

Когда нужно сделать выбор




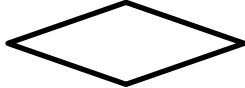
*Долженкова Мария Львовна,
кандидат технических наук, заведующий кафедрой ЭВМ
Нижегородова Маргарита Владимировна,
кандидат педагогических наук, доцент кафедры САУ
Шмакова Наталья Александровна,
старший преподаватель кафедры САУ*

Разветвленный алгоритм

Программы, которые мы писали до сих пор, работали строго последовательно. Каждая следующая инструкция выполнялась после завершения предыдущей, мы не могли повлиять на ход действий, даже если с нашей точки зрения решение задачи было не совсем корректным. Подобные алгоритмы называют линейными. Для написания более сложных программ необходима возможность принятия решения о выполнении или невыполнении того или иного участка кода в зависимости от сложившихся условий. В этом случае принято говорить о ветвлении или условных конструкциях.

Условная конструкция позволяет изменить поведение программы и используется, когда нужно сделать выбор в пользу какого-либо варианта развития алгоритма. Для того чтобы лучше понимать, как осуществляется выбор, обратимся к графической нотации, описывающей алгоритм решения задач – схеме алгоритма.

Определим необходимый нам минимальный набор символов.

Символ	Назначение
	Терминальный символ – отображает выход во внешнюю среду и вход из внешней среды. Является начальным или конечным символом схемы алгоритма.
	Символ ввода/вывода - сопровождает получение данных для обработки или формирования результата для передачи во внешнюю по отношению к алгоритму среду.
	Символ процесса – определяет выполняемые операции по обработке данных, включает арифметические, логические выражения и присваивание значений переменным.
	Символ решения – обеспечивает выбор направления выполнения алгоритма в зависимости от логического условия, которое в простейшем случае может принимать только два значения: истина, если условие выполняется, или ложь, если условие не выполняется.

Составим схему алгоритма для решения задачи поиска площади прямоугольника (рисунок 1).

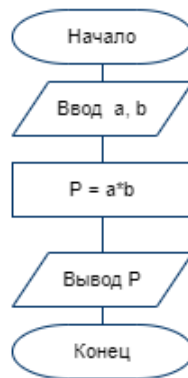


Рисунок 1 – Поиск площади прямоугольника

Сейчас наш алгоритм является линейным. Пользователь вводит два значения, соответствующие длине и ширине прямоугольника (символ ввода/вывода), вычисляется площадь по известной нам формуле (символ процесса) и результат выводится на экран. Пока все хорошо.

А что если вместо числовых значений пользователь введет буквы? Мы не сможем корректно выполнить операцию умножения или введем пользователя в заблуждение? Справиться с данной проблемой можно, если предварительно проверить правильность ввода, и, если данные не корректны, не выполнять вычисления, а вывести пустое значение (рисунок 2).

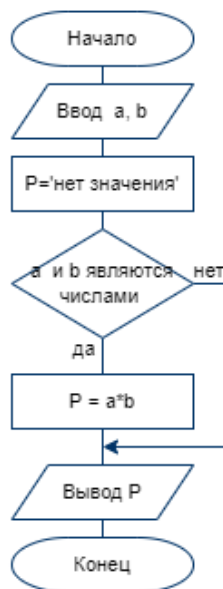


Рисунок 2 – Дополнительная проверка корректности ввода

Если при описании алгоритма нам потребовался символ решения, можно говорить, что такой алгоритм разветвленный, так как в зависимости от условия будет выполняться лишь одна из его ветвей.

В языках программирования для реализации подобной нелинейности используют условные операторы. В Python есть условный оператор if, который в простейшем случае, имеет следующий синтаксис.

if <условное выражение>:
оператор

Оператор выполняется лишь в том случае, если условное выражение является истинным. Графическая интерпретация этого оператора на языке схем представлена на рисунке 3.



Рисунок 3 – Графическая интерпретация условного оператора

Само решение задачи о площади прямоугольника представлено ниже.

Программа	На экране будет выведено
<pre>a,b = input('Введите значение ширины и длины прямоугольника через пробел\n').split() s = 0 if (a.isdigit() and b.isdigit()): s = int(a) * int(b) print('Площадь заданного прямоугольника ', s)</pre>	<p>Введите значение ширины и длины прямоугольника через пробел</p> <p>5 10</p> <p>Площадь заданного прямоугольника 50</p>

Теперь разберемся, что в Python является истинным (True). К истинным значениям в Python относятся: любое число не равно нулю, любая непустая строка, любой непустой объект. Само логическое выражение может быть достаточно сложным – содержать операторы сравнения и логические операторы, которые используются для сравнения значений и конструирования сложных логических выражений соответственно.

