Вятский государственный университет
Цифровые кафедры
Разработка прикладного программного обеспечения

Когда нужно сделать выбор

Авторы
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

Дата 19.10.2022

Срезы

Срезы в Python – это механизм, с помощью которого извлекается подстрока по указанным параметрам: начальный индекс (start), конечный индекс (stop) и шаг (step), которые указываются через символ «:».

[start:stop:step]

индекс	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
S	0	1	2	3	4	5	6	7	8	9

S[1:8] ⇔ S[-9:-2]

Hello World! ⇔ Hello word!

```
msg = 'Hello World!'
msg = msg[:6]+"w"+msg[7:]
```

Пример получения года из даты

- Есть дата в таком формате: 12-10-2022.
- Как извлечь из этой строки подстроку, в которую входит только год?

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
1	**2**	**-**	**1**	**0**	**-**	**2**	**0**	**2**	**2**
0	1	2	3	4	5	6	7	8	9

Методы работы со строками

- Метод – это встроенная функция, которая работает с объектом определенного типа данных. Существуют строковые методы, методы для работы с целыми числами, методы списков, словарей и так далее.

объект.имя_метода(аргументы)

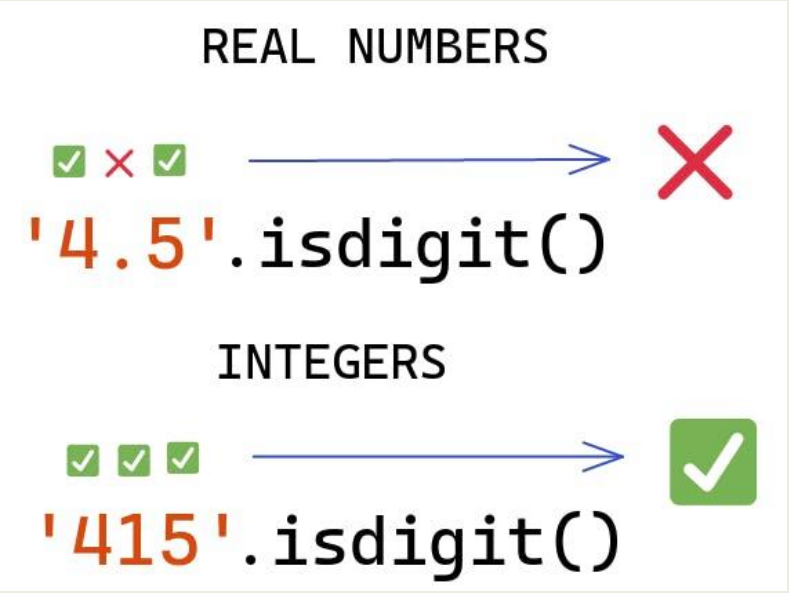
```
msg.upper( )
```

Основные методы для работы со строками

- `lower()` Преобразование строки к нижнему регистру
- `upper()` Преобразование строки к верхнему регистру
- `swapcase()` Переводит символы нижнего регистра в верхний, а верхнего – в нижний
- `find(sub[, start[, end]])` Возвращает индекс первого вхождения подстроки sub
- `replace(sub, new)` Заменяет подстроку sub на новую подстроку new
- `count(sub)` Возвращает число вхождений подстроки sub
- `strip([chars])` Удаление пробельных символов в начале и в конце строки, либо символов из параметра chars, если он передан.
- `split(sep[,maxsplit])` Разрезать строку (по пробельным символам)
- `'[char]'.join()` Склеивает все строки из переданного параметра, используя соединитель (возможно, пустой)

Проверка типа строки

- `islower()` Состоит ли строка из символов в нижнем регистре
- `isupper()` Состоит ли строка из символов в верхнем регистре
- `isalnum()` Состоит ли строка из букв и цифр
- `isalpha()` Состоит ли строка из букв
- `isdigit()` Состоит ли строка из цифр



Тайный агент

Вам пришла шифровка.
Необходимо расшифровать.

Известно, что необходимо заменить в тексте все цифры 1 на слово 'раз', развернуть последовательность, заключенную между первым и последним вхождением буквы 'р' в обратном порядке.

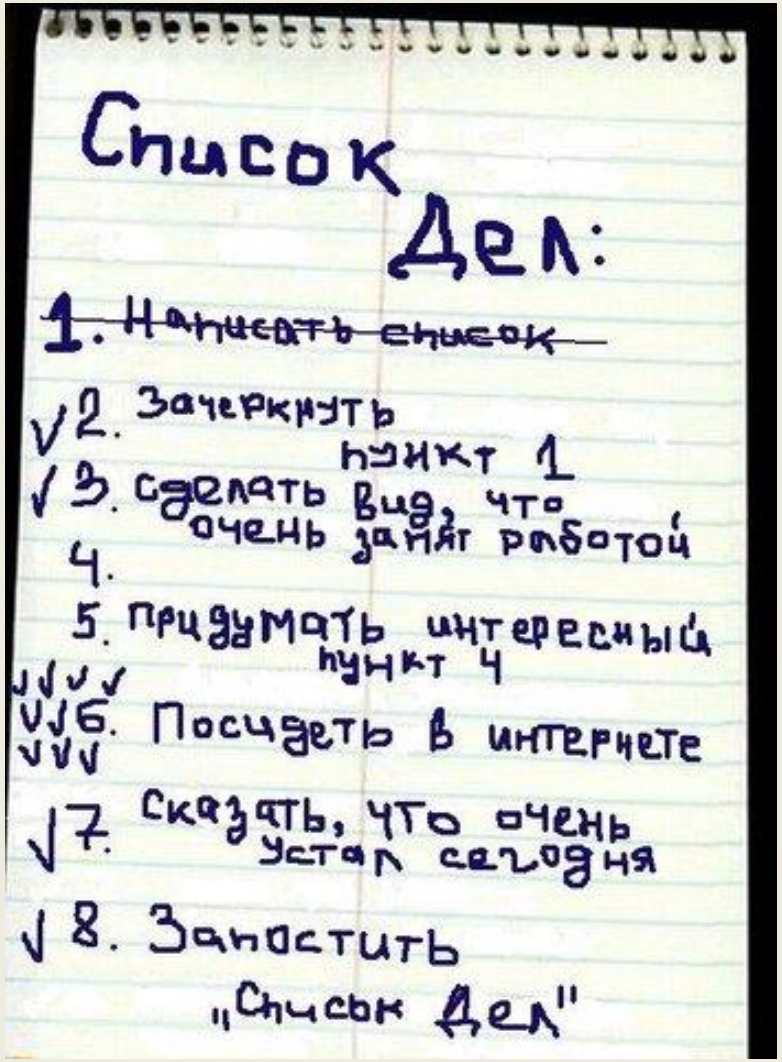
Составляем список дел

- Чтобы Python понял, что имеет дело со списком, нужно заключить все элементы в квадратные скобки ([]). Сами элементы разделяются запятыми.
- Мы можем создать пустой список

```
tasks = []
```

- Записать в него первую задачу

```
tasks = ['написать список’]
tasks[0] = 'зачеркнуть пункт 1'
```



Списки

Список в Python может содержать элементы любого типа, в том числе другие списки в качестве элементов. Это называется вложенностью, а подобные списки – вложенными. Например список дел на сегодня:

```
tasks = ['посмотреть фильм', 'сдать зачет']
```

```
print(tasks[0]) ⇔  посмотреть фильм
```

А на сегодня и на завтра.

```
tasks = [['посмотреть фильм', 'сдать зачет'], ['навестить маму', 'купить книгу']]
```

