1. What is the relationship between def statements and lambda expressions ?

def statements are used to create normal functions in python.

Whereas, lambda expressions are used to create anonymous, single use functions in python.

Both of them are functions, the internal working of both of them are same. Both belong to the class function.

1. What is the benefit of lambda?

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function

(a function that takes in other functions as arguments).

Lambda functions are used along with built-in functions like filter() , map() etc.

1. Compare and contrast map, filter, and reduce.

Similarities of these 3 functions.

These 3 are builtin python functions.

They each take a function name and a list of elements,

and then return the result of applying the function to each element in the list.

SYNTAX for map function : map(function, iterables)

SYNTAX for filter function: filter (function, iterables)

SYNTAX for reduce function: reduce(function, iterables)

In contrast, these 3 functions work differently as mentioned below.

map function takes another function as a parameter along with a sequence of iterables and returns an output

after applying the function to each iterable present in the sequence

def function1(a):

return 2\*a

x = map(function1, (1,2,3,4,5)) #x is the map object

print(x) # The output would be (2,4,6,8,10)

The filter() function is used to create an output list consisting of values for which the function returns true.

def function1(a):

if a%2==0:

return a

x = filter(function1, (1,2,3,4,5,6,7,8,9,10))

print(x) # It gives output (2,4,6,8,10)

The reduce() function applies a provided function to ‘iterables’ and returns a single value, as the name implies.

eg: reduce(lambda a,b: a+b,[1,2,3,4,5]) # it gives output 15

1. What are function annotations, and how are they used?

Function annotation links arbitrary metadata to the function arguments and its return value.

Function annotations are accessed using the ‘\_\_annotations\_\_’ attribute.

def fib(a:'float', b:'int')-> 'result':

pass

print(fib.\_\_annotations\_\_)

the output

{'b': 'int', 'a': 'float', 'return' : 'result'}

1. What are recursive functions, and how are they used?

Recursive functions are functions which call itelf inside its definition.

These functins call itself, until certain conditions are met.

def function1

....

function1 # recursive call. till the condition is met

The classic example of recursive fuction is finding factorial of a number

def factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* factorial(x-1))

num = 3

print("The factorial of", num, "is", factorial(num))

1. What are some general design guidelines for coding functions?

General guidelines for coding functions are

1. Use 4 space indentation and no tabs.

2. Use docstrings : There are both single and multi-line docstrings that can be used in Python.

3. Wrap lines so that they don’t exceed 79 characters

4. Use of comments so that its easy to understand for both coders and users.

7. Name three or more ways that functions can communicate results to a caller.

Functions communicate results to the caller either with print statement or using the return keyword.