Advanced Functions in Python

Session-2

Agenda

Lecture -1: Review Python and Jupyter notebook

- ❑ Understand Python functions
- ❑ Use built-in functions to perform common tasks in Python
- ❑ Create custom functions to perform custom analyses
- ❑ Construct complex functions with multiple parameters

Use built-in functions to perform common tasks in Python

Built-in Functions

```
#lower(), upper()
word = 'Hello'

print (word, word.lower(), word.upper())
```

Hello hello HELLO

```
#Concatenation

print ('1' +'2')

print ('Hello' + ' there')
```

'12'
'Hello there'

Built-in Functions (Contd.)

```
#Concatenation with join() method
#The method join() returns a string in which the string elements of sequence are joined by str separator.
str_separator=' '
list =['I', 'am', 'learning']
str_separator.join(list)
#or ' '.join(list)
```

'I am learning'

```
#split

s = 'Let\'s split the words'

s.split(' ')
```

["Let's", 'split', 'the', 'words']

Quiz

How can we add these two strings together to print 'Good Morning Dhaka!' ?

s1 = “Good Morning ”

s2 = “Dhaka!”

a. s1.join(s2)

b. s1.add(s2)

c. s1 = s1 + s2

d. s1 ++ s2

Answer: s1 = s1 + s2

Explanation - join() method takes all items in an iterable and joins them into one string. S2 is not a valid iterable, it is just a word.

Other options are invalid.

Understand Python functions

Function

Functions are block of codes that are written once and can be called any time when needed. Python functions are defined using the word def keyword as shown below:

```
def function_name(parameters):
    """docstring"""
    statement(s)
    return statement
```

```
def with_args(arg1=0,arg2=['a', 5]):
    """ A function with arguments """
    num = arg1 + 3
    mylist = arg2 + ['b',10]
    return [num, mylist]
with_args()
```

[3, ['a', 5, 'b', 10]]

Arguments passed as shown in the left panel below.

What would this print when you call the function as below?

with_args(5)

Function (Contd.)

```
def with_args(arg1=0,arg2=['a', 5]):
    """ A function with arguments """
    num = arg1 + 3
    mylist = arg2 + ['b',10]
    return [num, mylist]
with_args(5)
```

[8, ['a', 5, 'b', 10]]

What about this?

```
with_args(10, ['x',6])
```

[13, ['x', 6, 'b', 10]]

Function (Contd.)

```
def with_args(arg1=0,arg2=['a', 5]):
    """ A function with arguments """
    num = arg1 + 3
    mylist = arg2 + ['b',10]
    return [num, mylist]
```

Docstring in Function

```
# A Docstring is the first statement in the body of a function,
# which can be accessed with function_name.__doc__
with_args.__doc__
```

```
' A function with arguments '
```

Quiz

What is the output of the add() function call?

```
def add(a, b):
    return a+3, b+5
add(5, 6)
```

a. 19
b. 811
c. Syntax Error
d. (8, 11)

Answer: d

Lambda

A lambda function is a small anonymous function.

- It can take any number of arguments, but allow only one expression.

- Syntax
  - lambda [arguments] : expression

```
x = lambda a, b : a + b
print(x(5, 7))
```

Lambda (Contd.)

You can use the lambda functions as an anonymous function inside another function.

```
def dup_or_double(n):
    return lambda x : x * n

result = dup_or_double (2)

double = result (50)
print (double)
```

100

*args and **kwargs

- Using a * and ** you can pass a variable number of arguments to a function
- By convention, * is often used with the word args as *args but you can use any name with * to pass a variable number of arguments

```
# Use of *args for variable number of arguments
def myFun(*args):
    for arg in args:
        print (arg)

myFun( 'Welcome ', 'to ' ,'Washington DC')
```

```
Welcome
to
Washington DC
```

```
# Use of *args for variable number of arguments
def myFun(*some_text):
    for arg in some_text:
        print (arg)

myFun( 'Welcome ', 'to ' ,'Washington DC')
```

```
Welcome
to
Washington DC
```

```
# Use of **kargs for variable number
# of keyword arguments with one extra argument.

def myFun(arg1, **kwargs):
    for key, value in kwargs.items():
        print ("%s = %s" %(key, value))

# Driver code
myFun("Hi", Name ='Tom', Age = 40, Gender='Male')
```

```
Name = Tom
Age = 40
Gender = Male
```