Оператор	Действие	Пример
>	«Больше чем»: True, если левый операнд больше правого	$x > y$
<	«Меньше чем»: True, если левый операнд меньше правого	$x < y$
==	«Равно»: True, если операнды равны между собой	$x == y$
!=	«Не равно»: True, если операнды не равны между собой	$x != y$

Оператор	Действие	Пример
>=	«Больше или равно»: True, если левый операнд больше или равен правому	x >= y
<=	«Меньше или равно»: True, если левый операнд меньше или равен правому	x <= y

Определим, какое из двух чисел, введенных пользователем, больше.

Программа	На экране будет выведено
<pre>a = int(input()) b = int(input()) if a > b: print("a больше b") if a < b: print("b больше a") if a == b: print("a и b равны")</pre>	<p>2</p> <p>-4</p> <p>a больше b</p>

Оператор	Результат	Пример
and	True, если значения обоих операндов True	x and y
or	True, если значение одного из операндов True	x or y
not	True, если значение операнда False	not x

Давайте определим, является ли введенный пользователем год високосным. Напомним, что в соответствии с григорианским календарем, год является високосным, если его номер кратен 4, но не кратен 100, а так же если он кратен 400.

Программа	На экране будет выведено
<pre>year = int(input()) s = "" if not(year % 4 == 0 and year % 100 != 0 or year % 400 == 0): s = 'не' print('Год %d %s является високосным' % (year, s))</pre>	<p>1900</p> <p>Год 1900 не является високосным</p>

Обратим внимание на то, что, логическая операция с оператором and выполняется раньше, чем операция с оператором or. Так, например, если у нас есть три числа a, b и c, то выражение: a > 0 and b <= 0 and c <= 0 or b > 0 and a <= 0 and c <= 0 or c > 0 and a <= 0 and b <= 0 будет истинным, если только одно из них положительное.

В рассмотренных примерах мы использовали сокращенный условный оператор. Если же необходимо реализовать более сложный алгоритм, используют полную запись, или даже вложенные условные операторы.

На прошлой лекции мы создавали свою мини-Википедию. Но ее работа не была корректной в случае, если пользователь запрашивал информацию о человеке,

Основы программирования на Python. Когда нужно сделать выбор

которого нет в нашем словаре. Будем проверять результат запроса и выводить сообщение, если ключа с заданным значением нет. Для этого будем использовать полную форму if, которая реализует ветвление.

if <условное выражение>:

оператор1, выполняется, если условное выражение истинно

else:

оператор2, выполняется, если условное выражение ложно

Если условное выражение оказывается ложным, то выполняется блок кода инструкции else. Ситуация, при которой бы выполнились обе ветви, невозможна. Графическая интерпретация этого оператора на языке схем представлена на рисунке 4.



Рисунок 4 – Графическая интерпретация условного оператора

Обратите внимание на отступы во второй и четвертой строках – инструкция if является составной. Согласно общей форме полного варианта инструкции if, как в ветви if, так и в ветви else может быть указано не одно действие.

Иногда существует более двух возможностей выбора и требуется более двух ветвей. Один из способов выразить подобное вычисление – использовать связанные условия (вложенные условия).

if <условное выражение1>:

оператор1, выполняется, если условное выражение1 истинно

elif <условное выражение2>:

оператор2, выполняется, если условное выражение1 ложно, а условное выражение2 истинно

else:

оператор3, выполняется, если условное выражение1 и условное выражение2 ложно

elif – это аббревиатура для «else if». Ограничений на количество инструкций elif нет. Инструкции else должны быть указаны в конце, сколько бы их ни было. Графическая интерпретация этой формы представлена на рисунке 5.

Если в цепочке if-elif-elif-... истинными являются несколько условий, то «срабатывает» первое из них (как показано на рисунке 6).

Давайте попробуем

Задача 1. Определим скидку на реализуемый товар в зависимости от его цены. Максимальная скидка (10%) назначается, если цена товара более 5000 руб., для

товара с ценой до 1000 руб. – скидка не предусмотрена, при цене от 2000 до 5000 руб. скидка составляет 5%, а от 1000 до 2000 руб. – 2%.

