Brian Ross

CSC 201

March 18, 2016

Program 4

Objective:

Program 4 introduces the concepts of functions, regular expressions and modules. The capabilities of the regular expressions module allow for a different approach to manipulating data. In this program the re.search and re.split commands provide a quick and effective way of finding and manipulating the input string that commands the functionality of this program. Partitioning the program into different functions allows for a much cleaner looking program as well as a streamlined task procedure to make the program more concise.

Variables:

somestring – user input with the form (expression)\*(expression)…

sign – searches the string for characters other than[^(^)^\d^\s^+^/^\*^\d\.\d^-]. Used to determine if the sting contained any characters that were deemed invalid

othersign – searched to make sure that the input string contained a mathematical argument

digit – searched to make sure the input string contained digits

more\_conditions – used to determine if the input string contained the pattern “\* (“

more\_conditions – used to determine if the input string contained the pattern “) \*”

args – split the input string a the pattern “)\*(“ to make the previous calculator program valid

list\_of\_args – a list of the expressions from the input string after they had been split at the “)\*(“ indexes

m – replaced extra parentheses with whitespace for the program 2 calculator

memory – list where the calculated values of the expressions were sent

some\_constant – needed for mathematical logic, takes on the product of the numbers in list “memory”

total\_arguement – combines somestring and some\_constant and appends them to list last\_memory to be called by the user

last\_memory – empty list to be filled with the computations of the user’s input

Program:

This program operated by prompting the user for an expression or set of expressions in a certain format, the goal is to determine whether the user inputs a valid expression or not. Using regular expressions to isolate certain acceptable characters and patterns in the user input, the program determines if the string of characters is a valid expression with appropriate arguments and format. In the event that the user input is invalid the user is prompted to enter a new expression. If the user input is valid the program splits the string into its individual expressions in order to evaluate them independently. Once evaluated and appended to a list, the contents of the list are then multiplied together as is the constraint of the program’s desired purpose. The value of the user’s expression input is appended to a list that can be called by the user to print all of the previous user inputs. The program is designed as a while to continually prompt the user to enter expressions that will be verified and possibly evaluated until the user types “end”.

Problems:

Initially, the simpleExpressionIsValid function acts as a sort of gatekeeper for the rest of the program. That functions must have a very specific set of characters, in a very specific order, in order to provide the evaluateSimpleExpression function with a truly valid function. I had problems closing loopholes in the simpleExpressionIsValid function so that evaluateSimpleExpression wouldn’t crash. Having knowledge of regular expressions now, I would likely have completed Program 2 with a different strategy that I used and continued using in Program4.