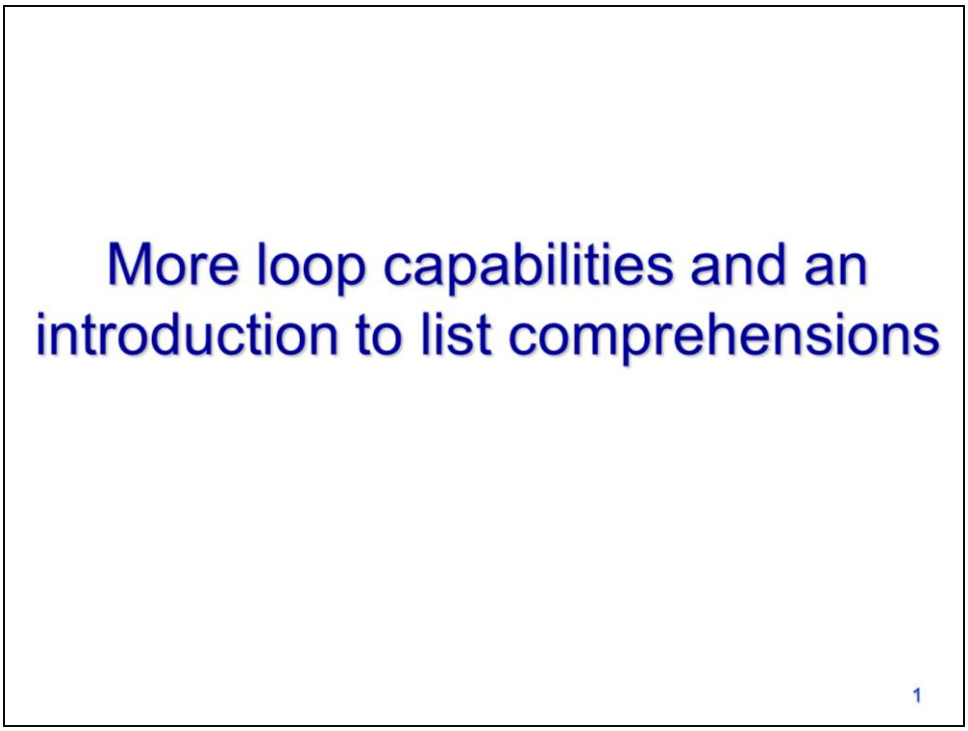
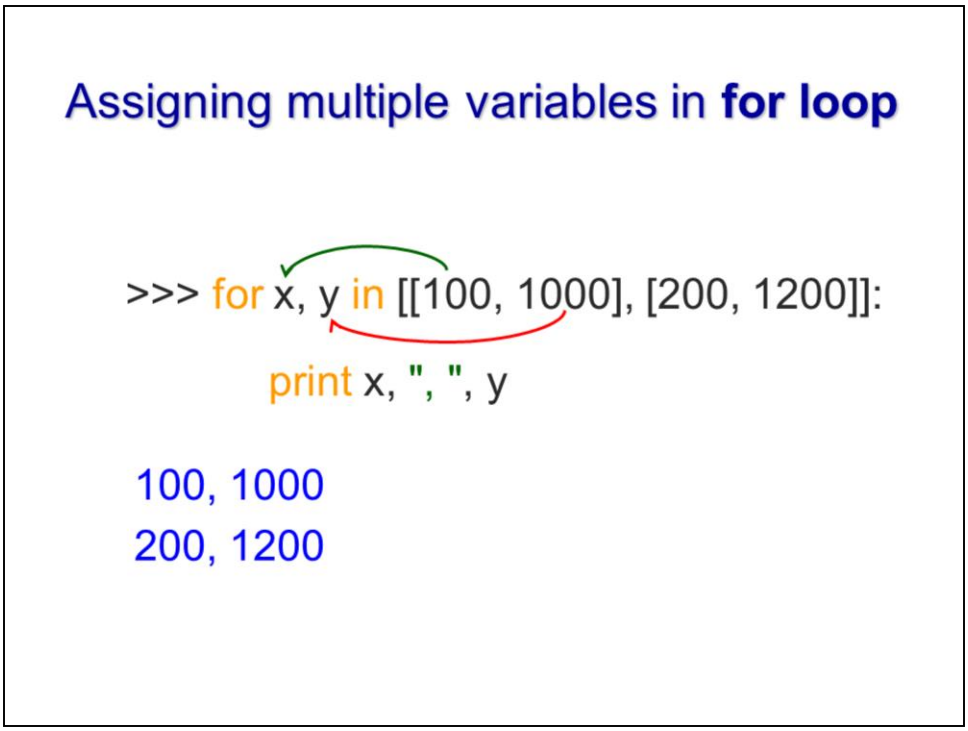
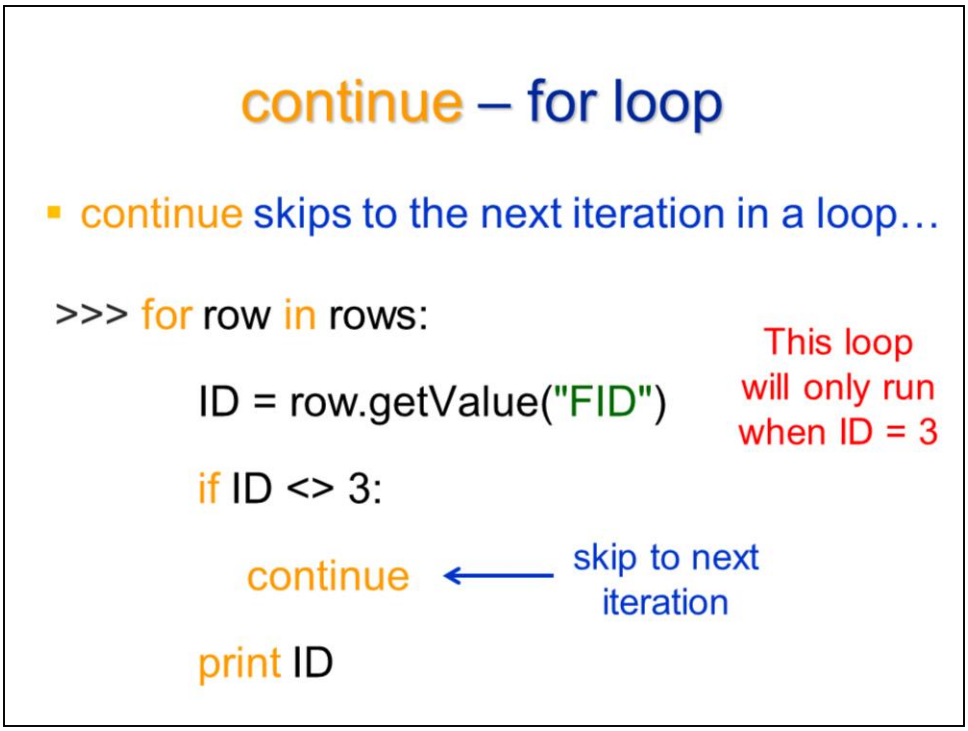