Программа	На экране будет выведено
<pre>cost =int(input()) if cost < 1000: print ('Скидок нет') elif cost < 2000: print('Скидка 2%') elif cost < 5000: print('Скидка 5%') else: print('Скидка 10%')</pre>	<p>1500</p> <p>Скидка 2%</p>

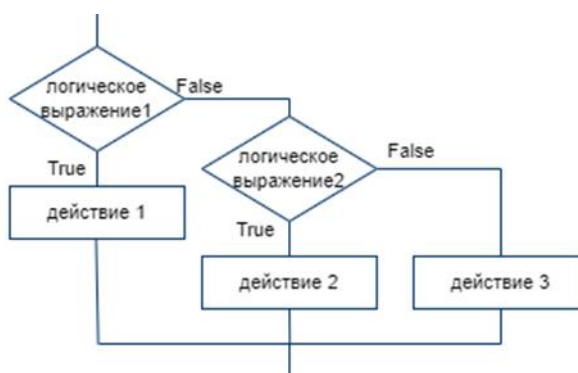


Рисунок 5 – Графическая интерпретация условного оператора

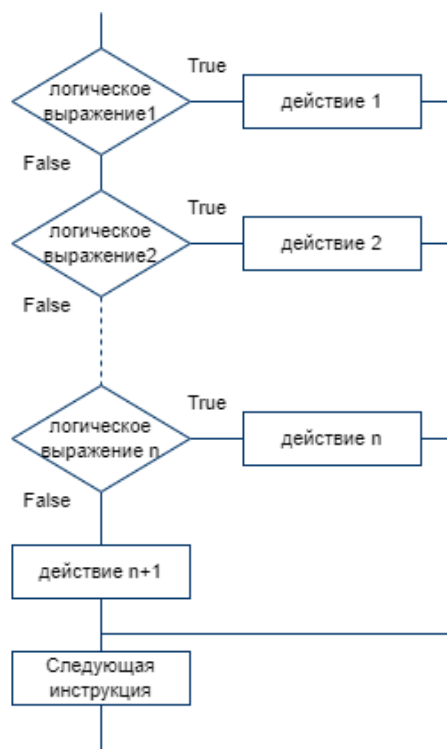


Рисунок 6 – Несколько условий в цепочке

Скидка составит 2%, хотя условие «cost < 5000» тоже выполняется. Вложенные условия проверяются сверху вниз до первого подходящего.

Задача 2. Пусть по введенным пользователем координатам x и y (целые числа) необходимо определить в какой из четвертей координатной плоскости находится точка.

Схема алгоритма решения задачи представлена на рисунке 7.

