Preliminary information
This is your first Scam assignment. To run your code, use the following command:

    scam FILENAME

or

    scam -r FILENAME

where FILENAME is replaced by the name of the file containing the program you wish to run. The -r option will automatically run a no-argument function named main on startup.

All assignment submissions should supply a program named author.scm. This program should look like:

    (define (main)
      (println "AUTHOR: Rita Recursion rrita@crimson.ua.edu")
    )

with the name and email replaced by your own name and email.

For each numbered task (unless otherwise directed), you are to provide a program named taskN.scm, with the N corresponding to the task number, starting at one (as in task1.scm, task2.scm, and so on). For example, if task 5 is:

5. Implement factorial so that it implements a recursive process. Name your function fact. It will take a non-negative integer argument.

you should create a program file named task5.scm. The program should look like:

    (define (main)
      (setPort (open (getElement ScamArgs 1) 'read))
      (define args (readExpr))
      (println (apply fact args))
    )

    (define (fact n)
      (if (< n 2) 1 (* n (fact (- n 1))))
    )

The expression beginning with setPort sets the input file pointer to the file named by the first command line argument. The file should contain a parenthesized list of the arguments to be passed to the fact function. The readExpr call in the second expression reads this list of arguments and returns them to the apply function, which passes these arguments to the fact function. Here is one way to run the task5 program.

    $ echo "(5)" > task5.args
    $ scam -r task5.scm task5.args
    120
    $

The filename task5.text is the first command-line argument. The -r option informs Scam to run the main function after the program has been loaded. Note that purpose of your main function is to test the functions you are to define; it should contain no other logic.

For printing, it may be of use to know that you can have actual tabs and newlines within a string, as in:
(println "The quick brown fox
    m
    u
    p
    j
    e
    d

ever the lazy dog")

which will print out as:

The quick brown fox
    m
    u
    p
    j
    e
    d

ever the lazy dog

A useful debugging function inspect. Here is an example usage:

(inspect (+ 2 3))

which produces the output:

(+ 2 3) is 5

Another useful debugging function is pause. It takes no arguments, stopping execution until a newline is entered from the keyboard.

If you have trouble loading a file, try using the comment-out-the rest-of-the-file marker (;$) in your code. You can move this marker around until you find the malformed expression in your code. **DO NOT** leave a ;$ marker in any code you submit for grading. I will be injecting code at the the bottom of your submissions and having that marker present will comment out this injected code.

You may not use assignment in any of the code you write. Nor may you use any looping function such as `while` or `for`. You may not use lists or arrays, unless otherwise specified.

**Tasks**

1. TBD

2. Consider colorizing a value between 0 and 100 so that each integer value corresponds to a CYM color. To determine the CYM color, the intensity of the cyan, yellow, and magenta colors are determined individually. To compute the cyan intensity, the integer value is scaled between 0 and 255 according to a quarter cycle of a left-shifted sine wave. For example, a value of 0 would correspond to a cyan value of 255 while a value of 100 would correspond to a cyan value of zero. The yellow value is computed likewise with a half-cycle of an inverted up-shifted sine wave. For example, values of 0 and 100 would correspond to a yellow value of 255, while a value of 50 would correspond to a yellow value of 0. Finally, a magenta value is computed with a three-quarters of a cycle of an left- and up-shifted sine wave. For example, a value of 0 would yield a magenta value of 255, while a value of 100 would yield a magenta value of 127.5. The `sin` and `cos` functions will be useful for this task.

Use a value of 3.14159265358979323846 for $\pi$.

Your task is to define a function named `cym` which takes a single value as its argument. Your function should return the corresponding CYM values as hexadecimal string. For example, if all the color values are zero, then the function should return the string

#000000

If all the values are 255, the resulting string should be:

#FFFFFF
You may find the `string+` function useful for concatenating substrings.

Note: all computed color values should be truncated. For example, if the actual value computed by a color function is 138.87645673, the function should report 138. The `int` function can be used for this purpose.

3. The Mandelbrot set (for examples, see http://www.softlab.ece.ntua.gr/miscellaneous/mandel/mandel.html) is a set of planar points, a point \((x,y)\) being in the set if the following iteration never diverges to infinity:

\[
r = r \times r - s \times s + x
\]

and

\[
s = 2 \times r \times s + y
\]

with \(r\) and \(s\) both starting out at 0.0. While we can’t iterate forever to check for divergence, there is a simple condition which predicts divergence: if \(r \times r + s \times s > 4\) is ever true, either \(r\) or \(s\) will tend to diverge to infinity. Processing of a point continues until divergence is detected or until some threshold number of iterations has been reached. If the threshold is reached, the point is considered to be in the Mandelbrot set. Obviously, the higher the threshold, the higher the confidence that the point actually is in the set. The points not in the Mandelbrot set can be categorized as to their resistance to divergence. These points are often colorized, as in the previous task.

