ICS3U: Basic Input Validation

Checking User Input

Up until this point, we have assumed that the user will do as instructed in a prompt. If a program requires the user to enter a capital letter, our code has assumed that they will do so. In reality, this does not always happen. Sometimes this is by accident, such as when the user mistypes a letter or enters a number that is above a certain value. In other cases, it is deliberate. Some people have an interest in making a program misbehave or crash, sometimes with malicious intent.

We would like to mak our programs more robust, which means that we will need to stop assuming that the user will do as they are told, and begin writing code that handles cases where they do not. Instead of moving directly to processing the information once it is entered, we can instead examine the input to see if it meets one or more criteria first. Since they are based on conditions, `while` loops are excellent tools to do this. If bad data is entered, it is possible to have the user re-enter it before it is processed.

Assume that as part of a program, we want the user to enter a positive integer. While we do not yet have the tools available to deal with non-integral values, such as the letter 'A', we can ensure that if the user *does* enter an integer value, that it is greater than zero. To do this, we use a `while` loop similar to the one below that runs as long as the condition `num <= 0` is `True`. Once the user enters a positive value and this condition is no longer met, the loop will stop.

```
num = int(input("Please enter a positive integer: "))
while num <= 0:
   num = int(input("That's not a positive value. Please try again: "))
print("Number is valid.")
```

If multiple conditions are required, we can use logical operators to combine them in the header of a `while` loop. The code below asks the user to enter a value between 2 and 5, and checks if the value entered falls outside of this range. If either of the conditions `val < 2` or `val > 5` is `True`, the loop will run, asking the user to enter a new value.

```
val = int(input("Pick a number between 2 and 5: "))
while val < 2 or val > 5:
   val = int(input("Invalid value. Enter a number between 2 and 5: "))
print("Number is valid.")
```

The same ideas can be used to handle string-based data. The following code requires the user to enter a vowel before it will move on. In this case, we use `in` to avoid linking ten separate conditions together with `or`. As long as the input is not contained in the string of vowels, the loop will continue to run.

```
letter = input("Please enter a vowel: ")
while letter not in "AEIOUaeiou":
   letter = input("Not a vowel. Try again: ")
print("Thank you.")
```

Even without typecasting user input to a numeric data type, this loop is not fool-proof. In fact, it is still possible for the code to result in logical errors due to bad input. Try running the code with the input "IOU". What happens, and why?

Let's put things together in a working application. The following program can be used to determine if a triangle suspected of being right-angled *actually* contains a right angle. Such a triangle would need to obey the Pythagorean Theorem, $a^2+b^2=c^2$. That is, the sum of the squares of the two shorter sides must be equal to the square of the hypotenuse. The program reads the three sides from the user and checks if the Pythagorean Theorem holds. Since it is a fairly large program (at least, compared to the ones we have previously written), some comments have been included to explain each major section.

```
# Obtain the lengths of the two shorter sides.
side1 = int(input("Enter the length of the first side: "))
while side1 <= 0:
   side1 = int(input("Side length must be greater than zero. Try again: "))
side2 = int(input("Enter the length of the second side: "))
while side2 <= 0:
   side2 = int(input("Side length must be greater than zero. Try again: "))

# Obtain the length of the hypotenuse. It must be the longest side, and
# its length must not exceed the sum of the other two sides.
hypot = int(input("Enter the length of the hypotenuse: "))
while True:
   if hypot < side1 or hypot < side2:
      print("The hypotenuse must the longest side. ", end="")
   elif hypot >= (side1 + side2):
      print("Hypotenuse is too long to form a triangle. ", end="")
   else:
      break
   hypot = int(input("Try again: "))

# Check if the triangle is a right triangle or not.
if hypot**2 == side1**2 + side2**2:
   print("The triangle contains a right angle.")
else:
   print("The triangle does not contain a right angle.")
```

The first portion of the code reads two integers from the user, representing the two shorter sides. Each side must have a length greater than zero, since negative side lengths are not permitted. Each of the two `while` loops runs as long as the user enters values that are zero or less.

The second portion deals with the hypotenuse. Since there are multiple conditions, an infinite loop is used with an `if` block inside of it to separate the cases. This is by design – a similar program could be written that lists all conditions in the loop's header instead. To ensure that the user has entered the correct measurements, the program first checks whether the hypotenuse is smaller than either of the other two sides. If it is, then it cannot be the hypotenuse. It then checks if the hypotenuse is equal to or larger than the sum of the other two sides. If it is, then a measurement error has likely occurred, since it cannot exceed this amount. If none of the excluding conditions are met, then the `else` containing the `break` statement is executed, and the loop terminates.

The final portion does the actual calculation. If the square of the hypotenuse is equal to the sum of the squares of the other two sides, then the triangle is right-angled. Otherwise, it is not.