Довольно часто программы имеют инструкции присваивания, в которых переменная на левой стороне от оператора = также появляется на правой от него стороне. Например,

counter = counter + 1

На правой стороне оператора присваивания 1 прибавляется к переменной counter. Полученный результат затем присваивается переменной counter, заменяя первоначальное значение. По сути, это строка кода добавляет 1 к counter. Еще один пример такой инструкции мы видели при подсчете суммы:

total = total + num

Эта инструкция присваивает значение выражения total + num переменной total. В результате исполнения этой инструкции число num прибавляется к значению total.

Различные инструкции присваивания (в каждой инструкции x = 6)ИнструкцияЧто она делаетЗначение x после инструкцииx = x + 4Прибавляет 4 к x10x = x - 3Вычитает 3 из x3x = x \* 10Умножает x на 1060x = x / 4Делит x на 41.5x = x // 4Делит нацело x на 41x = x % 4Находит остаток от деления x на 42

Эти типы операций находят широкое применение в программировании. Для удобства Python предлагает расширенные операторы присваивания. Расширенные операторы не требуют, чтобы программист дважды набирал имя переменной. Приведенную ниже инструкцию:

total = total + num

 можно переписать как

total += num

Точно так же инструкцию

counter = counter + 1

можно переписать как

counter += 1ОператорПример использованияЭквивалент+=x += 5x = x + 5-=x -= 2x = x - 2\*=x \*= 10x = x \* 10/=x /= 4x = x / 4//=x //= 4x = x // 4%=x %= 4x = x % 4

Примечания

Примечание 1. Аналогичным образом можно менять местами значения трех и более переменных.

a, b, c, d = b, c, d, a

Примечание 2. Очень часто сигнальные метки называют flag.

Примечание 3. Поскольку в Python есть встроенные функции max() и min(), то давать такие названия для максимального и минимального значения не очень хорошо. Куда лучше использовать названия largest и smallest или mx и mn.

Примечание 4. Сумму чисел от 1 до 100, можно вычислить и без цикла: Сумма=1+1002⋅100=5050.Сумма=21+100​⋅100=5050.Действительно, числа от 11 до 100100, можно разбить на 5050 пар, сумма в которых равна 101101: 1+100=101,2+99=101,3+98=101,…50+51=101.1+100=101,2+99=101,3+98=101,…50+51=101.В начальной школе, где учился математик Карл Фридрих Гаусс (6 лет), учитель, чтобы занять класс на продолжительное время самостоятельной работой, дал задание ученикам – вычислить сумму всех натуральных чисел от 11 до 100100. Маленький Гаусс ответил на вопрос почти мгновенно, применив указанный способ подсчета, чем невероятно удивил всех и, прежде всего, учителя.

**Количество чисел**

На вход программе подаются два целых числа �a и �b (�≤�)(a≤b). Напишите программу, которая подсчитывает количество чисел в диапазоне от �a до �b включительно, куб которых оканчивается на 44 или 99.

Формат входных данных  
На вход программе подаются два целых числа �a и �b (�≤�)(a≤b).

Формат выходных данных  
Программа должна вывести одно целое число в соответствии с условием программы.

Примечание. Куб числа �a – это его третья степень �3a3.

a=int(input())  
b=int(input())  
count=0  
for i in range(a, b+1):  
    if i%10==4 or i%10==9:  
        count+=1  
print(count)

**Сумма чисел**

На вход программе подается натуральное число �n, а затем �n целых чисел, каждое на отдельной строке. Напишите программу, которая подсчитывает сумму введенных чисел.

Формат входных данных  
На вход программе подаются натуральное число �n, а затем �n целых чисел, каждое на отдельной строке.

Формат выходных данных  
Программа должна вывести сумму данных чисел.

n=int(input())  
sm=0  
for \_ in range(n):  
    sm+=int(input())  
print(sm)  
      
**Знакочередующаяся сумма**

На вход программе подается натуральное число �n. Напишите программу вычисления знакочередующей суммы 1−2+3−4+5−6+…+(−1)�+1�.1−2+3−4+5−6+…+(−1)n+1n.

Входные данные  
На вход программе подается натуральное число �n.

Выходные данные  
Программа должна вывести единственное число в соответствии с условием задачи.

n=int(input())  
f=0  
for i in range(1, n+1):  
    f+=(-1)\*\*(i+1)\*i  
print(f)

**Последовательность Фибоначчи**

Напишите программу, которая считывает натуральное число �n и выводит первые �n чисел последовательности Фибоначчи.

Формат входных данных  
На вход программе подается одно число � (�≤100)n (n≤100) – количество членов последовательности.

Формат выходных данных  
Программа должна вывести члены последовательности Фибоначчи, отделенные символом пробела.

Примечание. Последовательность Фибоначчи – это последовательность натуральных чисел, где каждое последующее число является суммой двух предыдущих:1, 1, 2, 3, 5, 8, 13,  21, 34, 55, 89

n=int (input())  
a, b=1, 1  
for i in range(n):  
    print(a, end=' ')   
    a, b=b, a+b

**Тема урока: цикл while**

Цикл while;

Считывание данных до стоп значения;

Бесконечный цикл;

Решение задач.

Аннотация. Урок посвящен циклу while. Мы научимся создавать программы, повторяющие определенные действия пока выполняется некоторое условие.

Цикл while

Как уже было сказано в предыдущем уроке существуют две основные разновидности цикла:

циклы, повторяющиеся определенное количество раз (for, счетные циклы, counting loops);

циклы, повторяющиеся до наступления определенного события (while, условные циклы, conditional loops)

Мы уже рассмотрели работу цикла for, который является счетным циклом. Цикл for замечательно работает, если мы заранее знаем, сколько повторений (итераций) нам потребуется сделать. Но иногда нужно, чтобы цикл выполнялся до наступления некоторого события, и количество итераций в этом случае заранее оценить просто невозможно. И здесь на помощь приходит цикл while.

Структура цикла while в Python выглядит так:

while условие: блок кода

Двоеточие (:) в конце строки с инструкцией while сообщает Python, что дальше находится блок команд. В блок входят все строки, расположенные с отступом от строки с инструкцией while, вплоть до следующей строки без отступа.

Блок команд, который выполняется в цикле while, называется телом цикла.

Рассмотрим код, использующий цикл while, который распечатает 10 раз слово Привет:

i = 0 while i < 10: print('Привет') i += 1

Такой код можно легко заменить циклом for, поскольку мы заранее знаем сколько раз нужно выполнить тело цикла. Однако так бывает не всегда.

Напишем программу, которая считывает числа и выводит их квадраты, пока не будет введено -1. При такой постановке задачи мы не можем воспользоваться циклом for, так как не знаем сколько чисел будет предшествовать числу -1.

num = int(input()) while num != -1: print('Квадрат вашего числа равен:', num \* num) num = int(input())

В качестве начального значения переменной num, мы используем первое из чисел. Далее пока выполняется условие цикла, а именно, пока введенное число не равно -1, мы исполняем тело цикла. В тело цикла входит две команды:

напечатать квадрат введенного числа;

считать следующее число.