This video will expand on our capabilities with loops as well as introduce the concept of list comprehensions.



As we've seen in the past few weeks, hierarchical list structures can be quite common in scripting.

When iterating through a hierarchical list with a for loop, it can be convenient to assign variables to each item in a sublist. Assigning multiple values in a for loop header line requires that all the sublists contain the same number of items.



The **continue** function forces the loop to immediately skip to the next iteration. The script in this example will skip over any feature that does not have an ID value of 3.

continue – while loop

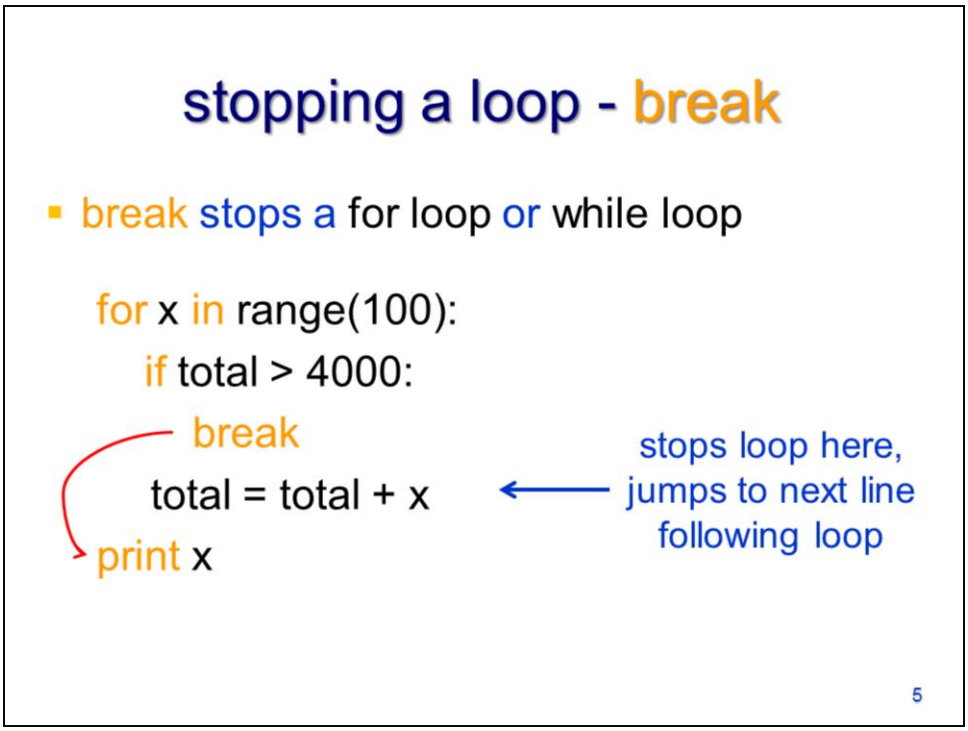
- remember to change the condition before you use continue…

```
>>> while row:
        ID = row.getValue("FID")
        if ID <> 3:
            row = rows.next()
            continue
        print ID
        row = rows.next()
```

