1. Vocabulary Matching: (15 points)

Write the number from the correct definition in the blank next to each term on the left:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>__1__Program</td>
<td>A sequence of instructions that specifies to a computer actions and computations to be performed.</td>
</tr>
<tr>
<td>__2__Keyword</td>
<td>A reserved word that is used by the compiler to parse a program; you cannot use things like if, def, and while as variable names.</td>
</tr>
<tr>
<td>__3__Runtime error</td>
<td>An error that does not occur until the program has started to execute but that prevents the program from continuing.</td>
</tr>
<tr>
<td>__4__Integer division</td>
<td>An operation that divides one integer by another and yields an integer. It yields only the whole number of times that the numerator is divisible by the denominator and discards any remainder.</td>
</tr>
<tr>
<td>__5__Variable</td>
<td>A name that refers to a value.</td>
</tr>
<tr>
<td>__6__Integer</td>
<td>A Python data type that holds positive and negative whole numbers.</td>
</tr>
<tr>
<td>__7__Operator</td>
<td>A special symbol that represents a simple computation like addition, multiplication, or string concatenation.</td>
</tr>
<tr>
<td>__8__Floating-point</td>
<td>A format for representing numbers with fractional parts.</td>
</tr>
<tr>
<td>__9__Type conversion</td>
<td>An explicit statement that takes a value of one type and computes a corresponding value of another type.</td>
</tr>
<tr>
<td>__10__Function</td>
<td>A named sequence of statements that performs some useful operation. They may or may not take parameters and may or may not produce a result.</td>
</tr>
<tr>
<td>__11__Syntax error</td>
<td>An error in a program that makes it impossible to parse (and therefore impossible to interpret).</td>
</tr>
<tr>
<td>__12__Print statement</td>
<td>An instruction that causes the Python interpreter to display a value on the screen.</td>
</tr>
<tr>
<td>__13__function call</td>
<td>A statement that executes a function. It consists of the name of the function followed by a list of arguments enclosed in parentheses.</td>
</tr>
<tr>
<td>__14__Semantic error</td>
<td>An error in a program that makes it do something other than what the programmer intended.</td>
</tr>
<tr>
<td>__15__Frame</td>
<td>A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function.</td>
</tr>
</tbody>
</table>

One point for each correct answer.
2. Vocabulary Matching - Part 2 (15 points)
Write the number from the correct definition in the blank next to each word:

<table>
<thead>
<tr>
<th>11</th>
<th>Modulus operator</th>
<th>1</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Conditional statement</td>
<td>3</td>
<td>Comparison operator</td>
</tr>
<tr>
<td>7</td>
<td>Block</td>
<td>13</td>
<td>Recursion</td>
</tr>
<tr>
<td>9</td>
<td>Base case</td>
<td>12</td>
<td>Temporary variable</td>
</tr>
<tr>
<td>10</td>
<td>None</td>
<td>8</td>
<td>Guardian</td>
</tr>
<tr>
<td>2</td>
<td>Incremental development</td>
<td>14</td>
<td>Multiple assignment</td>
</tr>
<tr>
<td>15</td>
<td>Encapsulate</td>
<td>4</td>
<td>Generalize</td>
</tr>
<tr>
<td>5</td>
<td>Iteration</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One point per correct answer.

1. An expression that is either true or false.
2. A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time.
3. One of the operators that compares two values: ==, !=, >, <, >=, and <=.
4. To replace something unnecessarily specific (like a constant value) with something appropriately general (like a variable or parameter).
5. Repeated execution of a set of statements using either a recursive function call or a loop.
6. A statement that controls the flow of execution depending on some condition.
7. A group of consecutive statements with the same indentation.
8. A condition that checks for and handles circumstances that might cause an error.
9. A branch of the conditional statement in a recursive function that does not result in a recursive call.
10. A special Python value returned by functions that have no return statement, or a return statement without an argument.
11. An operator, denoted with a percent sign (%), that works on integers and yields the remainder when one number is divided by another.
12. A variable used to store an intermediate value in a complex calculation.
13. The process of calling the function that is currently executing.
14. Making more than one assignment to the same variable during the execution of a program.
15. To divide a large complex program into components (like functions) and isolate the components from each other (by using local variables, for example).
3. Write Code (5 points)
Write a function **get_number** that prompts the user to enter a number and returns a floating point value. You do NOT need to check for errors. (Assume the user always enters a valid number).

```python
def get_number():
    num_str = raw_input("Enter a number!")
    num_float = float(num_str)
    return(num_float)
```

+1 pts = Function prompts user for number
+2 pts = Function returns the value
+2 pts = Function converts string to float.

4. Write Code (5 points)
Write a function **return_largest** that accepts 3 parameters \((x, y, z)\) and returns the largest of the three. For example, **return_largest**\((7, -34, 23.8)\) should return 23.8.

```python
def return_largest(X, Y, Z):
    if (X >= Y) and (X >= Z):
        return(X)
    elif (Y >= X) and (Y >= Z):
        return(Y)
    else:
        return(Z)
```

+2 pts = function takes 3 parameters
+2 pts = correct value is selected
+1 pts = value is returned.

