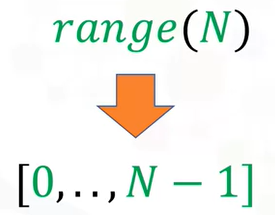
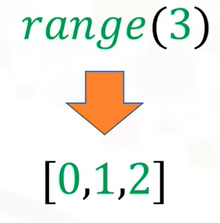
In this video we will cover Loops in particular for loops and while loops. We will use many visual examples in this video. See the labs for examples with data. Before we talk about loops, let's go over the range function.

**Range Function**

The range function outputs and ordered sequence as a list I. If the input is a positive integer, the output is a sequence.

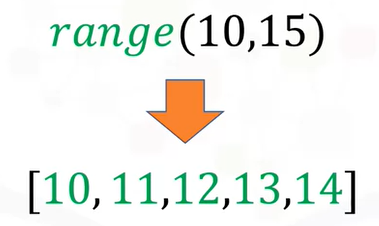


The sequence contains the same number of elements as the input but **starts at zero**. For example, if the input is three the output is the sequence zero, one, two.



If the range function has two inputs where the first input is larger than the second input, the output is a sequence that starts at the first input. Then the sequence iterates up to but not including the second number.

For the input 10 and 15 we get the following sequence. See the labs for more capabilities of the range function.



Please note, if you use Python 3, the range function will not generate a list explicitly like in Python 2.

E.g. range(3)

= range(0, 3)

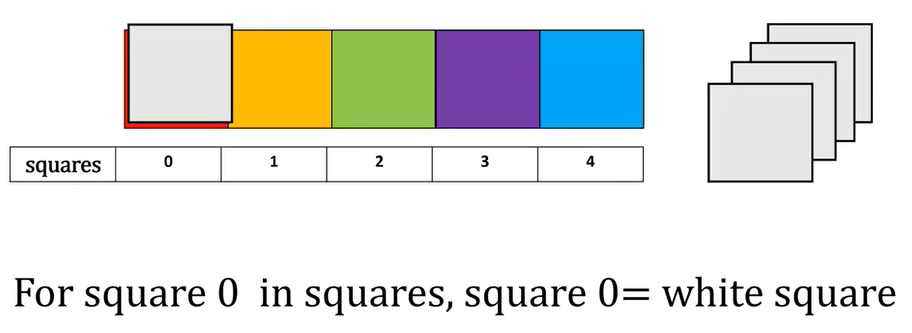
**For Loops**

In this section, we will cover for loops. We will focus on lists, but many of the procedures can be used on tuples.

Loops perform a task over and over. Consider the group of colored squares. Let's say we would like to replace each colored square with a white square.

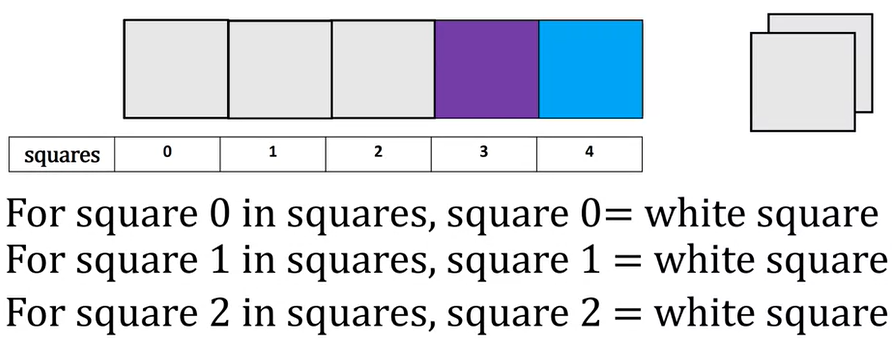
Let's give each square a number to make things a little easier and refer to all the group of squares as squares.

If we wanted to tell someone to replace squares zero with a white square, we would say equals replace square zero with a white square or we can say four squares zero in squares square zero equals white square.



Similarly, for the next square we can say for square one in squares, square one equals white square.

