Python for Informatics
Автор: Dr. Charles R. Severance
Содержание курса: http://pycode.ru/edu/why-python/

Глава 3. Условное выполнение

3.1. Логические выражения

Логическими (boolean expression) называются выражения, которые могут принимать одно из двух значений – истина или ложь. В следующем примере используется оператор ==, который сравнивает два операнда и возвращает *True*, если они равны (equal) или в противном случае *False*:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

True и *False* – специальные значения, которые принадлежат к типу *bool*, они не являются строками:

```
>>> type(True)
<type 'bool'>
>>> type(False)
<type 'bool'>
>>>
```

Оператор == является одним из операторов сравнения (comparison operators), другие операторы сравнения:

```
x != y # x не равны y
x > y # x больше y
x < y # x меньше y
x >= y # x больше или равно y
x <= y # x меньше или равно y
x is y # x такой же, как y
x is not y # x не такой же, как y
```

Все эти операции, возможно, вам знакомы, но в Python их смысл несколько отличается от принятого в математике. Распространенная ошибка заключается в использовании одиночного символа =, вместо двойного ==. Запомните, что = является оператором присваивания, а == является оператором сравнения. В Python нет подобного =< или =>.

3.2. Логические операторы

Существуют три логических оператора (logical operators): *and*, *or* и *not*. Смысл этих операторов схож с их смыслом в английском языке:

```
x > 0 and x < 10
```

это истинно в случае, если *x* больше 0 и меньше 10.

```
n%2 == 0 or n%3 == 0
```

это истинно, если одно из выражений является истинным.

В заключение, оператор *not* отрицает логическое выражение, так

```
not (x > y)
```

истинно, если *x* > *y* является ложью, т.е. *x* меньше или равно *y*.

Строго говоря, операнды логических операторов должны быть логическими выражениями, но Python не очень требователен к этому. Любое ненулевое число интерпретируется им как «истинное».

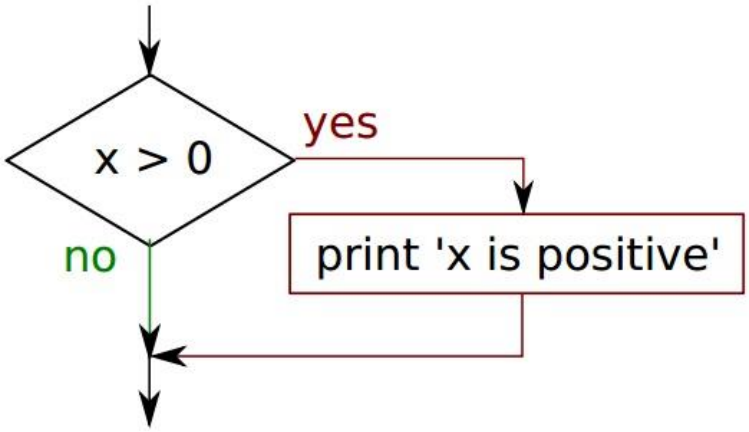
```
>>> 17 and True
True
```

3.3. Условное исполнение

Для того чтобы писать полезные программы, почти всегда необходимо проверять условия и изменять поведение программы. Условные инструкции (conditional statements) предоставляют нам такую возможность. Простейшей формой является инструкция *if*:

```
if x > 0 :
    print 'x is positive'
```

Логическое выражение после инструкции *if* называется условием (condition). Далее следует символ двоеточия (:) и строка (строки) *с отступом*.



Если логическое условие истинно, тогда управление получает выражение, записанное с отступами, иначе – выражение пропускается.

Инструкция *if* имеет схожую структуру с определением функции или цикла *for*. Инструкция содержит строку заголовка, которая заканчивается символом двоеточия (:), за которым следует блок с отступом. Подобные инструкции называются составными (compound statements), т.к. содержат больше одной строки.

