Objective – Module 4

Ethans
Learn from experts

UDF Functions and Object Functions

- What are various type of functions
- Create UDF functions
- Parameterize UDF function, through named and unnamed parameters
- Defining and calling Function
- The anonymous Functions - Lambda Functions
- String Object functions
- List and Tuple Object functions
- Dictionary Object functions

Slide 81

What are Functions?

Ethans
Learn from experts

- A function is a set of instructions organized, which is used to perform a single, related actions.

- Functions provide better modularity to a program and a high degree of freedom to reuse existing codes.

We can have different function in Python, most common functions are:

- Builtin functions
- Object functions
- User defined functions
- Module function

To take the help on function

>>> help('NAME OF THE FUNCTION')
Help on built-in function

Slide 82

www.ethans.co.in

UDF Functions

Ethans
Learn from experts

User defined functions are common Python function which we create:.

Here how we create UDF:

- Function blocks begin with the keyword **def** followed by the function name and parentheses.
- Input parameters or arguments should be placed within the parentheses. Parameters can also be defined inside these parentheses.
- The first non executable statement of a function can be an optional statement – the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- A return statement with no arguments returns None.

UDF Functions Example

Ethans
Learn from experts

```
def NoParam():
    '''
    DocString of the function
    '''
    print 'First Executable Statement'
    return

def OneParam(a):
    print 'First Executable Statement'
    print a
    return a

# Call function
NoParam()
value = OneParam(10)
print value
```

Pass by Reference or value

Ethans
Learn from experts

- All the parameters in Python are passed by reference as all datatypes are Objects
- If you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
def OneParam(a):
    a.append(4)
    return a

# Call function
a = OneParam([1,2,3])
print a
```


How to pass parameters

Ethans
Learn from experts

You can call a function by using the following types of arguments:

- Mandatory arguments
- Optional or Default arguments
- Any number of unnamed arguments
- Any number of named arguments

Mandatory Arguments

Ethans
Learn from experts

```
# Accept only one arguments
def OneParam(a):
    print a

# Call function
OneParam([1,2,3])

# If we pass more than one arguments it will raise an issue
OneParam([1,2,3], 10)

# Accept only two arguments
def twoParam(a, b):
    print a, b

# Calling with two arguments
twoParam ([1,2,3], 10)
```


Mandatory Arguments

Ethans
Learn from experts

```
# Accept only one arguments
def OneParam(a):
    print a

# Call function
OneParam(a = [1,2,3])  # Fine

# If we pass more than one arguments it will raise an issue
OneParam(a = [1,2,3], 10) # Error

# Accept only two arguments
def twoParam(a, b):
    print a, b

# Calling with two arguments
twoParam (b = [1,2,3], a = 10)  # Fine
twoParam (b = [1,2,3], 10)      # Error
```

Default Arguments

Ethans
Learn From experts

```
# Accept one default arguments
def OneParam(a=10):
    print a

# Call function
OneParam(a = [1,2,3])  # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Error

# Accept only two arguments
def twoParam(a, b=20):
    print a, b

# Calling with two arguments
twoParam (b = [1,2,3], a = 10)  # Fine
twoParam (10)                   # Fine
```


N unnamed Arguments

Ethans
Learn From experts

```
# Accept n number of default arguments
def OneParam(*a):
    print a

# Call function
OneParam([1,2,3])  # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Fine

# Accept one mandatory and any arguments
def twoParam(a, *b):
    print a, b

# Calling with two arguments
twoParam (a = 10, 20, 30)  #Error
twoParam (10, 20, 30m 40) # Fine
```

N named Arguments

Ethans
Learn From experts

```
# Accept n number of default arguments
def OneParam(**a):
    print a

# Call function
OneParam(c=[1,2,3])  # Fine

# If we pass more than one arguments it will raise an issue
OneParam() # Fine
OneParam(10, 20) # Error
OneParam(b=10, c=20) # Fine

# Accept one mandatory and any arguments
def twoParam(a, **b):
    print a, b

# Calling with two arguments
twoParam (a = 10, b=20, c=30)  #Fine
twoParam (10, 20, 30, 40) # Error
```


Lambda Functions

Ethans
Learn From experts

- lambda keyword to create one liner anonymous functions.
- These functions are called anonymous because they are not declared in the standard manner by using the def keyword.
- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- Anonymous functions cannot contain commands or multiple expressions.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Lambda Functions

Ethans
Learn from experts

```
# Accept n number of default arguments
def OneParam(a):
    print a + 10

# Function using Lambda
oneParam = lambda a : a + 10
print oneParam(10)
```


String functions

Ethans
Learn from experts

```
>>> string = 'ethans'

>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize',
'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format',
'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

String functions help

Ethans
Learn from experts

```
>>> help(string.capitalize)
Help on built-in function capitalize:

capitalize(...)
   S.capitalize() -> string

   Return a copy of the string S with only its first character
   capitalized.

>>> help(string.upper)
Help on built-in function upper:

upper(...)
   S.upper() -> string

   Return a copy of the string S converted to uppercase.
```


