

Алгоритмизация и программирование

**Преподаватель:
Кадырова Гульнара
Ривальевна**

Структура условного выбора

Команда **if-elif-else**

Рассмотрим задачу расчета площади прямоугольника.

Алгоритм решения:

- ввод двух размеров прямоугольника,
- расчет площади
- **ВЫВОД ОТВЕТА НА ЭКРАН**

```
File Edit Format Run Options Window Help
```

```
a = float(input('Введите первую сторону пмяоугольника -> '))
b = float(input('Введите вторую сторону пмяоугольника -> '))
s = a*b
print('Площадь пмяоугольника со сторонами',a,'и',b,'=', '% .2f' % s)
|
```

```
===== RESTART: E:/ПИТОН/Прямоугольник.py
Введите первую сторону пмяоугольника -> 5.4
Введите вторую сторону пмяоугольника -> 7.3
Площадь пмяоугольника со сторонами 5.4 и 7.3 = 39.42
>>> |
```

В данной программе обязательно должны выполняться все действия, указанные в ней.

В программах возможны ситуации, когда требуется выполнение не строго определенного действия, а одного из двух или более вариантов действия, при этом выбор того или иного варианта зависит от некоторого условия.

Возможны ситуации, когда некоторое действие должно выполняться не всегда, а только при определенном условии.

Два варианта действий

Пример. Дано целое число. Определить, является ли оно четным.

Решение:

Начальная часть программы решения задачи:

```
a = int(input('Введите целое число '))
```

Далее возможны два варианта действий (два варианта инструкции print() вывода информации на экран):

- 1) `print('Это число четное')`
- 2) `print('Это число нечетное')`

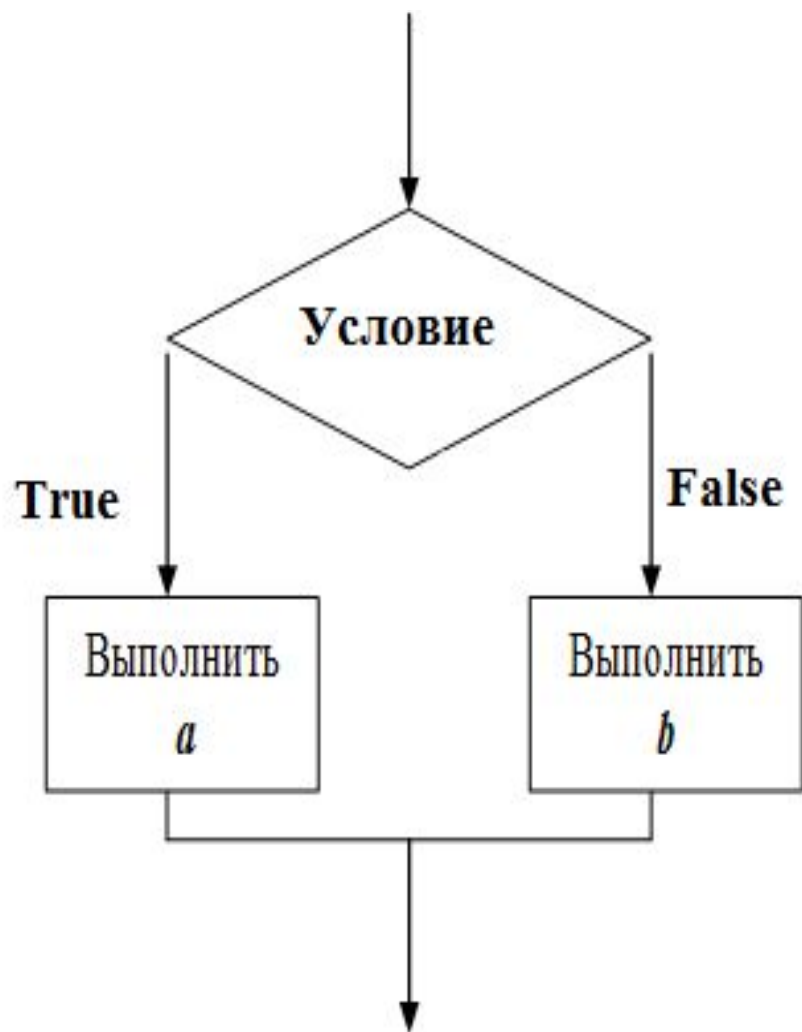
В таких случаях (возможны два варианта действий) в программе необходимо использовать инструкцию `if` в следующей форме:

if *<условие>*:

<Действия 1-го варианта (1-я серия инструкций)>

else:

<Действия 2-го варианта (2-я серия инструкций)>



В Python предусмотрено несколько возможных способов выражения логических значений: логическая константа False, 0, неопределенное значение Python None и пустые значения (например, пустой список [] или пустая строка "") — все эти значения интерпретируются как **False**. Логическая константа True и все остальные значения интерпретируются как **True**.

Для создания **логических условий** используются **операторы сравнения** (<, <=, ==, >, >=, !=, is, is not, in, not in) и **логические операторы** (and, not, or); все они возвращают **либо True, либо False**.

В языке Python есть 6 операций сравнения:

	Знак операции	Означает
1	<	Меньше
2	<=	Меньше либо равно
3	>	Больше
4	>=	Больше либо равно
5	==	Равно
6	!=	Не равно

Примеры:

$a < 0$

$x \leq a + b$

$(a + b)/2 > 0$

$2 * c \geq 7 * d - 1$

Применительно к рассмотренному примеру инструкция `if` оформляется так:

```
if a % 2 == 0:
```

```
    print('Это число четное')
```

```
else:
```

```
    print('Это число нечетное')
```

Вместо условия `a % 2 == 0` можно записать `not a % 2`.

`not a % 5` – *True* для кратных 5

False для не кратных 5

Задача. Даны два вещественных числа a и b . Если первое больше второго, то увеличить каждое число в 2 раза, иначе – уменьшить в два раза

Сложные условия

В инструкции `if` возможно также использование так называемых «сложных условий» – состоящих из двух или нескольких простых условий, соединенных логическими операторами **and** (И), **or** (ИЛИ) или **not** (НЕ).

Результат выполнения логических операций с оператором **and** над двумя простыми условиями `C1` и `C2`:

<code>C1</code>	<code>C2</code>	<code>C1 and C2</code>
<code>False</code>	<code>False</code>	<code>False</code>
<code>False</code>	<code>True</code>	<code>False</code>
<code>True</code>	<code>False</code>	<code>False</code>
<code>True</code>	<code>True</code>	<code>True</code>

Результат выполнения логических операций с оператором **or** над двумя простыми условиями C1 и C2:

C1	C2	C1 or C2
False	False	False
False	True	True
True	False	True
True	True	True

Результат выполнения логической операции с оператором **not** над условием C определяется так:

C	not C
False	True
True	False

Пример. Предположим, что компания набирает сотрудников, возраст которых – от 25 до 50 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: «Подходит» или «Не подходит» он по этому признаку.

Пусть возраст сотрудника задан и записан в переменной *age*. Тогда фрагмент программы, в котором выводится ответ, будет выглядеть так:

```
if age >= 25 and age <= 50:
```

```
    print('Подходит')
```

```
else:
```

```
    print('Не подходит')
```

Приоритет операций:

1. Операции сравнения (<, <=, >, >=, ==, !=)
2. Not
3. And
4. Or

Для изменения приоритета используют круглые скобки.

Иногда условия получаются достаточно длинными, и их хочется перенести на следующую строку. Сделать это в Python можно двумя способами:

- использовать обратный слэш („\“):

```
if v < 400 and v != 2 and v != 3 and v != 12 and \
v != 13 and v != 22 and v != 23:
```

...

- взять все условие в скобки (перенос внутри скобок разрешен):

```
if (v < 400 and v != 2 and v != 3 and v != 12 and v != 13
and v != 22 and v != 23):
```

...

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:

1) логические функции, то есть функции, возвращающие результат логического типа .

