I skipped the first three secontions and focused on sections four and five as directed because I have been programming in python for about two years now.

The most well known statement type in python is the if statement. The statement is if(condition) do (something) and else(condition is untrue) then do something else. But there does not need to be another statement. An if statement can actually exist on its own. In python, it is more proper to write, if, elif, and else in that order. If something is true, if something else is true, then if nothing is true.

A for statement in python is different from iterating arithmetically. Python lets you control the iteration and halting point by this type of statement.

The break statement breaks out of the innermost for or while loop. Most loops have an else, or a terminating statement but when a break statement is executed it immediately throws the program from the loops.

The pass statement does nothing, and it can be used to fill an empty class or function as python does not like to have classes or functions be empty.

I think the most interesting code in this section is the default argument values example.

**def** ask\_ok(prompt, retries=4, complaint='Yes or no, please!'):

**while** **True**:

ok = raw\_input(prompt)

**if** ok **in** ('y', 'ye', 'yes'):

**return** **True**

**if** ok **in** ('n', 'no', 'nop', 'nope'):

**return** **False**

retries = retries - 1

**if** retries < 0:

**raise** IOError('refusenik user')

print complaint

This shows me something I did not know before. You can assign a value for more than one argument. Normally I used and or or statements to get this job done. For example I would use if y and ye and yes are in the input return true. Or if n or no or nop or nope where in the input return false. This definitely streamlines the process and makes the code more efficient. This function can be called in several ways, and also introduces the [in](https://docs.python.org/2/reference/expressions.html#in) keyword. This tests whether or not a sequence contains a certain value.

Python can also use unpacked lists as arguments. If you have a function that takes in two numbers as arguments you can pass in the list instead of the argument individually. Before I have taken time to code around this problem so it is nice knowing I do not have to search through a list to get individual values, I can instead just give a function a list.

Part five talks about data structures in python. List.append adds an item to the end of the list. List.extend extends the list by appending all the items in the given list. List.insert adds an intel in the given position. List.remove removes the first item in the list whose value is at x. List.pop removes the item at the given position, if no position is given it removes the last item in the list. List.index gives the value of the item in the list at the position. List.count returns the amount of times a value is in a list. List.sort sorts the item of a list in place.List.reverse reverses the elements of a list in place.

In python you can use a list as a stack, and I have before. When I do not have a stack class made, I use a list as a stack and use .append and .pop to simulate a stack.

Python lists can be used and queues. To do this you have to use collections.deque and it is meant to simulate the fast appends and pops from both ends.

Python has three built-in functions that make coding easier with lists. The filter function returns a sequence with the terms listed. It returns true if the sequence exists or can return the sequence. The map function calls a function on the sequence. For example if you had a list of numbers to define cubes you can map every three numbers to a cube function. The reduce function calls a function and returns a single number. This can be used for recursive functions.

Another thing I learned is how to streamline list creation. Before I had to iterate through lists to be able to create things using a function.

For example, assume we want to create a list of squares, like:

>>> squares = []

>>> for x in range(10):

... squares.append(x\*\*2)

...

>>> squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

We can obtain the same result with:

Squares = [x\*\*2 for x in range (10)]

Matrices can also be easily understood by python. A 3x4 matrix can be done with three lists of length four. This is really useful for mathematical purposes.

These are some of the most interesting things I learned in the chapters. I took algorithms and data structures already so I am familiar with loops, tuples, sequences, trees, dictionaries. But it was useful to revisit everything and see it in a more direct capacity. I also love that instead of doing complex code with these structures I can just read about them :)