Question: How do I know which type of loop to use (e.g., a for-loop versus a while-loop)?

It all depends on the problem you are trying to solve. In most cases, you can use any of the two. Here are a few guidelines to keep in mind:

(1) For-loops work in scenarios where you know (or can compute) the number of iterations your loop will execute, **before** you enter the loop. For example, if you want to iterate over every element of a list or every character of a string or a range of numbers, then a for-loop would work great.

(2) In Python, anything you can solve with a for-loop, you can also solve with a while-loop. In some scenarios though, you might need to do some extra work in the while-loop case to make the two equivalent. Think of the simple example where we ask you to print every character of a string. Write both the for-loop and the while-loop code and compare them.

(3) If you cannot compute an upper bound on the number of your iterations, then you need to use a while-loop. While loops are the perfect candidate when what you do in the body of the while-loop determines the number of iterations. Remember the collatz conjecture example from the while-loop worksheets? You cannot solve this with Python's for-loop. You do not know the number of times you need to repeat the body of the loop; when you exit the loop solely depends on what is happening inside that loop.

Question: What is the difference between built-in functions len() and range()?

Built-in function len(X) returns the length of a string or a list X. The return value is of type integer, so we can use this value in an expression, assignment etc.

Range(stop) will return a list-like object of int, until and not including integer stop. Unlike len(), you can only use range() in a for-loop.

If you want to iterate over all elements of a list or a string X in a for-loop, you can combine the two by calling range(len(X)). The for-loop
```
for i in range(len(X)):
   #do something
```
will iterate over all indices of object X.

Question: When shall I use a for-loop over a range of numbers?

Let us assume you are processing a list X (this could be a string too). Here are your two for-loop options:

(a) for item in X:
(b) for index in range(len(X)):

Option (a) works great if you only care about the list items (what they are) and not about where they are located (their index). If you need to know the item's index, then you need to use a separate variable as a numerical accumulator, which you should increment in every loop iteration.

If you need to know an item's index, then option (b) is preferable as it allows you both to access the index (variable index) and the item (X[index]).

Question: I am still confused about loops. Can you summarize?

Here is a summary of all the types of loops we've learned in class:
(1) for val in x:

X can be a string or a list. If x were of any other type (e.g., number, bool, int), you would get an error. Remember that val is simply a variable name. You can use anything as that variable name. Just make sure it is meaningful for the problem you are solving.

If X is a non-empty string, in the first loop iteration variable *val* will refer to the first character of string X. If X is a non-empty list, then in the first loop iteration variable *val* will refer to the first element of list X.

(2) for val in range(stop):

This second loop type iterates over a range of numbers.

(3) while condition:

The last loop type is a while-loop:

```
while condition:
   #do something
```

This while loop will continue to execute as long as the condition, a boolean expression, evaluates to True. Since while loops do not execute for a predetermined number of iterations, you need to make sure that you have code within the loop body that will eventually set this condition to False. If you don't, then you'll end up with an infinite loop and an unresponsive program.