Важным являются два момента:

правильная инициализация переменной num;

изменение переменной num внутри цикла while.

￼Важно: если не изменять переменную num внутри цикла, то можно получить так называемый бесконечный цикл, который будет выполняться бесконечно много раз.

Цикл while очень похож на условный оператор if. Разница заключается в том, что в случае с условным оператором соответствующий блок кода будет выполняться только один раз, тогда как с циклом while блок кода будет выполнен многократно.

**Цикл for VS цикл while**

Мы всегда можем заменить цикл for с помощью цикла while. Следующие две программы выводят числа от 0 до 100:

# используем for for i in range(101): print(i) # используем while i = 0 while i < 101: print(i) i += 1

В первом цикле переменная i последовательно принимает значения от 0 до 100. Для цикла while, нам пришлось завести самостоятельно переменную i и придать ей начальное значение. Тело цикла while содержит аналогичную инструкцию вывода print(i), однако помимо этого мы самостоятельно увеличиваем значение переменной i на 1, что делается автоматически в случае с циклом for.

Напишем программу, выводящую все числа, кратные 3, используя цикл for и while:

# используем for for i in range(0, 100, 3): print(i) # используем while i = 0 while i < 100: print(i) i += 3

Не всегда, однако удается заменить цикл while с помощью цикла for. Если заранее не известно количество итераций, то необходимо использовать цикл while и только его.

Считывание данных до стоп значения

Часто при решении задач на цикл while, мы считываем данные, до тех пор пока пользователь не введет некоторое значение, которое называют стоп значением. Напишем программу, которая считывает числа и находит их сумму, до тех пор пока пользователь не введет слово stop:

text = input() total = 0 while text != 'stop': num = int(text) total += num text = input() print('Сумма чисел равна', total)

Такой код будет часто использоваться при решении задач.

Бесконечный цикл

Всегда, кроме редких случаев, цикл while должен содержать возможность завершиться. То есть в цикле что-то должно сделать проверяемое условие ложным. Если цикл не имеет возможности завершиться, то он называется бесконечным циклом. Бесконечный цикл продолжает повторяться до тех пор, пока программа не будет прервана. Бесконечные циклы обычно появляются, когда программист забывает написать программный код внутри цикла, который делает проверяемое условие ложным. В большинстве случаев следует избегать применения бесконечных циклов.

Пример бесконечного цикла:

i = 0 total = 0 while i < 10: total += i

Так как в теле цикла не происходит изменения переменной i, то условие i < 10 остается истинным и цикл выполняется бесконечно много раз.

Бесконечные циклы можно использовать в связке с оператором прерывания break. Об этом будет рассказано в следующих уроках.

Примечания

Примечание 1. Цикл while получил свое название из-за характера своей работы: он выполняет некую задачу до тех пор, пока условие является истинным. Слово while на английском означает как раз "пока".

Примечание 2. Цикл while называют циклом с предусловием, поскольку выполнению тела цикла предшествует проверка условия (сначала проверяется условие, а уже затем выполняется тело цикла).

Примечание 3. Однократное выполнение тела цикла называется итерацией цикла.

Примечание 4. Цикл while может не выполниться ни одного раза. Например, следующий код:

i = -1 while i > 0: print('Hello world!')

не выведет текст, поскольку условие i > 0 ложно с самого начала.

Примечание 5. Графическое представление цикла while имеет вид:

￼

Примечание 6. Условие в цикле while, как и в условном операторе if, может содержать логические операции or, and, not.

**До КОНЦА 1**

На вход программе подается последовательность слов, каждое слово на отдельной строке. Концом последовательности является слово «КОНЕЦ» (без кавычек). При этом само слово «КОНЕЦ» не входит в последовательность, лишь символизируя её окончание. Напишите программу, которая выводит члены данной последовательности.

Формат входных данных  
На вход программе подается последовательность слов, каждое слово на отдельной строке.

Формат выходных данных  
Программа должна вывести члены данной последовательности.

a=input()  
while a!='КОНЕЦ':  
    print(a)  
    a=input()

**До КОНЦА 2**

На вход программе подается последовательность слов, каждое слово на отдельной строке. Концом последовательности является слово «КОНЕЦ» или «конец» (большими или маленькими буквами, без кавычек). При этом сами слова «КОНЕЦ» и «конец» не входят в последовательность, лишь символизируя её окончание. Напишите программу, которая выводит члены данной последовательности.

Формат входных данных  
На вход программе подается последовательность слов, каждое слово на отдельной строке.

Формат выходных данных  
Программа должна вывести члены данной последовательности.

a=input()  
while (a!='КОНЕЦ') and (a!='конец'):  
    print(a)  
    a=input()

**Количество членов**

На вход программе подается последовательность слов, каждое слово на отдельной строке. Концом последовательности является одно из трех слов: «стоп», «хватит», «достаточно» (маленькими буквами, без кавычек). Сами эти слова в последовательность не входят, лишь символизируя её окончание. Напишите программу, которая выводит общее количество членов данной последовательности.

Формат входных данных  
На вход программе подается последовательность слов, каждое слово на отдельной строке.

Формат выходных данных  
Программа должна вывести общее количество членов данной последовательности.

a=input()  
n=0  
while a!='стоп' and a!='хватит' and a!='достаточно':  
    a=input()  
    n=n+1  
print(n)

**Пока делимся**

На вход программе подается последовательность целых чисел делящихся на 77, каждое число на отдельной строке. Концом последовательности является любое число, не делящееся на 77 (само это число в последовательность не входит, лишь символизируя её конец). Напишите программу, которая выводит члены данной последовательности.

Формат входных данных  
На вход программе подается последовательность чисел, каждое число на отдельной строке.

Формат выходных данных  
Программа должна вывести члены данной последовательности.

n=int(input())  
while n%7==0:  
  print(n)  
  n=int(input())  
    
**Сумма чисел**

На вход программе подается последовательность целых чисел, каждое число на отдельной строке. Признаком окончания последовательности является любое отрицательное число, при этом в саму последовательность оно не входит. Напишите программу, которая выводит сумму всех членов данной последовательности.

Формат входных данных  
На вход программе подается последовательность чисел, каждое число на отдельной строке.

Формат выходных данных  
Программа должна вывести сумму членов данной последовательности.

n=int(input())  
a=0  
while n>=0:  
    a=a+n  
    n=int(input())  
print(a)

**Количество пятерок**

На вход программе подается последовательность целых чисел от 11 до 55, характеризующее оценку ученика, каждое число на отдельной строке. Концом последовательности является любое отрицательное число либо число, большее 55. Напишите программу, которая выводит количество пятерок.

Формат входных данных  
На вход программе подается последовательность чисел, каждое число на отдельной строке.

