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

Ans: They both create function. The body of def statement is the code block which can contain multiple statements such as loops, conditions, exceptions, etc. Whereas lambda body is a single expression that returns a value.

2. What is the benefit of lambda?

Ans: It creates anonymous function with single expression that returns a value.

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

Ans: map() takes a function and one or more iterables as arguments. The output is an iterator

that returns the transformed items.  
Syntax:   
 map(function, iterable[, iterable1, iterable2,..., iterableN])  
Example:  
>>> z = [4,5,6]  
>>> y = list(map(lambda x: x\*\*2, z))  
>>> y  
[16, 25, 36]

filter()  processes an iterable and extracts the items that satisfy a given operation. The

function argument must be a single-argument function. It’s typically a boolean-valued function that

returns either True or False.  
Syntax:   
 filter(function, iterable)  
Example:  
>>> marks = [40,80,60,50,90,95,70]  
>>> over\_75 = list(filter(lambda x: x>75,marks))  
>>> over\_75  
[80, 90, 95]

reduce() takes a function of two arguments, applies cumulatively to all the items in

iterable, and returns a single final value.

Syntax:

reduce(function, iterable[, initial])

If initial is supplied, then it becomes the first argument to function and the first element in iterable

becomes the second element.

Example:

>>> from functools import reduce

>>> num = [1,2,3,4,5]

>>> sum = reduce(lambda x,y: x+y, num)

>>> sum

15

>>> sum = reduce(lambda x,y: x+y, num,-1)

>>> sum

14

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

Ans:

Function annotations allows to add arbitrary metadata to function parameters and return value.  They are evaluated at compile time and they do not exist at run time. They are used and interpreted by a third party or external python libraries. They are optional parameters that follow the parameter name.

1. Annotations for simple parameters :

def foobar(a: expression, b: expression):

Example: def sum(a: int, b: int ):

return a+b

1. Annotations for excess parameters :  
   def foobar(\*args: expression, \*kwargs: expression):
2. Annotations for return type :  
   def foobar(a: expression, b: expression)->expression:

Example: def sum(a: int, b: int )->int:

return a+b

1. Annotations for nested parameters:

def foobar((a: expression, b: expression), (c: expression, d: expression)):

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

Ans: Function that calls itself is said to be recursive, and the technique of employing a recursive function is called recursion. The function will continue to call itself until some condition is met to return a result.

1. Example of a recursive function to find the factorial of an integer:

def factorial(x):  
 if x == 1:  
 return 1  
 else:  
 return (x \* factorial(x-1))

1. Example of a recursive function to find the sum of an integer:

def sum(x):  
 if x==1:  
 return 1  
 else:  
 return x+sum(x-1)

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

Ans:

1. Use 4-space indentation and no tabs.   
 2. Use docstrings  
 3. Don't write more than 79 characters in a line  
 4. Use a space before and after an operator  
 5. Use snake\_case for functions and variables naming and PascalCase for classes naming.  
 For constants, use all uppercase letter separated with underscores.  
 6. Update comment while updating code.

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

Ans:

1. To call a function, specify the function name with the round brackets.  
2. Use of return keyword inside function which returns the results to the caller.  
3. Use of print statement inside function which prints the results to the caller.