Functions

Enumerate Function in Python

```
print(names_list)

#for i in names_list:
#         print(i)

# looping with index
#for i in range(len(names_list)):
#         print(i,names_list[i])

for i,j in enumerate(names_list,1):
         print(i,j)
```

Iterators

- Iterators and Iterables in Python
- Ex1 - lists

for loop

```
    list1=[1,2,3,5.5,6.7,”ganesh”]
for I in list1:
        print(i)
```

Ex2 – Tuples

```
  tup1 = (1,2,3,4,5.5,6.7,”ganesh”)
for I in tup1:
        print(i)
```

Iterators

- Ex3 - sets

  for loop

      set1={1,2,3,5.5,6.7,”ganesh”}

for I in set1:

        print(i)

Ex4 – Strings

  mystr = “Hello Ganesh”

for i in mystr:

      print(i)

Iterators

- To print list of elements without loop
- 2 methods
  - Using Index

Ex : days_list=[“Sun”,”Mon”,”Tue”,”Wed”,”Thu”]

```
i=0
while I <= len(days_list):
        print(days_list[i])
        i+=1
```

Iterators

- Next method is using iterator protocol
- Way of working iterators and iterables

```
for i in list1:  # list1 is iterables
    print(i)       # i is iterator objects
```

Inbuilt Function is

```
L1=iter(list1)
next(L1)
Next(L2)
```

Note : lists/tuples/sets/dictionaries all are called iterables

Example

```
import cx_Oracle
con=cx_Oracle.connect('hr/hr@xe')

mycursor=con.cursor()

mycursor.execute('select * from employees')
names_list=[ names[1] for names in mycursor]
#print(names_list)

mynames=iter(names_list)
print(next(mynames))
print(next(mynames))
print(next(mynames))
print(next(mynames))
con.close()
```

```
def print_iterables(n):
        names=iter(n)
        while True:
                try:
                        nxt=next(names)
                except StopIteration:
                        break
                else:
                        print(nxt)

x=print_iterables(names_list)
print(x)
```

Generators Function

- Generator function is a function which returns generator-iterator with the help of yield keyword

Generator-iterator is special type of iterator ie. generator function will Generate iterators

All generator-iterator are iterators but all iterators are not generator-iterators

yield is like a return in normal function

```
def disp_sq(mx_no):
        x=1
        while x<mx_no:
                print(x*x)
                x+=1
                if x>mx_no:
                        break

disp_sq(10)
```

Generator Example 1

```
def disp_nos(mx_no):
        x=1
        while x < mx_no:
                yield x*x
                x+=1
                if x>mx_no:
                        break


no=disp_nos(10)
print(next(no))
```

Generator Example 2

```
def disp_fibo_nos(nos):
        a,b=0,1
        while True:
                c=a+b
                if c<=nos:
                        yield c
                        a=b
                        b=c
                else:
                        break

n=disp_fibo_nos(10)
print(next(n))
print(next(n))
print(next(n))
print(next(n))
```

- Nested Function
- Function return function
- References
- Function as parameters

Namespace and Variable Scope

- names used to identify the object – identifiers

- Namespace allows us to reuse the Names (identifiers)
    - Built-in namespace
    - Global Namespace
    - Local Namespace

Scope of Variables

```
print(dir())
num=10
def f1():
        n=10
        print("Inside Function ",dir())

print("Outside function ",dir())

f1()
```

Closures

```
# Nested Function
def f1():
        x=10
        def f2():  # Nested Function
                x=20
                print("Inside f2 ",x)
        f2()
        print("Inside f1 ",x)
```

- f1()

Closures

```
def f1():
        x=10
        def f2():
                y=3
                result=x+y
                return result
        return f2

a=f1()
print(a())
```

- Closure is a function object that remembers values in enclosing scope even if they are not present in memory

```
def f1():
        x=10
        def f2():
                y=20
                r=x+y
                return r
        #return f2()
        return f2
#ans=f1()
#print(ans.__name__)

ans=f1()
print(ans())
```