String functions executions

Ethans
Learn from experts

```
>>> string.capitalize()
'Ethans'

>>> string.center(10, 'X')
'XXethansXX'

>>> string.count('e')
1
>>> string.encode('base64')
'ZXRoYW5z\n'

>>> 'ZXRoYW5z\n'.decode('base64')
'ethans'

>>> string.endswith('ns')
True

>>> string.find('a')
1
```

String functions executions

Ethans
Learn from experts

```
>> 'I am {0} and I am {1} years old'.format('Ethans', 2)
'I am Ethans and I am 2 years old'
>>> string.index('a')
3
>>> string.isalnum()
True

>>> string.isalpha()
True
>>> string.islower()
True
>>> string.isspace()
False
>>> string = 'This is the sample string'
>>> allWords = string.split(' ')
>>> allWords
['This', 'is', 'the', 'sample', 'string']
>>> '|'.join(allWords)
'This|is|the|sample|string'
```


List functions

Ethans
Learn from experts

```
>>> allWords
['This', 'is', 'the', 'sample', 'string']

>>> dir(allWords)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__delslice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

List functions help

Ethans
Learn from experts

```
>>> help(allWords.append)
Help on built-in function append:

append(...)
   L.append(object) -- append object to end

>>> help(allWords.extend)
Help on built-in function extend:

extend(...)
   L.extend(iterable) -- extend list by appending elements from the iterable
```


List functions Executions

Ethans
Learn from experts

```
>>> allWords.append('New Word')
>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word']

>>> allWords.count('is')
1
>>> allWords.extend(['New List1', 'New List2'])

>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2']

>>> allWords.index('is')
1
```

List functions Executions

Ethans
Learn from experts

```
>>> allWords.insert(10, 'New List3')
>>> allWords
['This', 'is', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2', 'New
List3']

>>> allWords.pop()
'New List3'

>>> allWords.remove('is')

>>> allWords
['This', 'the', 'sample', 'string', 'New Word', 'New List1', 'New List2']

>>> allWords.reverse()

>>> allWords
['New List2', 'New List1', 'New Word', 'string', 'sample', 'the', 'This']
```


Sorting

Ethans
Learn from experts

```
>>> help(allWords.sort)
Help on built-in function sort:

sort(...)
    L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
    cmp(x, y) -> -1, 0, 1
```

```
>>> numbers = [1,2,11,3,44,5,22,6,8,9]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 5, 6, 8, 9, 11, 22, 44]
```

```
>>> allWords.sort()
>>> allWords
['New List1', 'New List2', 'New Word', 'This', 'sample', 'string', 'the']
# ASCII based sorting
```

Sorting

Ethans
Learn from experts

```
>>> names = ['Amit', 'aakash', 'Ajay', 'Ashish']
>>> names.sort()
>>> names
['Ajay', 'Amit', 'Ashish', 'aakash']
>>> names.sort(key=str.upper)
>>> names
['aakash', 'Ajay', 'Amit', 'Ashish']
```

```
>>> names_age = [['Jatin', 35], ['Rahul', 20], ['Vijay', 47]]
>>> names_age.sort(key=lambda x:x[1], reverse=True)
>>> names_age
[['Vijay', 47], ['Jatin', 35], ['Rahul', 20]]

>>> def age_sort(list1):
        return list1[1]
>>> names_age.sort(key=age_sort, reverse=True)
>>> names_age
[['Vijay', 47], ['Jatin', 35], ['Rahul', 20]]
```


Tuple Functions

Ethans
Learn from experts

```
>>> names = ('Rahul', 'Vijay',' Aakash')
>>> type(names)
<type 'tuple'>
>>> dir(names)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
>>>
```

Tuple Functions

Ethan's Learn from experts

```
>>> help(names.count)
Help on built-in function count:

count(...)
   T.count(value) -> integer -- return number of occurrences of value


>>> names.count('Rahul')
1
>>> names.index('Rahul')
0
```


Dictionary functions

Ethan's Learn from experts

```
>>> dict1 = {1:2, 3:4}
>>> dir(dict1)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delite
m__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribut
e__', '__getitem__', '__gt__', '__hash__', '__init__', '__iter__', '_
_le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '_
_reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeo
f__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'g
et', 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'p
op', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewk
eys', 'viewvalues']
```

Dictionary functions example

Ethans Learn From experts

```
>>> help(dict1.clear)
Help on built-in function clear:

clear(...)
   D.clear() -> None.  Remove all items from D.
```

Example:

```
>>> dict1.clear()
>>> dict1
{}
>>> dict1 = {1:2, 3:4, 5:6}
>>> dict2 = dict1.copy()
>>> dict2
{1: 2, 3: 4, 5: 6}
>>> dict2[1] = 10
>>> print dict1[1]
2
```


Dictionary functions example

Ethans Learn From experts