Формат выходных данных  
Программа должна вывести количество пятерок.

n=int (input ())   
cnt\_5=0  
while 5>=n>=1:  
    if n==5:  
        cnt\_5+=1  
    n=int (input ())   
print (cnt\_5)

**Ведьмаку заплатите чеканной монетой**

Всем известно, что ведьмак способен одолеть любых чудовищ, однако его услуги обойдутся недешево. К тому же ведьмак не принимает купюры, он принимает только чеканные монеты. В мире ведьмака существуют монеты с номиналами 1,5,10,251,5,10,25.

Напишите программу, которая определяет, какое минимальное количество чеканных монет нужно заплатить ведьмаку.

Формат входных данных   
На вход программе подается одно натуральное число - цена за услугу ведьмака.

Формат выходных данных  
Программа должна вывести минимально возможное количество чеканных монет для оплаты.

n=int(input())  
a=0  
while n>=25:  
  a=a+1  
  n=n-25  
while n>=10:  
    a=a+1  
    n=n-10  
while n>=5:  
      a=a+1  
      n=n-5  
while n>=1:  
        a=a+1  
        n=n-1  
print(a)

**Тема урока: цикл while**

Использование цикла while для обработки цифр числа;

Решение задач.

Аннотация. Урок посвящен обработке цифр натурального числа, с помощью цикла while.

Обработка цифр числа

При изучении целых чисел (тип данных int), мы говорили про операцию целочисленного деления // и операцию нахождения остатка от деления одного целого числа на другое %. Используя цикл while и две данных операции, можно обработать цифры числа с произвольным количеством разрядов (цифр).

Пусть дано натуральное число n. Тогда:

результатом операции n % 10 – является последняя цифра числа;

результатом операции n // 10 – является число с удаленной последней цифрой.

Напишем программу, которая считывает натуральное число (целое положительное) и обрабатывает его цифры.

n = int(input()) while n != 0: # пока в числе есть цифры last\_digit = n % 10 # получить последнюю цифру # код обработки последней цифры n = n // 10 # удалить последнюю цифру из числа

Цикл while работает до тех пор пока в числе есть необработанные цифры. Тело цикла содержит:

процедуру получения последней цифры last\_digit = n % 10;

код обработки последней цифры;

процедуру удаления последней цифры из числа n = n // 10.

В качестве процедуры обработки может быть все, что угодно: вывод цифр, нахождение суммы, произведения цифр, нахождение наибольшей или наименьшей цифры, подсчет цифр удовлетворяющих некоторому условию и т.д.

Напишем программу, которая определяет есть ли в числе цифра 7.

num = int(input()) has\_seven = False # сигнальная метка while num != 0: last\_digit = num % 10 if last\_digit == 7: has\_seven = True num = num // 10 if has\_seven == True: print('YES') else: print('NO')

**Обратный порядок 1**

Дано натуральное число. Напишите программу, которая выводит его цифры в столбик в обратном порядке.

Формат входных данных   
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести цифры введенного числа в столбик в обратном порядке.

n=int(input())  
while n != 0:   
    a = n % 10   
    print(a)  
    n = n // 10

**Обратный порядок 2**

Дано натуральное число. Напишите программу, которая меняет порядок цифр числа на обратный.

Формат входных данных   
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести число, записанное в обратном порядке.

n=int(input())  
while n!=0:  
  a=n%10  
  print(a, end='')  
  n=n//10

**max и min**

Дано натуральное число �,(�≥10)n,(n≥10). Напишите программу, которая определяет его максимальную и минимальную цифры.

Формат входных данных   
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести максимальную и минимальную цифры введенного числа (с поясняющей надписью).

n=int(input())  
max1=0  
min1=9  
while n:   
    if (n % 10) < min1:  
      min1 = n % 10  
    if (n % 10) > max1:  
      max1 = n % 10  
    n = n // 10  
print('Максимальная цифра равна', max1)  
print('Минимальная цифра равна', min1)

**Все вместе**

Дано натуральное число. Напишите программу, которая вычисляет:

сумму его цифр;

количество цифр в нем;

произведение его цифр;

среднее арифметическое его цифр;

его первую цифру;

сумму его первой и последней цифры.

Формат входных данных   
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести значения указанных величин в указанном порядке.

n=int(input())  
a=0  
b=1  
c=0  
n1=n  
while n!=0:  
  first=n%10  
  last=n1%10  
  a=a+first  
  b=b\*first  
  n=n//10  
  c=c+1  
  d=a/c  
  e=first+last  
print(a)  
print(c)  
print(b)  
print(d)  
print(first)  
print(e)

**Вторая цифра**

Дано натуральное число �(�>9)n(n>9). Напишите программу, которая определяет его вторую (с начала) цифру.

Формат входных данных   
На вход программе подается одно натуральное число, состоящее как минимум из двух цифр.

Формат выходных данных  
Программа должна вывести его вторую (с начала) цифру.

n=int(input())  
while n>9:  
    last=n%10  
    n=n//10  
print(last)

**Одинаковые цифры**

Дано натуральное число. Напишите программу, которая определяет, состоит ли указанное число из одинаковых цифр.

Формат входных данных   
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести «YES» если число состоит из одинаковых цифр и «NO» в противном случае.

n=int (input ())   
flag="YES"   
last\_digit=n%10  
while n>0:  
    cur\_digit=n%10  
    if last\_digit != cur\_digit:  
        flag = "NO"   
    n //= 10  
print (flag)

**Тема урока: break, continue, else**

Оператор break;

Оператор continue;

Бесконечные циклы;

Ключевое слово else в циклах;

Решение задач.

Аннотация. Урок посвящен операторам break и continue, а также необязательному блоку else.

Оператор прерывания цикла break

Иногда бывает нужно прервать выполнение цикла преждевременно. Оператор break прерывает ближайший цикл for или while.

Усовершенствуем с помощью оператора break программу, проверяющую число на простоту,:

num = int(input()) flag = True for i in range(2, num): if num % i == 0: # если исходное число делится на какое-либо отличное от 1 и самого себя flag = False break # останавливаем цикл если встретили делитель числа if flag: # эквивалентно if flag == True: print('Число простое') else: print('Число составное')

Как только мы встречаем делитель отличный от 1 и num, мы меняем значение сигнальной метки и прерываем цикл, поскольку дальнейшее его выполнение лишено смысла: число гарантированно не является простым.

￼    Оператор прерывания цикла break позволяет ускорять программы, так как мы избавляемся от лишних итераций.

Напишем программу, использующую цикл for, которая считывает 10 чисел и суммирует их до тех пор, пока не обнаружит отрицательное число. В этом случае выполнение цикла прерывается командой break:

result = 0 for i in range(10): num = int(input()) if num < 0: break result += num print(result)

￼Оператор прерывания цикла break удобен в связке с сигнальными метками: когда после проверки некоторого условия нам нет смысла продолжать выполнение цикла.

Напишем программу, которая определяет, содержит ли введенное пользователем число цифру 7.

num = int(input()) number = num flag = False while num != 0: last\_digit = num % 10 if last\_digit == 7: flag = True break # прерываем цикл, так как число гарантированно содержит цифру 7 num //= 10 if flag: # эквивалентно if flag == True: print('Число', number, 'содержит цифру 7') else: print('Число', number, 'не содержит цифру 7')

Как только мы встретили цифру 7, мы меняем значение сигнальной метки и прерываем цикл с помощью оператора break. Мы можем и не прерывать цикл преждевременно, а дождаться его естественного завершения (условие num != 0, то есть все цифры числа обработаны), однако в таком случае мы будем совершать лишнюю работу, и в случае, если число очень большое, программа будет работать медленнее.