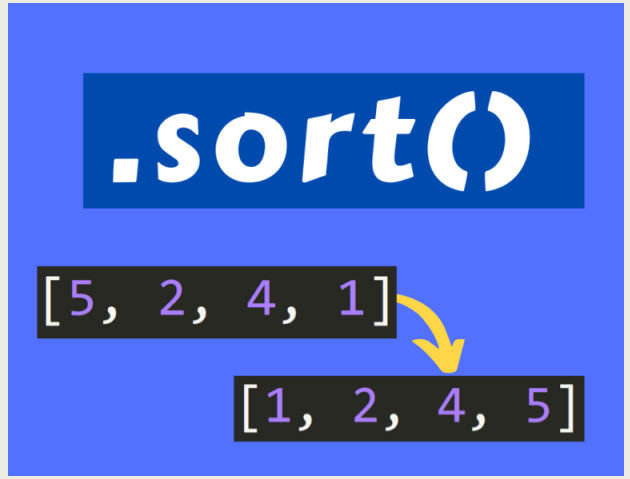
Тогда каждый элемент будет иметь два индекса: номер во внешнем и номер во внутреннем списке.

```
print(tasks[0][1]) ⇔ сдать зачет
```

Встроенные функции

- min() возвращает минимальный элемент.
- max() возвращает максимальный элемент.
- sorted() упорядочивает элементы списка

(sorted(s, reverse=True) ⇔ создан новый список, отсортированный в обратном порядке)



Встроенные функции

- Считывать и удалять последний элемент списка.

```
last_element = tasks.pop()
```

- Или элемент с указанным индексом

```
not_needed = tasks[1].pop(0)
```

- Если нам не нужен доступ к значению удаляемого элемента

```
del tasks[0]
```

- Если мы передумали выполнять задачу с известным значением

```
tasks[0].remove('сдать зачет')
```

Встроенные функции

- `append()` Добавляет элемент в конец списка
- `insert()` Вставляет элемент в указанное место списка
- `remove()` Удаляет элемент по значению
- `pop()` Удаляет последний элемент, либо элемент с указанным индексом
- `clear()` Очищает список (удаляет все элементы)
- `copy()` Возвращает копию списка
- `count()` Возвращает число элементов с указанным значением
- `index()` Возвращает индекс первого найденного элемента
- `reverse()` Меняет порядок следования элементов на обратный
- `sort()` Сортирует элементы списка

Срезы для списков

```
lst = ['Сергиев Посад', 'Владимир', 'Суздаль', 'Ярославль',
'Кострома', 'Иваново', 'Ростов Великий', 'Переславль-Залесский']
```

Второй и третий день путешествия

`lst[1:3]` ⇔ `['Владимир', 'Суздаль']`.

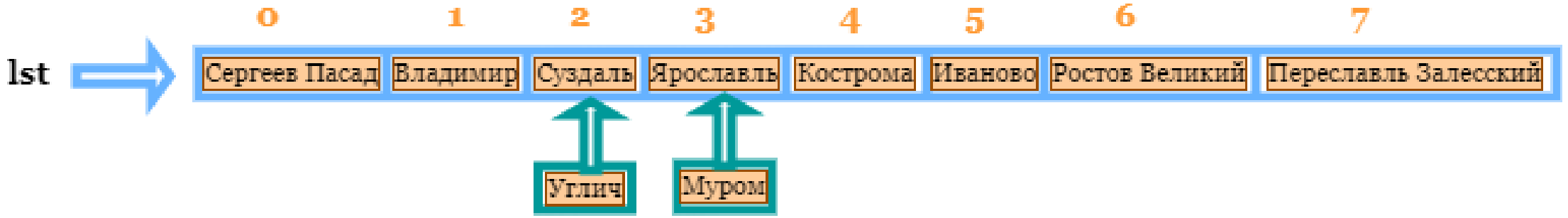
Где проведем предпоследний день тура?

`lst[-2:-1]` ⇔ `['Ростов Великий']`

Сменим маршрут: `lst[2:5] = 'Углич', 'Муром'`

`lst1=lst` - Не копия

Вот копия `lst1=lst[:]`



Словари

- Словарь – это неупорядоченная последовательность произвольных объектов с доступом по ключу – ассоциативный список.
- Пусть необходимо связать каждый город из маршрута с телефонным

```
dict1 = {'Сергиев Посад': 49654, 'Владимир': 4922,
         'Углич': 48532, 'Муром': 49234,
         'Кострома': 4942, 'Иваново': 4932,
         'Ростов Великий': 48536, 'Переславль-Залесский': 48535}
```

Ограничения

- Данные, представляющие собой ключ словаря, должны быть уникальны внутри множества ключей этого словаря. Не должно быть двух одинаковых ключей.
- Ключ должен быть объектом неизменяемого типа, то есть строкой, числом или кортежем.

На значения нет никаких ограничений. Они не обязаны быть ни уникальными, ни неизменяемыми. Зная ключ, мы всегда можем определить соответствующее ему значение.

```
print(Dict1['Владимир' ]) # Результат 4922
print('Владимир' in Dict1)  #True
print('Москва' in Dict1) #False
```

Методы работы со словарем

- `get(key [,d])` Возвращает значение ключа key. Если key не существует, возвращает d (по умолчанию None)
- `items()` Возвращает список кортежей словаря в формате (ключ, значение)
- `keys()` Возвращает список с ключами словаря
- `values()` Возвращает список со значениями словаря

Подведем итог

- Вам поручили написать модуль RPG игры с простой системой прокачки персонажа. На старте персонаж обладает минимальным списком сопротивлений. По ходу игры ему случайным образом выпадают заклинания, позволяющие пополнить их список.
- Что использовать?
  - список магических заклинаний ограничен (**кортеж**).
  - заклинание связно с резистом (**словарь**).
  - то что имеет и получает персонаж можно хранить вместе (**список**).

Полезные модули и библиотеки

В составе Python есть множество полезных модулей и библиотек, функции из которых можно использовать в своей программе.

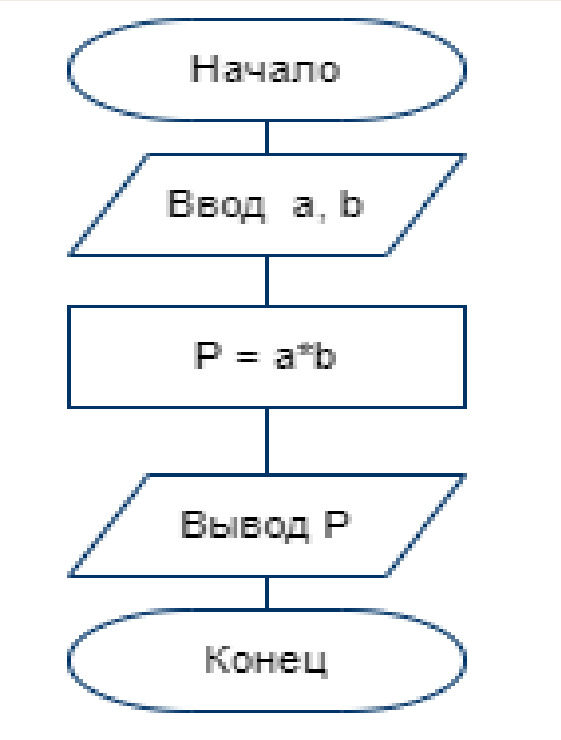
Мы уже говорили о модуле math, содержащем математические методы. Сейчас нам может потребоваться функция генерации случайных значений.

Набор таких функций реализован в модуле random.

