**Chapter 5**:

**Loops and iterations Part 1**

* Loops (repeated steps) have iteration variables that change each time through a loop. Often these change each time through a loop. Often these iteration variables go through a sequence of numbers.
* While: functions like if statement, question that leads to a true of false, colon then indented block, de-indent to end block. If its true it will run the code under the while statement, and then go back up and ask the question again, and it will continue to do so until its not true, then it will skip the while statement and move on.
  + To create a iteration variable, or how many times to count down or up, set value x = value , then create your while x > 0: then indent to a print(x) indent again to x = x-1 then DE indent and print a word and then print(x) again to print the last value of x.
  + If you do nothing with the iteration variable you create a infinite loop
* Zero trip loops are not even guaranteed to run once, so i.e x=0 while x > 0 print(word) unindent print(wordword) <- in this example the while will never run
  + Functions just like a if statement, if the while function isn’t true, it will skipp right over it.
* Breaking out of a loop: The break statement ends the current loop and jumps to the statement immediately following the loop. It is like a loop test that can happen anywhere in the body of the loop. To initiate a break, you create a if statement inside of a while statement, set your if value, then indent and type “break”. The way to make this a loop is to unindent from break, type a print(x) or some form of function(s) then unindent again and continue or finish your code. So it will run the loop until your if statement is true, until it is true, then it will break from the loop and wont ask the question again, it will just move on to the line of code following the while statement. Example of break inside of a while statement:

if line == 'done':

break

* + Can be 100s of lines, as soon as you hit a break statement, the code will exit to whatever the next line is beyond the end of the loop
* Continue: ends the current iteration and jumps to the top of the loop and starts the next iteration. Stop this iteration, were done with it, go back up to the top of the loop and ask the question again.

An example would be: while True:

line = input('> ')

if line[0] == '#':

continue

if line == 'done':

break

print(line)

print('Done!')

* + Exist in the while statement, as part of a if statement i.e) if x == 0 then on the next line indent and type continue, if the value of x is equal to zero it will ask the question again
* ( While = true )is a way to create a infinite loop
* Indefinite Loops: (While) loops are called indefinite loops because they keep going until a logical condition becomes false

**Loops and iterations Part 2**

* Definite Loops: Quite often we have a list of items of the lines in a file, a finite (limited) set of things. We can write a loop to run the loop once for each of the items in a set using the python “for” construct, this is called “Definite Loops” because they execute an exact number of times
  + Another way to say this is: Definite loops iterate through the members of a set
  + Using “for” and “in” together makes a simple definite loop, i.e) for I in (5, 4, 3, 2, 1): and then next line indent print(i), then next line unindent print(‘go’) will count down from 5 to 1 printing 54321 then printing go.
  + You can do the same thing with strings, by setting a variable with multiple string values i.e) friend = (‘sam’, ‘tom’, ‘mike’, ‘jim’) then writing next line for friend in friend: next line indent print(‘welcome: ‘, friend) next line unindent print(done) with say welcome sam then welcome tom, and continue to go through your variable values from left to right until there are no more values, and then it will stop the loop and go the the next line outside of the loop
  + Using for and in, your essentially asking if its done or has it gone through all the values you assigned to the variable
  + Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the sequence or set

**Loop Idioms: What we do in loops (Part 3 of loops and iterations)**

* Patterns that have to do with how we construct loops
* The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time.
* Set initial variable(s) equal to value(s), then for “thing” in data: next line with indent were looking for something or to do something to each entry (value) separately, updating the variable, and then continuing the loop, until the loop is over then the code stops the loop and looks at the variables, or you get your “pay off”
* Example for finding the largest value:

largest\_so\_far = -1

print(‘Before ‘, largest\_so\_far)

for the\_num in [9, 41, 12, 3, 74, 15] :

if the\_num > largest\_so\_far :

largest\_so\_far = the\_num

print(largest\_so\_far, the\_num)

print(‘After ‘, largest\_so\_far)