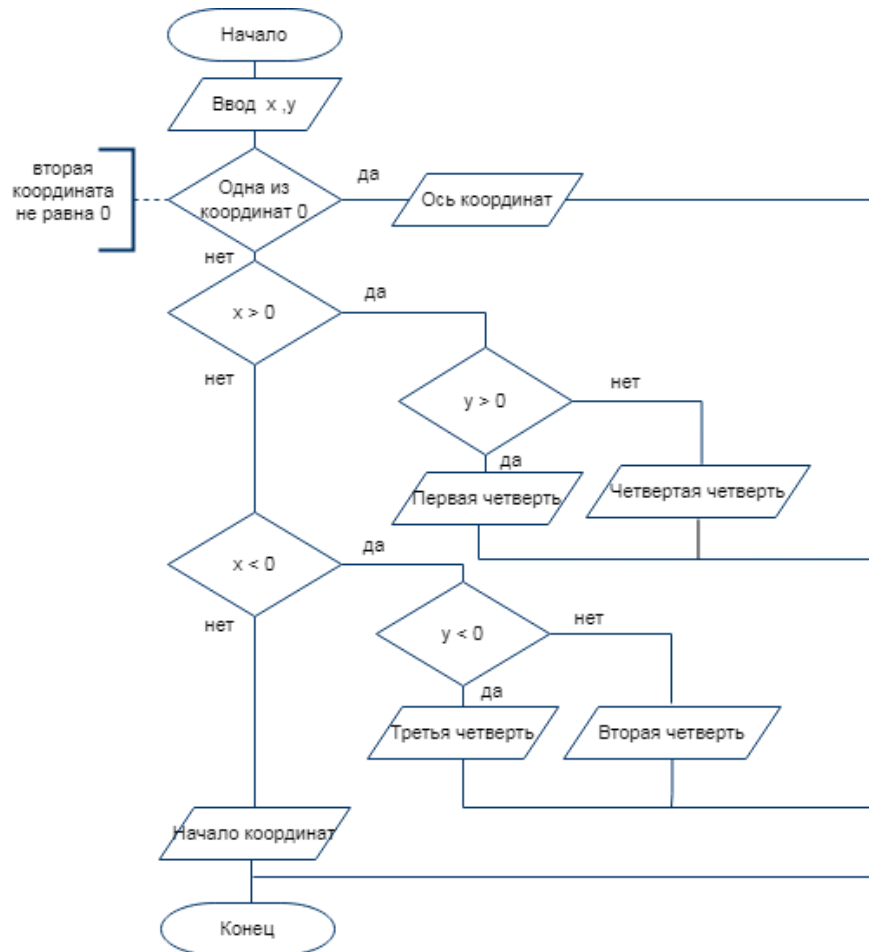


Рисунок 7 – Схема алгоритма решения задачи

Программа	На экране будет выведено
<pre> x = int(input()) y = int(input()) if y == 0 and x != 0 or x == 0 and y != 0: print('Ось координат') elif x > 0: if y > 0: print('Первая четверть') else: print('Четвертая четверть') elif x < 0: if y > 0: print('Вторая четверть') else: print('Третья четверть') </pre>	<p>-1</p> <p>3</p> <p>Вторая четверть</p>

<pre> print('Третья четверть') else: print('Начало координат') </pre>	
---	--

Некоторые замечания

В языке Python разрешены двойные неравенства, например.

if A < B < C: # Означает то же самое, что и if A < B and B < C:

В качестве условия в инструкции if можно использовать также:

– логические функции, то есть функции, возвращающие результат логического типа:

```

stroka = (input('Введите целое число '))
if stroka.isalpha():
    print('Это строка')
else:
    print('Это число')

```

– оператор in (оператор проверки принадлежности):

```

a = 5
if a in range(10):
    print('В диапазоне')

```

```

sim = input('Введите символ ')
s = input('Введите строку символов ')
if sim in s:
    print('В строке')

```

```

raduga = ['красный', 'оранжевый', 'желтый', 'зеленый', 'голубой', 'синий', 'фиолетовый']
color = 'зеленый'
if color in raduga:
    print('В списке raduga')

```

– операторы is/is not (операторы проверки идентичности).

Давайте попробуем

Задача 3. Даны три целых числа, каждое записано в отдельной строке. Выведите наибольшее из данных чисел.

Программа	На экране будет выведено
<pre> x = int(input()) y = int(input()) z = int(input()) if(x < y): x, y = y, x if(x < z): x, z = z, x print(x) </pre>	<pre> 2 6 1 6 </pre>

Задача 4. Даны три натуральных числа a , b , c , записанные в отдельных строках. Определите, существует ли невырожденный треугольник с такими сторонами. Если треугольник существует, выведите строку YES, иначе выведите строку NO.

Программа	На экране будет выведено
<code>x = int(input())</code>	3
<code>y = int(input())</code>	4
<code>z = int(input())</code>	5
<code>if(x < y):</code> <code>x, y = y, x</code>	YES
<code>if(x < z):</code> <code>x, z = z, x</code>	
<code>if(z + y > x):</code> <code>print('YES')</code>	
<code>else:</code> <code>print('NO')</code>	

Задача 5. Требуется определить, бьет ли ладья, стоящая на клетке с указанными координатами (номер строки и номер столбца), фигуру, стоящую на другой указанной клетке. Вводятся четыре числа: координаты ладьи (два числа) и координаты другой фигуры (два числа), каждое число вводится в отдельной строке. Координаты – целые числа в интервале от 1 до 8. Требуется вывести слово YES, если ладья сможет побить фигуру за 1 ход и NO – в противном случае.

Программа	На экране будет выведено
<code>x = int(input())</code>	1
<code>y = int(input())</code>	8
<code>x1 = int(input())</code>	2
<code>y1 = int(input())</code>	3
<code>if(x == x1 or y == y1):</code> <code>print('YES')</code>	NO
<code>else:</code> <code>print('NO')</code>	

Задача 6. Заданы координаты двух клеток шахматной доски (четыре целых положительных числа). Если клетки покрашены в один цвет, то выведите слово YES, а если в разные цвета – то NO.

Программа	На экране будет выведено
<code>x = int(input())</code>	1
<code>y = int(input())</code>	1
<code>x1 = int(input())</code>	4
<code>y1 = int(input())</code>	2
<code>if abs(x1 % 2 - y1 % 2) == abs(x % 2 - y % 2):</code> <code>print('YES')</code>	YES
<code>else:</code> <code>print('NO')</code>	

Задача 7. По данному числу n закончите фразу «На лугу пасется...» одним из возможных продолжений: « n коров», « n корова», « n коровы», правильно склоняя слово «корова». Дано число n ($n < 100$).