Criteria to Create Closures in Python

- Nested Function
- Nested Function must refer values in enclosing scope
- Enclosing function must return nested function

Advantages of Closures in Python

- Can avoid Global Values
- Provides some kind of data Hiding Capabilities

Decorators

- Decoration to make more presentable / attractive
- Python Decorators
- Any callable python object that is used to modify a function or class
  - It will take a function and it will add some functionality to it and will return

  Two Types – Function Decorator and Class Decorator

Functions as Parameters to another function

```
def f1():
        print(“Welcome to Function f1")

def f2(myf1):
        print(“Welcome to Function f2")
        f1()

f2(f1)
```

```
EX : def deco_upper(f1):
        def inner_f():
                mystr=f1()
                return mystr.upper()
        return inner_f

def disp_str():
        return("welcome to decoration ")

show_str=deco_upper(disp_str)
print(show_str())
```

```
Ex: def show():
        print("Ganesh")

#show()

def decorate(f):
        def inner1():
                print('---------')
                f()
                print('---------')
        return inner1

x=decorate(show)
x()
```

```
def decorate_string(f1):
        def inner():
                str1=f1()
                return str1.upper()
        return inner

@decorate_string
def show_string():
        return "Good Morning"

print(show_string())

#sh=decorate_string(show_string)
#print(sh())
```

- # Decorate function with Parameters

```
def decorate_div(f1):
        def inner(x,y):
                if y==0:
                        return "Invalid Input ....."
                return f1(x,y)
        return inner

@decorate_div
def div(a,b):
        return a/b

print(div(4,0))
```

- For Decorator Functions

    1. Need to take function as parameter
    2. Add Functionality to function
    3. Return another function

Multiple Decorators on single Function

```
# 2 decorator functions for a function

def convert_upper(f1):
        def inner():
                mystr=f1()
                return mystr.upper()
        return inner

def split_str(f1):
        def inner():
                str1=f1()
                return str1.split()
        return inner

@split_str
@convert_upper
def show_str():
        return "Good Morning "

#1. Decorator function to convert into upper
#2. Decorator function to split the String

print(show_str())
```

```
# Parameter [Passing for decorator function

def main_function(s):   # Outer function with Paramater
        def convert_upper(f1):
                def inner():
                        mystr=f1()
                        return mystr.upper() + s
                return inner
        return convert_upper

@main_function(" Ganesh ")  # Call decorator function with parameter
def show_str():
        return " Good Morning "

print(show_str())
```

```
# Genaral Decorator function for multiple functions

def general_decorator_f(f):  # General decorator function
        def inner(*args):
                L1=[]
                L1=args[1:]
                for i in L1:
                        if i==0:
                                return "Zero in Denominator....invalid!!!"
                return f(*args)
        return inner

@general_decorator_f
def divide_f1(x,y):
        return x/y

@general_decorator_f
def divide_f2(x,y,z):
        return x/y/z

print(divide_f1(10,0))
print(divide_f2(10,4,3))
```

Map / Filter / Reduce

Lambda Functions

Lambda Functions

- Functions defined without a name
- Also called as anonymous function
- Lambda functions does not contain any def keyword

- Syntax :

  lambda args : expression

Returns function object

```
def add(x,y):
        return x+y


res=add(10,290)
print(type(res))  # return is int object
print(res)



r1=lambda x,y : x+y
print(type(r1))     # Return is function object
print(r1(10,290))
```

Map() function

- Built in function
- Used to apply a function to all the elements of a sequence
- Syntax :

  map(function,sequence)

  Ex : To find all the squares of numbers in the given list of elements

1 Code for printing squares of numbers in list

```
mylist=[x for x in range(1,10)]
print(mylist)
new=[]
for item in mylist:
                    new.append(item**2)
print(new)
```

```
#map()
# to print squares of numbers in list
#1
nos_list=[x for x in range(1,5)]
print(nos_list)

#2
sq_nos_list=[x*x for x in range(1,5)]
print(sq_nos_list)

#3
for i in nos_list:
        print(i*i)
```

```
#4 Using Function
def square(n):
        return(n*n)

for i in nos_list:
        square(i)

#4 Using Map function
L1=list(map(square,nos_list))
print(L1)
```