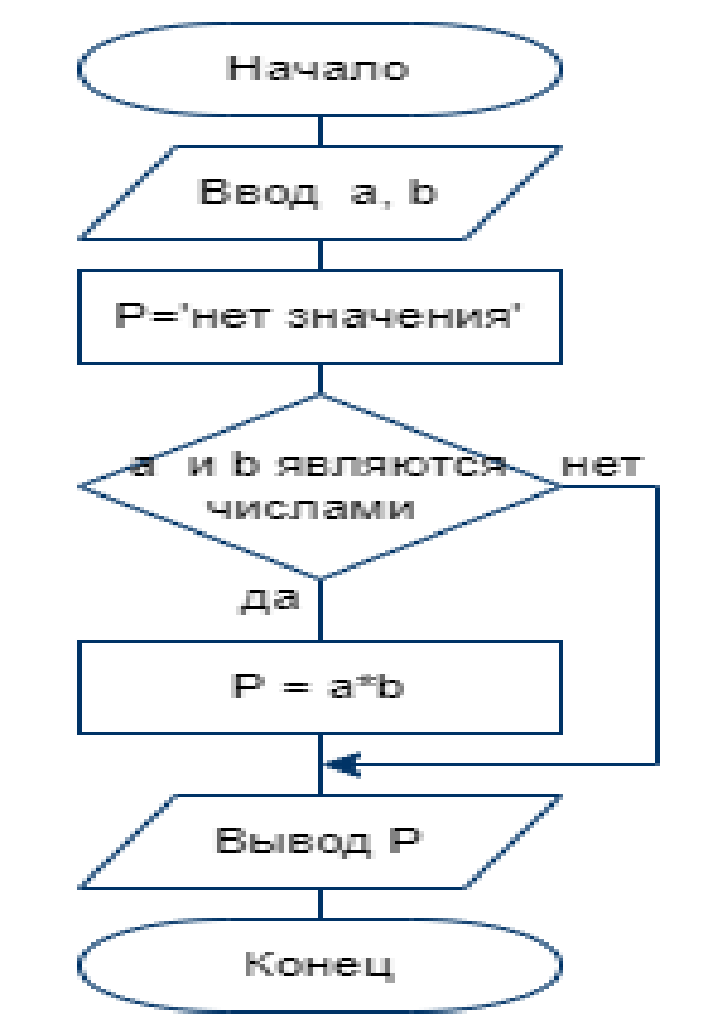
- Функция randint(start, end), генерирует случайное целое число в заданном диапазоне
- Функция randrange(n), генерирует случайное целое число со значением от 0 до n - 1.

Поговорим о выборе

Поиск площади прямоугольника



Линейный алгоритм

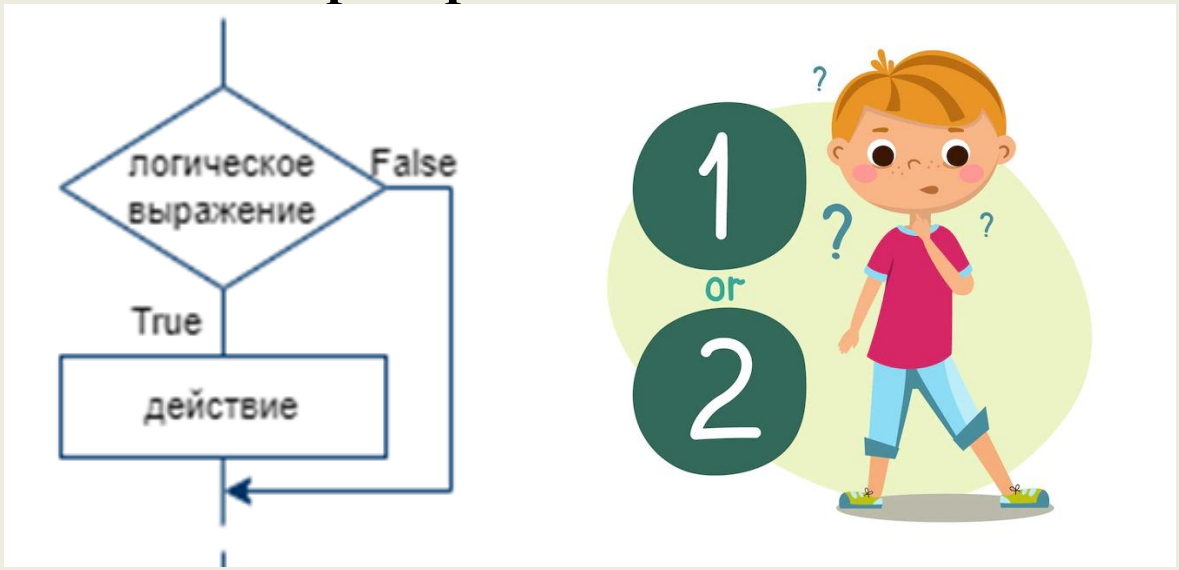


Разветвленный алгоритм с дополнительной проверкой

Разветвленный алгоритм

Позволяет выбрать дальнейшего решения только одну ветвь в зависимости от значения заданного логического условия

if <условное выражение>:
оператор



Действие выполняется лишь в том случае, если логическое выражение истинно

Все дело в отступах

if < условное выражение>:
    действие 1
    ...
    действие n
<инструкция следующая за if>

False
True

К истинным значениям в Python относятся:

- любое число не равно нулю,
- любая непустая строка,
- любой непустой объект.

Логическое выражение

Логическое выражение может быть достаточно сложным – содержать операторы сравнения и логические операторы, которые используются для сравнения значений и конструирования сложных логических выражений соответственно.

Оператор	Пример
>	x > y
<	x < y
==	x == y
!=	x != y
>=	x >= y
<=	x <= y

Оператор	Пример
and	x and y
or	x or y
not	not x

Високосный год

Попробуем по введенному пользователем значению года определить является ли он високосным.

Високосный год в соответствии с григорианским календарем, год является високосным, если его **номер кратен 4**, но **не кратен 100**, а так же если он **кратен 400**.

Операция с оператором and выполняется раньше, чем операция с оператором or.

a > 0 and b <= 0 and c <= 0 or

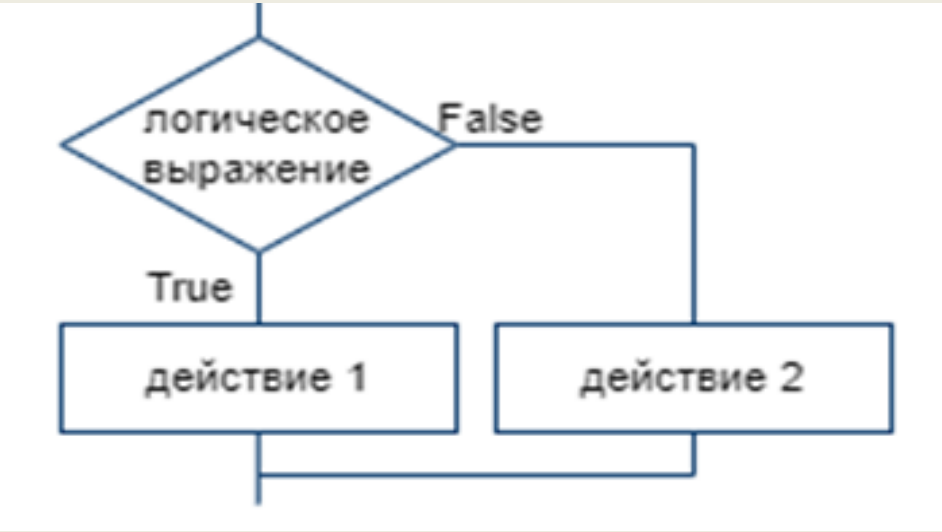
b > 0 and a <= 0 and c <= 0 or c > 0 and a <= 0 and b <= 0

Истинно если только одно из них положительное.