Define a function, named `mandelbrot-iter`, that takes a threshold as its single argument, and returns another function that can be used to test whether or not a point is in the Mandelbrot set using the given threshold. The returned function takes two arguments, the \(x\)-coordinate, and the \(y\)-coordinate of the point to be tested and it returns the resistance (i.e., the number of iterations until the divergence test succeeds). The return value should be 0 if the point described by the \(x\)- and \(y\)-coordinates is in the Mandelbrot set (i.e., reaches the threshold). You should test for divergence before you test for reaching the threshold.

Example usage:

```scm
(define mandelbrot-tester (mandelbrot-iter 100))
(if (= (mandelbrot-tester 2 3) 0)
  (print "point (2,3) is in the Mandelbrot set!\n")
  (print "point (2,3) is not in the Mandelbrot set.\n")
)
```

In the above example, the threshold for determining whether or not a number is in the Mandelbrot set is 100.

4. Define a function named `root-n` which creates a function for calculating the \(n\)th root of a given argument. Note that for a number \(x\) and a guess \(y\), a better guess for the second root of \(x\) is \(y + \frac{x}{y^2}\) and a better guess for the third root of \(x\) is \(2 \times y + \frac{x}{y^3}\). Extrapolate this pattern to figure out how to define and return a function that calculates the \(n\)th root. The form of your returned function should follow that in the text for square root.

Define a second function name `root` that takes two arguments, \(n\) and \(x\). This function should call `root-n` with \(n\) and then call the resulting function with \(x\). Example:

```scm
scam> (root-n 2)
<function anonymous(x)>
scam> (fmt \"%.2f\" (root 2 144))
12.00
```

Test for convergence by comparing consecutive guesses to see if they are close enough; do not compare with strict equality or against an absolute difference.

Example:

```bash
$ echo "(2 144)" > task4.args
$ scam -r task4.scm task4.args
12.000000000
$
5. Define a function, named \texttt{crazyTriangle}, that constructs a function that will print out \( n \) levels of Pascal's triangle, but with a twist. The leftmost and rightmost numbers at each level are not necessarily ones, as with Pascal's triangle, but are given as the first and second arguments of \texttt{crazyTriangle}. The returned function takes a single argument, which is the number of levels in the triangle to be printed. The output produced by \((\texttt{crazyTriangle 1 1) 6}) would be six levels of Pascal's triangle:

\begin{verbatim}
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
\end{verbatim}

The output produced by \((\texttt{crazyTriangle 1 2) 6}) would be:

\begin{verbatim}
      1
     1 2
    1 3 2
   1 4 5 2
  1 5 9 7 2
 1 6 14 16 9 2
\end{verbatim}

Note that the apex is always the first argument.

Your triangle printing function must print one level to a line with lower levels above upper levels. The widest level must have no preceding spaces; all other levels can only have spaces preceding the first value in the level. All levels must have only a newline following the last value in the level. Finally, your levels need to be centered around the apex (but don’t worry if the triangle skews rightward with multi-digit entries).

Your function must implement a tree-recursive process and should not overflow an integer \textit{while computing a triangle entry} (unless the final value itself overflows).

Example:

\begin{verbatim}
$ echo "((1 1)" > task5.args
$ echo "(5)" >> task5.args
$ scam -r task5.scm task5.args
  1
    1 1
      1 2 1
          1 3 3 1
              1 4 6 4 1
$\end{verbatim}

6. TBD
7. TBD
8. TBD
9. TBD
10. TBD

\textbf{Compliance}

Output format has to match exactly, spacing and all. There can be no whitespace other than a newline after the last printable character of each line in any output. No lines of output are indented, unless explicitly specified.

\textbf{Handing in the tasks}

To submit assignments, you need to install the \textit{submit system}:

- \textit{Linux or Windows Bash instructions}
- \textit{Mac instructions}

For preliminary testing, send me all the files in your directory by running the command:
submit proglan lusth test1

For your final submission, use the command:

submit proglan lusth assign1

The submit program will bundle up all the files in your current directory and ship them to me. Thus it is very important that only the files related to the assignment are in your directory (you may submit test cases and test scripts). This includes subdirectories as well since all the files in any subdirectories will also be shipped to me, so be careful. You may submit as many times as you want before the deadline; new submissions replace old submissions.