- Note : map function returns list in python 2

```
#5 Using Lambda function
s=tuple(map(lambda x:x*x,nos_list))
print(s)
```

```
# Adding two Lists using map

nos2_list=[x for x in range(5,9)]
print(nos2_list)

newlist=tuple(map(lambda x,y:x+y,nos_list,nos2_list))
print(newlist)
```

2 Second Method -- using function

```
mylist=[x for x in range(1,5)]
print(mylist)

def sq(n):
        return n**2

s=list(map(sq,mylist))
print(s)
```

filter() Function

- This function will filter the elements of the iterables based on some function
- Used to filter the / some unwanted elements
- Python3 returns the filter object
- Python2 returns the output in the list form

- Syntax :

  filter(function,iterables)

```
# filter function
# Ex : to print all the even numbers from list
# 1 method

for i in range(1,11):
        if i%2==0:
                print(i)

#2 method
list1=[x for x in range(1,11) if x%2==0]
print(list1)

# 3 Using filter function   filter() functions are faster
L1=list(filter(lambda x:x%2==0,range(1,11)))
print(L1)

L2=tuple(filter(lambda x:x%2==0,range(1,11)))
print(L2)
```

reduce() Function

- This reduce function will reduce a iterable to single element using some functions

Output from reduce function will be single element

To perform some computation on list or tuples etc

we use reduce function

Function can be applied only on one iterable

Syntax :

reduce(function,iterable)

In Python3 – import functools

```
# Reduce function
# To find the sum of all elements in list
#1 method
numlist=[x for x in range(1,6)]
print(numlist)
s=0
for i in numlist:
        s=s+i

print("sum ",s)

#2 method
from functools import reduce

s=reduce(lambda x,y:x+y,numlist)
print(type(s))
print(s)
```

```
#3 method
def ret_sum(m,n):
        return m+n

x=reduce(ret_sum,numlist)
print("last ",x)
```

Frozen sets

Frozen set keyword makes mutables objects immutable

```
s=set(x for x in range(1,10))
print(s)

s.add(20)
print(s)

s1=frozenset(s)
s1.add(35)
print(s1)
```

Shallow Copy

- Is a copying method (copies the reference)
- It creates a new object which stores the reference of original object
- Can be used in for different ways :
    1. builtin functions

        list() , set(), dict()
    2. slicing operator
    3. using list comprehension method
    4. copy function from copy module

Builtin Function list()

```
list1=[1,2,3,4]
list2=list(list1)  # builtin function list()

print("list1 is ",list1)
print("list2 is ",list2)

list1.append("ganesh")

print("list1 is ",list1)
print("list2 is ",list2)
```

Using slicing operator

```
list1=[1,2,3,4]
list2=list1[:]
print("list1 is ",list1)
print("list2 is ",list2)

list1.append("ganesh")

print("list1 is ",list1)
print("list2 is ",list2)
```

Using list comprehension

```
list1=[1,2,3,4]
list2=[x for x in list1]

print("list1 is ",list1)
print("list2 is ",list2)

list2[1]="NewElement"

print("list1 is ",list1)
print("list2 is ",list2)
```

Using copy function from copy module

```
import copy

list1=[1,2,3,4]
list2=copy.copy(list1)
print("list1 is ",list1)
print("list2 is ",list2)

list2[3]="Ganpati"

print("list1 is ",list1)
print("list2 is ",list2)
```

- Shallow copy behaves differently when applied on nested list

Deep Copy

- Deep Copy is a method of copying and it creates a new object and recursively adds copies of nested objects present in the original elements

- # nested list
- import copy

- list1=[1,2,[4,5]]
- print(list1)

- list2=copy.copy(list1)
- print(list2)

