FUNCTIONS

Functions blocks begin def followed by the function name and parentheses ().
There are input parameters or arguments that should be placed within these parentheses.
You can also define parameters inside these parentheses.
There is a body within every function that starts with a colon (:) and is indented.
You can also place documentation before the body.
The statement return exits a function, optionally passing back a value.

A variable that is declared inside a function is called a **local variable**. The parameter only exists within the function (i.e. the point where the function starts and stops).
A variable that is declared outside a function definition is a **global variable**, and its value is accessible and modifiable throughout the program. We will discuss more about global variables at the end of the lab.

Python's Built-In Functions

Method is a member function that is invoked upon a specific object using an object-oriented message passing syntax, **such as data.sort( )**

len()	return len of the object or variable
sum()	return sum of elements in list/tuple.
sorted()	create and retrun a new list with elements in ascending order
sort()	return the same list with elements in ascending order

The identifiers used to describe the expected parameters are known as **formal parameters,**
The objects sent by the caller when invoking the function are the **actual parameters.**

Python provides means for functions to support more than one possible calling signature. Such a function is said to be **polymorphic FUNCTIONS**

Common Built-In Functions

abs(x)	Return the absolute value of a number.
all(iterable,'/')	Return True if bool(e) is True for each element e.
any(iterable)	Return True if bool(e) is True for at least one element e.
chr(integer)	Return a one-character string with the given Unicode code point.
divmod(x, y)	Return (x // y, x % y) as tuple, if x and y are integers.
hash(obj)	Return an integer hash value for the object.
id(obj)	Return the unique integer serving as an "identity" for the object.
input(prompt)	Return a string from standard input; the prompt is optional.
isinstance(obj, cls)	Determine if obj is an instance of the class (or a subclass).
iter(iterable)	Return a new iterator object for the parameter (see Section 1.8).
len(iterable)	Return the number of elements in the given iteration.
map(f, iter1, iter2, ...)	Return an iterator yielding the result of function calls f(e1, e2, ...)
	for respective elements e1 ∈ iter1,e2 ∈ iter2, …
max(iterable)	Return the largest element of the given iteration.
max(a, b, c, ...)	Return the largest of the arguments.
min(iterable)	Return the smallest element of the given iteration.
min(a, b, c, ...)	Return the smallest of the arguments.
next(iterator)	Return the next element reported by the iterator (see Section 1.8).
open(filename, mode)	Open a file with the given name and access mode.
ord(char)	Return the Unicode code point of the given character.
pow(x, y)	Return the value xy (as an integer if x and y are integers);
	equivalent to x y.
pow(x, y, z)	Return the value (xy mod z) as an integer.
print(obj1, obj2, ...)	Print the arguments, with separating spaces and trailing newline.

range(stop)	Construct an iteration of values 0, 1, . . . , stop−1.
range(start, stop)	Construct an iteration of values start, start+1, . . . , stop−1.
range(start, stop, step)	Construct an iteration of values start, start+step, start+2 step, . . .
reversed(sequence)	Return an iteration of the sequence in reverse.
round(x)	Return the nearest int value (a tie is broken toward the even value).
round(x, k)	Return the value rounded to the nearest k decimal places
sorted(iterable)	Return a list containing elements of the iterable in sorted order.
sum(iterable)	Return the sum of the elements in the iterable (must be numeric).
type(obj)	Return the class to which the instance obj belongs.

Sr.No.	Syntax/command	Output	Remarks
1	def test(): print("this is my first function")		While creating or defining a function, using print command will change the type of data type to none and performing any kind of concatenation operations will give an error.
2	type(test())	this is my first function NoneType	
3	def test1(): return "this is my first function"		To avoid data type as Nonetype use return command. So here the data type will be maintained and you can perform concatenation operations.
4	test1()	'this is my first function'	
5	type(test1())	str	

6	def test2():
	    pass
			It will just pass the defined function or else it will give an error

7	def test3():
	    return 1,2,3,4,[5,6,7,8]
			It you return multiple data types in the defined function then it produces output in tuples.