Программа	На экране будет выведено
<pre>x = int(input()) print('На лугу пасется ', x, end = ' ') if(4 < x % 100 < 21): print('коров') elif(x % 10 == 1): print('корова') elif(1 < x % 10 < 5): print('коровы') else: print('коров')</pre>	<p>5</p> <p>На лугу пасется 5 коров</p>

Задача 8. Даны шесть целых чисел a, b, c, d, e, f . Определить, сколько из них положительных и определить их сумму.

Программа	На экране будет выведено
<pre>sum = 0 k = 0 x = int(input()) if x > 0: k = k + 1 sum = sum + x x = int(input()) if x > 0: k = k + 1 sum = sum + x x = int(input()) if x > 0: k = k + 1 sum = sum + x x = int(input()) if x > 0: k = k + 1 sum = sum + x x = int(input()) if x > 0: k = k + 1 sum = sum + x print(k, sum)</pre>	<p>6</p> <p>-1</p> <p>2</p> <p>1</p> <p>-3</p> <p>0</p> <p>3 9</p>

Хорошо, что чисел в задаче только шесть: посмотрим на код. Он состоит из абсолютно одинаковых участков. Для того чтобы организовать многократное повторение некоторых действий, введены циклические процессы.

Эх раз, еще раз, еще много, много раз

Циклы являются мощнейшим инструментом, предоставляемым высокоуровневыми языками программирования. Идея циклов очень проста. Пока не выполнится какое-либо условие, его принято называть условием выхода из цикла, повторять какие-либо действия, называемые телом цикла. Например, программа может запрашивать у пользователя пароль для входа в систему до тех пор, пока не будет введено правильное сочетание символов, но не более трех раз.

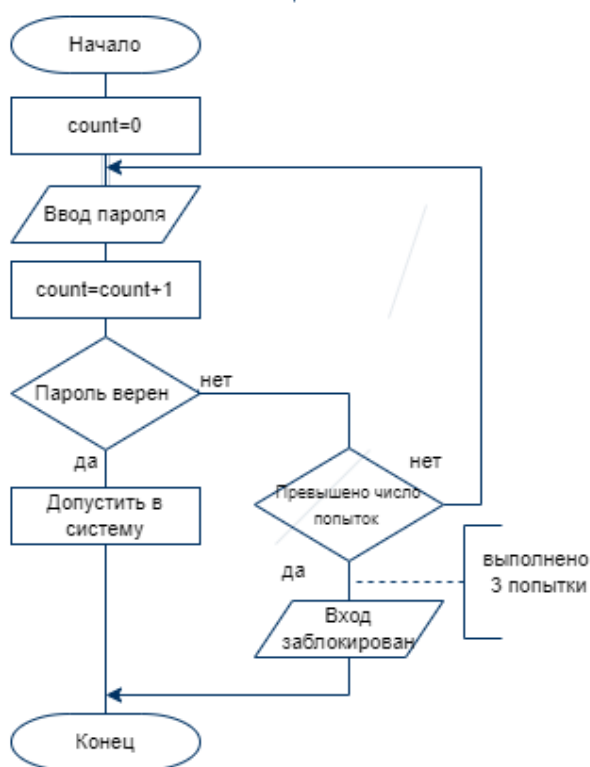


Рисунок 8 – Алгоритм запроса пароля

Именно такую функциональность обеспечивают циклы. Преимущества циклов заключается в следующем: нет необходимости повторять один и тот же фрагмент кода, можно перебирать элементы структур данных.

В Python есть два типа циклов.

- Цикл `for` – используется в тех случаях, когда некоторое действие необходимо выполнить определенное, известное перед его началом, число раз.
- Цикл `while` – используется, когда количество повторений заранее не известно, и тело цикла должно выполняться, пока не будет достигнуто определенное условие.

Введем некоторые понятия.

Тело цикла – последовательность инструкций, которую нужно выполнить несколько раз.

Итерация – однократное выполнение тела цикла.

Просматриваем все, что есть

В реальной жизни мы довольно часто сталкиваемся с циклическими процессами, когда необходимо последовательно выполнить однотипные действия с разными данными, например, выполнение задач на день, запланированные экзамены, покупка продуктов в магазине по списку. Действительно, когда мы приходим в магазин, то покупаем все, что есть в нашем списке.

Такие циклические процессы удобно реализовывать с помощью цикла for, особенность которого состоит, что при его выполнении происходит перебор элементов любой последовательности (списка, кортежа, строки, словаря и других) в том порядке, в котором они появляются, то есть совершается некоторое действие заранее известное число раз. Мы рассматривали перебор на прошлой лекции, когда говорили о словарях. В общем случае цикл for выглядит следующим образом.

for <переменная> in <последовательность>:
 <тело цикла>

Графическая интерпретация представлена на рисунке 9.