Полный условный оператор

Мини-Википедия. У нас есть список знаменитостей, о каждом из которых имеется информация, например, год и страна рождения, страна проживания, сфера деятельности, самое большое достижение и так далее, причем не у всех набор информации одинаков. Напишем программу, которая по запросу пользователя будет выводить имеющуюся о человеке информацию.

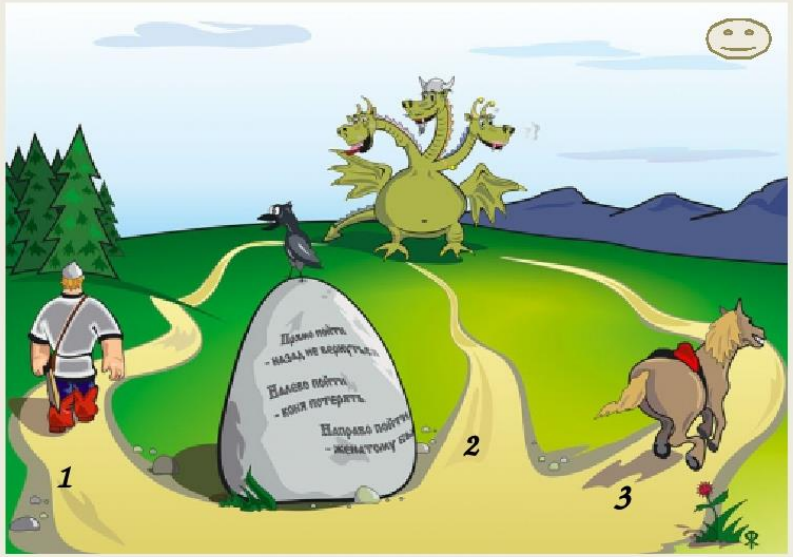
```
if <условное выражение>:
        оператор 1
else:
        оператор 2
```



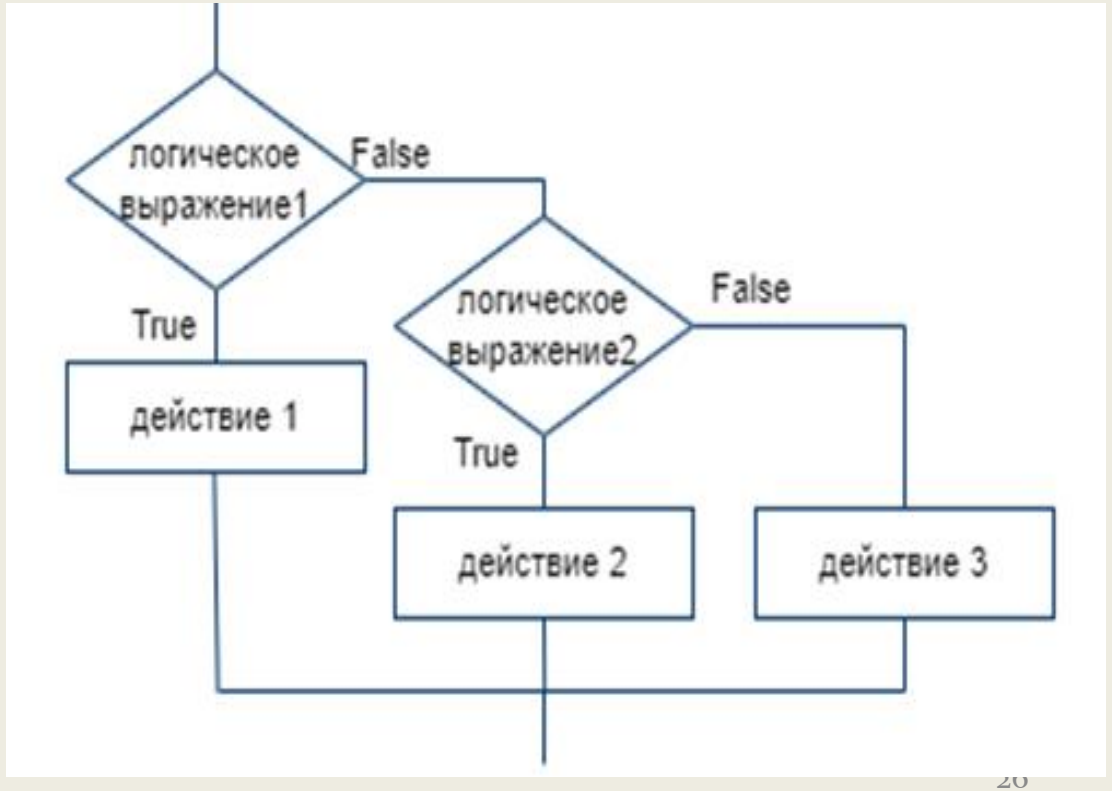
Действие 1 выполняется, если логическое выражение истинно
Действие 2 выполняется, если логическое выражение ложно

Вложенный условный оператор

Когда существует более двух ветвей развития алгоритма можно использовать связанные условия (вложенные условия).

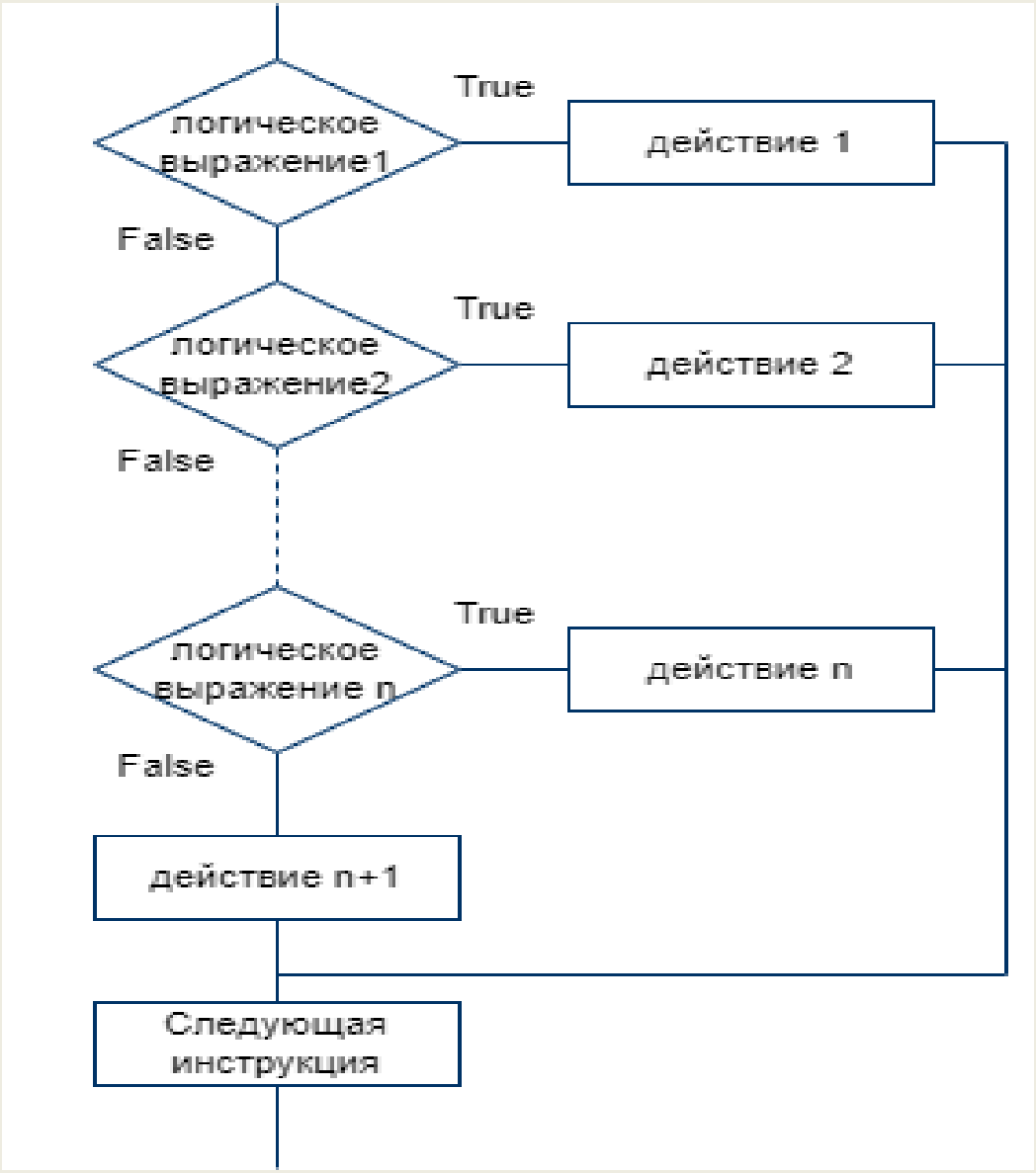



```
if <условное выражение1>:
    оператор 1
elif <условное выражение2>:
    оператор 2
else:
    оператор 3
```



