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

The def defined functions do not return anything if no return is given whereas the lambda function does return an object.def function always uses def keyword, lambda functions uses lambda keyword. Lambda functions are only one of code to perform operations, def statement contains multiple lines of codes to get the result.

def add(I,j) :

return i+j

After function call:

add(4,8) #gives o/p: 12

add=lambda x,y:x+y

print(add(4,8)) #gives o/p: 12

1. What is the benefit of lambda?

Lambda function is a one liner function to get the solution. It can take any number of arguments but takes only 1 expression. syntax:

Lambda arguments: expression

1. **Map Function** accepts another function & a sequence of iterable as a parameter which gives output after applying function to each iterable.

Syntax:

Map(function,itearble)

Map function takes both user-defined functions & lambda functions.

Eg:

tup= (5, 7, 22, 97, 54, 62, 77, 23, 73, 61)

newtuple = tuple(map(lambda x: x+3 , tup))

print(newtuple)

Output:

(8, 10, 25, 100, 57, 65, 80, 26, 76, 64)

**Filter Function** is used to give output as True or False when the function is called.

Syntax:

Filter(function,iterables)

Like Map Function, Filter function also takes both user-defined & lambda functions.

Eg.

*User Defined function with filter function-*

def func(x):

if x>=3:

return x

y = filter(func, (1,2,3,4))

print(y)

print(list(y))

It produces output as [3,4]

*Lambda with filter function-*

y = filter(lambda x: (x>=3), (1,2,3,4))

print(list(y))

It produces output as [3,4]

**Reduce Function** applies a provided function to ‘iterables’ and returns a single value.

Syntax:

reduce(function,iterables)

reduce(lambda a,b: a+b,[23,21,45,98])

Output:

187

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

**Function annotations** are arbitrary python expressions that are associated with various part of

functions. These expressions are evaluated at compile time and have no life in python’s runtime environment. Python does not attach any meaning to these annotations.

Function annotations are completely optional both for parameters and return value.

Function annotations provide a way of associating various parts of a function with arbitrary python expressions at compile time.

Syntax:

def foo(x: expression, y: expression = 20):

   ….

python doesn’t provide any semantics with annotations. It only provides nice syntactic support for associating metadata as well as an easy way to access it. Also, annotations are not mandatory.

1. Normally a [function](https://www.programiz.com/python-programming/function) can call other functions. It is even possible for the function to call itself. These types of functions are termed as recursive functions.

For Example: Recursive function to find the factorial of an integer.

def factorial(x):

if x == 1:

return 1

else:

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

num = 3

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

The output is:

The factorial of 3 is 6

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

Coding rules and guidelines ensure that software is:

* Safe: It can be used without causing harm.
* Secure: It can't be hacked.
* Reliable: It functions as it should, every time.
* Testable: It can be tested at the code level.
* Maintainable: It can be maintained, even as your codebase grows.

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

When a function gets executed in the program, the execution control is transferred from calling a function to called function and executes function definition, and finally comes back to the calling function. In this process, both calling and called functions have to communicate with each other to exchange information.