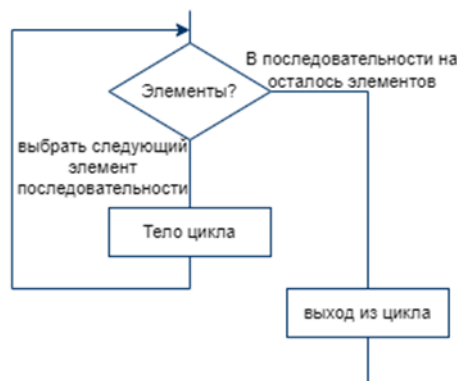


Рисунок 9 – Графическая интерпретация цикла for

Выведем на экран список дел, запланированных на выбранный пользователем день (вчера, сегодня, завтра), из прошлой лекции.

Программа	На экране будет выведено
<pre> tasks = [['вымыть окна', 'записаться на консультацию'], ['посмотреть фильм', 'сдать зачет'], ['навестить маму', 'сдать зачет']] day = int(input('Получить список дел на вчера - 1, на сегодня - 2, на завтра - 3 ')) count = 1 if day in (1, 2, 3): </pre>	<p>Получить список дел на вчера - 1, на сегодня - 2, на завтра - 3</p> <p>2</p> <p>1 . посмотреть фильм</p>

<pre> for i in tasks[day - 1]: print(count, '.', i) count += 1 else: print('Ошибка выбора') </pre>	2 . сдать зачет
--	-----------------

Для того чтобы автоматически получать номер элемента последовательности, можно воспользоваться функцией `enumerate`, которая создает кортеж на основании каждого элемента последовательности. Кортеж состоит из индекса элемента (начальное значение – ноль) и самого элемента.

Заменяем каждый шестой символ предложения на символ «\$».

Программа	На экране будет выведено
<pre> text = 'Как здорово, что все мы здесь сегодня собрались' new_text = '' for i, char in enumerate (text): if i % 6 == 0: new_text += '\$' else: new_text += char print(new_text) </pre>	\$ак зд\$рово,\$что в\$е мы \$десь \$егодн\$ собр\$лись

Отметим, что не желательно изменять элементы последовательности внутри тела цикла. Это может привести к трудно локализуемым ошибкам, избежать которых поможет предварительно созданная копия последовательности.

В качестве последовательности могут использоваться выражения различных типов. При этом на каждой итерации переменная будет принимать значение соответствующего типа.

Программа	На экране будет выведено
<pre> for i in 2, 1.5, 'cat', 2 + 2: print(i) </pre>	2 1.5 cat 4

Цикл `for` удобен, когда необходимо выполнить действие строго определенное число раз. Для этого можно использовать встроенную функцию `range()`.

Функция `range()` создает список из `n` элементов в заданном диапазоне, при этом в нее передается от одного до трех аргументов.

– `range(stop)` – указывает на то, что нужно проверить все числа от 0 и до `stop - 1`, например, `range (5)` – вернет объект диапазон, состоящий из пяти целых чисел: 0, 1, 2, 3, 4.

– `range(start, stop)` – перебрать нужно все числа, находящиеся между `start` и `stop`, например, `range(5, 10)` – вернет целые числа 5, 6, 7, 8, 9.

– `range(start, stop, step)` – сгенерируют список от `start` до `stop`, но с шагом, равным `step`, например, `range(1, 10, 2)` – вернет целые нечетные числа в диапазоне от 1 до 10.

Давайте выведем таблицу умножения для заданного пользователем числа.

Программа	На экране будет выведено
<pre>m = int(input(' Введите число для таблицы ')) for u in range (1, 11): print(m,'*', u, '=', m * u)</pre>	<p>Введите число для таблицы 6</p> <p>6 * 1 = 6</p> <p>6 * 2 = 12</p> <p>6 * 3 = 18</p> <p>6 * 4 = 24</p> <p>6 * 5 = 30</p> <p>6 * 6 = 36</p> <p>6 * 7 = 42</p> <p>6 * 8 = 48</p> <p>6 * 9 = 54</p> <p>6 * 10 = 60</p>

Python позволяет также создавать вложенные циклы, то есть использовать цикл внутри цикла. Как это работает? Сначала программа запустит внешний цикл и начнет его итерацию, затем перейдет во внутренний цикл, который будет выполняться до своего завершения. После чего продолжится выполнение итерации внешнего цикла. На следующей итерации будет вновь запущен внутренний цикл. Это будет происходить до тех пор, пока внешняя последовательность не завершится или не прервется.