Не существует ограничения на число инструкций, которые могут встречаться в теле *if*, но хотя бы одна инструкция там должна быть. Иногда полезно иметь тело *if* без инструкций (обычно оставляют место для кода, который еще не написан). В этом случае можно воспользоваться инструкцией *pass*, которая ничего не делает.

```
if x < 0 :
     pass # необходимо обработать отрицательные значения!
```

Если вы вводите инструкцию *if* в интерпретаторе Python, то приглашение сменится с трех угловых скобок на три точки (в версии 2.7 наблюдается отступ вместо точек), которые сигнализируют о том, что вы находитесь в блоке инструкций:

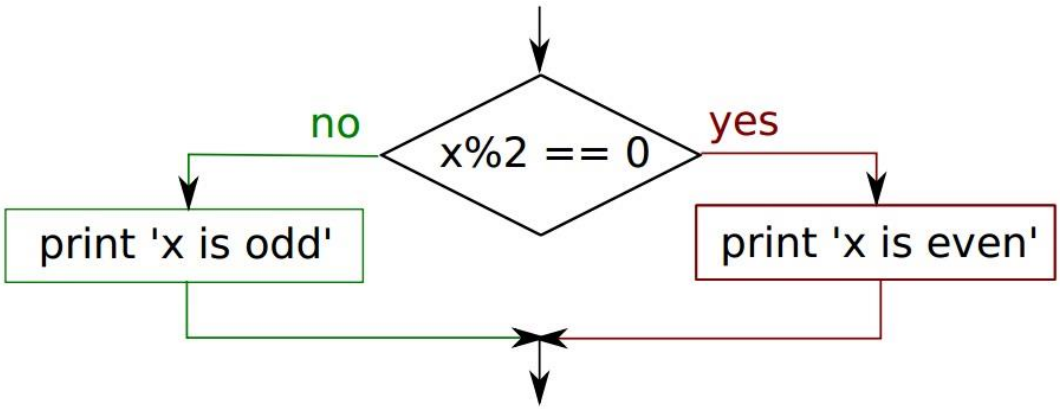
```
>>> x = 3
>>> if x < 10:
     print 'Small'


Small
>>>
```

3.4. Альтернативное исполнение

Второй формой инструкции *if* является альтернативное исполнение (alternative execution), в котором существуют два направления выполнения и условие определяет, какое из них выполнится. Синтаксис выглядит следующим образом:

```
if x%2 == 0 :
     print 'x is even'
else :
     print 'x is odd'
```

Если остаток от деления *x* на 2 равен 0, то *x*-четное, и программа выводит сообщение об этом. Если условие ложно, то выполняется второй набор инструкций.



Так как условие может быть либо истинным, либо ложным, тогда точно выполнится один из вариантов. Варианты называются ветвями (branches), потому что они являются ответвлениями в потоке исполнения.