Бесконечные циклы

В предыдущих уроках мы говорили о цикле, который не имеет возможности завершиться и назвали его бесконечным циклом. Самый простой способ создать бесконечный цикл в Python – записать следующий код:

while True: print('Python is awesome!')

Результатом выполнения такого кода будет бесконечное количество строчек текста:

Python is awesome! Python is awesome! . . . Python is awesome! Python is awesome! Python is awesome!

Бесконечный цикл продолжает повторяться до тех пор, пока программа не будет прервана. Изучив оператор break, мы получили механизм прерывания бесконечных циклов.

Возможно, вам может показаться, что бесконечные циклы лишены смысла, однако это не совсем так. Например, вы можете написать программу, которая запускается и работает, постоянно принимая запросы на обслуживание. Программный код такой программы может выглядеть так:

while True: query = get\_new\_query() # получаем новый запрос на обработку query.process() # обрабатываем запрос

Иногда с помощью бесконечного цикла удается сделать программный код более читабельным. Более простым может быть завершение цикла на основе условий внутри тела цикла, а не на основе условий в его заголовке:

while True: if условие 1: # условие для остановки цикла break ... if условие 2: # еще одно условие для остановки цикла break ... if условие 3: # еще одно условие для остановки цикла break

В подобных случаях, когда существует множество причин завершения цикла, часто их проще выделить из нескольких разных мест, чем пытаться указать все условия завершения в заголовке цикла.

￼Важно: бесконечные циклы могут быть очень полезными. Просто помните, что вы должны убедиться, что цикл в какой-то момент будет прерван, чтобы он действительно не становился бесконечным.

Оператор continue

Другая стандартная идиома циклов — пропуск отдельных элементов при переборе. Оператор continue позволяет перейти к следующей итерации цикла for или while до завершения всех команд в теле цикла.

Напишем программу, которая выводит все числа от 1 до 100, кроме чисел 7, 17, 29 и 78.

for i in range(1, 101): if i == 7 or i == 17 or i == 29 or i == 78: continue # переходим на следующую итерацию print(i)

Примечания

Примечание 1. Оператор break прерывает выполнение ближайшего цикла, а не программы, то есть далее будет выполнена команда, следующая сразу за циклом.

Примечание 2. Графическое представление операторов break и continue имеет вид:

**Наименьший делитель**

На вход программе подается число �>1n>1. Напишите программу, которая выводит его наименьший отличный от 11 делитель.

Формат входных данных  
На вход программе подается одно натуральное число �n.

Формат выходных данных  
Программа должна вывести наименьший делитель отличный от 11.

Примечание. Используйте оператор break при обнаружении делителя.

n=int(input())  
for i in range (2, n+1):  
    if n%i==0:  
        break  
print(i)

**Следуй правилам**

На вход программе подается натуральное число �n. Напишите программу, которая выводит числа от 11 до �n включительно за исключением:

чисел от 55 до 99 включительно;

чисел от 1717 до 3737 включительно;

чисел от 7878 до 8787 включительно.

Формат входных данных  
На вход программе подается одно натуральное число �n.

Формат выходных данных  
Программа должна вывести числа в соответствии с условием задачи, каждое на отдельной строке.

Примечание. Используйте оператор continue .

n=int(input())  
for i in range (1, n+1):  
  if 5 <=i<= 9 or 17<=i<= 37 or 78 <=i<=87:  
    continue  
  print(i)

**Блок else в циклах**

Python допускает необязательный блок else в конце циклов while и for. Это уникальная особенность Python, не встречающаяся в большинстве других языков программирования. Синтаксис такой конструкции следующий:

while условие: блок кода1 else: блок кода2 # или for i in range(10): блок кода1 else: блок кода2

Блок кода2, указанный в else, будет выполнен, когда штатным образом завершается цикл while или for.

Сейчас вы можете подумать: «Как это может быть полезным?» Ведь мы можем сделать то же самое, поместив блок кода2 сразу после цикла while или for без else:

while условие: блок кода1 блок кода2 # или for i in range(10): блок кода1 блок кода2

В чем разница?

￼

Если слово else отсутствует в описании цикла, то блок кода2 будет выполняться после завершения цикла, несмотря ни на что. Если же слово else присутствует, то блок кода2 будет выполняться только в том случае, если цикл завершается штатным образом. Под штатным завершением цикла подразумевается его завершение без использования оператора прерывания break.

Рассмотрим следующий программный код:

n = 5 while n > 0: n -= 1 print(n) else: print('Цикл завершен.')

Данный цикл повторяется до тех пор, пока истинно условие n > 0. Поскольку цикл завершился штатным образом, то блок кода в инструкции else будет выполнен. Таким образом, результатом выполнения такой программы будут строки:

4 3 2 1 0 Цикл завершен.

Рассмотрим следующий программный код:

n = 5 while n > 0: n -= 1 print(n) if n == 2: break else: print('Цикл завершен.')

Этот цикл преждевременно завершается с помощью оператора прерывания break, поэтому блок кода в инструкции else не будет выполнен. Результатом выполнения такой программы будут строки:

4 3 2

￼Вам может показаться, что инструкция else в циклах while и for не совсем соответствует тому, что реально происходит. Гвидо ван Россум, создатель Python, сказал, что если бы он проектировал язык Python заново, то избавился бы от else в циклах.

Напишем программу, которая определяет, содержит ли введенное пользователем число цифру 7. Вместо программного кода, написанного ранее:

num = int(input()) n = num flag = False while n != 0: last = n % 10 if last == 7: flag = True break # прерываем цикл, так как число гарантированно содержит цифру 7 n //= 10 if flag is True: print('Число', num, 'содержит цифру 7') else: print('Число', num, 'не содержит цифру 7')

мы можем использовать:

num = int(input()) n = num while n != 0: last = n % 10 if last == 7: print('Число', num, 'содержит цифру 7') break n //= 10 else: print('Число', num, 'не содержит цифру 7')

Примечания

Примечание 1. Оператор continue не влияет на выполнение блока else в циклах.

Примечание 2. Блок else в циклах часто применяется для обработки отсутствия элементов.

Примечание 3. Блок кода else в циклах встречается не так часто на практике. Однако если вы обнаружите ситуацию, в которой применение else оправдано, то не стесняйтесь его использовать. Это может добавить ясности вашему коду!

**Тема урока: ревью кода**

Ревью кода;

Поиск ошибок;

Производительность кода.

Аннотация. Урок посвящен проведению ревью кода, поиску ошибок и анализу производительности.

Ревью кода

Ревью кода – проверка исходного кода программы с целью обнаружения и исправления ошибок и неточностей, которые остались незамеченными при начальной разработке.

