Function, List ,Tuple & String

Functions in Python

Functions

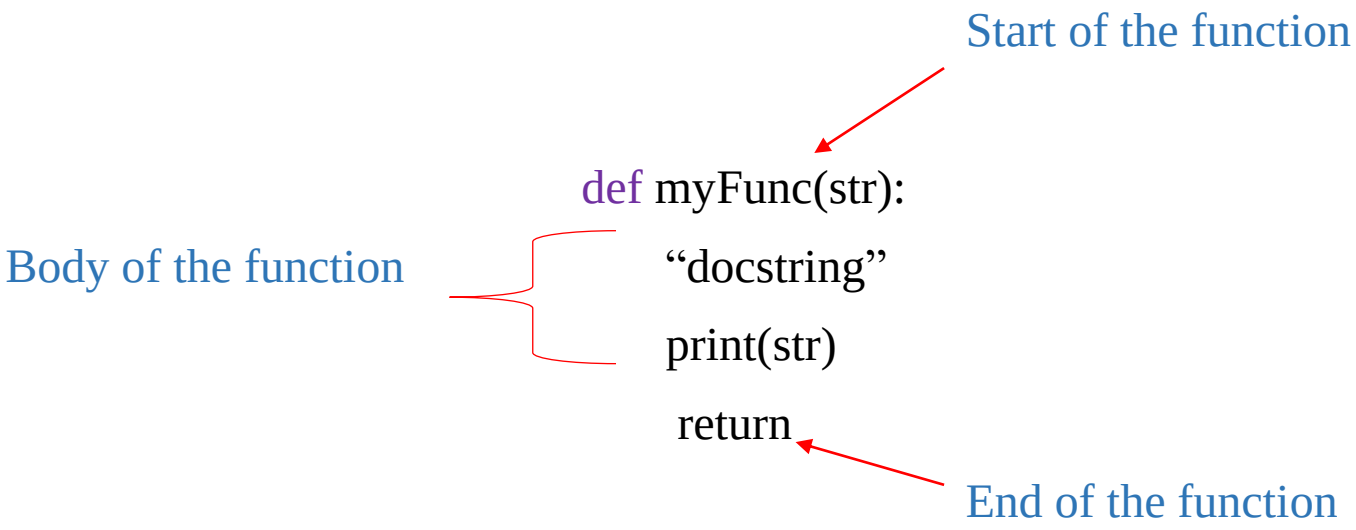
- A Function is a set of instructions that is used to perform a task of some kind.
- Function are :
  - ✓ Organized
  - ✓ Re-usable
  - ✓ Modularized
- Python has many **built-in** functions like print(), input(), int(), len() but you can also create your own functions.
- These function are called User-defined Function

Syntax :

```
def  <function_name>( parameter1, parameter2, ……., parameter ):
        Statement1
        Statement 2
        return [ expression ]
```

Defining and Calling a Function

Defining a Function



Calling of Function

Pass by Object Reference

- Python supports call by value (Where the value is always an object reference, not the value of the object)
- Hence many professionals term it as Pass by Object reference

Example 1:

In this example, we are appending value to an existing list. Hence the change is reflected in the list whose reference has been passed to the function.

```
>>> def add(list):
        list.append("INDIA")


>>> list=["CDAC"]
>>> add(list)
>>> print(list)
['CDAC', 'INDIA']
>>>
```

Pass by Object Reference

Example 2:

In this example we have assigned a new list. Hence the change is not reflected in the list whose reference has been passed to the function

```
>>> def add(list):
        list=["CDAC","INDIA"]


>>> list=["Patna"]
>>> add(list)
>>> print(list)
['Patna']
```

Function (Local and Global Variables)

- Local Variables
  - ✓ The variable defined within the function has a local scope and hence they are called local Variable.
  - ✓ Local scope means they can be accessed within the function only
  - ✓ They appear when the function is called and disappear when the function exits.
- Global Variable
  - ✓ The variable defined outside the function has a global scope and hence they are called global variable
  - ✓ Global scope means they can be accessed within the function as well as outside the function.
  - ✓ The value of a global variable can be used by referring the variable as global inside a function.