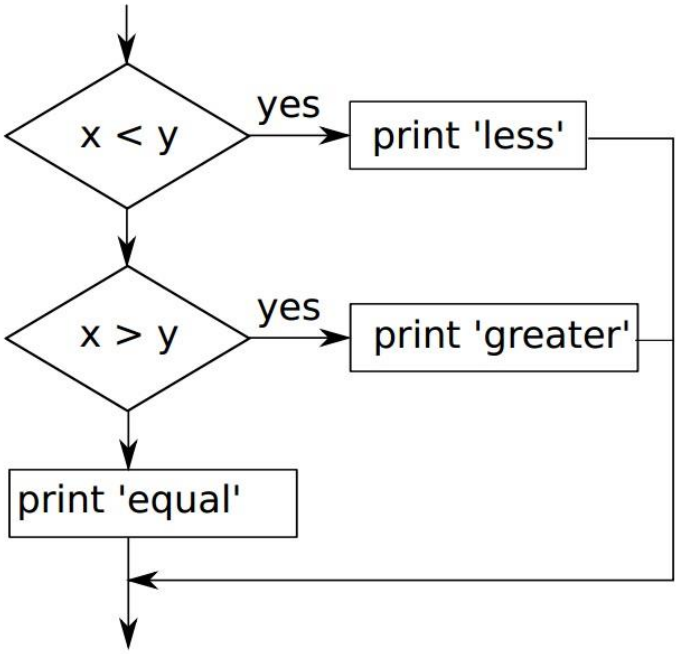
3.5. Последовательность условий

Иногда имеется больше двух вариантов выполнения, тогда нам необходимо больше двух ветвей. В этом случае, можно воспользоваться сцепленными условиями (chained conditional):

```
if x < y:
     print 'x is less than y'
elif x > y:
     print 'x is greater than y'
else:
```

```
    print 'x and y are equal'
```

elif является аббревиатурой от «else if». Снова будет исполнена точно одна ветвь.



Не существует ограничения на количество инструкций *elif*. Если встречается оператор *else*, то он должен быть в конце, но может не быть ни одного.

```
if choice == 'a':
    print 'Bad guess'
elif choice == 'b':
    print 'Good guess'
elif choice == 'c':
    print 'Close, but not correct'
```

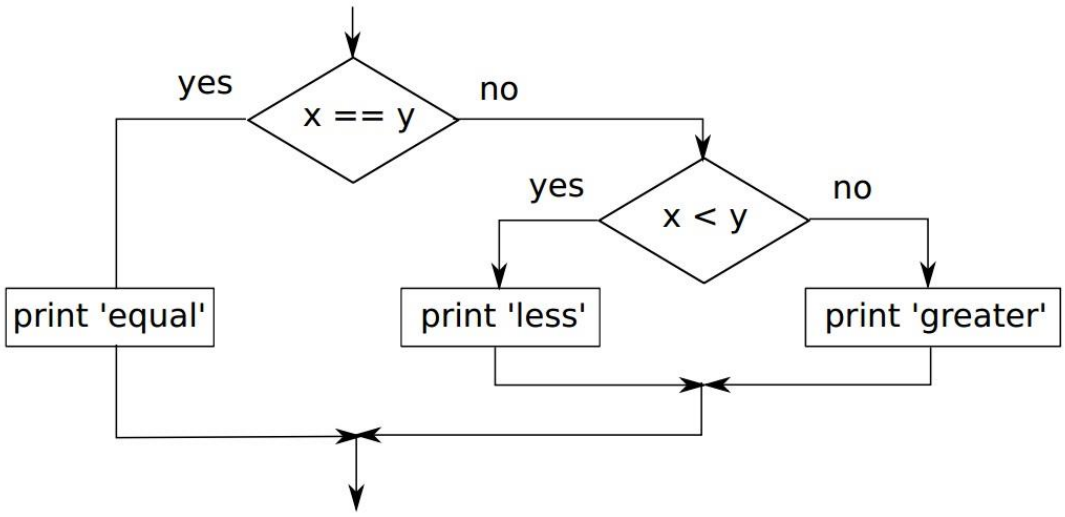
Каждое условие проверяется в порядке расположения. Если первое условие ложно, то проверяется следующее и т.д. Если одно из условий истинно, то выполняется соответствующая ветка, и инструкция завершается. Даже если верно более чем одно условие, все равно выполняется только первая истинная ветка.

3.6. Вложенные условия

Одно условие может быть вложено в другое. Мы можем записать пример трихотомии (trichotomy):

```
if x == y:
     print 'x and y are equal'
else:
     if x < y:
          print 'x is less than y'
     else:
          print 'x is greater than y'
```

Внешнее условие содержит две ветки. Первая ветка содержит простую инструкцию. Вторая ветка содержит еще одну инструкцию *if*, которая имеет две ветки. Эти две ветки являются простыми инструкциями, хотя они также могут содержать инструкции.



Хотя отступ инструкций делает структуру более очевидной, вложенные условия (nested conditionals) усложняют чтение исходного кода. Избегайте их по возможности.