- list2[2][0]="ganesh"
- print(list1)
- print(list2)
- list3=copy.deepcopy(list1)
- print(list3)
- print(list1)

- list3[2][0]="Bhosale" # Only new copy changed
- print(list3)
- print(list1) # old copy not affected

Dictionaries

```
# Creating Dictionary object

#Method1
months_dict={}    # Creating empty_dictionary
print(months_dict)
print(type(months_dict))

months_dict["Jan"]=31
months_dict["Feb"]=28
months_dict["March"]=31
months_dict["April"]=30
months_dict["May"]=31
months_dict["June"]=30
months_dict["July"]=31

print(months_dict)
```

```
#Method2

names_dict={"Ganesh":49,"Manish":45,"Nilesh":40}

print(names_dict)
```

```
# Access
# keys  using keys()

#1.1
print("Keys in names Dict ",names_dict.keys())
print("Keys in Months Dict ",months_dict.keys())

#Using for loop
for months in months_dict.keys():
        print(months)

#Using Iter function
mnths=iter(months_dict.keys())
print(next(mnths))

#Using List Comprehension
mnths_list=[mnths for mnths in months_dict.keys()]
print(mnths_list)
```

```
# Access
# Values using values()

print(months_dict.values())
print(names_dict.values())


# Access Values using Keys
print("Months in Jan is ",months_dict["Jan"])
```

```
# Dictionary Operations

#1 copy a dictionary

nos_dict={1:"One",2:"Two",3:"Three",4:"Four"}
print(nos_dict)

nos2_dict=dict(nos_dict)
#print(nos2_dict)

# Display length
print(len(nos_dict))

# Delete key-value pair from dictionary
del nos_dict[1]
print(nos_dict)

# Check for Existence
print(2 in nos_dict)
```

```
# Sorting Tuple and Dictionary using builtin function sorted()
#List
L1=[4,55,33,56,2,1]
print(sorted(L1))     # Sorted in Ascending Order
print(sorted(L1,reverse=True)) # Sorted in Descending Order

# Tuples
T1=(4,55,33,56,2,1)
print(sorted(T1))
print(sorted(T1,reverse=True))

T2=((3,66),(22,43),(45,44),(5,3))
print(sorted(T2))  # Sorted on First Index of every Element

nos_dict={3:"Three",1:"One",2:"Two",4:"Four"}
print(sorted(nos_dict)) # Sorting only Keys

print(sorted(nos_dict.values()))  # Sorting only Values

sorted_dict=sorted(nos_dict.items()) # Sorting entire dict
print(sorted_dict)
```

```
#for rows in mycursor:
        print(rows[1],int(rows[7]))

emp_dict={}
emp_dict={rows[1]:rows[7] for rows in mycursor}
print(emp_dict)

#print(sorted(emp_dict)) #sorted on names
print(sorted(emp_dict.items())) # Sorted Dictionary

# sort based on Salary Ascending
print(sorted(emp_dict.items(),key=lambda n:n[1]))

# sort based on Salary Descending
print(sorted(emp_dict.items(),key=lambda n:n[1],reverse=True))
```

Modules – sys / os / time / math / dir

- Modules are the group of functions / instructions / statements

sys module

sys module provides functions and variables used to manipulate different parts of the Python runtime environment.

sys.argv returns a list of command line arguments passed to a Python script. The item at index 0 in this list is always the name of the script.

Ex : import sys

```
print(sys.argv[1],sys.argv[2],sys.argv[3])
```

F:\cs_examples\modules.py ganesh manish hello 100 200

Note : argv[0] is always filename

sys

- sys.exit
- sys.maxsize - larges integer a variable can take
- sys.path
- sys.version
- getssizeof() function – returns memory occupied by object

OS

- The **OS module in python** provides functions for interacting with the operating system.
- **OS**, comes under **Python's** standard utility **modules**.
- This **module** provides a portable way of using operating system dependent functionality.
- The *__os__* and *__os__.path* **modules** include many functions to interact with the file system.