Global Variable

Local Variable

```
>>> var = 10
>>> def fun():
...     global var
...     print(var)
...     varlocal = var * 2
...     print(varlocal)
...
>>> fun()
10
20
>>>
```

Various Forms of Function Arguments)

You can call a function by using the following arguments

Positional/Required Argument : Arguments passed should match the function parameter from left to right.

```
# Postional Arguments
def  myfun(a,b,c):
            print(a,b,c)
# Calling of function
myfun(1,2,3)

  1 2 3
```

Default Arguments : We can assign default value for arguments to receive if the call passes too few values

```
# Default Argument
def  myfun(a,b=3,c=3):
            print(a,b,c)
# Calling of function
myfun(1)

1 3 3
```

Various Forms of Function Arguments)

Keyword/Named Arguments : We can call a function by specifying the keyword argument in the form **argument-name** = **value**

```
# Keyword Arguments
def  myfun(a,b,c):
            print(a,b,c)
# Calling of function
myfun( b=2, c=3, a=1)

1 2 3
```

Various Forms of Function Arguments

- Arbitrary/Variable Argument Lists :Sometimes, we do not know in advance the number of arguments that will be passed into a function.
- Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.
- These arguments are not named in the function definition. To add an arbitrary argument in the function definition, start the variable name with * and **

```
>>> def fun( *var_arg ):
```

```
# Arbitrary Arguments
def fun(*arr):
        for val in arr:
                print(val,)
# Calling the function
fun(1,2,3,4)
1 2 3 4
```

List in Python

List

- Lists are the most versatile data type of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets [ ].
- List items are indexed based and **mutable**
- List can store duplicate items

Example :

mylist = ['abcd', 123 , 23.4, ‘python', 34.5]

```
print (mylist)          # Prints complete list
print (mylist[0])       # Prints first element of the list
print (mylist[1:3])     # Prints elements starting from 2nd till 3rd
print (mylist[2:])      # Prints elements starting from 3rd element till last
print (mylist[:2])      # Prints elements starting from first to 2nd element
```

Built-in List Methods

- ✓ **append(object):**appends a object to list
- ✓ **count(object):**counts the occurrences of passed object
- ✓ **index(object):**finds the zero based index of an objects in list
- ✓ **insert(index , object):**inserts object at specified index
- ✓ **pop(index):**removes object from specified index if exists else error
- ✓ **remove(object):**removes the passed object if exists else error
- ✓ **clear():**clears whole list
- ✓ **reverse():**reverse the list items
- ✓ **sort():**sorts the list items

More on List

mylist=[1,2,3,4,5]

- ✓ **List Length:** print(len(mylist))
- ✓ **Min of list:** print(min(mylist))
- ✓ **Max of list:** print(max(mylist))
- ✓ **Sum of list values:** print(sum(mylist))
- ✓ **Extending list:**

  mylist.extend([6,7,8]) ; print(mylist)

  print(mylist+[6,7,8])
- ✓ **Reverse of list:** print(mylist[::-1])
- ✓ **Changing sub-parts of the list(With Slicing):** mylist[:2]=[8,9]; print(mylist)
- ✓ **Deleting elements from list:**

  del mylist[2] ; del mylist[2:5] ; del mylist ; mylist[2:5]=[ ]
- ✓ **Nested indexing:**

  mylist = ["CDAC",[1,2,3,4,5],[6,7,8,9]]

More on List

- ✓ **List Comprehension:** A list comprehension consists of the following parts:
  - An Input Sequence.
  - A Variable representing members of the input sequence.
  - An Optional Predicate expression.
  - An Output Expression producing elements of the output list from members of the Input Sequence that satisfy the predicate.