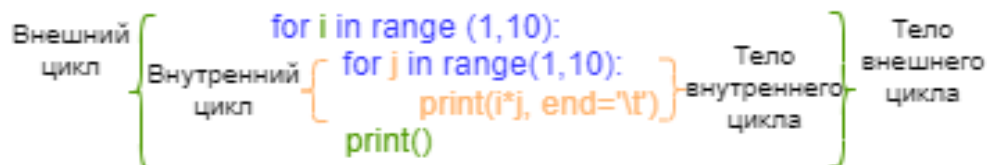


Рисунок 10 – Схематичное выполнение вложенного цикла

Перед Вами таблица Пифагора – результат выполнения кода из рисунка 10.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36

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

Еще одна задача – написать программу для вывода узора по образцу, используя вложенный цикл.

Образец:

```
*
**
***
****
***
**
*
```

Программа	На экране будет выведено
<pre>n = int(input()) for i in range(1, n + 1): if i < n // 2 + n % 2: for j in range(1, i + 1): print('*', end = '') else: for j in range(n - i): print('*', end = '') print()</pre>	<pre>10 * ** *** **** ***** **** *** ** *</pre>

Для управления циклом можно использовать специальные операторы break и continue. Оператор break осуществляет преждевременный выход из цикла, используется, когда в теле цикла возникает условие несовместимое с его дальнейшим выполнением (рисунок 11).

В ресторанном меню есть блюдо, которое Вы не любите, и как только встречаете его в предлагаемом списке, завершаете обед.

Программа	На экране будет выведено
<pre>menu = ['курица', 'рыба', 'овощи', 'свинина'] for dish in menu: if dish == 'рыба': print('Я не ем рыбу')</pre>	<pre>Отлично, принесите курица Я не ем рыбу Спасибо за обед</pre>

<pre>break print('Отлично, принесите ' + dish) print('Спасибо за обед')</pre>	
---	--

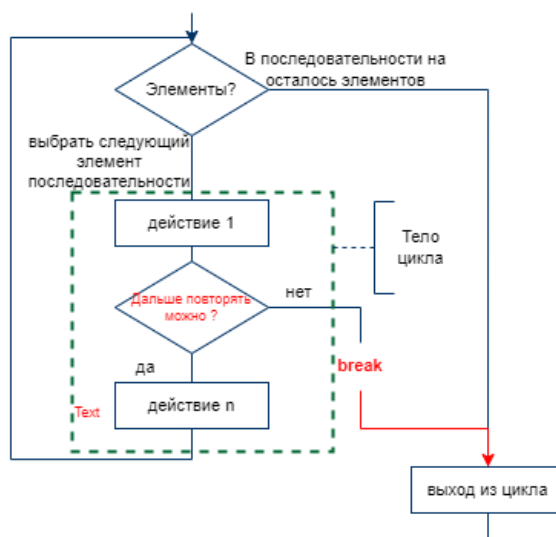


Рисунок 11 – Пример работы оператора break

Оператор continue осуществляет переход к следующей итерации цикла, все оставшиеся до конца тела цикла инструкции пропускаются (рисунок 12).

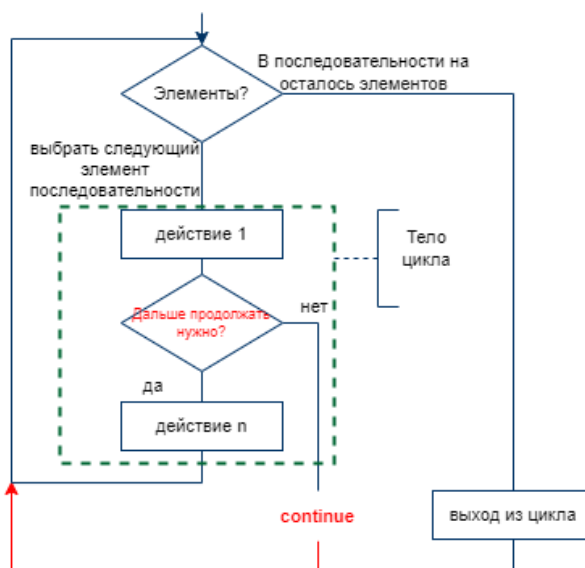


Рисунок 12 – Пример работы оператора continue

Вариант программы, когда вы пропускаете нелюбимое блюдо и продолжаете делать заказ дальше.