`Chet()` – функция, возвращающая результат `True`, если ее параметр (значение, указанное в скобках) является четным числом, и `False` – в противном случае.

```
n = int(input('Введите целое число '))
```

```
if Chet(n):
```

```
    print('Это число четное')
```

```
else:
```

```
    print('Это число нечетное')
```

2) оператор **in** (оператор проверки принадлежности), который проверяет, принадлежит ли некоторый объект (число, символ, переменная и т. п.) набору значений (списку, строке, диапазону чисел и т. п.):

а) **a = 3**

```
if a in range(10): #Если a попадает в диапазон 0...9
```

б) **sim = input('Введите символ ')**

```
s = input('Введите строку символов ')
```

```
if sim in s:      #Если символ sim имеется в строке s
```


3) операторы `is/is not` (операторы проверки идентичности), которые определяют, ссылаются ли (или не ссылаются) две переменные на один и тот же объект.

Конечно, можно комбинировать простые (или сложные) условия с логическими функциями, операторами `in` и `is/is not`, а также с логическими константами `True` и `False`.

Один, но не обязательный вариант действий

Пример. Даны три целых числа, среди которых есть отрицательные. Вывести на экран отрицательные числа на одной строке.

Используем в программе три переменные – a , b и c .

Начальная часть программы решения задачи:

```
print('Введите 3 целых числа')  
a = int(input())  
b = int(input())  
c = int(input())  
print('Среди них отрицательные:')
```

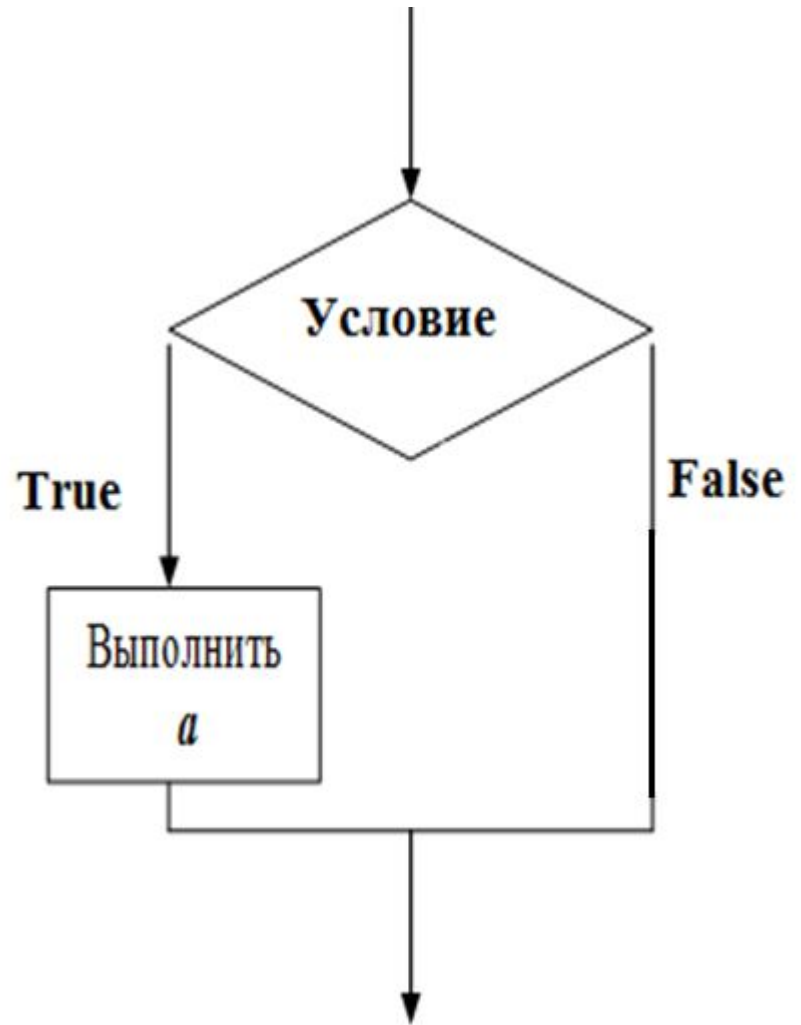
В каком случае выводить значение первого числа a ? Если $a < 0$. Аналогично и для двух других

В таких случаях (когда какие-то действия в программе выполняются не всегда, а только при определенном условии) используется инструкция **if** в следующей форме:

if *<условие>*:

<Действия (серия

Такой вариант инструкции **if** называют *инструкцией с полным вариантом*».