В процессе ревью кода могут быть исправлены:

фактические ошибки;

производительность кода;

читабельность кода и ошибки форматирования кода.

Целью ревью кода является улучшение качества программного кода и совершенствование навыков программиста.

Как правило ревью кода выполняет программист с большим опытом.

￼

**Фактические ошибки**

К фактическим ошибкам в коде относятся ошибки из-за которых код может работать неверно. По сути, это ошибки относящиеся к алгоритму, который используется в программе для решения задачи.

Среди частых фактических ошибок встречаются:

отсутствие начальной инициализации переменной;

неправильная начальная инициализация переменной;

отсутствие отступа (в Python блоки кода выделяются отступами);

неправильные числовые граничные значения, например при использовании функции range();

неправильные граничные сравнения (путаница с >, >= или <, <=);

путаница логических операций or и and и т.д.

**Производительность кода**

Под производительностью кода в простейшем случае можно подразумевать то, сколько времени программа тратит на решение задачи. При написании программы, программист должен думать над тем, сколько времени в худшем случае потребуется его программе для решения задачи.

Рассмотрим задачу, которая проверяет число на простоту.

1 версия программы: Перебираем все числа от 2 до num - 1 и делаем проверку делимости числа num на i:

num = int(input()) flag = True for i in range(2, num): if num % i == 0: flag = False if num > 1 and flag == True: print('Число простое') else: print('Число составное')

Если программе на вход подается простое число num = 1934234249, то она будет работать примерно 270270 секунд = 4.54.5 минуты.🙄

2 версия программы: Несложно понять, что перебирать все числа от 2 до num - 1 не имеет смысла. Достаточно проверить числа от 2 до num // 2 + 1:

num = int(input()) flag = True for i in range(2, num // 2 + 1): if num % i == 0: flag = False if num > 1 and flag == True: print('Число простое') else: print('Число составное')

Если программе на вход подается простое число num = 1934234249, то она будет работать примерно 130130 секунд = 2.22.2 минуты. По сути мы улучшили время работы программы вдвое! 😊

3 версия программы: Правую границу num // 2 + 1 проверки можно улучшить, если заметить, что у любого составного числа есть делитель (отличный от 1), не превосходящий квадратного корня из числа. Таким образом, имеет смысл перебирать делители от 22 до ���+1num​+1.

num = int(input()) flag = True for i in range(2, int(num \*\* 0.5) + 1): if num % i == 0: flag = False if num > 1 and flag == True: print('Число простое') else: print('Число составное')

Если программе на вход подается простое число num = 1934234249, то она будет работать примерно 0.0660.066 секунд. По сути мы улучшили время работы программы в 4000 раз! 😎

4 версия программы: Предыдущие оптимизации касались случая, когда проверяемое число является простым. В случае если число является составным и мы нашли его первый делитель (отличный от 1), мы прерываем цикл с помощью оператора break:

num = int(input()) flag = True for i in range(2, int(num \*\* 0.5) + 1): if num % i == 0: flag = False break if num > 1 and flag == True: print('Число простое') else: print('Число составное')

**Читабельность кода**

Следует помнить, что наш код должен легко читаться другими программистами. Чтобы этого достичь, следует придерживаться стандарта PEP 8. Обращайте внимание на следующие моменты:

отступы и пробелы: используйте 4 пробела на один уровень отступа и никогда не смешивайте символы табуляции и пробелы;

названия переменных: используйте говорящие названия для переменных (total, counter, product) и следуйте стилю lower\_case\_with\_underscores (слова из маленьких букв с подчеркиваниями);

пустые строки: дополнительные отступы пустыми строками могут быть изредка полезны для выделения группы логически связанных частей программы: инициализация переменных, основной алгоритм, завершающая проверка и т.д.;

комментарии: комментарии должны являться законченными предложениями. И помните, комментарии, которые противоречат коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код!

￼   О стандарте PEP 8 на русском языке можно почитать тут.

Примечания

Примечание 1. Хорошие статьи о ревью кода можно почитать на хабре: статья 1, статья 2.

Примечание 2. Программные ошибки, часто называют багами 🦟

Примечание 3. Все программисты создают баги, особенно в начале своего карьерного пути. Это норма 😀

Примеры

**Пример 1.** Требовалось написать программу, которая выводит символ А 10 раз. Вы ревьювите следующий код:

print('A') print('A') print('A') print('A') print('A') print('A') print('A') print('A') print('A') print('A')

Что вы о нем скажете? Правильно ли он работает? Как его улучшить?

**Ревью.** Приведенный код выполняет поставленную задачу, однако его можно улучшить. Поскольку действия в строках одинаковые, то можно использовать цикл. Поскольку мы знаем количество повторений (итераций), то нам подходит цикл for:

for \_ in range(10): print('A')

**Пример 2.** Требовалось написать программу которая должна вывести все числа от 1 до 100 кратные 7. Вы ревьювите следующий код:

i = 1 while i < 101: if i % 7 == 0: print(i) i += 1

Что вы о нем скажете? Правильно ли он работает? Как его улучшить?

**Ревью.** Приведенный код содержит достаточно распространенную ошибку: неверно поставленный отступ. Поскольку управление циклом while происходит при помощи переменной i, то ее необходимо инкрементировать (увеличивать) каждую итерацию. В приведенном коде она инкрементируется только если выполняется условие i % 7 == 0, которое ложно для начального значения i = 1. Таким образом, получаем бесконечный цикл. Для исправления ошибки необходимо удалить отступ у строки i += 1:

i = 1 while i < 101: if i % 7 == 0: print(i) i += 1

В данном случае лучше использовать цикл for с шагом, равным 7. Это позволит сделать код более наглядным и сократит время выполнения программы, так как нет необходимости просматривать и проверять все числа на делимость на 7:

for i in range(7, 101, 7): print(i)

**Пример 3.**Требовалось написать программу которая выводит все числа от 100 до 1 в порядке убывания. Вы ревьювите следующий код:

for i in range(1, 100): print(101 - i)

Что вы о нем скажете? Правильно ли он работает? Как его улучшить?

**Ревью.** Приведенный код содержит достаточно распространенную ошибку: неправильная правая граница цикла for. Следует помнить, что правая граница в цикле for всегда не включительна. Таким образом, для исправления ошибки необходимо заменить число 100 на 101:

for i in range(1, 101): print(101 - i)

В данном случае лучше использовать цикл for с шагом, равным -1:

for i in range(100, 0, -1): print(i)

**Пример 4.** Требовалось написать программу которая находит сумму всех нечетных чисел от 1 до 1000. Вы ревьювите следующий код:

a = 1 for i in range(1, 1000): if i % 2 == 1: a = a + 1 print(a)

Что вы о нем скажете? Правильно ли он работает? Как его улучшить?