This loop will only run when ID = 3

get next row before continue

This while loop example performs the same operation as we saw with the for loop on the previous slide. With a while loop, it is important to change the loop condition before using continue in order to make the loop progress.



The **break** command will stop a loop completely. In this example, the for loop will stop when the total is greater than 4000. When the loop breaks, the script will immediately proceed to the code that follows the loop.

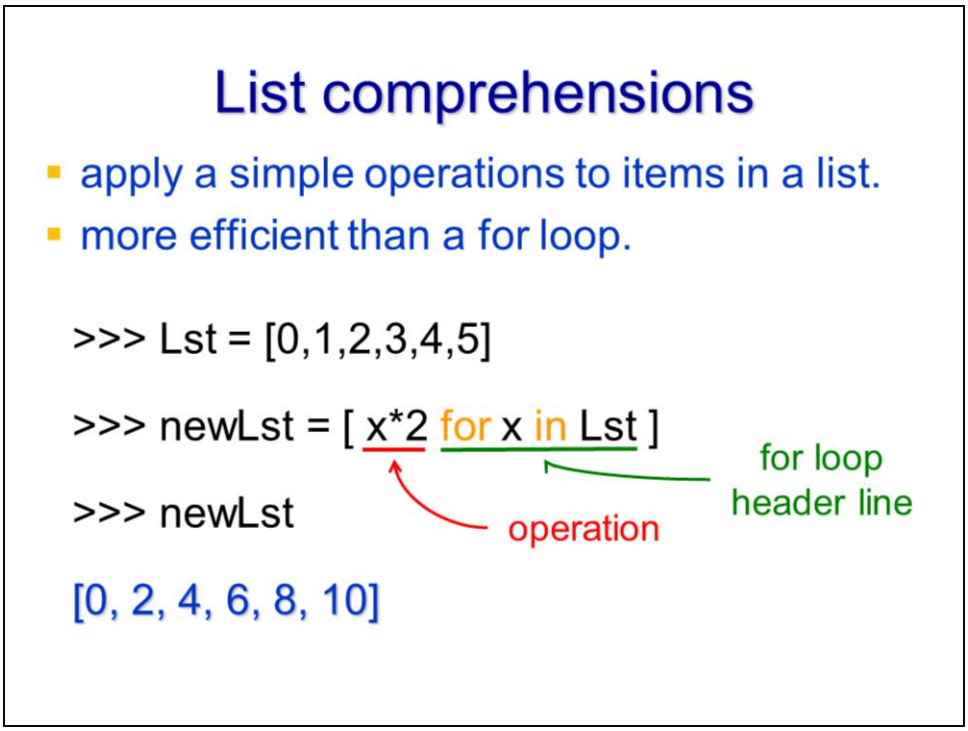
for / else loops

- else part runs if loop doesn't break

```
for value in List:
    if value == "NAN":
        print "Null data found…aborting"
        break
    total = total + value
else:
    print total
```