Логические операторы часто позволяют упростить вложенные условные инструкции. Например, мы можем записать следующий код, используя одно условие:

```
if 0 < x:
     if x < 10:
```

```
        print 'x is a positive single-digit number.'
```

Инструкция *print* выполнится только, если мы зададим ее после обоих условий, также мы получим похожий эффект, используя оператор *and*:

```
if 0 < x and x < 10:
     print 'x is a positive single-digit number.'
```

3.7. Перехват исключений с использованием try и except

Ранее мы рассматривали код, где использовались функции *raw_input* и *int*. Мы также видели их ненадежное исполнение:

```
>>> speed = raw_input(prompt)
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int()
>>>
```

Когда мы запускали эти инструкции в интерпретаторе Python, то получали строку приглашения, затем набирали наш текст.

Пример простой программы, которая перевод температуру по Фаренгейту в температуру по Цельсию:

```
inp = raw_input('Enter Fahrenheit Temperature:')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print cel
```

Если мы запустим этот код и введем некорректные данные, то получим недружелюбное сообщение об ошибке:

```
Enter Fahrenheit Temperature:72
22.2222222222


Enter Fahrenheit Temperature:fred
Traceback (most recent call last):
```

```
File "fahren.py", line 2, in <module>

fahr = float(inp)

ValueError: invalid literal for float(): fred
```

Существует структура условного выполнения, встроенная в Python, которая обрабатывает эти типы ожидаемых и неожиданных ошибок, она называется «try / except». Идея *try* и *except* заключается в следующем: вы знаете, что некоторая последовательность инструкций может иметь проблемы, и вы хотите добавить некоторые инструкции, которые бы выполнялись в случае возникновения ошибки. Эти дополнительные инструкции (блок *except*) игнорируются, если ошибок не произошло.

Вы можете представить *try* и *except* как страховой полис для последовательности инструкций.

Перепишем нашу программу о температуре следующим образом:

```
inp = raw_input('Enter Fahrenheit Temperature:')

try:

     fahr = float(inp)

     cel = (fahr - 32.0) * 5.0 / 9.0

     print cel

except:

     print 'Please enter a number'
```

Python начинает выполнение с последовательности инструкций в блоке *try*. Если все выполняется без ошибок, то блок *except* пропускается. Если произошло исключение (exception) в блоке *try*, то Python покидает блок *try* и выполняет последовательность инструкций внутри блока *except*.

```
Enter Fahrenheit Temperature:72

22.2222222222


Enter Fahrenheit Temperature:fred

Please enter a number
```

Обработка исключения с помощью инструкции *try* называется перехватом (catching) исключения. В рассмотренном примере, блок *except* выводит на

экран сообщение об ошибке. В общем, перехват исключения предоставляет вам шанс решить проблему, попытаться заново или хотя бы красиво завершить работу программы.

3.10. Словарь

Тело (body): последовательность инструкций в составной инструкции.

Логическое выражение (boolean expression): выражение, значением которого может быть либо True, либо False.

Ветка (branch): одна из возможных последовательностей инструкций в условной инструкции.

Цепочное условие (chained conditional): условная инструкция с несколькими возможными ветками.

Оператор сравнения (comparison operator): один из операторов, который сравнивает операнды: ==, !=, >, <, >= и <=.

Условная инструкция (conditional statement): инструкция, которая контролирует поток выполнения в зависимости от некоторого условия.

Условие (condition): логическое выражение в условной инструкции, которое определяет, какая ветка выполнится.

Составная инструкция (compound statement): инструкция, которая содержит заголовок и тело. Заголовок заканчивается (:). Тело смещается относительно заголовка.

Логический оператор (logical operator): один из операторов, которые объединяют логические выражения: and, or и not.

Вложенное условие (nested conditional): условная инструкция, которая встречается в одной из веток другой условной инструкции.

3.11. Упражнения

1. Какие выражения называются логическими?
2. В чем отличия оператора = от == ?
3. Напишите программу с использованием инструкции if.
4. В чем заключается перехват исключений?