Test cases to try out on your code:
return_largest(4,4,5)
return_largest(5,5,4)

5. Write Code (5 points)
Write a function **draw_a_square()** that will drive your scribbler robot in a square (polygon with 4 equal sides and four 90 degree corners), and beep at each of the four corners. You may assume that `turnRight(1, 0.5)` will turn your scribbler exactly 90 degrees. You do not need to include the `from myro import *` or `initialize()` calls, and you may assume they have already been done for you. Do NOT use a loop, and do NOT use recursion.

```python
def draw_a_square():
    forward(1,1)
    turnRight(1,0.5)
    beep(1,800)
    <above repeated 3 more times>
```

+2 = does a forward, turn, and beep each time, in any order (but order must be consistent)
+2 = repeats 4 times without for/while loops
+1 = in a function
def n_lines(n):
    if n > 0:
        print "Line!"
    n_lines(n-1)

6a. Program Comprehension (1 point)
How many times will the string “Line!” be printed when n_lines is called with n=4?
Number Four Note that 0 > 0 is False, so the print statement does not execute when N=0.

6b. Simple Stack Diagram (4 points)
Draw a stack diagram for the function n_lines called with n = 4. (i.e. n_lines(4) )
Include the value of any local variables, and remember to start with __main__.

| __main__ | + 2 points - stack has 5 function calls of n_lines. |
| n_lines: N = 4 | +1 point - starts with __main__ |
| n_lines: N = 3 | +1 point - N = 4,3,2,1,0 in the 5 calls. |
| n_lines: N = 2 |
| n_lines: N = 1 |
| n_lines: N = 0 |

7. Write Code (4 points)
Write a function with infinite recursion named run_forever. Your function should have no parameters, and it should run forever when called (on an ideal computer, in a real computer it would eventually run out of memory.) You may add a print statement if you wish.

```python
def run_forever():
    run_forever() # This is a recursive call
```

+2 points - it's a function
+2 points - that calls itself, without a termination clause.
8. Write Code (6 points)
Rewrite the function `n_lines` from question 6 using a for loop instead of recursion.

```python
def n_lines(N):
    for i in range(N):
        print "Line!"
```

+1 pt = it's in a function
+1 pt = function accepts N as a parameter
+2 pt = The for statement is correct (executs N times)
+2 pt = Function has the same behavior as the original: Prints Line! N times.

9. Write Code (8 points)
Rewrite the function `n_lines` from question 6 and 8 using a while loop instead of recursion or a for loop. You may use n as your looping variable.

```python
def n_lines(N):
    while N > 0:
        print "Line!"
        N = N - 1
```

+1 pt = it's in a function
+1 pt = Function accepts N as a parameter
+2 pts = While statement is correct
+2 pts = Looping Variable is decremented
+2 pts = Function has same behavior as the original: Prints Line! N times.

10. Stack Diagram (4 points)
Draw a stack diagram for the code you wrote for problem 9 when `n_lines` is called with n=4 (e.g. `n_lines(4)`). Remember to start with `__main__` and to show all local variables. If local variables change during execution, strike them out and show the new value each time they change.

```
__Main__

__Main__
n_lines:

N = 4
N = 3
N = 2
N = 1
N = 0
```

+ 2 = has 2 frames, __main__ and n_lines
+2 = Shows N (or looping variable) decrementing as the while loop runs. (-1 point if they forget the last value of the looping variable that makes the continuation condition turn false.)
11. Python Expression Evaluation (14 points)
Pretend that you are the Python Interpreter (IDLE window). What do you print or return when each of the following statements are entered?

Example: \((7+4) / 2\) \hspace{1cm} Result: \underline{5}

Example: \(\text{range}(4)\) \hspace{1cm} Result: \underline{[0, 1, 2, 3]}

1. \((7.0 + 4) / 2\) \hspace{1cm} Result: \underline{5.5}
2. \(7 + 3 / 2\) \hspace{1cm} Result: \underline{8}
3. \(\text{range}(4, 8)\) \hspace{1cm} Result: \underline{[4, 5, 6, 7]}
4. \(\text{range}(4, 8, 2)\) \hspace{1cm} Result: \underline{[4, 6]}
5. \(7.0 > 5.0\) \hspace{1cm} Result: \underline{True}
6. \(7 + 3 / 2 > 8\) \hspace{1cm} Result: \underline{False}
7. \(\text{print} \ "\text{Pumpkin %.3f} \ %3.1459\) \hspace{1cm} Result: \underline{Pumpkin 3.146}

Grading: 2 points for each correct answer.

12. Extra Credit (1 point)

What is the name you gave your robot? \underline{_____________

Grading: Any name gets a point!

13. Extra Credit (2 points)

Where does Python get it's name?

Monty Python's Flying Circus, a British comedy TV show.

14. Extra Credit (3 points)

What are Isaac Asimov's 3 laws of robotics?

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given to it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Grading: 1 point for each law that is close.