OS

```
import os
print(os.name)
print(os.getcwd())
os.rename()

import subprocess
command="dir"
os.system(command)
```

```
#import os
#print(os.name)
#print(os.getcwd())

import os
import subprocess
command="cmd"
oraconnect="sqlplus / as sysdba"
#os.system(command)
#subprocess.Popen(command)
subprocess.Popen(oraconnect)
```

time

```
# time module
import time
print(time.time())

#Display current time
curr_time=time.localtime(time.time())
print(curr_time)
print(curr_time[0],curr_time[1],curr_time[2])

#Formatted Time - Accepts a time-tuple and
#               returns a readable 24-character string
curr_time=time.asctime(time.localtime(time.time()))
print(curr_time)

# Displays current CPU time as a floating-point number of seconds.
print(time.clock())
x1=time.perf_counter
x2=time.process_time
print(x1())
```

- time.sleep(2) # suspends the program execution for 2 seconds

Glob()

- Python’s glob module has several functions that can help in listing files under a specified folder.
- We may filter them based on extensions, or with a particular string as a portion of the filename.
- All the methods of Glob module follow the Unix-style pattern matching mechanism and rules.
- However, it doesn’t allow expanding the tilde (~) and environment variables.
- Ex :

display all .py files from a directory

```
import glob
files=glob.glob('d:\\\cs_examples\*.py')
print(files)
```

- glob.lglob # glob generator object

Picking and unplicking in Python

- Python pickle module is used for serializing and de-serializing a Python object structure.
- Any object in Python can be pickled so that it can be saved on disk.
- What pickle does is that it “serializes” the object first before writing it to file.
- Pickling is a way to convert a python object (list, dict, etc.) into a character stream.
- The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

- Q. Find all installed modules in the system

```
import pickle

'''mynames=["Ganesh Bhosale","Manish","Nilesh","Nitin"]

myfile="names.pkl"
fo=open(myfile,"wb")
pickle.dump(mynames,fo)
fo.close()
'''

fo=open("names.pkl","rb")
str=pickle.load(fo)
print(type(str))
fo.close()
```

```
class Student:
        def __init__(self,st_rno,st_name,st_add):
                self.st_rno = st_rno
                self.st_name = st_name
                self.st_add = st_add
#       def show_student(self):
#               print(self.st_rno+" "+self.st_name)

Student1=Student(100,"ganesh","pune")
fname="student.pkl"
fp=open(fname,"wb")

pickle.dump(Student,fp)
print("Pickling of Student Done.....")

fp.close()
```

- Pickling saves processing time

```
fname="student.pkl"
fp1=open(fname,"rb")
obj=pickle.load(fp1)
obj.show_student()
fp1.close()
```

Profiling

- Profiling helps us find the bottlenecks in the program code
- Profiling is less Effort but big Performance gain
- Options for profiling in Python
  - Timers
    - Are easy to implement and can be used anywhere in the program to measure the execution time

- import time

```
start=time.time()
print("Hellooooooo Python.....")
end=time.time()
print("Time Consumed {} seconds".format(end-start))
```

```
def myfunction():
        a=5 + 3
        b=4 + 4
        c=a+b
        d=c/b
        return d
strt=time.time()
myfunction()
end=time.time()
print("Time Consumed {} seconds".format(end-strt))
```

```
strt1=time.time()
for rows in mycursor:
        print(rows[0],rows[1])
end1=time.time()
print("Time Consumed {} sec. by for loop".format(end1-strt1))



strt2=time.time()
names=[rows[1] for names in mycursor]
end2=time.time()
print("Time Consumed {} sec. ".format(end2-strt2))
```

