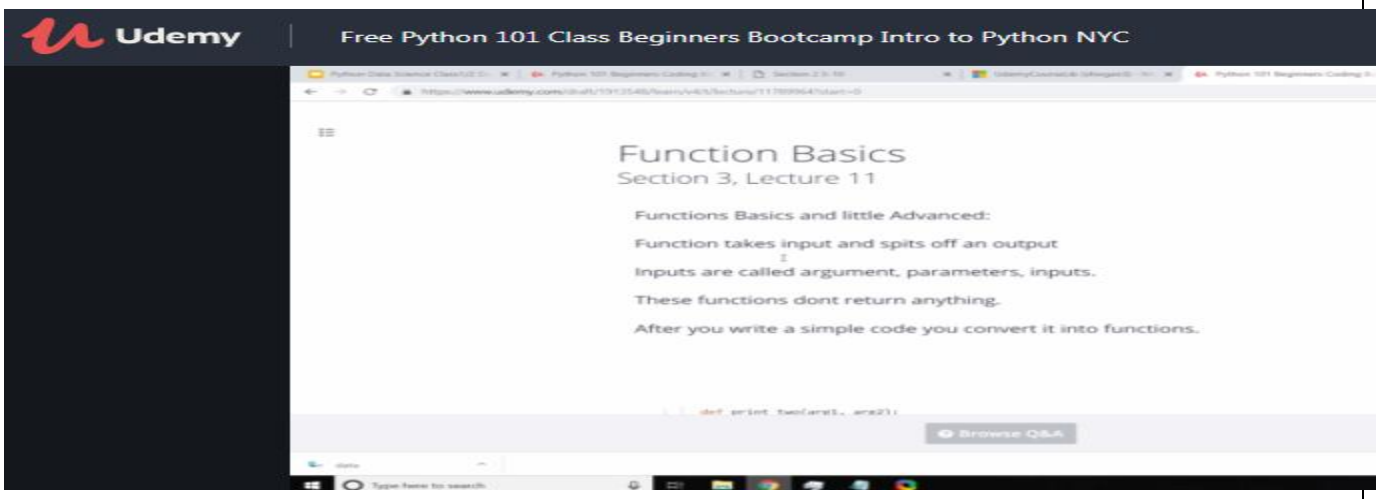
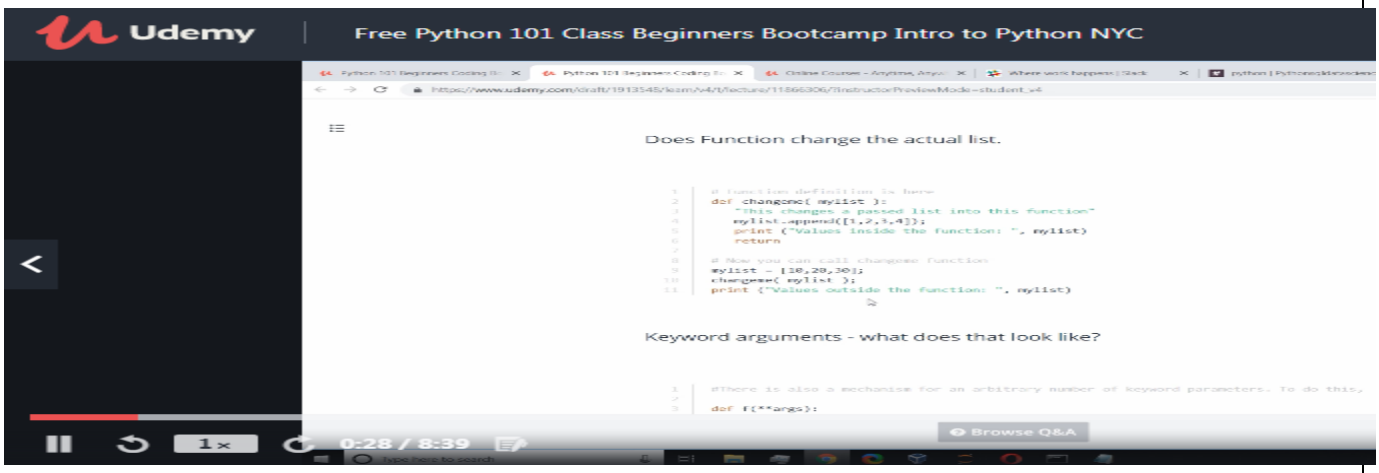
DAILY ASSESSMENT FORMAT

Date:	**21 may 2020**	**Name:**	**Mahima Shetty**
Course:	**Python**	**USN:**	**4Al15EC045**
Topic:	**Function**	**Semester & Section:**	**8th A**
Github Repository:	**Mahima**		

FORENOON SESSION DETAILS

Image of session





Function Basics & Advanced (Recursion, optional arg, key word arg)

Functions Basics and little Advanced:

Function takes input and spits off an output

Inputs are called argument, parameters, inputs, passing values/reference.