**Ревью.** Приведенный код содержит достаточно распространенные ошибки:

неправильная начальная инициализация переменной a;

неправильная правая граница цикла for;

неправильно записанная операция накапливания значения суммы.

a = 0 for i in range(1, 1001): if i % 2 == 1: a = a + i print(a)

Для улучшения читабельности кода изменим название переменной a на total и используем расширенный оператор присваивания:

total = 0 for i in range(1, 1001, 2): total += i print(total)

**Пример 5.** Требовалось написать программу которая вычисляет факториал числа. Вы ревьювите следующий код:

n = input() a = 0 for i in range(n): a = a \* i print(a)

Что вы о нем скажете? Правильно ли он работает? Как его улучшить?

**Ревью.** Приведенный код содержит достаточно распространенные ошибки:

отсутствие преобразования строки текста в целое число;

неправильная начальная инициализация переменной a;

неправильная работа с границами итерирования: переменная i принимает значения от 0 до n - 1.

n = int(input()) a = 1 for i in range(1, n + 1): a = a \* i print(a)

Для улучшения читабельности кода, изменим название переменной a на factorial и используем расширенный оператор присваивания:

n = int(input()) factorial = 1 for i in range(1, n + 1): factorial \*= i print(factorial)

￼В модуле math есть функция factorial(), которая вычисляет факториал числа. Поэтому вместо реализации своей версии, куда правильнее и удобнее воспользоваться уже готовой.

**Ревью кода-3**

На обработку поступает последовательность из 77 целых чисел (каждое на отдельной строке). Известно, что вводимые числа по абсолютной величине не превышают 106106. Нужно написать программу, которая подсчитывает и выводит сумму всех чётных чисел последовательности или 00, если чётных чисел в последовательности нет. Программист торопился и написал программу неправильно.

Найдите все ошибки в этой программе (их ровно 44). Известно, что каждая ошибка затрагивает только одну строку и может быть исправлена без изменения других строк.

Примечание 1. Число �x не превышает по абсолютной величине 106106, если −106≤� ≤106−106≤x ≤106.

Примечание 2. При необходимости вы можете добавить необходимые строки кода.

s = 0  
for \_ in range(7):  
    n = int(input())   
    if n % 2 == 0:  
        s +=  n  
print(s)

**Ревью кода-6**

На обработку поступает натуральное число. Нужно написать программу, которая выводит на экран произведение цифр введенного числа. Программист торопился и написал программу неправильно.

Найдите все ошибки в этой программе (их ровно 33). Известно, что каждая ошибка затрагивает только одну строку и может быть исправлена без изменения других строк.

n = int(input())  
product = 1  
while n > 0:  
    d = n % 10  
    product = product \* d  
    n //= 10  
print(product)

**Тема урока: вложенные циклы**

Вложенные циклы

Операторы break и continue во вложенных циклах

Решение задач

Аннотация. Вложенные, находящиеся внутри других циклов, циклы.

Вложенные циклы

Вложенный цикл расположен в еще одном цикле. Часы служат хорошим примером работы вложенного цикла. Секундная, минутная и часовая стрелки вращаются вокруг циферблата.Часовая стрелка смещается всего на 1 шаг для каждых 60 шагов минутной стрелки. И секундная стрелка должна сделать 60 шагов для 1 шага минутной стрелки. Это означает, что для каждого полного оборота часовой стрелки (12 шагов), минутная стрелка делает 720 шагов.

￼

Рассмотрим цикл, который частично моделирует электронные часы. Он показывает секунды от 0 до 59:

for seconds in range(60): print(seconds)

Можно добавить переменную minutes и вложить цикл написанный выше внутрь еще одного цикла, который повторяется 60 раз:

for minutes in range(60): for seconds in range(60): print(minutes, ':', seconds)

Для того, чтобы сделать модель законченной, можно добавить еще одну переменную для подсчета часов:

for hours in range(24): for minutes in range(60): for seconds in range(60): print(hours, ':', minutes, ':', seconds)

Результатом работы такого кода будет:

0 : 0 : 0 0 : 0 : 1 0 : 0 : 2 ... 23 : 59 : 58 23 : 59 : 59

Самый внутренний цикл сделает 60 итераций для каждой итерации среднего цикла. Средний цикл сделает 60 итераций для каждой итерации самого внешнего цикла. Когда самый внешний цикл сделает 24 итерации, средний сделает 24⋅60=144024⋅60=1440 итераций, и самый внутренний цикл сделает 24 ⋅60⋅60=8640024 ⋅60⋅60=86400 итераций!

Пример имитационной модели часов подводит нас к нескольким моментам, имеющим отношение к вложенным циклам:

вложенный цикл выполняет все свои итерации для каждой отдельной итерации внешнего цикла;

вложенные циклы завершают свои итерации быстрее, чем внешние циклы;

для того, чтобы получить общее количество итераций вложенного цикла, надо перемножить количество итераций всех циклов.

￼    Мы можем вкладывать друг в друга циклы как for, так и while.

Операторы break и continue во вложенных циклах

Оператор break выполняет прерывание ближайшего цикла в котором он расположен. Аналогично, оператор continue осуществляет переход на следующую итерацию ближайшего цикла.

Рассмотрим программный код:

for i in range(3): for j in range(3): print(i, j)

Результатом его выполнения будут 9 строк:

0 0 0 1 0 2 1 0 1 1 1 2 2 0 2 1 2 2

Изменим код, добавив во вложенный цикл условный оператор с оператором break:

for i in range(3): for j in range(3): if i == j: break print(i, j)

Результатом выполнения нового кода будут 3 строки:

1 0 2 0 2 1

Изменим оператор прерывания break на оператор continue:

for i in range(3): for j in range(3): if i == j: continue print(i, j)

Результатом выполнения нового кода будут 6 строк:

0 1 0 2 1 0 1 2 2 0 2 1

￼Если необходимо добиться прерывания внешнего цикла из-за выполнения условия во внутреннем, то стоит сделать это через сигнальную метку.

Примеры решения задач

Один интересный способ узнать о работе вложенных циклов состоит в их использовании для вывода на экран комбинаций символов. Давайте взглянем на один простой пример. Предположим, что мы хотим напечатать на экране звездочки в виде прямоугольной таблицы:

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

Таблица состоящая из звездочек состоит из 8 строк и 6 столбцов. Приведенный ниже фрагмент кода можно использовать для вывода одной строки звездочек:

for i in range(6): print ( '\*', end='')

Для того чтобы завершить весь вывод таблицы звездочек, нам нужно выполнить этот цикл восемь раз. Мы можем поместить этот цикл в еще один цикл, который делает восемь итераций, как показано ниже:

for i in range(8): for j in range(6): print('\*', end='') print()

Внешний цикл делает восемь итераций. Во время каждой итерации этого цикла внутренний цикл делает 6 итераций. (Обратите внимание, что в строке 4 после того, как все строки были напечатаны, мы вызываем функцию print(). Мы должны это сделать, чтобы в конце каждой строки продвинуть экранный курсор на следующую строку. Без этой инструкции все звездочки  
будут напечатаны на экране в виде одной длинной строки.)

