ICS3U: Nested If Blocks

Indentation

When we first introduced the `if` keyword, we noted that Python uses indentation to group lines of code. In the example below, the mathematical calculation and `print` statement are associated with the `if` statement because they share the same level of indentation.

```
if x > 5:
   y = x * 2
   print(y)
```

While whitespace usually does not matter within a line of code itself, indentation can cause problems if not used appropriately. Consider the code below, where the second `print` statement has been wrongly indented.

```
print("This is some text.")
   print("Here is more text.")
```

This will cause an error, like the one below.

```
Traceback (most recent call last):
  File "/usr/lib/python3.8/ast.py", line 47, in parse
    return compile(source, filename, mode, flags,
  File "/home/Code/indentation.py", line 2
    print("Here is more text.")
    ^
IndentationError: unexpected indent
```

This becomes particularly important when nesting `if` statements inside of each other.

Nested If Blocks

We have already seen examples of nesting with functions, such as when typecasting user input as a `float` or rounding a value in a `print` statement.

```
num = float(input("Please enter a number with lots of decimal digits: "))
print("Your number rounded to two decimals is", round(num, 2))
```

When nesting, one function (`input`) is embedded *inside* of another function (`float`). We can use a similar technique with `if` blocks (and later, with loops) whereby we insert an `if` statement *inside* of another.

Consider a program that monitors inventory for a small company that produces two types of vehicles. Sedans come in two- and four-door models, while minivans have both dual-door and hatch rear door options. The user must first select the type of vehicle (sedan or minivan) and then, depending on the type of vehicle that was chosen, the door style.

A possible algorithm for searching inventory for available vehicles is below.

```
GET type of vehicle
IF type is sedan
  GET number of doors
  IF two doors
    SEARCH inventory for sedans with two doors
  ELSE IF four doors
    SEARCH inventory for sedans with four doors
  ELSE
    DISPLAY an error message (invalid number of doors)
ELSE IF type is minivan
  GET rear door style
  IF dual-door
    SEARCH inventory for minivans with dual-doors
  ELE IF hatch
    SEARCH inventory for minivans with hatch doors
  ELSE
    DISPLAY an error message (invalid rear door option)
ELSE
  DISPLAY an error message (invalid vehicle type)
```

A Python program would look very similar to the algorithm.

```
vehicle_type = input("What type of vehicle (type 'sedan' or 'minivan')? ")
if vehicle_type == "sedan":
   number_of_doors = int(input("How many doors (enter 2 or 4)? "))
   if number_of_doors == 2:
      print("Searching for sedans with two doors...")
      # SEARCH CODE GOES HERE
   elif number_of_doors == 4:
      print("Searching for sedans with four doors...")
      # SEARCH CODE GOES HERE
   else:
      print("You must select either two or four doors.")
elif vehicle_type == "minivan":
   rear_door_style = input("Rear door style (enter 'dual' or 'hatch')? ")
   if rear_door_style == "dual":
      print("Searching for minivans with dual-doors...")
      # SEARCH CODE GOES HERE
   elif number_of_doors == 4:
      print("Searching for minivans with hatch doors...")
      # SEARCH CODE GOES HERE
   else:
      print("You must select either dual or hatch rear doors.")
else:
   print("You must select either a sedan or a minivan.")
```

There are two levels of indentation here. There is an "outer" `if` block that handles the type of vehicle. For each vehicle type, there is an "inner" `if` block that handles the door type. In both cases, the `if`, `elif` and `else` statements share the same level of indentation, signalling to the interpreter that they belong to the same block.

In many programs, `if` statements may not appear to be nested because there is a large amount of code before the next `if` statement is encountered, as in the following example.

```
num = int(input("Enter a value: "))
if num > 10:
   print("Your value is large.")
   # ...
   # 20 LINES OF CODE
   # ...
   if num % 2 == 0:
      print("This is an even value.")
      # ...
      # MORE CODE
      # ...
   else:
      print("This is an odd value.")
      # ...
      # MORE CODE
      # ...
```

Again, the important thing is to ensure that each block of related commands is indented to the same level, so that they are all executed when a particular condition is `True`.