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

Answer =

Define statements define named functions, while lambda expressions define anonymous functions (functions without a name).

Both can be used to create functions in Python, but lambda expressions are generally considered to be more concise and are used when the function is only needed in one place.

2. What is the benefit of lambda?

Answer =

Fewer Lines of Code

Lambda functions are inline functions and thus execute comparatively faster Many times lambda functions make code much more readable by avoiding the logical jumps caused by function calls

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

Answer =

map applies as a transformation to an element.

The map() function iterates through all items in the given iterable and executes the function we passed as an argument on each of them. Syntax : map(function, iterable(s))

filter accumulates only elements matching a condition.

filter() forms a new list that contains only elements that satisfy a certain condition, i.e. the function we passed returns True Syntax : filter(function, iterable(s))

reduce accumulates all elements to a single value, by using immutable values

reduce() works by calling the function we passed for the first two items in the sequence. The result returned by the function is used in another call to function alongside with the next (third in this case), element Syntax : reduce(function, sequence[, initial])

Animal = ["Anteater", "Dog", "Elephant", "Giraffe", "Cat"]

map\_object = map(lambda s: s[0] == "A", Animal)

for i in map\_object:

print(i)

True

False

False

False

False

#Filter function

Animal = ["Ant", "Dog", "pig", "Giraffe", "Cat", “Tiger”]

filter\_object = filter(lambda s: s[0] == "G", Animal)

for i in filter\_object:

print(i)

Giraffe

# Reduce function

from functools import reduce

list = [4, 3, 8, 3]

print(reduce(lambda x, y: x + y, list))

print("With an initial value: " + str(reduce(lambda x, y: x + y, list, 10)))

18

With an initial value: 28

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

Answer =

The common method for gaining access to function information, including its parameters and return value, is through function annotation.

These are only some arbitrary Python expressions that are optional and connected to other functions' components. They are only assessed at build time and are meaningless when the programme is running. Before some third-party libraries can access them, they have no value or meaning attached to them. By stating the type of the functions' parameters and return value, they are used to type-check the functions. We may enhance the assistance messages with the aid of the string-based annotations.

* Syntax :

def func(a: 'int') -> 'int':

pass

* Annotations for simple parameters:

def func(x: 'float'=10.8, y: 'argument2'):

* In the above code the argument, ‘x’ of the function func, has been annotated to float data type and the argument ‘y’ has a string-based annotation. The argument can also be assigned to a default value using a ‘=’ symbol followed by the default value. These default values are optional to the code.

Annotations for return values:

def func(a: expression) -> 'int':

The annotations for the return value is written after the ‘->’ symbol.

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

pass

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

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

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

Answer =

A recursive function is a function that calls itself during its execution. This means that the function will continue to call itself and repeat its behavior until some condition is met to return a result

def fact(x):  
 if x == 1:  
 return 1  
 else:  
 return x \* fact(x - 1)  
  
print(fact(3))

6

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

Answer =

1. Single Responsibility Principle: A function should have a single, well-defined purpose.
2. Readability: Use descriptive names for functions and variables, write clear and concise code, and follow standard formatting conventions.
3. Separation of Concerns: Keep related code together in functions and separate unrelated code into separate functions.
4. Reusability: Design functions to be flexible and reusable, allowing them to be used in multiple contexts.
5. Error Handling: Consider and handle error conditions, such as invalid inputs, in a consistent and robust manner.
6. Avoid Global State: Limit the use of global variables and avoid making changes to shared state within a function.
7. Avoid Side Effects: Functions should only perform actions that are necessary for their defined purpose and should not have any unintended side effects.
8. Testability: Design functions to be easily testable, with clear inputs and outputs and a well-defined behavior

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

Answer =

1. Function can return single value
2. Can return multiple values, tuple
3. can return list,dictionary
4. can return function object