Lab 2: Basic Flow Control

CSE/IT 107

NMT Computer Science

“When you come to a fork in the road, take it.”

— Attributed to Yogi Berra

“Simplicity is the ultimate sophistication.”

— Leonardo Da Vinci

“How do we convince people that in programming simplicity and clarity – in short: what mathematicians call “elegance” – are not a dispensable luxury, but a crucial matter that decides between success and failure?”

— Edsger Dijkstra

---

1 Introduction

The purpose of this lab is to introduce you to the fundamentals of flow control - if, else, and while loops.

2 Boolean logic

A common activity when programming is determining if something value is true or false. For example, if a variable is less than five or if the user entered the correct password. Any statement that can be resolved into a true or a false value is called a boolean statement, the value it resolves into (true or false) is called a boolean value.

```
>>> x = 5
>>> print(x < 3)
False
>>> print(x < 6)
True
```

In the above example, the boolean values are `True` and `False`. The boolean statements are `x < 3` and `x < 6`.

In addition to `<`, we can also test for other inequalities.

```
>>> x = 3
>>> y = 6
>>> print(x < y)
True
>>> print(x > y)
False
>>> print(x <= y)
True
>>> print(x >= y)
False
```

Note that `<=` means "less than or equal to" and `>=` means "greater than or equal to".

Finally, we can test if two values are equal (`==`) or not equal (`!=`).

```
>>> x = 3
>>> y = 3
>>> z = 4
>>> print(x == y)
True
>>> print(x == z)
False
>>> print(y != 5)
True
>>> print(y != x)
False
```

It is important to remember that we use `=` to assign a value to a variable and `==` to test if two values are equal.

3 `if` and `else`

The primary use for boolean values is to determine which branch in your code to follow. This is accomplished using `if` and `else`, as shown in the program below.

```python
x = 1
y = float(input("Please input a number: "))

if x == y:
    print("x and y are equal.")
else:
    print("x and y are not equal")
```

Try running the above program, inputting different numbers for `y`. If the number input is 1, then the first `print` statement will output. If not, then the second one will.

The way this works is very simple: when Python sees an `if` statement, it checks if the boolean statement immediately following it is `True`. If it is, then it will run indented section of code starting on the next line. Once that section is done, it will skip past the indented `else` block of code (if the `else` exists – it is optional), and continue running the rest of the program. If the statement was `False`, then it will skip the block of indented code, run the `else` block if it exists, and then continue running the rest of the program.

There are many uses for branching statements, such as to ensure that a given variable is not negative:

```python
x = float(input("Please input a number: "))

if x < 0:
    x = 0

print("x = " + str(x))
```

You can perform other operations as part of a boolean statement, such as this convenient way to check if a number is even:

```python
x = 5

if x % 2 == 0:
    print("x is even.")
else:
    print("x is odd.")
```

When using `if` and `else`, you will generally be dealing with user input. This is done using the function `input`, which you can see used in the above examples. When you use `input` it will display whatever string you pass to it, then pause while it waits for the user to type something and then hit enter. It will return whatever was entered as a string. We will be learning more about strings in future labs, but for now just know that they are basically groups of letters, like what you pass to a `print` statement, and are declared by surrounding something in quotes.

The main thing to know about strings for now is that they cannot be used as numbers. This is why we use the `float` function to convert the value the user gives us into a number.

```
>>> 5.5 == "5.5"
False
>>> 5.5 == float("5.5")
True
```

It is important to understand the order that things happen in a statement like `x = float(input("Please input a number: "))`. Though both `x =` and `float` appear first in the line, the first statement to execute is `input`. This is because `input` is inside of `float`'s parentheses and is therefore being passed as a parameter to `float`. Therefore, `float` cannot run until `input` is finished and has returned a value to be used by `float`. Similarly, `x =` will not happen until `float` has finished converting the value into a number.

If you are comparing strings, then you do not need to go through the extra step of converting the user's input into a number:

```python
password = "hunter2"

user_pass = input("Please input the password: ")

if password == user_pass:
    print("Password is correct. Welcome!")
else:
    print("Invalid password.")
```

4 `while` loops

The syntax of a `while` loop is very similar to that of an `if` statement, but instead of only running the indented block of code once, the `while` loop will continue running it until the given boolean statement is no longer true.

```python
x = 10

while x > 0:
    print(x)
    x = x - 1
```

The above program will print out the numbers 10 to 1. Try stepping through this program on paper, writing out the value of `x` at each time through the loop. Then repeat for this modified version of the program:

```python
x = 10

while x > 0:
    x = x - 1
    print(x)
```

This version of the program will print out the numbers 9 to 0. This might seem a bit strange, since the condition of the loop says it will stop when `x` is no longer larger than 0. And yet, it prints out the value 0 before the loop ends. This is because the loop condition is only checked whenever the end of the indented section is looped. If the condition is `True`, then the indented section will be executed again. If the condition is `False`, then the loop will end.

If the condition starts out `False`, then the loop will never execute. The following program will not print anything:

```python
x = 0
```

```python

while x > 0:
    x = x - 1
    print(x)
```

`if` and `else` can be combined with `while`, as shown below:

```python
x = 10

while x > 0:
    if x % 2 == 0:
        print(str(x) + " is even.")
    else:
        print(str(x) + " is odd.")
    x = x - 1
```

5 Turtle

6 .format()

7 Exercises

8 Submitting

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab2.zip` or `cse107_firstname_lastname_lab2.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed.

Upload your tarball or .zip file to Canvas.

8.1 Linux

Tar is used much the same way that Zip is used in Windows: it combines many files and/or directories into a single file. Gzip is used in Linux to compress a single file, so the combination of Tar and Gzip do what Zip does. However, Tar deals with Gzip for you, so you will only need to learn and understand one command for zipping and extracting.

In the terminal (ensure you are in your `lab1` directory), type the following command, replacing `firstname` and `lastname` with your first and last names:

```
tar czvf cse107_firstname_lastname_lab2.tar.gz *.py
```

This creates the file `cse107_firstname_lastname_lab1.tar.gz` in the directory. The resulting archive, which includes every python file in your `lab1` directory, is called a tarball.

To check the contents of your tarball, run the following command:

```
tar tf cse107_firstname_lastname_lab2.tar.gz *.py
```

You should see a list of your Python source code files.