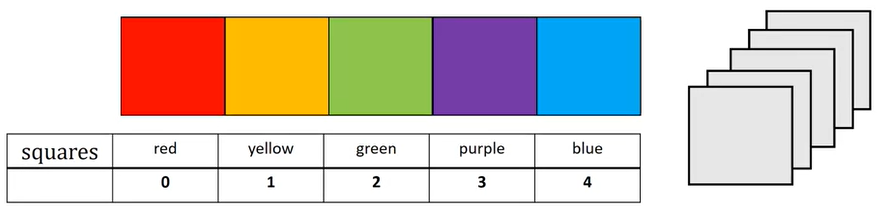
For the next square we can say for square two in squares, square two equals white square. We repeat the process for each square.



The only thing that changes is the index of the square we are referring to. If we're going to perform a similar task in Python we cannot use actual squares. So let's use a list to represent the boxes.

Each element in the list is a string representing the color. We want to change the name of the color in each element to white. Each element in the list has the following index.

This is a syntax to perform a loop in Python. Notice the indent, the range function generates a list.



= ['white', 'white', 'white', 'white', 'white']

The code will simply repeat everything in the indent five times. If you were to change the value to six it would do it 6 times. However, the value of I is incremented by one each time.

In this segment we change the I element of the list to the string white. The value of I is set to zero. Each iteration of the loop starts at the beginning of the indent. We then run everything in the indent.

**for x in range(0,3):**

**print(x)**

= 0

1

2

The first element in the list is set to white. We then go to the start of the indent, we progress down each line. When we reach the line to change the value of the list, we set the value of index one to white. The value of I increases by one. We repeat the process for index two. The process continues for the next index, until we've reached the final element.

**Iterate through lists or Tupples**

We can also iterate through a list or tupple directly in python, we do not even need to use indices.

Here is the list squares. Each iteration of the list we pass one element of the list squares to the variable square. Lets display the value of the variable square on this section.

**squares=['red','yellow','green']**

**for square in squares:**

**square**

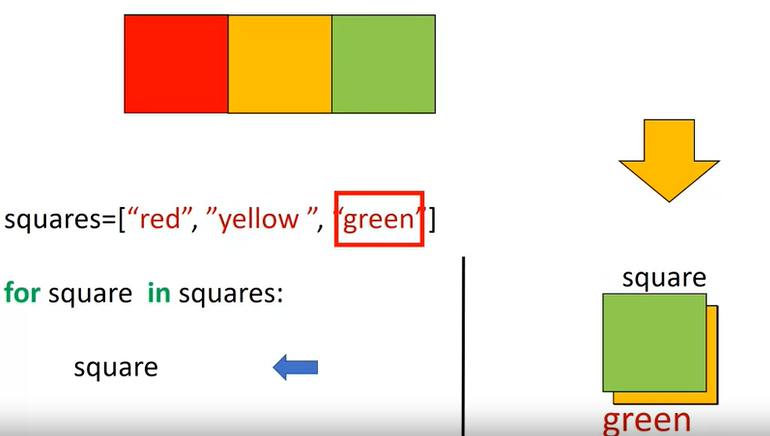
**print(square)**

= green

For the first iteration, the value of square is red, we then start the second iteration. For the second iteration, the value of square is yellow. We then start the third iteration. For the final iteration, the value of square is green.

**Enumerate**

A useful function for iterating data is enumerate. It can be used to obtain the index and the element in the list.



for x in ['A','B','C']:

print(x+'A')

= AA

BA

CA

Let's use the box analogy with the numbers representing the index of each square. This is the syntax to iterate through a list and provide the index of each element.