Программа	На экране будет выведено
<pre>menu = ['курица', 'рыба', 'овощи', 'свинина'] for dish in menu: if dish == 'рыба':</pre>	<p>Отлично, принесите курица</p> <p>Я не ем рыбу</p>

<pre>print('Я не ем рыбу') continue print('Отлично, принесите ' + dish) print('Спасибо за обед')</pre>	Отлично, принесите овощи Отлично, принесите свинина Спасибо за обед
--	---

В цикле for может быть дополнительный блок else, который выполняется, если элементы последовательности закончились и только в том случае, если не было преждевременного выхода из цикла с помощью оператора break.

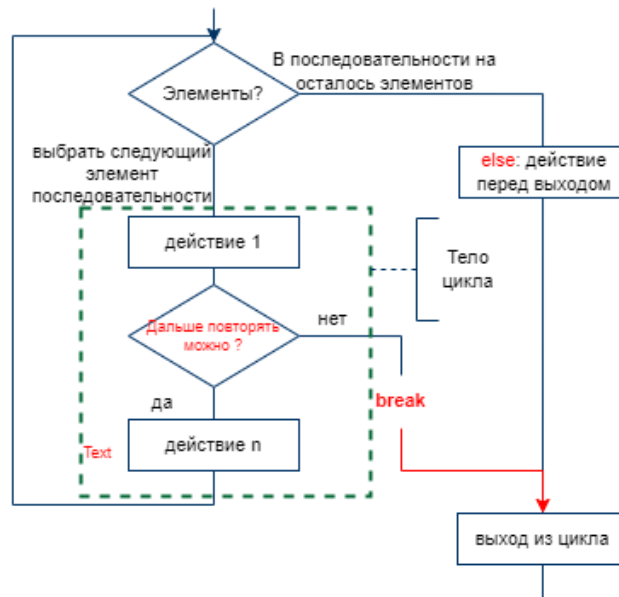


Рисунок 13 – Пример else в цикле for

Вернемся в ресторан.

Программа	На экране будет выведено
<pre>menu = ['курица', 'рыба', 'овощи', 'свинина'] for dish in menu: if dish == 'рыба': print('Я не ем рыбу') break print('Отлично, принесите ' + dish) else: print('Здорово, что в меню нет рыбы') print('Спасибо за обед')</pre>	Отлично, принесите курица Я не ем рыбу Спасибо за обед

Удалим рыбу из меню.

Программа	На экране будет выведено
<pre>menu = ['курица', 'овощи', 'свинина'] for dish in menu: if dish == 'рыба': print('Я не ем рыбу') break print('Отлично, принесите ' + dish)</pre>	Отлично, принесите курица Отлично, принесите овощи Отлично, принесите свинина Здорово, что в меню нет рыбы

<pre> else: print('Здорово, что в меню нет рыбы') print('Спасибо за обед') </pre>	Спасибо за обед
---	-----------------

Циклы очень важная конструкция языка программирования, именно они делают повторение простым, логичным и очень понятным.

Еще один пример

Давайте напишем игру «Виселица». Заведем константы, определяющие изображение виселицы за каждый неверный шаг.

PICTURE = ("",
 "")









Основы программирования на Python. Когда нужно сделать выбор

<pre> elif ans == word: print('ПОБЕДА это слово - ', ans) break print('Игра окончена') </pre>	<p>Задуманное слово --А--- --</p> <p>Введите букву a</p> <p>Такая буква уже была</p> <p>Задуманное слово --А--- --</p> <p>Введите букву</p>
---	--

Практическая работа

1) Определите количество трехзначных чисел, сумма цифр которых равна некоторому значению n.

2) Напишите программу для вывода следующего изображения

```

* * *
** *** **
*** ***** ***
* * *
** *** **
*** ***** ***

```

Контрольные вопросы

1) В каких случаях в программе используется полный вариант инструкции if? Как он оформляется? Как он «работает» (что происходит при его выполнении)? Нарисуйте схему выполнения.

2) Что представляет собой условие, записываемое в инструкции if в простейшем случае? Какие знаки операций сравнения (отношения) могут использоваться в нем?

3) Что является результатом операции сравнения?

4) Что такое сложное условие? Какие логические операции могут использоваться в нем?

5) Каков порядок (приоритет) выполнения логических операций? Как изменить этот порядок?

6) Что может быть использовано в инструкции if, кроме простых и сложных условий?

7) В каких случаях в программе используется неполный вариант инструкции if? Как он оформляется? Как он «работает» (что происходит при его выполнении)? Нарисуйте графическую схему выполнения.

8) В каких случаях в программе используется вложенная инструкция if? Как она оформляется? Как она «работает» (что происходит при ее выполнении)?