```
>>> dict1 = {1: 20, 3: 4, 5: 6}
>>> dict1.fromkeys(dict1, 10)
{1: 10, 3: 10, 5: 10}
>>> dict1.get(1)
20
>>> dict1.get(2, 10)
10
>>> dict1.get(1, 10)
20
>>> dict1.get(4)
>>> dict1.has_key(1)
True
>>> dict1.has_key(10)
False
```

Open function

Ethans
Learn from experts

Open is the built-in function in Python, which is used to interact with the flat files in all the operating systems.
Help on the function is bellow:

```
>>> help(open)
Help on built-in function open in module __builtin__:

open(...)
    open(name[, mode[, buffering]]) -> file object

    Open a file using the file() type, returns a file object.  This is the
    preferred way to open a file.  See file.__doc__ for further information.
```


Open mode

Ethans
Learn from experts

Mode	What it do
r	Open the file in read mode
w	Open the file in write mode
a	Open the file in append mode
r+	Open the file in read and write mode
w+	Open the file in write and read mode
a+	Open the file in append and read mode
rb	Open the file in read binary mode
wb	Open the file in write binary mode
ab	Open the file in append binary mode

Open file

Ethans Learn From experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt')

>>> dir(infile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
'__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate',
'write', 'writelines', 'xreadlines']
```

- Default mode is read
- Default buffering is enable
- r'' – is used to make file path raw
- Dir(object) – display all file object functions

Open file

Ethans Learn From experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt')

>>> dir(infile)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__',
'__getattribute__', '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines',
'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate',
'write', 'writelines', 'xreadlines']
```

- Default mode is read
- Default buffering is enable
- r'' – is used to make file path raw
- Dir(object) – display all file object functions

www.ethans.co.in

Dictionary functions example

Ethans
Learn from experts

```
>>> help(dict1.items)
Help on built-in function items:

items(...)
   D.items() -> list of D's (key, value) pairs, as 2-tuples


>>> employee = {'Jatin':3500, 'Rahul':1500, 'Aakash':1000}
>>> employee1 = employee.items()
>>> print employee1
>>> [('Jatin', 3500), ('Rahul', 1500), ('Aakash', 1000)]


>>> for i, j in employee.iteritems():
            print i, j
Aakash 1000
Jatin 3500
Rahul 1500
```


Dictionary functions example

Ethans
Learn from experts

```
>>> for i in employee.itervalues():
            print i
1000
3500
1500
>>> for i in employee.iterkeys():
            print i
Aakash
Jatin
Rahul

>>> employee.keys()
['Aakash', 'Jatin', 'Rahul']
>>> employee.values()
[1000, 3500, 1500]
>>> employee.pop('Aakash')
1000
>>> employee.popitem()
('Jatin', 3500)
```

Did we know?

Ethans
Learn from experts

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. It is built over the Simple DirectMedia Layer (SDL) library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games (mainly the graphics part) can be abstracted from the game logic, making it possible to use a high-level programming language, such as Python, to structure the game.

Pygame was built to replace PySDL after its development stalled.[4] Pygame was originally written by Pete Shinners and is released under the open source free software GNU Lesser General Public License. It has been a community project since 2004 or 2005.

Objective – Module 5

Ethans
Learn from experts

File Handling with Python

- Process text files using Python
- Read/write and Append file object
- File object functions
- File pointer and seek the pointer
- Truncate the file content and append data
- File test operations using os.path

Open file and read data

Ethans
Learn from experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
>>> help(infile.read)
Help on built-in function read:

read(...)
   read([size]) -> read at most size bytes, returned as a string.

   If the size argument is negative or omitted, read until EOF is reached.
   Notice that when in non-blocking mode, less data than what was requested
   may be returned, even if no size parameter was given.

>>> print infile.read()
This is line number 1 in the file
This is line number 2 in the file
This is line number 3 in the file
```


File Pointer operations

Ethans
Learn from experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')

>>> print infile.read(5)
This

>>> print infile.read(5)
is li

>>> print infile.read()
ne number 1 in the file
This is line number 2 in the file
This is line number 3 in the file
```

File Pointer operations with tell

Ethans
Learn from experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')

>>> print infile.tell()
0
>>> print infile.read(5)
This
>>> print infile.tell()
5
>>> print infile.read()
is line number 1 in the file
This is line number 2 in the file
This is line number 3 in the file
>>> print infile.tell()
103
```


File Pointer operations with seek

Ethans
Learn from experts

```
>>> infile = open(r'C:\Users\jatin\Desktop\Ethans\test.txt', 'r')
>>> print infile.read(5)
This
>>> help(infile.seek)
Help on built-in function seek:

seek(...)
  seek(offset[, whence]) -> None.  Move to new file position.

  Argument offset is a byte count.  Optional argument whence defaults to
  0 (offset from start of file, offset should be >= 0); other values are 1
  (move relative to current position, positive or negative), and 2 (move
  relative to end of file, usually negative, although many platforms allow
  seeking beyond the end of a file).  If the file is opened in text mode,
  only offsets returned by tell() are legal.  Use of other offsets causes
  undefined behavior.
  Note that not all file objects are seekable.
```