Quiz

What is the output of the following salary() function call?

```
def salary(**kwargs):
    for i in kwargs:
        print(i)
salary(name="Jeni", salary=90000)
```

a. TypeError
b. Jeni
   90000
c. name
   salary
d. ('name', 'Jeni')
   ('salary', 90000)

Answer: c

Lab - 1

Session2a-Functions.ipynb

Generators & Decorators

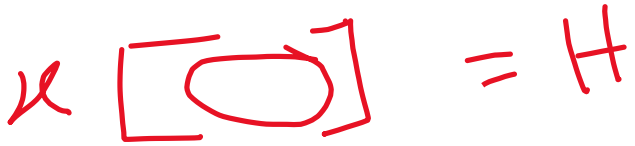
Iterator

- In Python, an iterator is an object which implements the iterator protocol. The iterator protocol consists of two methods:
  - The **__iter__()** method, which must return the iterator object, and
  - The **next()** method, which returns the next element from a sequence.

```
x = "Hi"
a_iterator = iter(x)

print(next(a_iterator))
print(next(a_iterator))
```

```
H
i
```



Generators

Generator functions are a special kind of function that return a lazy iterator.

- These are objects that you can loop over:
  - Like a list but, unlike lists, lazy iterators do not store their contents in memory, Or
  - Like a function but unlike functions, which return a whole array, a generator yields one value at a time which requires less memory.
- Any python function with a keyword “yield” may work as generator

Many Python functions like pandas read_csv can return a generator object that you can loop through

Generators (Contd.)

Here is an example.

```
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row

csv_gen = csv_reader("../data/modpop.csv")
row_count = 0

for row in csv_gen:
    row_count += 1

print(f"Row count is {row_count}")
```

Decorators

Decorators are used to dynamically add new behaviors to some python objects.

- A function that takes another function as its argument, and returns yet another function
- Useful as they allow the extension of an existing function, without modifying the original function code

Here is an example.

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print("Before call")
        result = func(*args, **kwargs)
        print("After call")
        return result
    return wrapper
```

```
@my_decorator
def add(a, b):
    return a + b

add(4, 2)
```

Before call

After call

6

Quiz - 1

Every generator is an iterator in Python, not every python iterator is a generator

a. True

b. False

Answer: True

Quiz - 2

To write a python generator, you can either use a Python function or a comprehension. But for an iterator, you must use the iter() and next() functions.

a. True

b. False

Answer: True

Quiz - 3

What will be the output of the following Python code?

```
def deco_f(x):
    def f1(a, b):
        print("Hello")
        if b==0:
            print("Zero Divisor")
            return
        result = f(a, b)
        return result
    return f1
@deco_f
def f(a, b):
    return a%b
f(4,0)
```

1. Hello
   Zero Division Error

2. Hello
   Zero Divisor

3. Zero Division Error

4. Hello

Answer: 2

Map, Filter, Reduce

Map()

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

Syntax: map(function, iterable)

```
list1=['cat', 'dog', 'cow']
```

map() takes a function as an argument

```
animals=map(lambda x: x.upper(),list1 )
animals
```

```
<map at 0x7fb0f5e605f8>
```

```
next(animals)
```

```
'CAT'
```

```
next(animals)
```

```
'DOG'
```

Using a loop or iterables you can access all of them one by one

```
list(map(lambda x: x.upper(),list1 ))
```

```
['CAT', 'DOG', 'COW']
```

Filter()

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

Syntax: filter(function, sequence)

```
list(filter(lambda x: 'a' in x, ['cat', 'dog', 'cow','man']))
```

```
['cat', 'man']
```

Reduce()

The **reduce(fun,seq)** function is used to **apply a particular function passed in its argument to all of the list elements** mentioned in the sequence passed along.

This function is defined in "**functools**" module.

```
from functools import reduce
reduce(lambda p, x: f'{p}+{x}', ['cat', 'dog', 'cow'])
```

```
'cat+dog+cow'
```

Thank You