Coding, or programming, is how we tell the computer to perform computation on the data and tell us the result. The most basic code will allow us to perform calculator arithmetic, like adding and subtracting. Beyond basic computation, we can evaluate and construct relationships in data. One important operation is comparison. Comparison operators take two inputs and tell you whether the first item is greater than (>), less than (<), or equal to (==:)) the second. These operators can also be combined (>= greater than or equal, <= less than or equal, != not equal). There are other operators and you should feel free to look them up.

Operations can be generalized by packaging them into a function using the **def** keyword. A function needs a name, and can take an argument (data passed into the function). A function can **return** a value using the return keyword. One way to explore a function is to follow the data as it moves through the function, diagramming out the control flow the program steps through. Then write or print out the variables as it moves through the function. Another quick way to learn about functions is to change the data used by the function, and see what happens. Yet another way is to change the arithmetic or code in some way, and test what happens.

Data are the values that are used in computing. They are the content that is manipulated, stored, and operated on. Data comes in different **types**. Numbers are easy to recognize. They can be whole numbers, which can be split further into integers and floats. **Integers** are whole numbers like 0, -1, 4.   **Floats** are numbers with decimal points, like 0.4, -24.5643. Whole integers are great for counting. There's a function you should know called **range** that returns a **list** of consecutive numbers. In contrast, floats are good for precision calculations. Be aware that in python when you divide two integers, you will get a rounded down integer result. If you want a precise result, you have to use floats. You can cast a number into a float by putting a decimal in it it (e.g. 4 becomes 4.0) or you can specify **float**(4). Similarly, you can cast a float into an integer by typing **int**(4.5) to chop off the decimal and give you 4.

Variables are the names given to keep track pf data. You declare a variable by using the = sign. If you assign a variable, then you can change the value. An example of changing the value of variable ***a*** is below:

```
a = 10
print a    #prints 10
a = a +3
print a   #prints 13, the value of a has been changed
```

Another type of data are alphanumeric characters. In python, a letter inside single or double quotes is the way you specify a character (e.g. 'a'). A sequence of alphanumeric characters is called a **string**. You can create a string by putting it between matching double or single quotes like this 'hello world' or "Madam I'm Adam." Strings can also be added and multiplied, so "abc" * 3 = "abcabcabc" and "abc"+"def" = "abcdef"

Strings and other sequences (like lists) can have **length**, a property describing how many items are in the sequence**.** A **list** is a sequence of ordered items separated by commas, enclosed by square brackets [ ]. Sequences also have a handy feature called **slices**, which is a shorthand for looking at different portions of the string. Get familiar with how to use slices and indexes in strings.

There are ways to identify a particular character inside the sequence using its **index** location**.** For example "Madam I'm Adam"[0] will return 'M' (the letter in position 0) and len('hello') will return 5 (the number of characters in the string). Note that index number for always start at 0. The range and slice notation have the same convention, where the first argument is the start_index, and the range returned is exclusive of th second argument, the **end index**/ Both range and slices have a third argument, a skip amount. Know how to use these arguments for slices and ranges.

You'll be asked to create functions in python. Know how to use the def keyword, return statements, and the aggregator recipe. Know how to **iterate on** (python can consider each item in a sequence one at a time) using **for** loops. Strings have particular operations like *string*.upper(), *string*.find(), Lists have a lot of operations built in, like reverse (*lis*t.reverse()) and sort (l*ist*.sort()). You can convert from a list to string using *string*.join(l*ist*). You can convert from a string to a list by using *striing*.split(l*ist*) with different delimiter arguments.

Be mindful of indentations, as they are meaningful. Statements that start subordinate blocks will end in colons, with subsequent blocks indented. Likewise, the type of parenthesis you use is important (square for lists, round for function calls). Misspelling or mixing up variables will cause unintended errors. Python error statements can usually help pinpoint these problems.

One of the best ways to study for the quiz is to look through all the code in the lecture notes and homework, and see if you can understand them. If you have time, try watching the Programming Fundamentals in the Real World class on Lynda.com and following through the exercises in your ipython notebook.

Examine and guess what the following code does. Type it into ipython to confirm your guess:

```
print 4 / 0.1
```

```
print 3 / 2
```

```
print ‘grape’ + ‘vine’
```

```
print ‘867’ + ‘5309’
```

```
print 867 + 5309
```

```
a = 14
b = 20
```

```
if (a == b):
    print ‘tie’
else print a > b
```

```
mynumber = 111
mystring = ‘yay’
print mynumber * 2
print mystring * 2
```

```
alist = range(0,5)
print type(alist)
```

```
print alist[0]
```

```
print alist[-1]
```

```
print alist[::-1]
```

```
print alist * 2
```

```
print alist[3] * 2
```

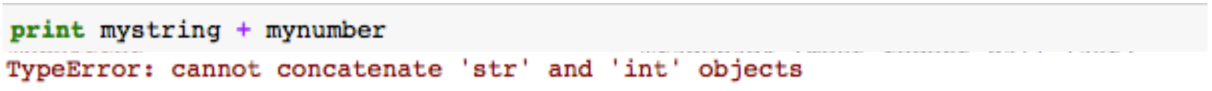
```
blist = range(100,150,10)
print len(blist)
```

```
print blist[1]
```

```
print blist[-2]
```

Why does `print 5/10` return 0?

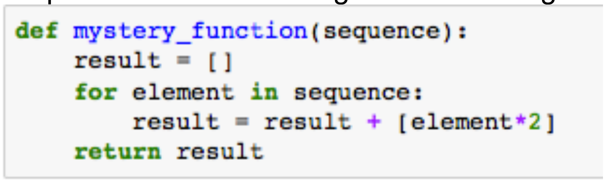
Explain why the following expression produces an error.



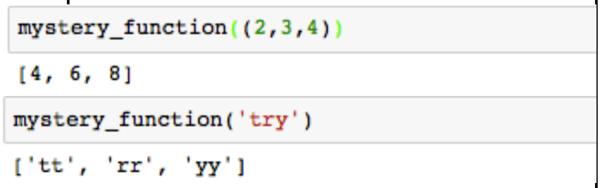
```
print mystring + mynumber
TypeError: cannot concatenate 'str' and 'int' objects
```

How might you fix the above error?

Explain what the following function does given the outputs shown.



```
def mystery_function(sequence):
    result = []
    for element in sequence:
        result = result + [element*2]
    return result
```



```
mystery_function((2,3,4))

[4, 6, 8]

mystery_function('try')

['tt', 'rr', 'yy']
```

How might you change the mystery_function to do something different? Give a brief explanation and show how you would change the code.