Давайте рассмотрим еще один пример. Предположим, что вы хотите напечатать звездочки в комбинации, которая похожа на приведенный ниже звездный треугольник:

\* \*\* \*\*\* \*\*\*\* \*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

Представим эту комбинацию звездочек, как сочетание строк и столбцов. В этой комбинации всего восемь строк. В первой строке один столбец. Во второй строке – два столбца. В третьей строке – три. И так продолжается до восьмой строки, в которой восемь столбцов.

for i in range(8): for j in range(i + 1): print('\*', end='') print()

**Таблица-1**

Дано натуральное число � (�≤ 9)n (n≤ 9). Напишите программу, которая печатает таблицу размером �×3n×3, состоящую из данного числа (числа отделены одним пробелом).

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести таблицу размером �×3n×3, состоящую из данного числа.

Примечание. В конце строки может быть пробел.

n = int(input())  
for i in range(n):  
    for j in range(3):  
       print(n,end=' ')   
    print ()

**Таблица-2**

Дано натуральное число � (�≤ 9)n (n≤ 9). Напишите программу, которая печатает таблицу размером �×5n×5, где в �i-ой строке указано число �i (числа отделены одним пробелом).

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести таблицу размером �×5n×5 в соответствии с условием.

Примечание. В конце строки может быть пробел.

n=int(input())  
for i in range(1, n+1):  
  for j in range(5):  
    print(i, end=' ')  
  print()

**Таблица-3**

Дано натуральное число � (�≤ 9)n (n≤ 9). Напишите программу, которая печатает таблицу сложения для всех чисел от 11 до �n (включительно) в соответствии с примером.

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести таблицу сложения для всех чисел от 11 до �n.

Примечание 1. Таблицу сложения подразумеваем от 1 до 9 (включительно).

Примечание 2. В конце строки может быть пробел.

n=int(input())  
for i in range(n):  
  for j in range(9):  
    print(i+1,'+',j+1,'=',i+j+2)  
  print()

**Численный треугольник 1**

Дано натуральное число �n. Напишите программу, которая печатает численный треугольник в соответствии с примером:

1 22 333 4444 55555 ...

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести треугольник в соответствии с условием.

Примечание. Используйте вложенный цикл for.

n=int(input())  
for i in range(n):  
  for j in range(i+1):  
    print(i+1, end='')  
  print()

**Звездный треугольник**

Дано нечетное натуральное число �n. Напишите программу, которая печатает равнобедренный звездный треугольник с основанием, равным �n в соответствии с примером:

\* \*\* \*\*\* \*\*\*\* \*\*\* \*\* \*

Формат входных данных  
На вход программе подается одно нечетное натуральное число.

Формат выходных данных  
Программа должна вывести треугольник в соответствии с условием.

Примечание. Используйте вложенный цикл for.

n=int(input())  
for i in range(1,n+1):  
  for j in range(i):  
    if i+j<=n:  
      print('\*', end='')  
  print()

Использование вложенных циклов при решении уравнений

Вложенные циклы можно использовать для решения уравнений с несколькими переменными. Зная, что решения (корни) уравнения являются натуральными (целыми) числами, несложно написать программу, содержащую вложенный цикл и перебирающую все возможные значения переменных.

**Решение задач**

**Задача 1.** Найдите все пары натуральных чисел (и их количество), являющихся решением уравнения 12�+13�=77712x+13y=777.

Решение. Поскольку по условию числа �x и �y являются натуральными, то �≤64,�≤59x≤64,y≤59 (при �=65x=65, 12⋅65=78012⋅65=780, что больше, чем 777777, поэтому �≤64x≤64, т.к. �y не может быть отрицательным по условию; аналогично получается, что �≤59y≤59). Напишем программу, которая перебирает всевозможные пары чисел (�;�)(x;y) и проверяет для них выполнение условия 12�+13�=77712x+13y=777.

total = 0 for x in range(1, 65): for y in range(1, 60): if 12 \* x + 13 \* y == 777: total += 1 print('x =', x, 'y =', y) print('Общее количество натуральных решений =', total)

В результате выполнения такого кода, мы получим:

x = 3 y = 57 x = 16 y = 45 x = 29 y = 33 x = 42 y = 21 x = 55 y = 9 Общее количество натуральных решений = 5

**Задача 2.** Найдите все пары натуральных чисел (и их количество), являющихся решением уравнения �2+�2+�2=2020x2+y2+z2=2020.

Решение.  Поскольку по условию числа �,�x,y и �z являются натуральными, то �,�,�<2020≈45x,y,z<2020​≈45. Напишем программу, которая перебирает всевозможные тройки чисел (�;�;�)(x;y;z) и проверяет для них условие �2+�2+�2=2020x2+y2+z2=2020.

total = 0 for x in range(1, 45): for y in range(1, 45): for z in range(1, 45): if x \*\* 2 + y \*\* 2 + z \*\* 2 == 2020: total += 1 print('x =', x, 'y =', y, 'z =', z) print('Общее количество натуральных решений =', total)

В результате выполнения такого кода, мы получим:

x = 18 y = 20 z = 36 x = 18 y = 36 z = 20 x = 20 y = 18 z = 36 x = 20 y = 36 z = 18 x = 36 y = 18 z = 20 x = 36 y = 20 z = 18 Общее количество натуральных решений = 6

**Численный треугольник 2**

Дано натуральное число �n. Напишите программу, которая печатает численный треугольник с высотой равной �n, в соответствии с примером:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 ...

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести треугольник в соответствии с условием.

Примечание. Используйте вложенный цикл for.

n=int(input())  
m=1  
for i in range(1,n+1):  
    for j in range(1,i+1):  
        print(m, end=' ')  
        m=m+1  
    print()

**Численный треугольник 3**

Дано натуральное число �n. Напишите программу, которая печатает численный треугольник с высотой равной �n, в соответствии с примером:

1 121 12321 1234321 123454321 ...

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести треугольник в соответствии с условием.

Примечание. Используйте вложенный цикл for.

n=int(input())  
for i in range(1, n+1):  
  for j in range(i):  
    print(j+1, end='')  
  for k in range(i-1, 0, -1):  
    print(k, end='')  
  print()

**Делители-1**

На вход программе подается два натуральных числа �a и �b (�< �a< b). Напишите программу, которая находит натуральное число из отрезка [�;�][a;b] с максимальной суммой делителей и сумму его делителей. Если таких чисел несколько, то выведите наибольшее из них.

Формат входных данных  
На вход программе подаются два числа, каждое на отдельной строке.

Формат выходных данных  
Программа должна вывести два числа на одной строке, разделенных пробелом: число с максимальной суммой делителей и сумму его делителей.

a=int(input())  
b=int(input())  
counter=0  
largest=0  
for i in range(a, b+1):  
  total=0  
  for j in range(1, i+1):  
    if i % j == 0:  
      total+=j  
    if total >= counter:  
      counter = total  
      largest = i  
print(largest, counter)  
        
**Делители-2**

На вход программе подается натуральное число �n. Напишите программу, выводящую графическое изображение делимости чисел от 11 до �n включительно. В каждой строке надо напечатать очередное число и столько символов «+», сколько делителей у этого числа.