Вложенный условный оператор

Если в цепочке if-elif-elif-… истинными являются несколько условий, то «срабатывает» первое из них



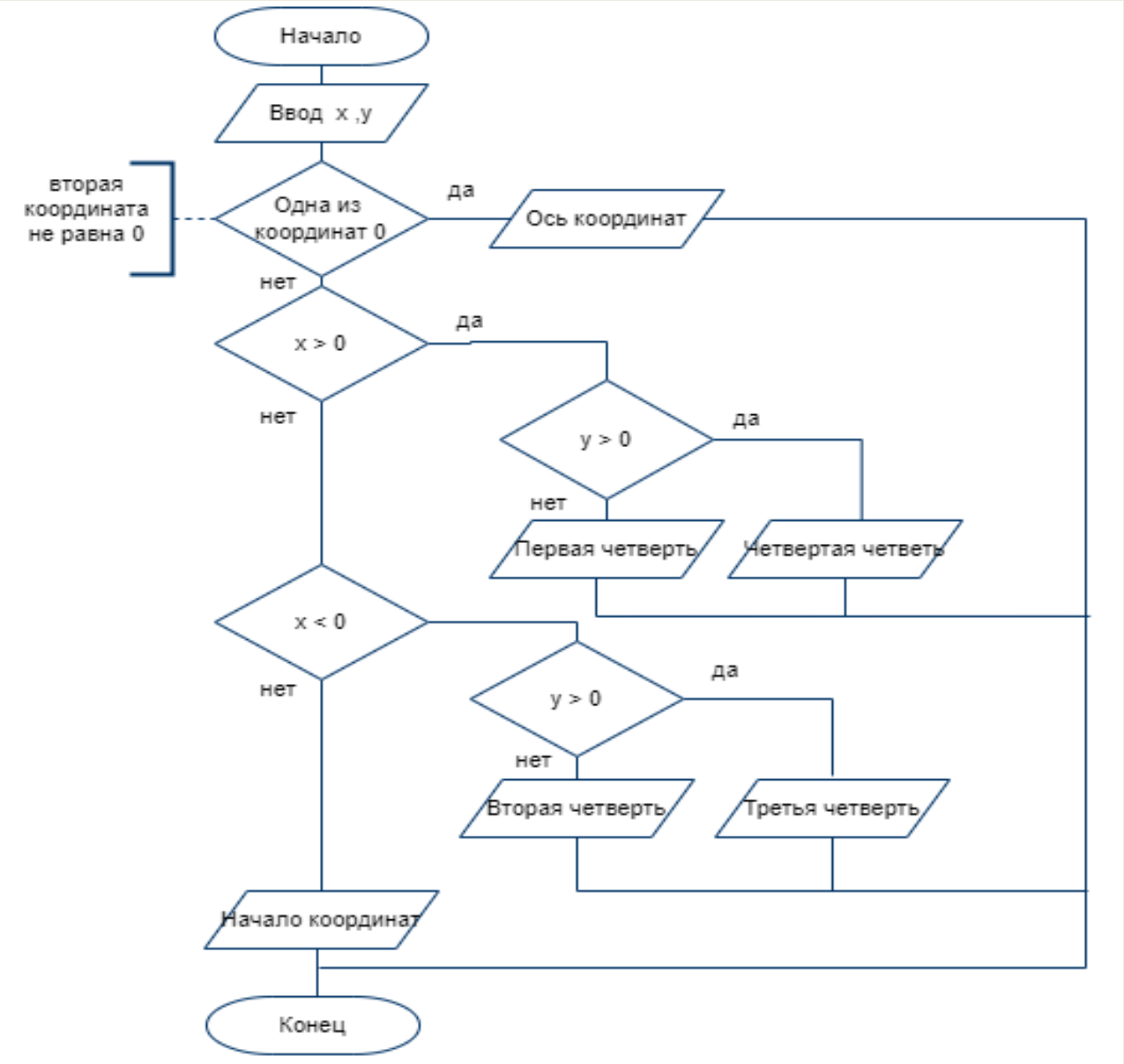
Давайте попробуем

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%.

В какой мы четверти?

По введенным пользователям координатам x и y (целые числа) необходимо определить в какой из четвертей координатной плоскости находится точка.



Некоторые замечания

- В языке Python разрешены двойные неравенства

  if A < B < C: ⇔ if A < B and B < C:
- В качестве условия в инструкции if можно использовать любой оператор или функцию возвращающую результат логического типа

Еще одна задача

По данному числу n закончите фразу «На лугу пасется...» одним из возможных продолжений: «n коров», «n корова», «n коровы», правильно склоняя слово «корова».

Число n < 100.

Подведем итоги

- Мы познакомились с **управляющей структурой**.

Это составной операторы, которые изменяет порядок выполнения операторов программы

- Мы разобрали оператор if, который позволяет **условно** выполнять набор действий на основе оценки данных программы.

Он может быть сокращенный, полный и вложенный

- Мы узнали, что все дело в отступах.

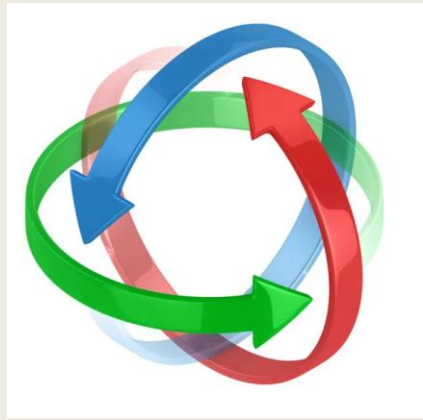
Если два оператора находятся на одном уровне отступа, то они являются частью одного и того же блока

Посчитаем

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

Хорошо, что чисел в задаче только шесть)))

Для того чтобы организовать многократное повторение некоторых действий, введена еще одна управляющая конструкция – циклический процесс.



Эх раз, еще раз, еще много, много раз

Циклы – мощнейший инструмент, предоставляемым высокоуровневыми языками программирования.