6

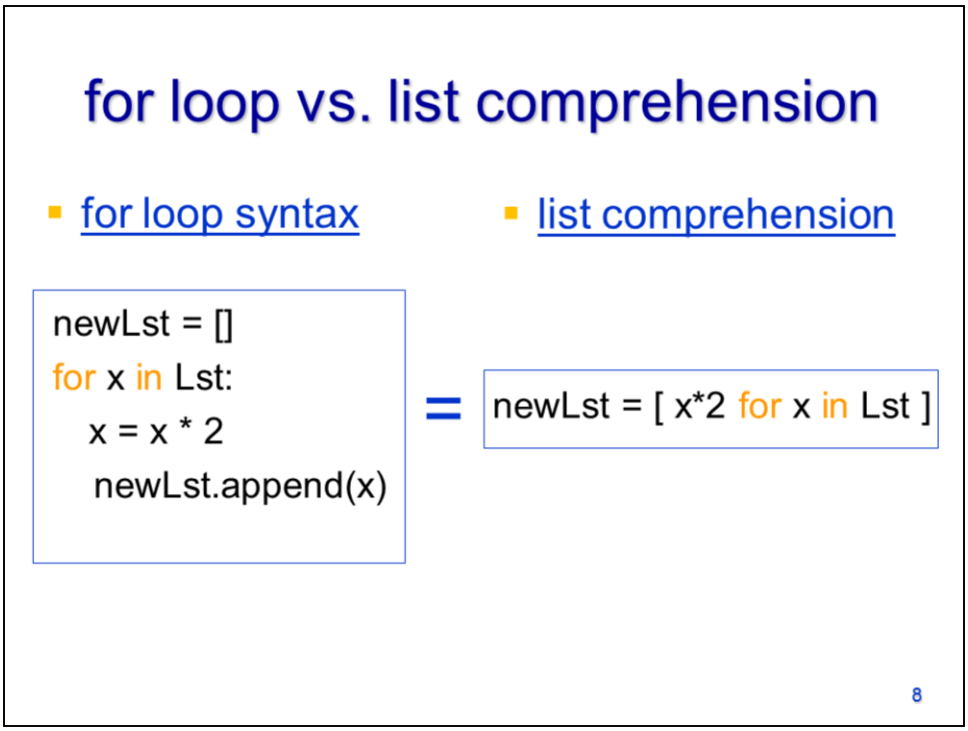
A for loop can be used with an else part. The else part will run only if the loop does not break. In this example, the total will only print if no null value is found in the List. Note that the for and else header lines have the same level of indentation.



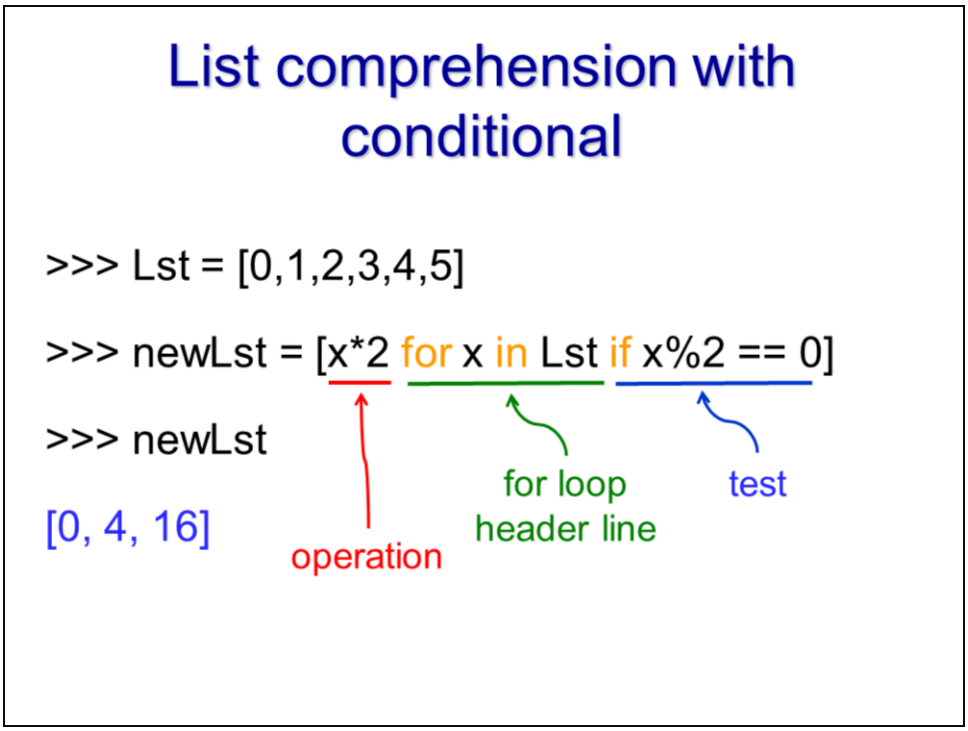
A list comprehension is a concise and efficient way to apply simple iterative processes to a list. The example here demonstrates the syntax for a list comprehension.

The list comprehension is defined within a list – i.e. enclosed in square brackets. It starts with the operation

which is followed by the for loop header line. The variable in the loop header must match the variable in the preceding operation. The result of the list comprehension is a new list which contains the same number of items as the original list.



For loops can perform the same tasks as a list comprehension – in this example, both the for loop syntax and list comprehension syntax give the exact same result. However, the for loop uses several lines of code where the list comprehension uses a single line. Concise code is generally preferred in a script because it usually improves script readability. A more important advantage of list comprehensions is that they operate faster than for loops performing an equivalent task. List comprehensions have the disadvantage of being useful only for relatively simple iterative operations.



List comprehensions can include a condition test (i.e. if test).

The general syntax in this case is the operation

Followed by the for loop header,

Followed by the if test. Note that when a conditional is included, the output list will not necessarily contain the same number of items as the original.