ICS3U: Making Decisions

Conditional Processing

So far, all of your programs have been **sequential**: they have performed one action, then moved on to the next, then the next, and so on. Sequential processing is an essential part of any programming language. It allows us to specify an ordering of commands, so that certain actions can precede others.

Another important feature of a modern programming language is its ability to execute a specific piece of code if one or more conditions are met, and a *different* piece of code if they are not. This is known as **conditional processing**. For example, consider a simple game where the player must guess a randomly-generated number within a fixed number of attempts. If the player enters the correct number, then the program should halt and display a congratulatory message. On the other hand, if the player does not enter the correct value, then the secret number should be revealed along with an alternate message.

In the previous example, both scenarios (winning or losing) were prefaced by the word "if". If a condition is met (guessing the number), perform an action. If the condition is not met (failing to guess the right number), perform a different action. Python allows us to specify these conditions using the `if` keyword. The general format of `if` is as follows.

```
if CONDITION(S):
   CODE TO EXECUTE IF CONDITION(S) TRUE
   MORE CODE …
```

There are two things to note here. The first is that the `if` statement ends with a colon (:). This signifies to the interpreter that a block of code, associated with the conditions, will follow. The second is that the block of code is indented. Python uses indentation to group code. This is different from other programming languages that use symbols, such as {} or (). All code that is to be executed as part of the block must share the same level of indentation. In most cases, the IDE will automatically indent the code once a colon is encountered.

The Boolean Data Type

In addition to numeric and sequence data types, like integers and strings, Python also has a **boolean** data type. A boolean data type can assume one of two possible values: `True` or `False`. Both of these are reserved keywords. Note that they are capitalized. The following command creates a variable, *x*, and assigns it a value of `True`.

```
>>> x = True
>>> x
True
```

This may seem strange to restrict a data type to only two options, but Python uses boolean values when doing comparisons. All conditions will be either `True` or `False`, and the interpreter will use these values to determine whether or not a block of code is executed.

Comparison Operators

Comparisons are made by checking if values are equal to, greater than, or less than other values. There are six **comparison operators** in Python: equal to (==), not equal to (!=), greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=).

Checking if a condition is met (`True`) or not (`False`) usually involves an expression that uses one of the comparison operators above. The code below displays a message if the user enters a positive integer.

```
num = int(input("Please enter a positive integer: "))
if num > 0:
   print("Thank you for doing as you were told.")
print("DONE")
```

The first line reads a value from the user, converts it to an integer, and assigns it to `num`. The second line compares the value of `num` to 0. If `num` is greater than zero, then the condition `num > 0` is `True`. The interpreter will then enter the `if` block and execute the code on the third line. On the other hand, if `num` is not greater than zero, then the condition `num > 0` is `False`, and the code inside of the `if` block will not be executed. Below are two sessions showing output for each scenario.

```
Please enter a positive integer: 8
Thank you for doing as you were told.
DONE

Please enter a positive integer: -2
DONE
```

Here is another example, which asks the user to enter two numbers and checks if their sum is twelve.

```
num1 = int(input("Please enter an integer: "))
num2 = int(input("Please enter another integer: "))
if num1 + num2 == 12:
   print("The sum of", num1, "and", num2, "is 12.")
print("DONE")
```

If the condition `num1 + num2 == 12` is `True`, then the message on the fourth line will be displayed. If it is `False`, then the message will not be displayed. Try running the code using various values for `num1` and `num2`.

A common error is to confuse the equal to comparison operator, ==, with the assignment operator, =, as in the second line of the code below.

```
x = 5
if x = 10:
    print("ten")
```

Doing this will cause a syntax error.

```
Traceback (most recent call last):
  File "/home/Code/Basics/program.py", line 2
    if x = 10:
         ^
SyntaxError: invalid syntax
```

Alternative Options

In the first example where we asked the user to enter a positive integer, a message was only displayed if they did so. No alternate message was displayed if the user entered a negative integer. It would be a nice feature to add this alternate message instead of merely halting the program. One way to do this is to add a second `if` statement with another condition.

```
num = int(input("Please enter a positive integer: "))
if num > 0:
   print("Thank you for doing as you were told.")
if num <= 0:
   print("That is a negative integer.")
```

If the condition `num > 0` is `False`, then the program will not display the message on the third line. It will check the condition `num <= 0` on the fourth line and, if it is `True`, it will display the message there.

```
Please enter a positive integer: -3
That is a negative integer.
```

This approach is not as efficient as it could be. For example, if the first condition is `True` then the first message will be displayed in the shell; however, the program will still perform the second comparison. Since any number is either greater than zero, or less than or equal to zero, one of the two conditions *must* be `True`, but both conditions cannot be `True` simultaneously. They are **mutually exclusive** conditions. To provide alternate options for mutually exclusive conditions, use the `elif` keyword, short for "else if".

```
num = int(input("Please enter a positive integer: "))
if num > 0:
   print("Thank you for doing as you were told.")
elif num <= 0:
   print("That is a negative integer.")
```

If the condition `num > 0` is `True`, then the message on the third line will be displayed. Unlike the earlier code, however, the comparison on the fourth line will be skipped entirely. Once a condition is `True`, any `elif` statement will be ignored and the program will move on to the next line of code after the `if` block.

An `elif` statement cannot occur before an `if` statement, or an error will occur. It is possible to chain multiple `elif` statements after another, as in the following example, as long as they have the same level of indentation.

```
num = int(input("Please enter a positive integer: "))
if num > 0:
   print("Thank you for doing as you were told.")
elif num == 0:
   print("Zero is not a positive integer.")
elif num < 0:
   print("That is a negative integer.")
```

As in the previous example, once a condition is `True`, all subsequent `elif` statements will be skipped.

Consider a program that reads a single letter from the user and displays messages if the user enters an uppercase ‘A’, and another message for any other letter. It would be extremely time-consuming, and inefficient, to write something like the following:

```
letter = input("Enter a letter: ")
if letter == "A":
   print("You entered an uppercase A.")
elif letter == "B":
   print("That is not an A.")
elif letter == "C":
   print("That is not an A.")

… 44 lines omitted …

elif letter == "Z":
   print("That is not an A.")
```

This 53-line program could result in up to 26 comparisons being made, and that does not include any lowercase letters. Furthermore, nothing will be displayed if the user enters a non-alphabetic character. Fortunately, Python provides an option to handle the case where all `if` and `elif` conditions are `False`: the `else` keyword. Since it covers “everything else”, an `else` statement is not followed by a condition.

```
letter = input("Enter a letter: ")
if letter == "A":
   print("You entered an uppercase A.")
else:
   print("That is not an A.")
```

If the user enters an uppercase ‘A’, then the message on the third line will be displayed because the condition `letter == "A"` will be `True`. Any other character (or sequence of characters, in this case) will result in the condition `letter == "A"` being `False`. The program will then enter the `else` block, and display the message on line five. Try entering some letters to see what happens.

Like `elif`, an `else` statement must have an `if` statement somewhere before it or an error will occur. Attempting to attach a condition to an `else` statement will also result in an error. Unlike `elif`, there can only be one `else` statement in an `if` block.