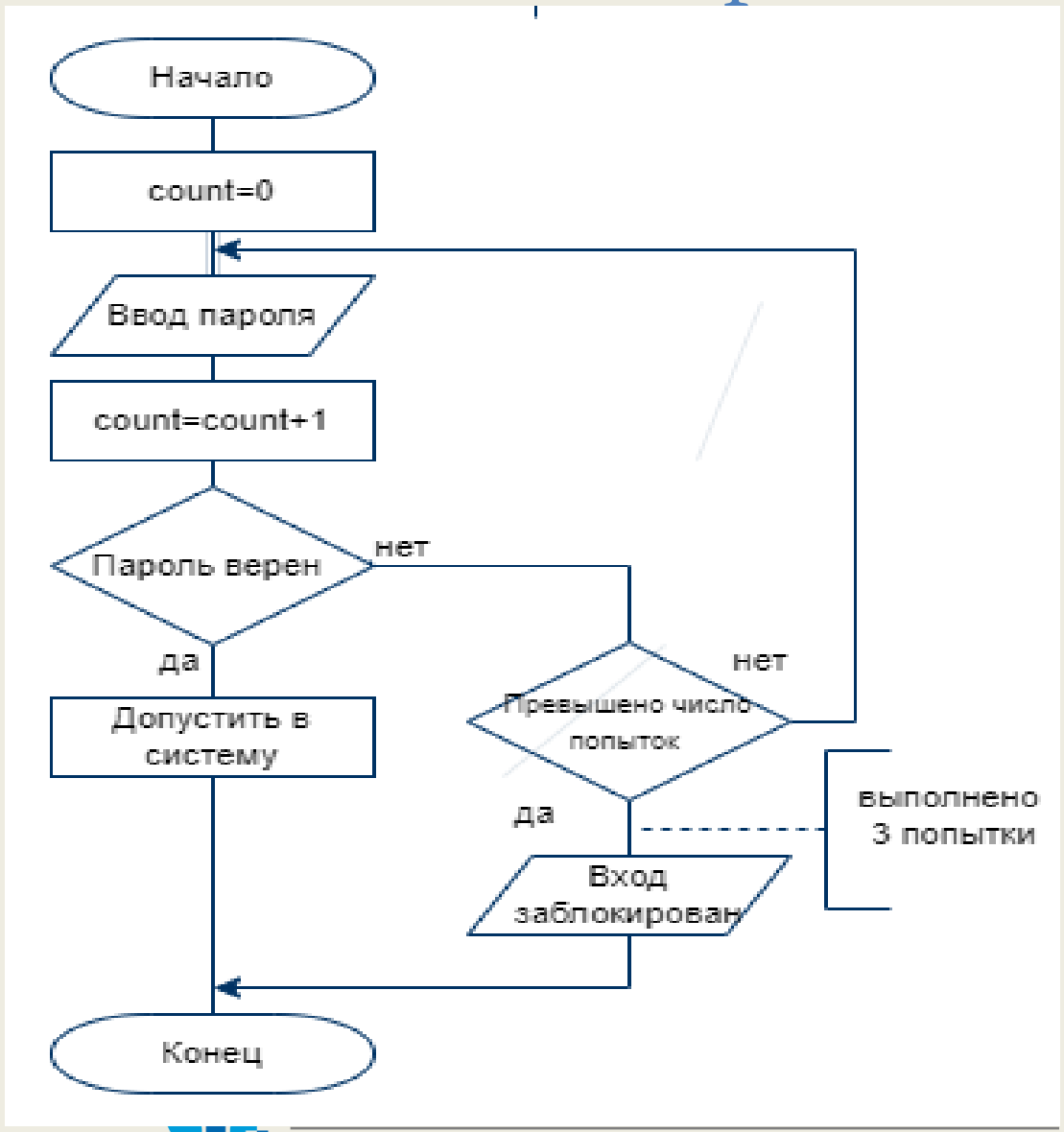
Идея циклов:

Пока не выполнится какое-либо условие (**условием выхода из цикла**), повторять какие-либо действия (**тело цикла**).

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

Итерация – однократное выполнение тела цикла.

Алгоритм запроса пароля



Преимущества циклов:

- нет необходимости повторять один и тот же фрагмент кода,
- можно перебирать элементы структур данных.

Два типа циклов в Python

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.

Просматриваем все что есть

Мы довольно часто сталкиваемся с цикличными процессами, например, выполнение задач на день, запланированные экзамены, покупка продуктов в магазине по списку.

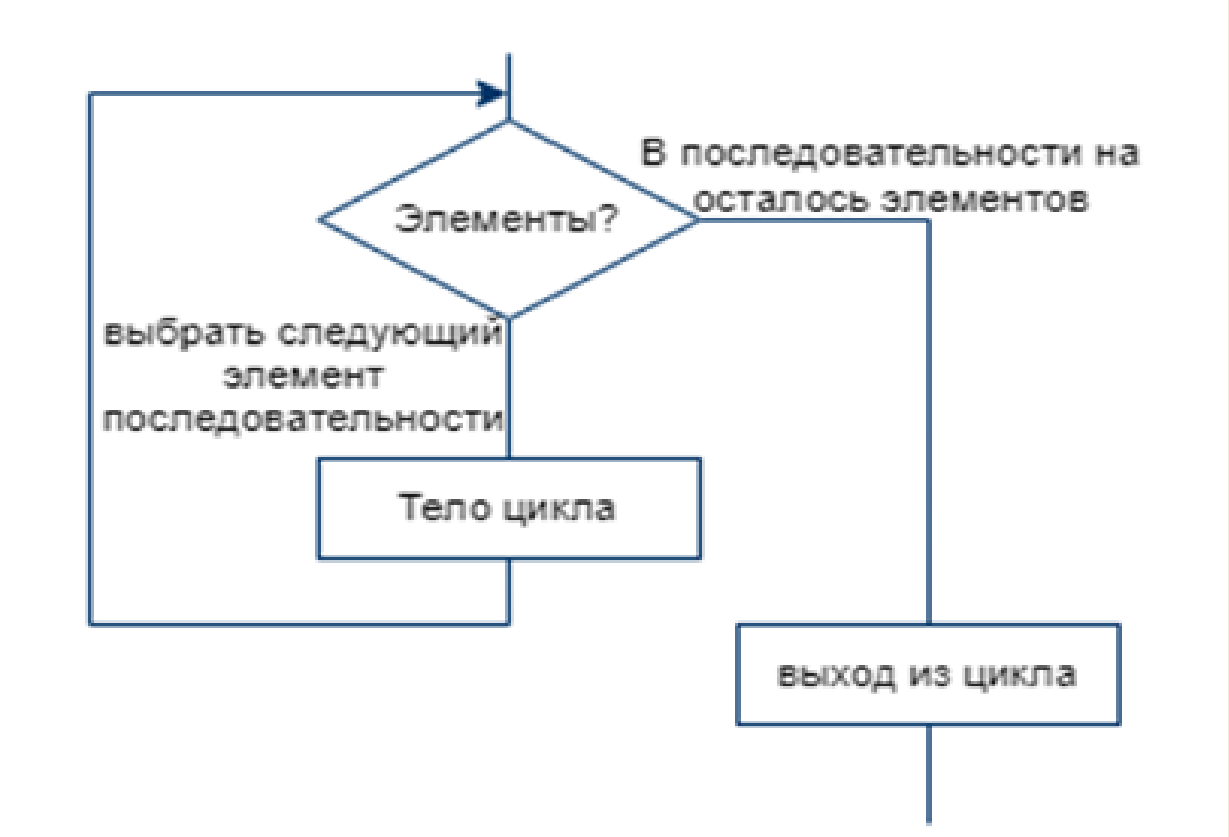
Такие циклические процессы удобно реализовывать с помощью **цикла for**.

Особенность:
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.