Применительно к рассмотренному примеру завершающая часть программы решения задачи оформляется так:

```
if a < 0:
```

```
    print(a, end = '')
```

```
if b < 0:
```

```
    print(b, end = '')
```

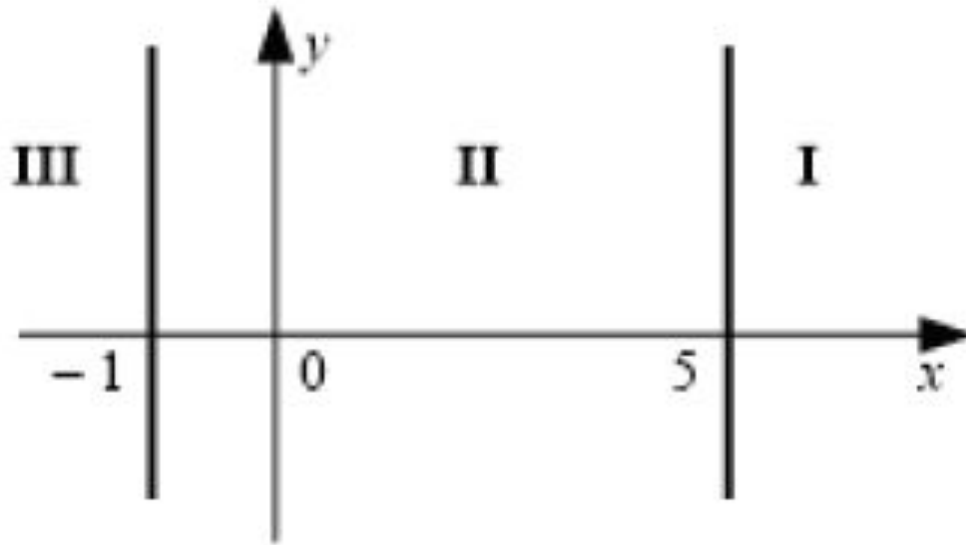
```
if c < 0:
```

```
    print(c)
```

Еще раз обратим внимание на то, что нет альтернативных вариантов действий, как в случае, рассмотренном ранее, а есть единственно возможный, но не обязательный, вариант.

Три и более вариантов действий

Рассмотрим задачу: «На плоскости выделены три зоны (I, II, III).



Дана координата x точки. Определить, в какую зону попала эта точка. Принять, что x не равно границам зон (-1 и 5)».

Решение

Можно в программе использовать 3 неполных варианта инструкции **if** (без ветви **else**):

```
if x < -1:
```

```
    print('В зону III')
```

```
if x > 5:
```

```
    print('В зону I')
```

```
if x > -1 and x < 5:
```

```
    print('В зону II')
```

Можно ли «сэкономить» одно слово **if** и использовать полный вариант инструкции во втором **if** ?

```
if x < -1:  
    print('В зону III')  
if x > 5:  
    print('В зону I')  
else:  
    print('В зону II')
```

Проверим.

При $x == 10$ ответ будет правильным.

При $x == 0$ – то же.

При $x == -8$ на экран будет выведено:

В зону III

В зону II

то есть ответ будет ошибочным.

Правильный вариант:

```
if x < -1:  
    print('В зону III')  
else:  
    if x > 5:  
        print('В зону I')  
    else:  
        print('В зону II')
```

Получается, что одна инструкция if «вложена» в другую.

В Python можно записать так:

```
if x < -1:  
    print('В зону III')  
elif x > 5:  
    print('В зону I')  
else:  
    print('В зону II')
```


Ветвь **elif** может присутствовать несколько раз;
наличие ветви **else** необязательно:

if ...:

...

elif ...:

...

elif ...:

...

elif ...:

...

