ICS3U: Better Input Validation

We have previously covered ways in which we can use conditional loops to screen user input and redirect them toward certain values. For example, if the user must enter a 'Y' or an 'N' in response to a question, we can use a while loop similar to the one below.

```
choice = input("Continue (Y/N)? ")
while choice != "Y" and choice != "y" and choice != "N" and choice != "n":
   choice = input("Please enter Y or N: ")
```

By using functions, we can reduce the amount of duplicated code in our programs. One task that we find ourselves repeating in most of our Python programs is reading a positive integer from the user. Sometimes we need to do this multiple times. Take for example the following code, which generates a bill for a movie theatre that sells child tickets for $6, adult tickets for $12 and senior tickets for $8.

```
num_children = int(input("How many children: "))
while num_children < 0:
   num_children = int(input("Invalid value, try again: "))
num_adults = int(input("How many adults: "))
while num_adults < 0:
   num_adults = int(input("Invalid value, try again: "))
num_seniors = int(input("How many seniors: "))
while num_seniors < 0:
   num_seniors = int(input("Invalid value, try again: "))
total = 6 * num_children + 12 * num_adults + 8 * num_seniors
print("The total is $", round(total, 2), sep="")
```

In the code, the same action was repeated three times. We can eliminate this repetition by creating a function, `get_positive_integer`, that will prompt the user to enter an appropriate value. Our first attempt might look something like this.

```
def get_positive_integer():
   integer = int(input("Enter a positive integer: "))
   while integer < 0:
      integer = int(input("Invalid value, try again: "))
   return integer
```

The code is nearly identical to that of the original program (only cosmetic changes have been made), but it allows us to rewrite the main program in a much more concise and easy-to-follow way.

```
num_children = get_positive_integer()
num_adults = get_positive_integer()
num_seniors = get_positive_integer()
total = 6 * num_children + 12 * num_adults + 8 * num_seniors
print("The total is $", round(total, 2), sep="")
```

If you run the program, you might notice that the prompts for the user are not very informative. It always asks the user to enter a positive integer, rather than asking for the number of children or adults. We can change this by passing a string as an argument to `get_positive_integer` that will be displayed as the initial prompt to the user.

Our new function might look like this.

```
def get_positive_integer(prompt):
   integer = int(input(prompt))
   while integer < 0:
      integer = int(input("Invalid value, try again: "))
   return integer
```

This will require a slight change to each of our calls to `get_positive_integer` in the main program.

```
num_children = get_positive_integer("Enter the number of children: ")
num_adults = get_positive_integer("Enter the number of adults: ")
num_seniors = get_positive_integer("Enter the number of seniors: ")
total = 6 * num_children + 12 * num_adults + 8 * num_seniors
print("The total is $", round(total, 2), sep="")
```

Remember that although this is slightly longer than our original program, the extendability and flexibility of using functions makes this a better overall approach. Imagine a much longer program which requires obtaining a positive value from the user dozens of times. To ensure that the user does so each time, we would need only a single line of code (a call to `get_positive_integer`) rather than three lines for separate values. Furthermore, changes to the code need only be done in a single function rather than scattered throughout the code.

As another example, the following code obtains an integer that falls between two values, `low` and `high`. This allows us to set lower- and upper-bounds for user input. Try it to verify that it works.

```
def get_integer(low, high, prompt):
   integer = int(input(prompt))
   while integer < low or integer > high:
      integer = int(input("Invalid value, try again: "))
   return integer

num = get_integer(10, 20, "Pick a number between 10 and 20: ")
print("You entered", num)
```

Python allows the programmer to set **default values** for arguments. These are values that will be used if other values are not provided. The code below uses default values for all three arguments, `low`, `high` and `prompt`. If you use them in a function definition, then each argument following the first default value must have its own default value, or an error will occur.

```
def get_integer(low=0, high=100, prompt="Enter a value: "):
   integer = int(input(prompt))
   while integer < low or integer > high:
      integer = int(input("Invalid value, try again: "))
   return integer

num1 = get_integer(10, 20, "Pick a number between 10 and 20: ")
num2 = get_integer(50, 80)
num3 = get_integer()
```

The first call behaves exactly the same as the first program, obtaining a value between 10 and 20. The second call, `get_integer(50, 80)`, obtains a value between 50 and 80 but uses the default prompt `Enter a value`. The third call, `get_integer()`, provides no arguments and so it will use the default prompt and will obtain a value between 0 and 100.

An alternate method of specifying values is to use **keyword arguments**. These allow some flexibility in ordering, and also make it possible to override any number of default values.

```
num4 = get_integer(high=200)
num5 = get_integer(prompt="The secret number is between 1-100. Guess? ")
```

To make this function truly useful as one that will obtain an integer value, we might combine some of the options from previous versions. We would like to provide the programmer with the ability to specify a prompt for the user. We would also like to specify a `low` and `high` value, but our current limits of 0 and 100 are fairly restrictive. There might be many instances where we wish to allow negative values, or values far larger than 100. By using the value of `inf` from the `math` module as default values, we can allow the user to enter any value if they wish; otherwise, these values will be overridden by those passed to `get_integer` as arguments.

```
import math

def get_integer(low=-math.inf, high=math.inf, prompt="Enter a value: "):
   integer = int(input(prompt))
   while integer < low or integer > high:
      integer = int(input("Invalid value, try again: "))
   return integer

val1 = get_integer()
val2 = get_integer(high=50)
val3 = get_integer(prompt="Guess a value between 1 and 5", low=1, high=5)
```

Note that this will still not prevent the program from crashing on input that cannot be represented as an `int`, but we will soon be at a point where we can deal with situations like those.