Lab 4: Control Flow

Purpose

To increase exposure to conditional and iterative statements, two foundational types of control flow manipulation in computation.

Description

Conditional and iterative statements are means by which you can control the flow of execution of your program. Specifically, conditional statements allow your program to make "either-or" decisions, executing a block of code based on a provided Boolean expression. Iterative statements allow your program to repeat a block of code an indefinite number of times.

Conditional statements are created using the keywords `if`, `elif`, and `else`. Each conditional statement must start with an `if` branch and can be followed by any number of `elif` branches. A final `else` branch may be added at the end as a default case.

Iterative statements - or loops - are created using the keyword `while`. Like conditional statements, loops require a Boolean expression to be used as a test to determine whether another iteration of the loop should be performed. If this test is always true, the loop will never terminate (a so-called infinite loop). Thus, it is important to include code in the body of a `while` statement that allows values used in the test to change so that it can eventually become false.

Implementation

For each function below, write at least 3 example cases as you did in lab 3 unless otherwise noted.

Create lab4.py file and write the following 4 function definitions in it. Add the module docstring t the top of your file as you did in lab 3. For each function, add the docstring and write three examples before you start writing code for the function. You need to import the doctest module and call its testmod() function at the bottom of your lab4.py file as follows:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

`max_of_two(x, y)`

Return the largest of two given integer numbers. Do not use the built-in `max` function; reason through the logic yourself. Your implementation should not have more than two branches and must contain an `else` branch. If both numbers are equal, this function may return either one.

`max_of_three(x, y, z)`

Return the largest of three given integer numbers. This function has the same restrictions as `max_of_two` and should **not** call `max_of_two` but may contain up to three branches.

`mul(x, y)`

Return the product of `x` and `y` without using the multiplication operator. This function must use a `while` loop. You can assume that both x and y are integers and >= 0 (0 or larger).

`exp(x, y)`

Return x^y without using the multiplication or exponentiation operators. This function must use a `while` loop and make calls to the `mul` function. You can assume that x is a positive integer and y is an integer that is >= 0 (0 or larger).

Submission

Show your work to the instructor or TA first. S`ubmit lab4.py file to Canvas`.