Тема 8. Логические выражения и операторы

1. Логические выражения и логический тип данных

Часто в реальной жизни мы соглашаемся с каким-либо утверждением или отрицаем его. Например, если вам скажут, что сумма чисел 3 и 5 больше 7, вы согласитесь, скажете: «Да, это правда». Если же кто-то будет утверждать, что сумма трех и пяти меньше семи, то вы расцените такое утверждение как ложное.

Подобные фразы предполагают только два возможных ответа – либо "да", когда выражение оценивается как правда, истина, либо "нет", когда утверждение оценивается как ошибочное, ложное. В программировании и математике если результатом вычисления выражения может быть лишь истина или ложь, то такое выражение называется логическим.

Например, выражение 4 > 5 является логическим, так как его результатом является либо правда, либо ложь. Выражение 4 + 5 не является логическим, так как результатом его выполнения является число.

На позапрошлой лекции мы познакомились с тремя типами данных – целыми и вещественными числами, а также строками. Сегодня введем четвертый – логический тип данных тип bool. Его также называют булевым. У этого типа всего два возможных значения: True правда и False ложь.

```
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = False
>>> type(b)
<class 'bool'>
```

Здесь переменной a было присвоено значение True, после чего с помощью встроенной в Python функции type() проверен ее тип. Интерпретатор сообщил, что это переменная класса bool. Понятия «класс» и «тип данных» в данном случае одно и то же. Переменная b также связана с булевым значением.

В программировании False обычно приравнивают к нулю, а True – к единице. Чтобы в этом убедиться, можно преобразовать булево значение к целочисленному типу:

```
>>> int(True)
1
>>> int(False)
0
```

Возможно и обратное. Можно преобразовать какое-либо значение к булевому типу:

```
>>> bool(3.4)
True
>>> bool(-150)
True
>>> bool(0)
False
>>> bool(' ') True
>>> bool('')
False
```

И здесь работает правило: всё, что не 0 и не пустота, является правдой.

2. Логические операторы

Говоря на естественном языке например, русском мы обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике: > больше, < меньше, >= больше или равно, <= меньше или равно, == равно, != не равно.

Не путайте операцию присваивания значения переменной, обозначаемую в языке Python одиночным знаком "равно", и операцию сравнения два знака "равно". Присваивание и сравнение – разные операции.

```
>>> a = 10
>>> b = 5
>>> a + b > 14
True
>>> a < 14 - b
False
>>> a <= b + 5
True
```

```
>>> a != b
True
>>> a == b
False
>>> c = a == b
>>> a, b, c
(10, 5, False)
```

В данном примере выражение `c = a == b` состоит из двух подвыражений. Сначала происходит сравнение == переменных `a` и `b`. После этого результат логической операции присваивается переменной `c`. Выражение `a, b, c` просто выводит значения переменных на экран.

3. Сложные логические выражения

Логические выражения типа kByte >= 1023 являются простыми, так как в них выполняется только одна логическая операция. Однако, на практике нередко возникает необходимость в более сложных выражениях. Может понадобиться получить ответа "Да" или "Нет" в зависимости от результата выполнения двух простых выражений. Например, "на улице идет снег или дождь", "переменная news больше 12 и меньше 20".

В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два

оператора – так называемые логические И and и ИЛИ or.

Чтобы получить True при использовании оператора and, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным.

Чтобы получить True при использовании оператора or, необходимо, чтобы результат хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора or сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

Допустим, переменной x было присвоено значение 8 x = 8, переменной y присвоили 13 y= 13. Логическое выражение y < 15 and x > 8 будет выполняться следующим образом. Сначала выполнится выражение y < 15. Его результатом будет True. Затем выполнится выражение x > 8. Его результатом будет False. Далее выражение сведется к True and False, что вернет False.

```
>>> x = 8
>>> y = 13
>>> y < 15 and x > 8
False
```

Если бы мы записали выражение так: x > 8 and y < 15, то оно также вернуло бы False. Однако сравнение y < 15 не выполнялось бы интерпретатором, так как его незачем выполнять. Ведь первое простое логическое выражение x > 8 уже вернуло ложь, которая, в случае оператора and, превращает все выражение в ложь.

