Lesson Plan

Warmup

```
#Write a program that switches the first and last elements of a list with 5 elements.
#For example, if myList is [1,2,3,4], after your code executes it should be [4,2,3,1].

#As an extension, can you find a way to make your code work for a list of any length?
#Hint for the extension: len(myList) is a function that gives the length of the list.
```

Lists Continued

Lists become even more useful as we can add and remove elements. Adding elements is done through the `append` function.

```
myList = [1,2,3]
myList.append(4)
print(myList) #Should print out '[1,2,3,4]'
```

The new element is always added to the end of the list.

Deletion is done through the `del` command:

```
myList = [1,2,3]
del myList[2] #What element corresponds to this index?
print(myList)
```

This will delete the third element.

When an element is deleted, all of the elements ahead of it will shift back to "close the gap". For example, we have `myList = [1,2,3,4]` and execute `del myList[1]` (deleting the second element). Before, the number 3 had index 2, but now we access it with `myList[1]`. The number 4 is shifted similarly.

There are some other useful things we can do with lists.

```
first = [1,2,3,4]
last = [5,6,7,8]
myList = first + last
print(myList)
```

Here, Python has linked together the two lists.

Finally, we can look at subsets of lists:

```
names = ["Alice", "Bob", "Carol", "Dan", "Eve", "Fred", "George"]
lessNames = names[1:5]
print(lessNames)
```

When we put `[a:b]` after the name of a list, Python interprets it as a new list that starts at the element with index `a` and includes every element after that, up to ***but not including*** index `b`.

Exercise:

```
#Given a three digit number a, turn it into a list so that the elements of the list
are its digits in reverse.
#For example, suppose a=345. You would write code that would return the list [5,4,3].
#For this problem, you'll need to use the '%' operator. Python evaluates "a % b" as
the remainder when you divide a by b. (the term for it is mod)
```

If/Else

In programming, we often want code to do something only upon a ***condition***. For example, let's say we have a variable `rainyDay`: it should increase by 1 every time there's a rainy day, but not in any other scenario. *Aside: What's the condition here?*

Conditions are either `True` or `False`, and are **booleans**. Here's our above example implemented in Python:

```
itRained = False #This is a boolean!
rainyDay=0
if itRained:
  rainyDay++
```

This is a ***conditional***. If we run this and print the value of `rainyDay`, you'll see that it remains zero. When we use the `if` structure, Python checks if our conditional is true. If so, then it executes whatever code is underneath. In this case, `itRained` is false, so Python doesn't run the code underneath.

The condition can be any boolean expression.

```
print("What's seven times eight?")
answer = int(input())
if answer == 56:
  print("That's correct.")
```

In this case, the condition is the inputted answer being equal to 56.

Very commonly, if the condition is false, we want our code to do something entirely different. This is the role of the `else` clause.

```
print("What's seven times eight?")
answer = int(input())
if answer == 56:
  print("That's correct.")
else:
  print("That's incorrect.")
```

Python will execute the code in the `else` clause if the conditional above isn't true.

Finally, there is one more clause: `elif`. This stands for "else if". Let's see it in action:

```
age = 16
if age<14:
  print("You're in middle school")
```

```
elif age<18:
  print("You're in high school")
else:
  print("You're old")
```

What happens here? First, Python checks if `age` is less than 14. If so, it executes the code in the clause, and then exits; it won't print out "You're in high school". If `age<14` isn't true, then it moves on to the `elif` clause directly below it. This clause's conditional is true, and "You're in high school" is printed out. If it weren't true, as when `age` is 18 or more, then it runs the final code in the `else` clause.

It's possible to stack many `elif` clauses, as seen with the following joke code:

```
#Determine if a is odd or even
if a == 1: print("It's odd")
elif a==2: print("It's even")
elif a==3: print("It's odd")
elif a==4: print("It's even")
#....
```

You can think of the if/else structure as a way for your code to take multiple paths.

An important nuance with `elif` clauses is that they only trigger when their condition is satisfied *and* the above conditions aren't satisfied. We can see this with the following puzzle:

```
a=10
if a<0: print("a")
elif a>8: print("b")
elif a==10: print("c")
else: print("d")
#What will be printed?
```

Exercises

```
#Have the user input two integers a and b. Then have them enter in a third number,
either 0 or 1.
#If the user enters in 0, print out the sum of a and b.
#If the user enters in 1, print out the product of a and b.
#If the user enters in anything else, print out "You've made an error" or some
equivalent.
```

```
#There are three roller coasters at an amusement park: Arctic Adventure(A), Beach
Bender(B), and Colossal Chase(C).
#For safety, roller coasters have minimum height requirements. The minimums are as
follows: 40in for A, 45in for B, 50in for C.
#Have the user input in their height as a number (in inches). Then create a list with
the coasters the user is allowed to ride on, and print it out.
```

```
#Create an integer num, and have the user input its value.
#If num is a multiple of 3, print 'fizz'.
#If num is a multiple of 5, print 'buzz'.
```

```
#If num is a multiple of 3 AND 5, your program should print 'fizzbuzz'.

#Remember that 'a%b' gives you the remainder when a is divided by b.
```