Перебор последовательности – цикл с известным числом повторений

for <переменная> in <последовательность>:

    <тело цикла>



Давайте попробуем

Выведем на экран список дел, запланированных на выбранный пользователем день (вчера, сегодня, завтра)

Еще задача

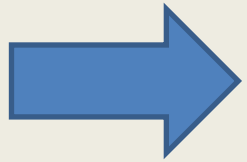
Заменим каждый шестой символ предложения на символ «$».

Для того чтобы автоматически получать номер элемента последовательности используют функцию ***enumerate()***, которая создает кортеж состоящий из индекса элемента (начальное значение – ноль) и самого элемента.

Замечание

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

```
for i in 2, 1.5, 'cat', 2 + 2:
        print(i)
```



```
2
1.5
cat
4
```

Работаем в диапазоне

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

- range(stop) –нужно проверить все числа от 0 и до stop -1

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

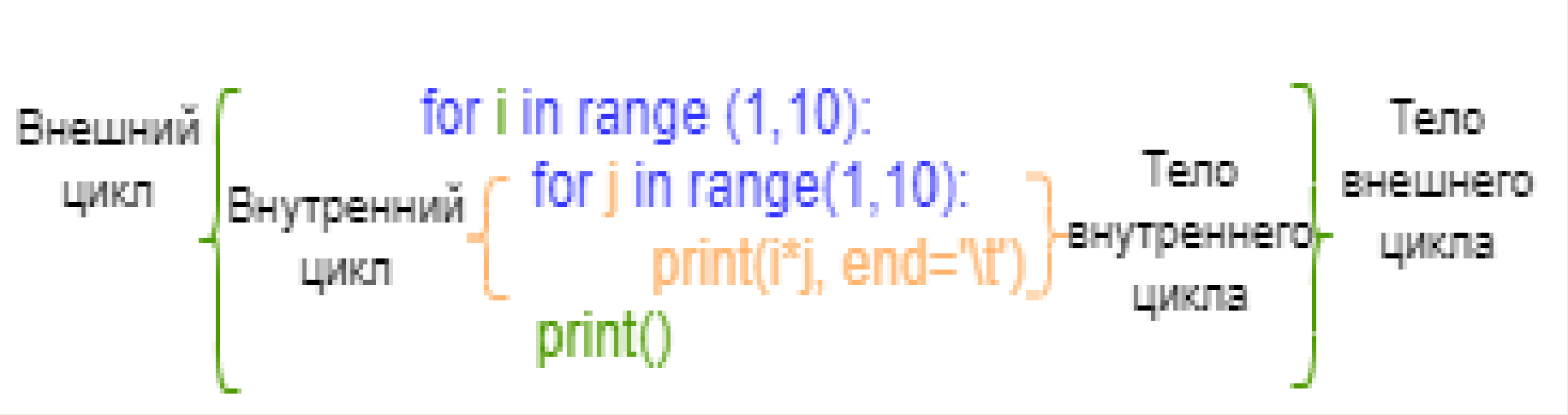
range (5, 10) – вернет целые числа 5, 6, 7, 8, 9

- range(start, stop, step) – сгенерируют список от start до stop, но с шагом, равным step