**squares=['red','yellow','green']**

**for i,square in enumerate(squares):**

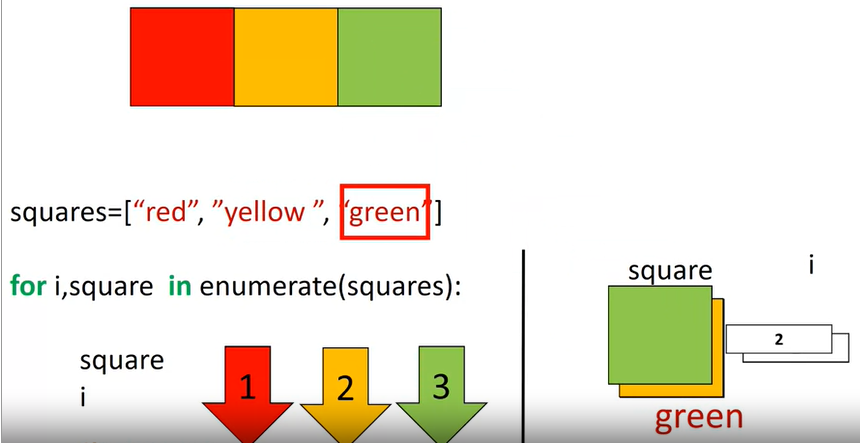
**square**

**i**

We use the list squares and use the names of the colors to represent the colored squares. The argument of the function enumerate is the list. In this case squares the variable I is the index and the variable square is the corresponding element in the list.

Let's use the left part of the screen to display the different values of the variable square and I for the various iterations of the loop.

For the first iteration, the value of the variable is red corresponding to the zeroth index, and the value for I is zero for the second iteration. The value of the variable square is yellow, and the value of I corresponds to its index i.e. 1. We repeat the process for the last index.



E.g

**for i,x in enumerate(['A','B','C']):**

**print(i,x)**

0 A

1 B

2 C

**While Loops**

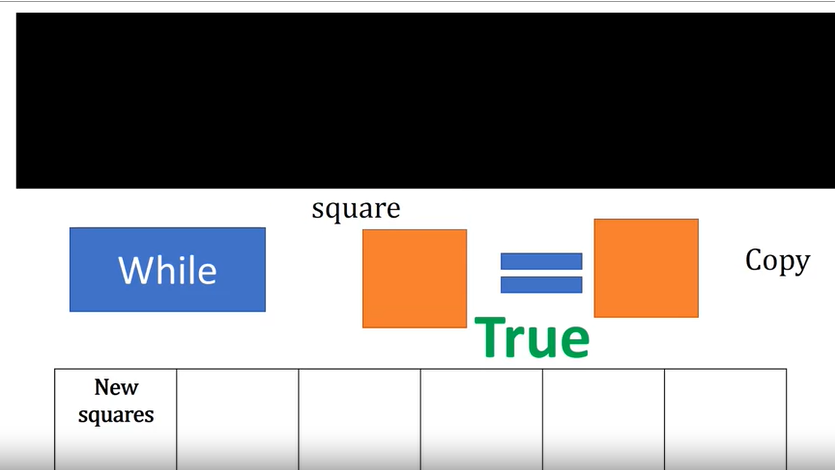
While loops are similar to for loops but instead of executing a statement a set number of times a while loop will only run if a condition is met.

Let's say we would like to copy all the orange squares from the list squares to the list New squares. But we would like to stop if we encounter a non-orange square.

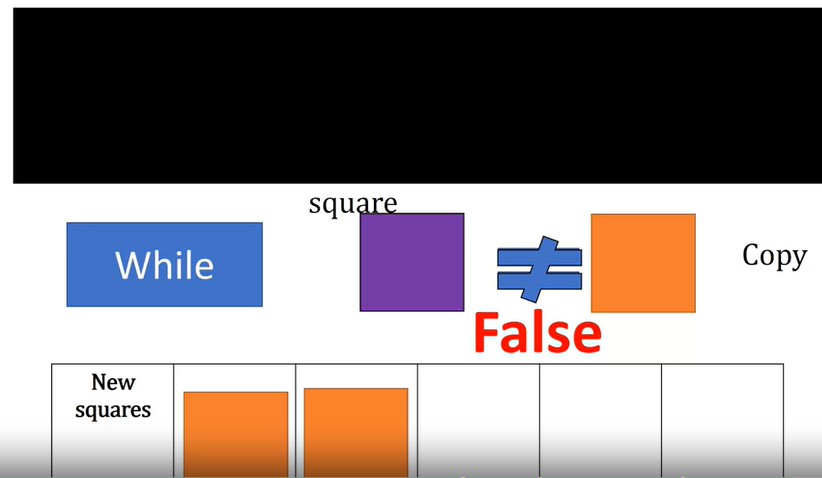


We don't know the value of the squares beforehand. We would simply continue the process while the square is orange or see if the square equals orange. If not, we would stop.