Если в цепочке if-elif-elif-... истинными являются
несколько условий, то «срабатывает» **первое из
НИХ!**

Например, программа:

```
if cost < 1000:
```

```
    print ('Скидок нет')
```

```
elif cost < 2000:
```

```
    print('Скидка 2%')
```

```
elif cost < 5000:
```

```
    print('Скидка 5%')
```

```
else:
```

```
    print('Скидка 10%')
```

при `cost == 1500` выводит на экран:

Скидка 2%

хотя условие `cost < 5000` тоже выполняется.

в общем случае:

if *<условие 1>*:

<действия 1>

elif *<условие 2>*:

<действия 2>

elif *<условие 3>*:

<действия 3>

elif ...:

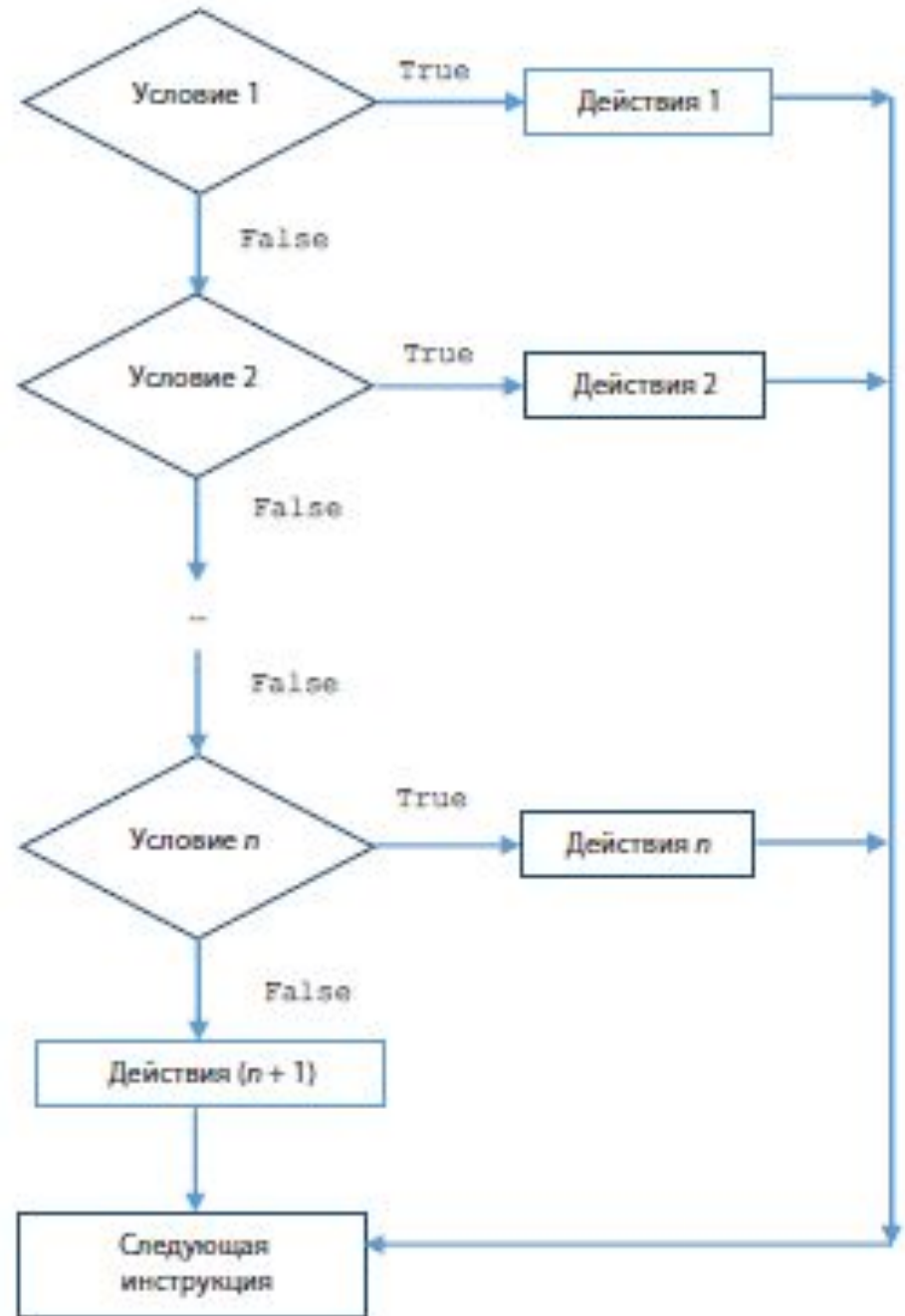
...

