Lab 3: Control Flow and Object Equality

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo

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 test cases using `assert` statements, unless otherwise noted.

Create pset3.py file and write the following 4 function definitions in it. Create test_pset3.py file and write at least 3 tests for each function using assert statement. You need to import your functions from pset3 to the test_pset3.py file by adding import statement at the top of the test_pset3.py file.

`max_of_two(x, y)`

Return the largest of two given 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 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.

`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.

`Class Point`

Create point.py file and write a class definition for Point which represent a point in 2D space. Implement all three boilerplate methods(__init__, __repr__, and __eq__).

`Class Circle`

Create circle.py file and write a class definition for Circle which represent a circle in 2D space. Implement all three boilerplate methods(__init__, __repr__, and __eq__). The __init__ method has to take a Point object and an int number as its arguments.

`test_equalities.py`

Create test_equalities.py file and create two Circle objects cir1 and cir2 whose center points and radii are the same. With assert statement, assert that cir1 == cir2 results in True. Create another Circle object cir3 with different center and radius than cir1 and cir2. With assert statement, assert that cir1 == cir3 results in False and cir2 == cir3 results in False. You need to import your classes from point and circle to the test_equalities.py file by adding import statement at the top of the test_equalities.py file.

Submission

Zip `pset3.py, test_pset3.py, point.py, circle.py, and test_equalities.py as one zip file named lab3_[id].zip, where [id] should be replaced with your Cal Poly id, and submit the zip file to polylearn`.