software carpentry (http://software-carpentry.org)

Programming with Python

(index.html)

Making Choices

Learning Objectives

- Explain the similarities and differences between tuples and lists.
- Write conditional statements including `if`, `elif`, and `else` branches.
- Correctly evaluate expressions containing `and` and `or`.

In our last lesson, we discovered something suspicious was going on in our inflammation data by drawing some plots. How can we use Python to automatically recognize the different features we saw, and take a different action for each? In this lesson, we'll learn how to write code that runs only when certain conditions are true.

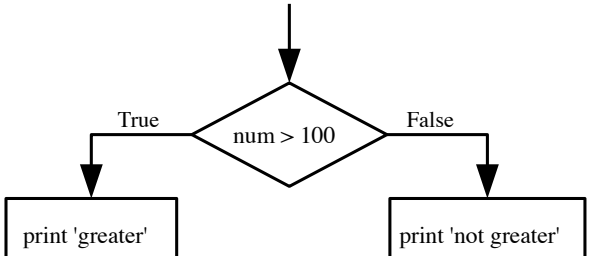
Conditionals

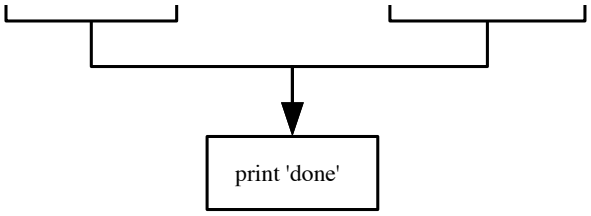
We can ask Python to take different actions, depending on a condition, with an if statement:

```
num = 37
if num > 100:
    print 'greater'
else:
    print 'not greater'
print 'done'
```

```
not greater
done
```

The second line of this code uses the keyword `if` to tell Python that we want to make a choice. If the test that follows the `if` statement is true, the body of the `if` (i.e., the lines indented underneath it) are executed. If the test is false, the body of the `else` is executed instead. Only one or the other is ever executed:





Conditional statements don't have to include an `else`. If there isn't one, Python simply does nothing if the test is false:

```
num = 53
print 'before conditional...'
if num > 100:
    print '53 is greater than 100'
print '...after conditional'
```

```
before conditional...
...after conditional
```

We can also chain several tests together using `elif`, which is short for "else if". The following Python code uses `elif` to print the sign of a number.

```
num = -3

if num > 0:
    print num, "is positive"
elif num == 0:
    print num, "is zero"
else:
    print num, "is negative"
```

```
"-3 is negative"
```

One important thing to notice in the code above is that we use a double equals sign `==` to test for equality rather than a single equals sign because the latter is used to mean assignment.

We can also combine tests using `and` and `or`. `and` is only true if both parts are true:

```
if (1 > 0) and (-1 > 0):
    print 'both parts are true'
else:
    print 'one part is not true'
```

```
one part is not true
```

while `or` is true if at least one part is true:

```
if (1 < 0) or (-1 < 0):
    print 'at least one test is true'
```

```
at least one test is true
```

Checking our Data

Now that we've seen how conditionals work, we can use them to check for the suspicious features we saw in our inflammation data. In the first couple of plots, the maximum inflammation per day seemed to rise like a straight line, one unit per day. We can check for this inside the `for` loop we wrote with the following conditional:

```
if data.min(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
    print 'Suspicious looking maxima!'
```

We also saw a different problem in the third dataset; the minima per day were all zero (looks like a healthy person snuck into our study). We can also check for this with an `elif` condition:

```
elif data.min(axis=0).sum() == 0:
    print 'Minima add up to zero!'
```

And if neither of these conditions are true, we can use `else` to give the all-clear:

```
else:
    print 'Seems OK!'
```

Let's test that out:

```
data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
    print 'Suspicious looking maxima!'
elif data.min(axis=0).sum() == 0:
    print 'Minima add up to zero!'
else:
    print 'Seems OK!'
```

```
Suspicious looking maxima!
```

```
data = numpy.loadtxt(fname='inflammation-03.csv', delimiter=',')
if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
    print 'Suspicious looking maxima!'
elif data.min(axis=0).sum() == 0:
    print 'Minima add up to zero!'
else:
    print 'Seems OK!'
```

```
Minima add up to zero!
```

In this way, we have asked Python to do something different depending on the condition of our data. Here we printed messages in all cases, but we could also imagine not using the `else` catch-all so that messages are only printed when something is wrong, freeing us from having to manually examine every plot for features we've seen before.

How many paths?

Which of the following would be printed if you were to run this code? Why did you pick this answer?

A B C B and C

```
if 4 > 5:
    print 'A'
elif 4 == 5:
    print 'B'
elif 4 < 5:
    print 'C'
```

What is truth?

`True` and `False` are special words in Python called `booleans` which represent true and false statements. However, they aren't the only values in Python that are true and false. In fact, *any* value can be used in an `if` or `elif`. After reading and running the code below, explain what the rule is for which values are considered true and which are considered false. (Note that if the body of a conditional is a single statement, we can write it on the same line as the `if`.)

```
if '': print 'empty string is true'
if 'word': print 'word is true'
if []: print 'empty list is true'
if [1, 2, 3]: print 'non-empty list is true'
if 0: print 'zero is true'
if 1: print 'one is true'
```

Close enough

Write some conditions that print `True` if the variable `a` is within 10% of the variable `b` and `False` otherwise. Compare your implementation with your partner's: do you get the same answer for all possible pairs of numbers?

In-place operators

Python (and most other languages in the C family) provides in-place operators (reference.html#in-place-operator) that work like this:

```
x = 1  # original value
x += 1 # add one to x, assigning result back to x
x *= 3 # multiply x by 3
print x
```

```
6
```

Write some code that sums the positive and negative numbers in a list separately, using in-place operators. Do you think the result is more or less readable than writing the same without in-place operators?

Tuples and exchanges

Explain what the overall effect of this code is:

```
left = 'L'
right = 'R'

temp = left
left = right
right = temp
```

Compare it to:

```
left, right = right, left
```

Do they always do the same thing? Which do you find easier to read?