IMMUTABILITY

- What does it mean?

- Mutable : you can change the object, and that variable retains or keeps that change

- lists, dictionaries, sets

- Immutable : Whatever you create the first time, it stays. (You will have to redefine what that variable holds)

- strings, numbers, tuples

```
my_string = “Hello World“

 # takes string and creates a NEW string that’s the old string reversed

 # notice how I’m not assigning a variable to it ?

my_string[:-1]


 # still prints original “Hello World”

print (my_string)
```

```
my_list = [1,2,3,4,"5", 6]

# takes list and removes last element
my_list.pop()

# prints the same list now, with one less element
print (my_list) # prints [1,2,3,4,”5”]
```

FUNCTIONS

- Function have their own namespace separate from the global namespace

```
def function( object1, object2):    # take objects from the another namespace (usually global) into this one!
    # create variables in local namespace
    sum_amount = object1 + object 2
    return  sum_amount        # means I want to give this local variable into the global namespace


A = 2
B = 1
C = sum_amount (A , B )    # C is taking whatever function is giving (what return means)
```

LIST COMPREHENSIONS

They are just a great way of condensing the process of adding elements to a list

Lets say, I want to get all of the characters of a string into a list I could do …

```
S = "Cyndy Ishida"

L = [] # start with empty list

for c in S:
    L.append(c)
```

```
S = "Cyndy Ishida"

L = [c for c in S]
```

Both do the exact same thing

IS VS ==

- The reason why this is so confusing, is because it has to do with memory
- Every object created in python, has a memory address, don't worry about how, when, where, and why,
- Is : returns a Boolean that checks if two objects are pointing to the same place in memory
- == : checks to see if the information are the same

SHALLOW VS. DEEP COPY

- Very confusing topic for most people.

- Shallow Copy: creates a new object that points to the same place in memory

- When I change one of the objects, they both are changed

- Deep Copy: copies the same value in a new place in memory.

- When I change one of the objects, both are not changed

- When you copy lists, you are always doing a shallow copy except in 2 cases.

L2 = copy.deepcopy(L1) # L1[:] isn't a perfect deep copy!

Think of "is" is saying True for Shallow Copy, False for Deep Copy