range (1, 10, 2) – вернет целые нечетные числа в диапазоне от 1 до 10

Напечатаем таблицу умножения

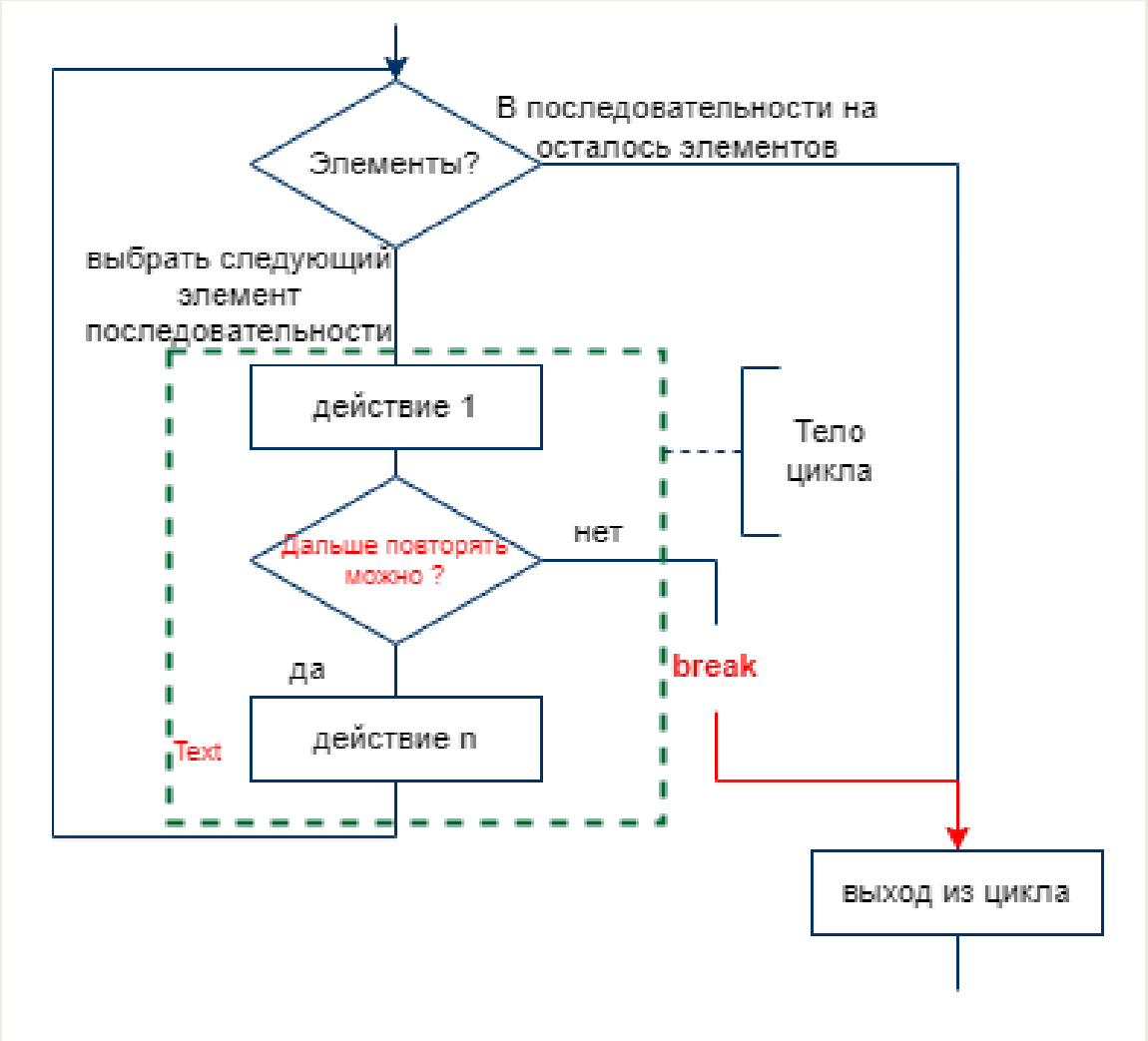
Вложенные циклы



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

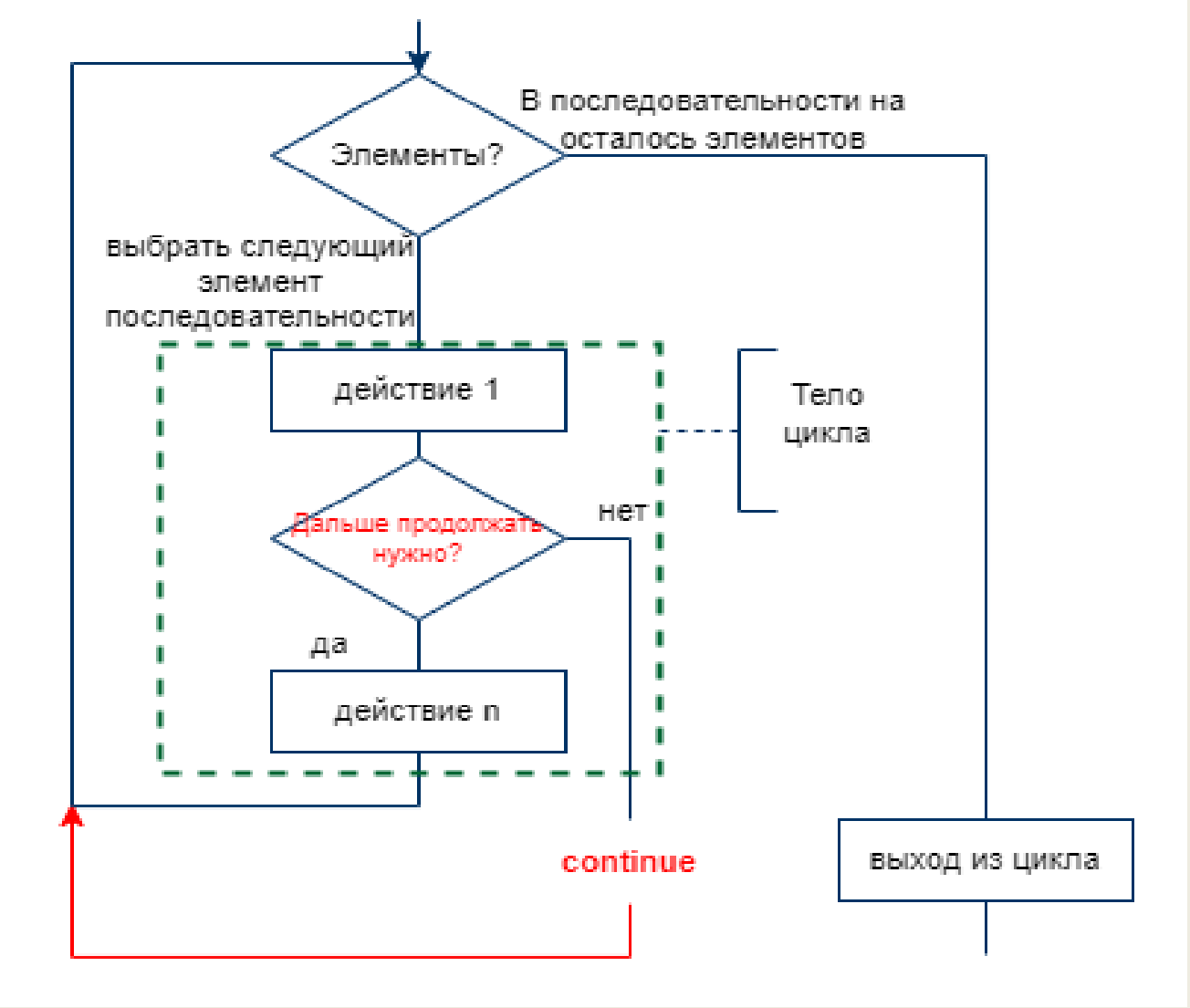
Управляем циклами

Оператор ***break*** осуществляет преждевременный выход из цикла, используется, когда в теле цикла возникает условие несовместимое с его дальнейшим выполнением



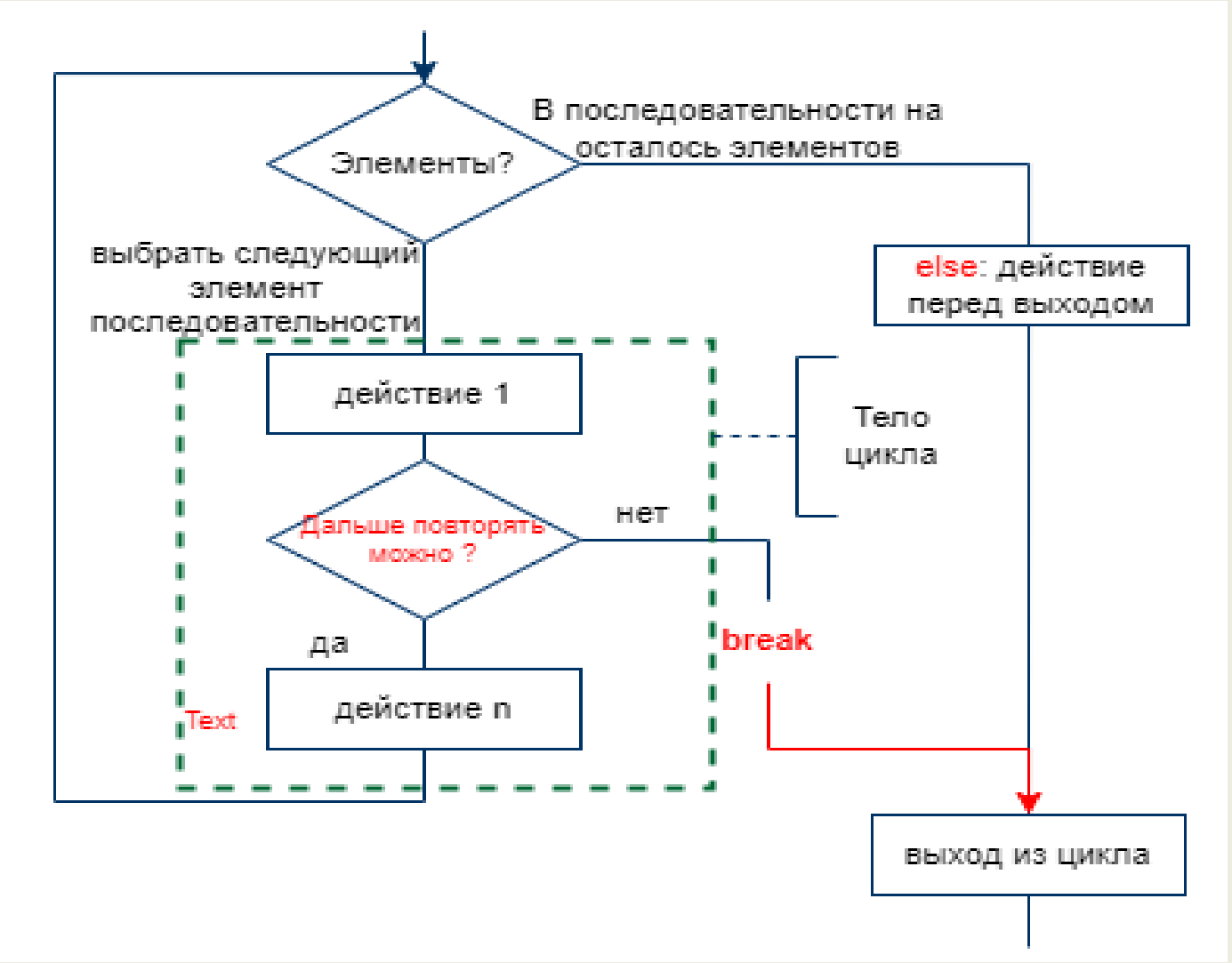
Управляем циклами

Оператор ***continue*** осуществляет переход к следующей итерации цикла, все оставшиеся до конца тела цикла инструкции пропускаются



Управляем циклами

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.***



Подведем итог

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

For – цикл перебора последовательностей и цикл с известным числом повторений

Циклом можно управлять через break /continue

Давайте поиграем

Напишем игру в слова «Виселица»

Продолжение следует
Спасибо за внимание