elif *<условие n>*:

<действия n>

else:

<действия (n + 1)>



Рассмотрим задачу, которая часто встречается в программировании: «Определить максимальное значение из двух заданных различных вещественных чисел».

Решение

Заданным переменным дадим имена a , b и максимальному из них - Max .

1. Так как возможны два варианта ответа, то в программе можно использовать полный вариант инструкции **if**:

if $a > b$:

$Max = a$

else:

$Max = b$

2. Можно использовать неполный `if`:

```
Max = a
```

```
if b > a:
```

```
    Max = b
```

3. В Python есть стандартная функция `max()`, которая возвращает максимальное значение среди указанных для нее аргументов:

```
Max = max(a, b)
```

Команда if-elif-else

Выполняется блок кода после первого истинного условия (в if или elif). Если ни одно условие не равно True, то выполняется блок кода после else:

```
x = 5
if x < 5:
    y = -1
    z = 5
elif x > 5:
    y = 1
    z = 11
else:
    y = 0
    z = 10
print(x, y, z)
```

Присутствие секций **elif** и **else** не обязательно, а количество секций **elif** не ограничено.

Для выделения блоков используются отступы. Каждый блок состоит из одной или нескольких команд, разделенных символами новой строки. Все эти команды должны снабжаться отступами одного уровня. (5 0 10)

Часть тело после команды `if` является обязательной. Впрочем, в нее можно включить команду `pass` (как и в любой точке Python, где нужна команда). Команда `pass` размещается там, где должна находиться команда, но никаких действий она не выполняет:

```
if x < 5:
```

```
    pass
```

```
else:
```

```
    x = 5
```

В Python не существует конструкции `case` (или `switch`). Там, где в других языках использовалась бы команда `case` или `switch`, Python обычно прекрасно обходится цепочкой `if... elif... elif... else`. Если же цепочка становится слишком громоздкой, обычно можно воспользоваться словарем функций

Управляющие конструкции (if-elif-else, циклы while и for) являются составными командами и в них используются блоки и отступы.

Python использует отступы для определения границ блоков (или тел) управляющих конструкций.

секция составной команды:

блок

секция составной команды:

блок

Составная команда состоит из одной или нескольких секций, за каждой из которых следует блок с отступом. Составные команды также могут находиться в блоках, как и любые другие команды. В этом случае они создают вложенные блоки.

В одной строке можно разместить сразу несколько команд, разделив их символом «точка с запятой» (;).

Блок, содержащий одну строку, может быть размещен в той же строке после двоеточия (:), завершающего секцию составной команды:

```
>>> x = 1; y = 0; z = 0
>>> if x>0: y = 1; z = 10
else: y = -1

>>> print(x, y, z)
1 1 10
>>>
```

Что касается базового интерактивного режима и оболочки Python IDLE, после внешнего уровня отступов необходима дополнительная строка. Дополнительная строка не нужна при размещении кода в файле программы.

Контрольные вопросы

1. В каких случаях в программе используется полный вариант инструкции if-else? Как он оформляется? Как он «работает»? (Что происходит при его выполнении?) Нарисуйте графическую схему выполнения.
2. Что представляет из себя условие, записываемое в инструкции if в простейшем случае? Какие знаки операций сравнения 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?

Контрольные вопросы

7. Какие варианты использования команды `if` существуют? Их отличия? Как они записываются?
8. Как работает полный вариант команды `if-elif-else`?
9. Сколько блоков `elif` может быть в команде?
10. Какие команды в Python относятся к составным командам?