More on List

- ✓ **List Comprehension:**
  - List comprehension is an elegant and concise way to create new list from an existing list.
  - It consists of an expression followed by **for statement** inside square brackets.
  - Example:

```
pow3 = [3 ** x for x in range(10)]
print(pow3)

pow3 = [3 ** x for x in range(10) if x > 5]
print(pow3)

even = [x for x in range(10) if x % 2 == 0]
print(even)

data = ['4', '7.5', '8.3']
new_data = list(float(n) for n in data)
print(new_data)
```

Tuple

- A tuple is a fixed size grouping of elements.
- A Tuple is defined in the same way as a list except that the whole set of elements are enclosed in parenthesis instead of square brackets.
- A tuple is a immutable list. A tuple can not be changed in any way once it is created.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Example:

```
my_tup = ( 'Python‘ , 'Hadoop‘ , ‘Java‘ , ‘C‘ )
print( my_tup[0])
print( my_tup[-1])
print( my_tup[1:3])
```

Tuple and its Method

- ✓ Adding elements to the tuple is not possible. Tuples have no append or extend method
- ✓ Similarly removing elements from a tuple is not allowed. Tuple have no remove or pop method
- ✓ Tuple have index method. However earlier versions of python doesn't support index method
- ✓ Tuple supports Boolean expression like **in** and **not in** like **LIST**

Tuple Methods

- ✓ index(object , [start , stop])
- ✓ count(object)

Enumerate Function

- enumerate()

- ✓ The enumerate function returns a generator object - a kind of object that supports the iteration protocol
- ✓ **enumerate()** is a built-in python function which works on sequences like list, tuple etc. Enumerate returns (index, value) tuple ,each time through the loop of the sequence given.

Example:

```
my_tup=("hello","hi")
print(tuple(enumerate(my_tup)))
```

Standard Library

- ✓ Standard Library is a collection of tools that come with Python. The standard library includes the following:
  - ➢ Built-in Functions
  - ➢ Modules
  - ➢ Packages
- ✓ Python's tools are packaged in modules. Hence to use them, you have to import the module it is stored in
- ✓ The recommended way to import modules is by typing: import <modulename>

Example: import sys

Useful Built-in Functions

✓ Some common built-in functions:

➢ sorted()	abs()	all()	any()
➢ bin()	hex()	oct()	chr()
➢ ord()	enumerate()	eval()	input()
➢ int()	len()	open()	print()
➢ divmod()	dir()		

Lambda Functions

- ✓ In case you wish to make your functions more concise, easy to write and read, you can create Lambda functions
- ✓ Anonymous Lambda function can be defined using the keyword **lambda**
- ✓ However there are few constraints, that you need to follow:
  - ➢ They are syntactically restricted to a single expression (i.e they are one-line functions)
  - ➢ Lambda function can take any number of arguments (including optional arguments) and returns the value of a single expression
- ✓ Lambda functions are used along with built-in functions like **filter(), map()**

Syntax:

lambda arguments: expression

Lambda Functions

Examples:

#Lambda Function

```
increment=lambda num,incr=5:num+incr
print(increment(10))
print(increment(10,20))
```

#Using lambda function within a function

```
def increment():
    return lambda num,incr=5:num+incr
f=increment()
res=f(10)
print(res)
```

#Using lambda function with filter function

```
mylist = [1, 3, 4,6, 7, 8, 11, 10, 12]
new_list =list(filter(lambda x: (x%2 == 0) , mylist))
print(new_list)
```

Modules in Python

The code written in interpreter gets lost as soon as the interpreter is closed.

```
>>> # Function gets lost as soon as the interpreter is closed
>>> def fun():
...     print 'Lets assume this is a very Handy Function'
...     print 'This can used in other Applications as well'
...     sys.exit()
...
>>> fun()
Lets assume this is a very Handy Function
This can used in other Applications as well

Process finished with exit code 0
```

Modules in Python

- ✓ A module is a python file that (generally) has only definitions of variables, function, and classes.