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести графическое изображение чисел от 11 до �n, каждое на отдельной строке.

n=int(input())  
for i in range(1, n+1):  
  count=0  
  for j in range(1, i+1):  
    if i % j == 0:  
      count+=1  
  print(i,'+'\*count,sep='')  
        
**Цифровой корень**

На вход программе подается натуральное число �n. Напишите программу, которая находит цифровой корень данного числа. Цифровой корень числа �n получается следующим образом: если сложить все цифры этого числа, затем все цифры найденной суммы и повторять этот процесс до тех пор, пока в результате не будет получено однозначное число (цифра), которое и называется цифровым корнем изначального числа.

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести цифровой корень введенного числа.

Примечание. Используйте вложенные циклы while.

n=int(input())  
while n > 9:  
  total=0  
  while n > 0:  
    last\_digit=n%10  
    total+=last\_digit  
    n//=10  
  n=total  
print(n)

**Сумма факториалов**

Дано натуральное число �n. Напишите программу, которая выводит значение суммы 1!+2!+3!+…+�!1!+2!+3!+…+n!.

Формат входных данных  
На вход программе подается одно натуральное число.

Формат выходных данных  
Программа должна вывести значение суммы 1!+2!+3!+…+�!1!+2!+3!+…+n!.

Примечание 1. Факториалом натурального числа �n, называется произведение всех натуральных чисел от 11 до �n, то есть�!=1⋅2⋅3⋅…⋅�n!=1⋅2⋅3⋅…⋅n

Примечание 2. Задачу можно решить без вложенного цикла. Напишите две версии программы =)

n=int(input())  
total=0  
counter=1  
for i in range(1, n+1):  
  counter\*=i  
  total+=counter  
print(total)

**Простые числа**

На вход программе подается два натуральных числа �a и �b (�< �a< b). Напишите программу, которая находит все простые числа от �a до �b включительно.

Формат входных данных  
На вход программе подаются два числа, каждое на отдельной строке.

Формат выходных данных  
Программа должна вывести все простые числа от �a до �b включительно, каждое на отдельной строке.

Примечание. Число 11 простым не является.

a=int(input())  
b=int(input())  
for i in range(a, b+1):  
  count=0  
  for j in range(1, i+1):  
    if i%j==0:  
      count+=1  
  if count==2:  
     print(i)

Квадратное уравнение

**import** math

**print**("Введите коэффициенты для уравнения")

**print**("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 **\n**x2 = %.2f" % (x1, x2))

**elif** discr == 0:

x = -b / (2 \* a)

**print**("x = %.2f" % x)

**else**:

**print**("Корней нет")

**Звездная рамка**

На вход программе подается натуральное число �*n*. Напишите программу, которая печатает звездную рамку размерами �×19*n*×19.

**Формат входных данных**  
На вход программе подаётся натуральное число �∈[3;19]*n*∈[3;19] — высота звездной рамки.

**Формат выходных данных**  
Программа должна вывести звездную рамку размерами �×19*n*×19.

**Подсказка.** Для печати звездной линии используйте умножение строки на число.

**Sample Input 1:**

3

**Sample Output 1:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Sample Input 2:**

4

**Sample Output 2:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Sample Input 3:**

5

**Sample Output 3:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Решение:

n=int(input())

print('\*'\*19)

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

print('\*'+' '\*17+'\*')

print('\*'\*19)

## Третья цифра

Дано натуральное число �(�>99)*n*(*n*>99). Напишите программу, которая определяет его третью (с начала) цифру.

**Формат входных данных**  
На вход программе подается одно натуральное число, состоящее как минимум из трех цифр.

**Формат выходных данных**  
Программа должна вывести его третью (с начала) цифру.

Решение:

n=int(input())

while n>99:

last=n%10

n=n//10

print(last)

**Все вместе 2**

Дано натуральное число. Напишите программу, которая вычисляет:

* количество цифр 33 в нем;
* сколько раз в нем встречается последняя цифра;
* количество четных цифр;
* сумму его цифр, больших пяти;
* произведение цифр, больших семи (если цифр больших семи нет, то вывести 11, если такая цифра одна, то вывести ее);
* сколько раз в нем встречаются цифры 00 и 55 (всего суммарно).

**Формат входных данных**  
На вход программе подается одно натуральное число.

**Формат выходных данных**  
Программа должна вывести значения указанных величин в указанном порядке.

Решение:

n=int(input())

piat=0 # Создаем счетчик для подсчета суммы цифр больших 5-и равный 0

tri=0 # Создаем счетчик для подсчета 3-ек равный 0

counter\_last = n % 10 # Создаем счетчик counter\_last равный n % 10

last=0 # Создаем счетчик для последней цифры равный 0

sem=1 # Создаем счетчик для подсчета произведений цифр больших 7-и равный 1

nolpiat=0 # Создаем счетчик для подсчета кол-ва цифр 0 и 5 равный 0

chet=0 # Создаем счетчик для четных цифр равный 0

while n > 0: # Создаем условный цикл пока считываемое число не равно 0

a = n % 10 # Создаем переменную 'a' для присвоения её последней цифры числа

if a == 3: # Создаем условие для проверки кол-ва троек если a == 3

tri += 1 # Счетчик троек плюс 1

if a == counter\_last: # Создаем условие для проверки последней цифры если a == counter\_last

last += 1# Счетчик для последней цифры плюс 1

if a % 2 ==0: # Создаем условие для проверки на четность

chet += 1 # Счетчик плюс один

if a > 5: # Создаем условие для проверки цифр больших пяти

piat += a # Счетчик суммы плюс 'a'

if a > 7: # Условие для проверки цифр больших 7

sem \*= a # Счетчик умножить на 'a'

if a in (0, 5): # Если 'a' равно 0 или 5 if a in (0,5)

nolpiat += 1 # Счетчик плюс один

n=n//10 # Избавляемся от остатка

print(tri) # Выводим ответ

print(last)

print(chet)

print(piat)

print(sem)

print(nolpiat)

## Числа Рамануджана 🌶️

Сриниваса Рамануджан – индийский математик, славившийся своей интуицией в области чисел. Когда английский математик Годфри Харди навестил его однажды в больнице, он обмолвился, что номером такси, на котором он приехал, было 17291729, такое скучное и заурядное число. На что Рамануджан ответил: "Нет, нет! Это очень интересное число. Это наименьшее число, выражаемое как сумма двух кубов двумя разными способами". Другими словами: 1729 =13+123=93+103.1729 =13+123=93+103.

Напишите программу, которая находит аналогичные интересные числа. В ответе запишите первые 55 чисел в порядке возрастания, включая число 17291729.

**Примечание.** Используйте вложенный цикл.

for a in range (1, 33):  
  for b in range (1, 33):  
    for c in range (1, 33):  
      for d in range (1, 33):  
        if ((a \*\* 3 + b \*\* 3) == (c \*\* 3 + d \*\* 3)) and a != b and a != c and a != d and b != c and b != d and c != d:  
          print(a \*\* 3 + b \*\* 3)

1729  
4104  
13832  
20683  
32832