В случае с оператором or второе простое выражение проверяется, если первое вернуло ложь, и не проверяется, если уже первое вернуло истину. Так как для истинности всего выражения достаточно единственного True, неважно по какую сторону от or оно стоит.

```
>>> y < 15 or x > 8
True
```

В языке Python есть еще унарный логический оператор **not**, т. е. отрицание. Он превращает правду в ложь, а ложь в правду. Унарный он потому, что применяется к одному выражению, стоящему после него, а не справа и слева от него как в случае бинарных and и or.

```
>>> not y < 15
False
```

Здесь y < 15 возвращает True. Отрицая это, мы получаем False.

```
>>> a = 5
>>> b = 0
>>> not a
False
>>> not b
True
```

Число 5 трактуется как истина, отрицание истины дает ложь. Ноль приравнивается к False.

Отрицание False дает True.

Тема 9. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ ИНСТРУКЦИИ ЯЗЫКА PYTHON

Основные структуры алгоритмов (ОСА) – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

Любой алгоритм может быть записан с помощью трёх алгоритмических конструкций: последовательного выполнения команд (линейных алгоритмов), условных операторов и циклов.

Важно знать, что в языке Python синтаксис обладает следующей особенностью: дело в том, что в коде нет операторных скобок (begin..end или {...}); вместо них **отступы** указывают, какие операторы выполнять внутри той или иной конструкции.

1. Последовательность действий (линейные алгоритмы)

Алгоритм называется ***линейным***, если он содержит N шагов, и все шаги выполняются последовательно друг за другом от начала до конца.

Для реализации алгоритмов линейной структуры используются следующие операторы:

1) оператор **input() -** осуществляет ввод данных;

2) оператор **print() -** осуществляет вывод данных;

3) оператор **присваивания (=) -** устанавливает связь между данными и переменными.

Последовательные действия описываются последовательными строками программы. Все операторы, входящие в последовательность действий, должны иметь один и тот же отступ. Например:

```
a = 1
b = 2
a = a + b
b = a - b
a = a - b
print (a, b)
```

2. Оператор условия и выбора (алгоритмы ветвления)

Алгоритм называется ***разветвляющимся***, если последовательность выполнения шагов алгоритма изменяется в зависимости от выполнения некоторых условий. Условие - это логическое выражение, которое может принимать одно из двух значений: True - если условие верно (истинно), и False - если условие неверно (ложно).

В условиях используют знаки отношений: < (меньше), > (больше), <= (меньше или равно), >= (больше или равно), == (равно) и != (не равно).
В качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) «И», «ИЛИ» и «НЕ» (and, or и not**)**.
Операторы сравнения в Python можно объединять в цепочки (в отличие от большинства других языков программирования, где для этого нужно использовать логические связки), например, a == b == c или 1 <= x <= 10.

Разветвляющийся алгоритм можно реализовать в программах с помощью простого, сокращенного, составного операторов, а также конструкции многозначных ветвлений.
Условная инструкция в Питоне имеет следующий синтаксис:
if *Условие****:***
Блок инструкций 1
else*:*
Блок инструкций 2
Блок инструкций 1 будет выполнен, если *Условие* истинно. Если *Условие* ложно, будет выполнен *Блок инструкций 2*.
Обратите внимание, что слова **if** и **else** начинаются на одном уровне, а все команды внутренних блоков сдвинуты относительно этого уровня вправо на одно и то же расстояние (4 отступа).
В условной инструкции может отсутствовать слово **else** и последующий блок. Такая инструкция называется ***неполным ветвлением***.
Внутри условного оператора могут находиться любые операторы, в том числе и другие условные операторы. Получаем ***вложенное ветвление*** – после одной развилки в ходе исполнения программы появляется другая развилка. При этом вложенные блоки имеют больший размер отступа (например, 8 пробелов).
if *Условие 1****:***
Блок инструкций 1
else*:*
if *Условие 2****:***
Блок инструкций 2
else*:*
Блок инструкций 3
Если нужно последовательно проверить несколько условий, используется форма с дополнительным оператором **elif** (сокращение от **else if**) **- *оператор выбора***:
if *условие 1:*
Блок инструкций 1
elif *условие 2:*
Блок инструкций 2

***else**:*
Блок инструкций 3
Дополнительных условий и связанных с ними блоков **elif** может быть сколько угодно, но важно отметить, что в такой сложной конструкции будет выполнен всегда только один блок кода. Другими словами, как только некоторое условие оказалось истинным, соответствующий блок кода выполняется, и дальнейшие условия не проверяются.