Computer Science 1
Conditional Execution (Lecture Part 1)

09/01/2013

September 2, 2013
1 Conditional Execution

Simple sequences of imperative statements are fine when you are, for example, directing the turtle to draw a predefined shape. Often, however, we do not know ahead of time what we want a program to do when it executes. It depends on some kind of information available only when the program is running. At this point, the most likely situation is due to user input. Here are a few examples.

```
Ask the user if he wants a thick line.
If the user replied with "yes",
    set the turtle to draw a thick line;
    otherwise set the turtle to draw a thin line.

Ask the athlete for her running time.
If the time entered is less than 223.13
    indicate that the athlete broke the world record!
Otherwise, if the time is less than 270,
    congratulate the athlete on an excellent run;
    otherwise encourage the athlete to keep training.
```

Below we see how you might write the above instructions in Python.

```
def getThickness():
    answer = input( "Do you want a thick line? " )
    if answer == "yes":
        turtle.pensize( 4 )
    else:
        turtle.pensize( 1 )

def evaluateRun():
    result = input( "Please enter the time for your mile run: " )
    time = float( result )
    if time < 223.13:
        print( "New world record. Congratulations!" )
    elif time < 270:
        print( "Preverbatimy good. You’re one of the best." )
    else:
        print( "Thanks for the data. Keep training!" )
```

Note the following things in the code.

Conditional statements are created using an if statement, followed by indenting and possible use of else and elif. The entire block of statements provides a set of alternative paths through the execution of the code, based on the tests described on the if line and on any elif lines that may be present.

Tests must be expressions that evaluate to the Python values True or False. These values form a type called Boolean, or in Python's vernacular, bool. (The name comes from one of the pioneers in the mathematics of the algebra of logic George Boole.)

In general form, an if block of statements is as follows.

```
if bool-expression0:
    statements to execute
    if bool-expression0 is True
elif bool-expression1:
    statements to execute
    if bool-expression1 is True
elif bool-expression2:
    statements to execute
    if bool-expression2 is True
:
:
elif bool-expressionN:
    statements to execute
    if bool-expressionN is True
else:
    statements to execute
    if none of the expressions above are True
```

Of course, you don't need any elif sections at all. If there are none, you just have a two-choice scenario based on a single test.

It turns out that the else section is also optional. It just means that, if the test results in a value of False, none of the code in the if block executes.

Remember! Once the statements in one section of an if block are executed, no other tests are evaluated, so no other statements get executed.

The tests we see in the above examples use comparison operators. The "`==`" operator can be read as "is equal to" and in most cases means that it's checking to see if the values of the expressions on its left and right sides evaluate to the same thing. The "`<`" operator is an inequality operator and works as you would expect. Other comparison operators are "`!=`" (not equal to), "`<=`" (less than or equal to), "`>`", and "`>=`".

Although they don'e show up in the examples here, there are also Boolean operators that work on Boolean expressions.

1. “`x and y`” is `True` only if both `x` and `y` are `True`.
2. “`x or y`” is `True` if either `x` or `y` is `True`.
3. “`not x`” is `True` only if `x` is `False`, and vice-versa.

For example:

```
if age > 10 and height > 35:
    print( "You may get on the ride." )
else:
    print( "Sorry, not allowed." )
```

We have already seen that we can cause text to appear in the window where the Python interpeter is running by passing a string to the input function. However, if we just want to “print” something in that window, the print function can be used. By default, the text is followed by a carriage return, so further prints occur on the following line. Although we will not elaborate here, it is possible to send the print function multiple arguments. In that case, they all end up on the same line. See http://docs.python.org/3/library/functions.html#print for details.

In the first example, we wanted a text string as the answer from the user. This is what the input function returns. However, in the second case we wanted a floating-point number. (“Floating point” refers to the fact that the decimal point is not fixed. We can therefore represent very large numbers and any number with a fractional part.) Therefore we use float, a function that converts a string containing any Python-legal number into a value of type float (yes, the same word), which can then be compared with other numbers using the comparison operators.