The above would look like;

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

* + In the above example, we set a varible (largest\_so\_far) = to -1, basically creating a largest so far “folder” then in our next line is a print statement, then next were setting ANOTHER varible “the\_num” and using it in a for and in loop, and setting multiple values
  + Then our next line is running through a if statement inside of our for loop, saying if the\_num is greater then largest\_so\_far, set largest \_so\_far = the\_num.
    - Basically we start off with the largest\_so\_far “folder” with the value or “file” set to -1 initially, then were setting a loop up to ask the question “is this number bigger then the number that’s inside the “folder” largest so far” and if it is, then we replace that number with the bigger number, and if its not, then we move onto the next value that is in the set for the\_num. Once that loop is done, we end up with our largest number inside on largest\_so\_far and we print that to screen

**More Loops (part 4 of loops and iterations)**

* Count Loop: To count how many times we execute a loop we introduce a counter variable that starts at 0 and we add one to it each time through the loop. So you set your variable equal to zero, then run your for and if statement, and inside of that you put your counter which would look like x = x +1 after setting x = 0 when you initially set your variable. This will count how many times you’re for and in statement ran
* Summing Loop: To add up a value we encounter in a loop
  + Introduce a dum variable that starts at 0 and we add the value to the sum each time through the loop
  + So your adding the for value to the variable you set (0) and then running the loop, and each time it runs and adds, its saving the new variable value and adding the next for value to it, and repeating until there are no more for values, the end result is all of the for values added together, the adding like this is known as running total, and when the loop is finished you have your actual total
  + Example for summing loop:

Zork = 0

Print(Before ‘, zork)

For thing in [9, 41, 12, 3, 74, 15] :

Zork = zork + thing

Print(‘After ‘, zork)

* Finding the average: in a loop is just taking the counting and sum patterns and divides when the loop is done, so if you had a sum of 6 numbers equal to 154, your counter would be at 6 by the end of the loop, the total of all the numbers added together would be 154 and you would divide 154 by 6 to get 25.666. it would look like: print(‘After ‘ sum /count) if your variables were named sum and count.
* Filtering in a loop: We use an if statement in the loop to catch / filter the values we are looking for
  + You set a if statement where you ask is x greater then a number you set: x >20, after this you could print to screen each time the value is bigger then 20. Then you run through your for variables values and whatever numbers are larger then 20 will print to screen, if they aren’t it wont print to screen, and once the loop is done it will continue on to the next line outside of the loop
* Search using a Boolean (only two values, true and false) value: If we just want to search and know if a value is found, we use a variable that starts at false and is set to true as soon as we find what were looking for
  + Example of this:

Found = false

Print(‘Before ‘, found)

For value in [9, 41, 12, 3, 74, 15] :

If value == 3 :

Found = True

Print (found, value)

Print(‘After ‘, found)

* + - In this example it runs the loop until the specified number 3 is found, then sets found equal to true, stops the loop and exits to the next line outside of the loop and prints the found value “true” to the screen.
* None value: setting the value of a variable to “None”, used to indicate empty, but it is assigned a flag. Example of none value:

Smallest = None

Print(‘Before ‘)

For value in [ 9, 41, 12, 3, 74, 35]

If smallest is None:

smallest = value

elif value < smallest :

smallest = value

print(smallest, value)

print(‘After’, smallest)

* + - First line is setting None to smallest, capturing the notion that the smallest so far is nothing. Thelistn after the for statement, we use if and is, saying if smallest number is none, set the new value of smallest to the first for value. So now for the rest of the code execution, the “if, is” statement where smallest is none, will never run again, and the loop will continue to the elif and run until the for values have been exhausted.
* Is and Is not: “is” is an operator used in logical expressions and implies “is the same as”, similar to == but stronger in that it demands the same type AND value of the variable. “Is not” is also a logical operator. Usually only use is one Booleans and None types.