8	test3()	(1, 2, 3, 4, [5, 6, 7, 8])	

9	def test3():
	    return 1,2,3,4,[5,6,7,8]
			In case you define function with multiple data types, you can assign multple variables to each of those data types.

10	a,b,c,d,e = test3()		
11	a	1	
12	b	2	

13	def test4():
	    a = 4*5
	    b = 6+4
	    return a , b
			Assigning multiple variables to the output of the function. You can also assign symbols like _ to the output

14	test4()	(20, 10)	
15	g = test4()		
16	k,v = g		
17	k	20	
18	v	10	
19	test3()	(1, 2, 3, 4, [5, 6, 7, 8])	Incase you don’t want to define any variable you can simply use _.
20	_,_,_,_,g = test3()		
21	g	[5, 6, 7, 8]	
22	_	4	

23	def test5():
	    a = 1
	    b = 10
	    l = [ ]
	    while a <= b:
	        l.append(a)
	        a = a+1
	. return l
			You can define a entire code as a function wherein variables values are fixed. Always use return command inplace of print while defining a code within the function.

24	test5()	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	

25	def test5(a,b):
	  l = [ ]
	  while a <= b:
	    l.append(a)
	    a = a+1
	. return l
			Incase you don't want fixed variable in a defined function you can provide those variables inside the bracket. Always use return command inplace of print while defining a code within the function.

26	test5(1,5)	[1,2,3,4,5]	

27	def test6(l):
	  l1 = []
	  for i in l:
	    if type(i) == int:
	      l1.append(i)
	  return l1
			Here you are defining code as a function and can be used on any kind of list to filter out int.

28	def tuple_to_list(*a):		(*a) - To give any number of inputs to the function within the bracket. Such argument will always return a tuple

29	def test5(a,b,c,d,*m):
	  return a,b,c,d,m
		(33, 44, 2, 2, (1, 222, 3, 444, 66, 77, 99, 654))	

30	def test6(*m, a,b,c,d):
	  return m,a,b,c,d
	test6(23,31,23,43,4,5,554,332,a=232,b=111,c=2334,d=43)
		((23, 31, 23, 43, 4, 5, 554, 332), 232, 111, 2334, 43)	Unless you do not specify the values that a,b,c,d will take, it will show an error to you.

31	def test7(**args):
	  return args
	test7(a=1,b=2,c=3,d=4)
		{'a': 1, 'b': 2, 'c': 3, 'd': 4}	(**args) to get inputs in the form of dict

32	n = lambda a , b : a + b
	n(4,5)
		9	Lambda is unanymous function as there is no name for the function but is a reserved keyword in python and can perform same function as normal functions.

33	[i for i in t]	[1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 8, 8, 9]	In a single statement you can convert tuple to list. Return should be entered on left corner within the square bracket.
34	[i*i for i in range(10)]	[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]	It will give consecutive number squares within the list.
35	l1 = lambda *x : [i**2 for i in x] l1(1,2,3,4,5)	[1, 4, 9, 16, 25]	Lambda function

36	def add(a):
	  """
	  add 1 to a
	  """
	  b = a + 1
	  print(a, "if you add one", b)
	  return(b)
		"
		add 1 to a
		"
			Text to be displayed in documentation should be placed within the inverted commas

37	def printer(artist):
	  global internal_var
	  internal_var= "Whitney Houston"
	  print(artist,"is an artist")
			To use local variable within function as global variable.

38	def contains(data, target):
	for item in datat:
	if item == target: # found a match
	return True
	return False
			When True statement is executed, the function immediately ends. If the for loop reaches its conclusion without ever finding the match, the final return False statement will be executed.

39	max(a, b, key=abs).		It will compare abs(a) to abs(b)
40	print (any([False, False, False, False]))	FALSE	Any Returns true if any of the items is True. It returns False if empty or all

41	print (any([False, True, False, False]))	TRUE	are false **(like or condition)**
42	print (all([True, True, True, True]))	TRUE	All Returns true if all of the items are True (or if the iterable is empty). All can be thought of as a sequence of **AND operations** on the provided iterables
43	print (all([False, True, True, False]))	FALSE	

44	a ='1'
	b = '2'
	c = '3'
	print(a,b,c,sep=':')
		1:2:3	