Initializing a module

- ➢ Importing a module for the first time in a particular program, or in interactive mode, causes Python to perform a series of action initializing the module.
- ✓ Python creates a module namespace that stores the names defined in the module
- ✓ Python runs the code in the module
- ✓ Python stores the name of the module in the local namespace

Modules (Cont…)

- ✓ Accessing functions and other members inside a module
  - mymodule.myfunction(x)
- ✓ Other ways of accessing module attributes are
  - To import a specific item, type from, the module name, import, and the item name

    Example:

    ```
    from mymodule import myfunction
    ```
  - To import all of a module's functions, type from, the module name, import, and an asterisk(*), a wildcard character that stands for all

    Example:

    ```
    from mymodule import *
    ```
- ✓ **Importing a module using a different name**
  - To import a module (but not a function) using a different name, type import, the module name, as, and the name you want to use

    Example:

    ```
    import mymodule as mymod
    ```

sys Module

- ✓ The sys module is for controlling and interacting with the Python interpreter. It includes information about the operating system and the version of Python you're using
- ✓ To use sys module type : import sys

```
sys.exit()
```

sys Module

Common Data, Functions present in sys module

- sys.exit() function : Tells the python interpreter to quit. It's the most direct way to end a Python script while it's running
- sys.argv : Stores any command line arguments passed when you started Python. Also includes the name of the program you're running
- sys.stdin: Used by the input() function; accept input from user
- sys.stdio: Used by the print() In interactive mode, prints to the screen

os and subprocess Module

- ✓ The os and subprocess modules include code that lets Python work with your operating system - they even run operating system command

- ✓ The os module is best for the following tasks:

  - ➢ Working with paths and permissions (test for access to a path, changing directories, changing access permissions and user/group IDs)
  - ➢ Working with files (open, close, write, truncate)

Functions of OS module

- ✓ os.system() : Executing a shell command
- ✓ os.environ : Get the users environment
- ✓ os.getcwd() : Returns the current working directory.
- ✓ os.getpid() : Returns the real process ID of the current process.
- ✓ os.listdir(path) : Return a list of the entries in the directory given by path.
- ✓ os.mkdir(path) : Create a directory named path with numeric mode.
- ✓ os.remove(path) : Remove (delete) the file path.
- ✓ os.removedirs(path) : Remove directories recursively.
- ✓ os.rename(src, dst) : Rename the file or directory src to dst.
- ✓ os.rmdir(path) : Remove (delete) the directory path.
- ✓ os.open(file,flag,mode) :Opens a file to read/write data

Math Module

- ✓ In order to achieve the final result, you have to code many computational logics using mathematical functions
- ✓ To ease this task various mathematical functions are predefined and can be used using **math** module

Math Number-theoretic Function

→ math.ceil(x)

» Return the ceiling of x as a float, the smallest integer value greater than or equal to x

```
>>> # Using math.ceil(x)
>>> math.ceil(10.01)
11.0
```

→ math.copysign(x, y)

» Return x with the sign of y. On a platform that supports signed zeros, copysign(1.0, -0.0) returns -1.0

```
>>> # Using math.copysign(x, y)
>>> math.copysign(10, -1)
-10.0
```

→ math.fabs(x)

» Return the absolute value of x

```
>>> # Using math.fabs(x)
>>> math.fabs(-10)
10.0
```

random Module

→This module implements pseudo-random number generators for various distributions

Functions for integers:

→random.randrange(stop)

» Generates a random integer within the given range

```
>>> # Using random.randrange(stop)
>>> random.randrange(100)
60
```

→random.randrange(start, stop[, step])

» Return a randomly selected element from range(start, stop, step). This is equivalent to choice(range(start, stop, step)), but doesn't actually build a range object

```
>>> # Using random.randrange(start, stop, step)
>>> random.randrange(0,100,10)
20
```

→random.randint(a, b)