XLS module

- XlsWriter is a python module for files in xlsx file format.
- It can be used to write text,numbers and formulates to multiple worksheets
- Supports features such as formatting, images, charts,page setup,

  autofilters, conditional formatting etc

```
import xlsxwriter

workbook = xlsxwriter.Workbook('helloGanesh.xlsx')
worksheet = workbook.add_worksheet()

worksheet.write('A1', 'RollNo')
worksheet.write('B1', 'Name')
worksheet.write('C1', 'Marks1')
worksheet.write('D1', 'Marks2')
worksheet.write('E1', 'Marks3')

worksheet.write('A2', '1001')
worksheet.write('A3', '1011')
worksheet.write('A4', '1101')
worksheet.write('A5', '2001')
worksheet.write('A6', '3001')
```

```
worksheet.write('B2', 'Ganesh')
worksheet.write('B3', 'Manish')
worksheet.write('B4', 'Nilesh')
worksheet.write('B5', 'Nitin')
worksheet.write('B6', 'Mangesh')

worksheet.write('C2', '65')
worksheet.write('C3', '45')
worksheet.write('C4', '55')
worksheet.write('C5', '75')
worksheet.write('C6', '55')

worksheet.write('D2', '66')
worksheet.write('D3', '55')
worksheet.write('D4', '44')
worksheet.write('D5', '76')
worksheet.write('D6', '56')

worksheet.write('E2', '76')
worksheet.write('E3', '88')
worksheet.write('E4', '67')
worksheet.write('E5', '77')
worksheet.write('E6', '88')

workbook.close()
```

```
import xlsxwriter

workbook = xlsxwriter.Workbook('employee.xlsx')
worksheet = workbook.add_worksheet()

# Start from the first cell.
# Rows and columns are zero indexed.
row = 0
column = 0

import cx_Oracle
con=cx_Oracle.connect('scott/tiger@orcl')
mycursor=con.cursor()
mycursor.execute('select * from emp')

names=[rows[1] for rows in mycursor if len(rows[1])==4]
#print(names)
```

- Note :
- Use xlrd Module for Reading Excel files
- Use Use Pandas for Excel

Regular Expressions

- Regular Expression is a tool for matching pattern in text
- Using regular Expression we can match / find / replace text or word in strings
- Why Regular Expressions :
  - String functions has some limitations

String Functions

```
mystr = "My Name is Ganesh"
newstr=mystr.replace("Ganesh","Manish")
print(mystr)
print(newstr)

str2="We have a broad road ahead"
# replace road by rd
newstr2=str2.replace("road","rd")
print(newstr2)

nn=str2[0:16]+str2[16:20].replace("road","rd")
print(nn)
```

Regular Expression Patterns

- Regular Expression is a tool for matching Patterns in text
- Patterns like
  - ^
  - $
  - .
  - [1-9] - any no from 1 to 9
  - [^1-9] - any no except 1 to 9
  - * - matches 0 or more occurences of preceding character
    - Ex ‘g*’ - 0 or more occurences of g

    + - matches 1 or more occurences of preceding character
      ex : ‘g+’ 1 or multiple g

    ? – zero or one occurrence of preceding character
      ex ‘g?’ 0 or 1 g

    {} - multiplication
      ex ‘g{2} ‘ i.e exactly gg
        ‘g{4,}’ 4 or more
        ‘g{3,7} – 3 till 7 g

    g|a - will match g or a

    () - group

    \s – matches space

    \S – matches non white space (any other character)

    \d – any cingle digit character

    \D – any single character but not digit

Python Regular Expressions Functions

- Match
- Search
- Replace
- Findall
- Split

- import re
- match –
- used to match any word or text in the string

```
# match    syntax : match(pattern,str,flag=0)
import re
str1="ganesh bhosale"
# match will match only start of the string
if re.match('ganesh',str1):
        print('yes')

if re.match('bhosale',str1):
        print('yes')
```

search syntax : match(pattern,str,flag=0)

- Will search text anywhere in string

```
import re
str1="ganesh bhosale"

if re.search('ganesh',str1):
        print('yes')

if re.search('bhosale',str1):
        print('yes')
```

- x=re.search('(g.*h)(.*s)',str1)

- print(x.group(0))
- print(x.group(1))
- print(x.group(2))