These functions don’t return anything.

After you write a simple code you convert it into functions.

```
def print two(arg1, arg2):

    print(f"arg1: {arg1}, arg2: {arg2}")

# this just takes one argument
def print one(arg1):
    print(f"arg1: {arg1}")

# this one takes no arguments
def print none():
    print ("I got nothing' to return.")
.
.
. print two("S", "J")
. print one("S!")
. print none()
```

Below are the functions that would return the calculation.

Create errors and give only one or three arguments.

```
def add(a, b):
   print(a,b)
   return a + b

def subtract(a, b):
   print(a,"minus",b)
   return a - b

def multiply(a, b):
.   print("multiple",a,b)
.   return a * b
.
. def divide(a, b):
```

```
.    print("DIVIDING",a,b)
.    return a / b
.
. print("Let's see some functions!")
. age = add(30, 5)
. height = subtract(78, 4)
. weight = multiply(90, 2)
. iq = divide(100, 2)
```

Try to see if you can find out the order of execution for the above...

```
def xxx(*args):
    for xy in args:
        print(xy)
xxx('aaa','bbb','ccc','eee')
```

This taken in multiple arguments.

Search yourself key word arguments and how they use keys in the function input. These key word arguments are used a lot in Django!

```
x = "I like the way python works"
y = x.split()

def print_first_word(words):
    """Prints the first word after popping it off."""
    word = words. Pop(0)
    print(word)

print_first_word(y)
```

First recursive function:

What is recursion? There is theory that when we find out about this matrix we fund recursion in fabric of code for this universe.

Don’t worry if you don’t get recursion, it is rarely used in day to day programming unless you are too worried about speed and efficiency when writing big code for big data!

```
def factorial( n ):
```

```
  if n <1:   # base case
     return 1
  else:
     returnNumber = n * factorial( n - 1 )  # recursive call
     print(str(n) + '! = ' + str(returnNumber))
     return returnNumber
```

```
def foo(*args):
    for a in args:
        print (a)
foo(1,2,3,5,6)
```

Lambda function- create functions on the fly in python in minimal code. Only JS allows you to the same. And now scala.

```
def short function(x):
   return x * 2

f = lambda x: x * 2

f(6)

short function(6)
```

Passing function as an argument

```
def apply_to_list(some list, short function):
   return [short function(x) for x in some list]

apply_to_list([1,2,3],short function )


ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)

strings = ['foo', 'card', 'bar', 'aaaa', 'abab']

strings. Sort(key=lambda x: len(set(list(x))))

strings
```

Try to convert, if you cannot convert return back what you have.

```
def attempt_float(x):

    try:
        return float(x)
    except:
        return x
```

Difference between printing and returning.

```
def joshif(x, y, z):

    print("function ran - message from inside of function")


    print ("end of the function")
    return (x + y) * z
```

Optional argument with if else:

```
def add_and_maybe_multiply(a, b, c=None):

    result = a + b


    if c is not None:
        result = result * c

    return result
```

We have looked at functions that are:

taking input vs not taking input

spitting output vs not spitting output

fixed number of arguments vs variable number of arguments

optional arguments And we see that most of times when we do something we convert it into functions so that we can use it later when needed.

Another way to look at Functions - Input and Output

One input and three output!

Capturing three output and unpacking the output.

```
def secret formula(started):
    samosa = started * 500
    dosa = samosa / 1000
    idli = dosa / 100
    return samosa, dosa, idli

start point = 10000
s,d,i = secret formula(start point)
tupleoutput =  secret formula(start_point)
. # remember that this is another way to format string
. print(f"We'd have {s} , {d} , and {i} .")

 print("We'd have {s} , {d} , and {i}".format(*tupleoutput ))
```

If we don't want to capture 3 outputs in separate variables

```
start_point = 10000
print ("We'd have %d beans, %d jars, and %d crabapples." % secret formula(start_point))
```

This was another function to understand function.