» Return a random integer N such that a <= N <= b

```
>>> # Using random.randint(a, b)
>>> random.randint(0, 100)
82
```

Date & Time Module

- ✓ Computers usually store time as the number of seconds that have passed since some specific date (called the epoch - 12:00am, January 1, 1970)
- ✓ Python has two primary modules with date and time tools:
  - ➢ The **datetime** module includes tools for working with dates, times, and combinations of both
  - ➢ The **time** module includes tools for working with times and dates in the recent past to near future. It focuses mostly on manipulating time based on the computer's internal representations of time

Date Time Module

✓ An object that stores date and time information looks like this:

datetime.datetime(1969, 7, 20, 22, 56)

- datetime.datetime is the type of object
- The arguments are integers, separated by commas, in order:

  year, month, and day_of_month

- Optional arguments :

  hours, minutes, seconds, microseconds.

- Another optional argument is time zone information, which defaults to None

Example:

```
import datetime
 # Datetime object
 str(datetime.datetime(1997,7,20,22,56))
# '1997-07-20 22:56:00‘
```

Strings in Python

String

- ✓ Python does not support a character type, these are treated as a string of length one.
- ✓ String are immutable.

Strings are ordered blocks of text

- ✓ Strings are enclosed in single or double quotation marks.
- ✓ Double quotation marks allow the user to extend strings over multiple lines without backslashes.

Example : ‘abc’ , “ABC”

Concatenation and repetition

- ✓ Strings are concatenated with the + sign:

```
>>>’abc’+’def’
‘abcdef’
```

- ✓ Strings are repeated with the * sign:

```
>>>’abc’*3
‘abcabcabc’
```

Operation on String

Indexing and Slicing Operation

✓ Python start indexing at 0.

✓ A string **name** will have indexing running from 0 to len(name)-1

Example :

```
s=‘CDAC INDIA’
print(len(s))
print(s[2])
```

Output :

```
10
A
```

Operation on String

Slicing Operation

- ✓ A subset of string is called “slice"
- ✓ You can get a subset of a string and other data structure, called a "slice", by specifying two indices
- ✓ Slicing works if one or both of the slice indices is negative
- ✓ Example :

```
s=‘CDAC INDIA’

print(s[ 0 : 3 ] )

print(s[ 1 : -1])

print(s[3::2])
print(s[::3])
print(s[::-2])
```

Operation on String

Membership checking

✓ **in** – Return true if the character exists in the given string.

Example:

```
>>> ‘D’ in ‘CDAC INDIA’
True
```

✓ **not in** - Return true if the character not exists in the given string.

Example:

```
>>> ‘D’ not in ‘CDAC INDIA’
False
```

String Formatting Operator %

✓ This function is unique to strings and is similar to functions from C’s printf() family

Operation on String

String Formatting Operator %

```
x=5; y=7

#formatted output

print("value of x:%05d\nvalue of y:%d"%(x,y))

#formatted output indexing /positional

 print("value of x:{1}value of y:{0}".format(y,x))

#keyword formatted output

 print("value of x:{k}value of y:{v}".format(k=x,v=y))

#width formatting

a1="Python"
a2="Java"
a3="Android"
print("|{:<10}|{:^20}|{:>10}".format(a1,a2,a3))
print("|{:<10}|{:^20}|{:>10}".format(a1,a2,a3))
print("|{:<10}|{:^20}|{:>10}".format(a1,a2,a3))
```

Built-in String Methods

- ✓ capitalize()
- ✓ len(str) //global function
- ✓ count(str, beg=0, end = len(str))
- ✓ encode(encoding='utf-16')
- ✓ index(str, beg=0, end=len(str))
- ✓ max(str) //Global Function
- ✓ min(str) //Global Function
- ✓ replace(old, new)
- ✓ upper()
- ✓ lower()
- ✓ title()
- ✓ split('delimeter')
- ✓ strip(chars)
- ✓ endswith()
- ✓ startswith()

THANK YOU!!