Comparison Operators

Let's take two numbers a and b.

```
a=1
b=2
```

Many times in programming we'll check if two numbers (and other kinds of data) are equal to each other. How we do this is with the "==" operator.

```
print(a==b)
```

When we run this, it outputs "False". What Python has done is check if a equals b. If their values are the same, then it is true that they are equal. Otherwise, it is false. So Python evaluates and stores "a==b" as a boolean.

Note: a common mistake is to confuse '==' with '='. '=' is used for assigning values to variables, '==' is used for comparison.

We can do other comparisons as well.

```
greaterthan = (a>b)
lessthan = (a<b)
geq = (a>=b) #checks if a is greater than or equal to b.
leq =(a<=b) #checks if a is less than or equal to b.
```

In addition, we can do operations on booleans themselves.

```
raining = True
sunday = True
rainySunday = raining and sunday
```

What the 'and' operator does is check if both of the booleans are true. Only then will rainySunday also be true.

```
pizza = True
sushi = False
goodLunch = pizza or sushi
```

What the 'or' operator does is check if either of the booleans are true. If so, goodLunch will be true. Only if both of the booleans are false will it return false.

Exercise:

```
#Largest Number
a=
b=
c=
isCtheLargest = #
print(isCtheLargest)
#Write an expression using comparisons that will output true if c is the largest out of a, b, c and false otherwise.
```

Lists

Variables give us a way to store information, but for every bit of data, we have to create a new variable. This is annoying, especially if we want to hold a lot of data related to something.

This is where we introduce the **list**. We declare a list like so:

```
myList = [14,56,23,52,41]
```

The brackets indicate the list, and we separate the data inside the list with commas.

We can put variables inside lists:

```
bigNumber = 1000
smallNumber = 10
myList = [bigNumber, smallNumber]
```

In fact, we can put different kinds of data inside a list:

```
age = 16
name = "Richard"
isPresent = True

myList = [age, name, isPresent]
```

Lists wouldn't be very useful if there wasn't some way to access the data inside. Every element inside a list has an *index*. In most programming languages, we start counting from zero, so the first element in a list has index 0, the second element index 1, and so on.

```
myList = [14,56,23,52,41]
print(myList[0]) #Should output 14
print(myList[1]) #Should output 56
print(myList[2]) #Should output 23
```

We need to be careful about accessing elements. Notice our list has five numbers in it.

```
myList = [14,56,23,52,41]
print(myList[5]) #What element will this access?
```

When we run this, we get an error. Python tries to retrieve the sixth element, but it doesn't exist! Thus, we should make sure to access indices within the bounds.

Very importantly, lists are *changeable*. We can change any element in a list given its index.

```
myList = [14,56,23,52,41]
print(myList) #Prints out "[14,56,23,52,41]"
myList[0] = 100 #Sets the first element of the list to 100
print(myList) #Prints out "[100,56,23,52,41]"
```

Note that if we try to change an element with an index out of bounds, it'll obviously give us an error.

A subtle detail about lists:

```
age = 16
myList = [age, "Richard"]
age = 17
print(myList) #What will it print?
```

As it turns out, it prints "[17, 'Richard']". The list retains the previous value of age.

Exercises:

```
#Lists can hold all kinds of data, even other lists!
list0 = [0,1,2,3]
list1 = [4,5,6,7]
list2 = [8,9,10,11]
list3 = [12,13,14,15]
megaList = [list0,list1,list2,list3]
#When printed out, megaList looks like this:
#[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]]
#You'll be given a number, a, that will range from 1 to 16.
#Write code that will output the ath number in the 'list'.
#For example, if a=5, the fifth number is 4, so your code should output 4.
```

```
#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 [4,3,2,1].

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

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`.

Exercises:

```
#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.
```

```
```