For the first example, we would check if the square was orange. It satisfies the conditions so we would copy the square. We repeat the process for the second square. The condition is met. So we copy the square.



In the next iteration, we encounter a purple square. The condition is not met. So we stop the process. This is essentially what a while loop does.



Let's use the figure on the left to represent the code.

**squares=['orange','orange','purple','blue']**

**Newsquares=[]**

**i=0**

**while(squares[i]=='orange'):**

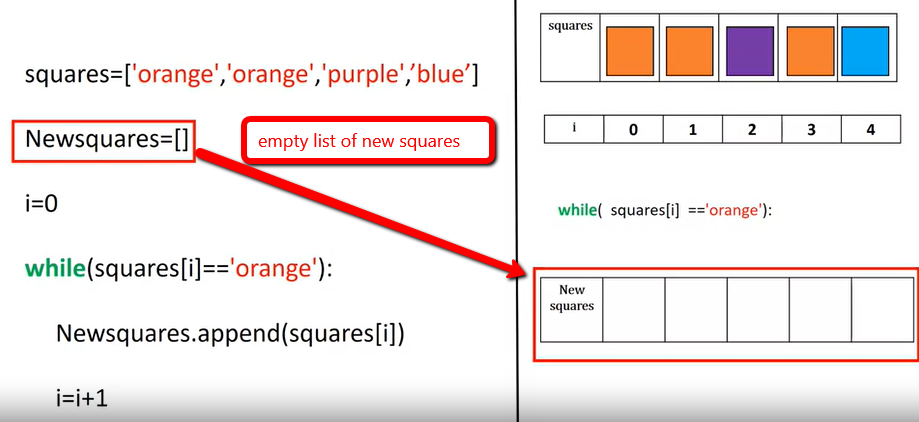
**Newsquares.append(squares[i])**

**i=i+1**

**print(Newsquares)**

We will use a list with the names of the colour to represent the different squares.

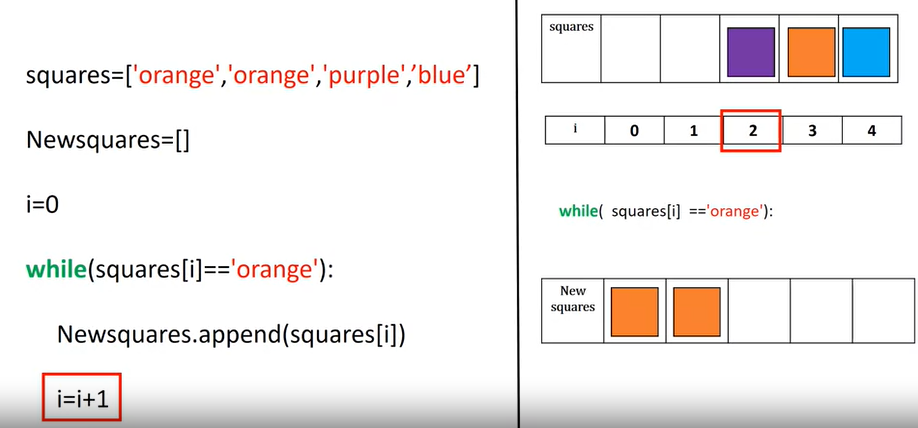
We create an empty list of new squares. In reality the list is of indeterminate size.



We start the index at zero the while statement will repeatedly execute the statements within the indent until the condition inside the bracket is false.

We append the value of the first element of the list squares to the list new squares. We increase the value of I by one. We append the value of the second element of the list squares to the list new squares.

We increment the value of I. Now the value in the array squares is purple;



therefore, the condition for the while statement is false and we exit the loop. Check out the labs for more examples of loop many with real data.