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

Ans: As an expression, lambda returns a value that can optionally be assigned a name. In contrast, the def statement always assigns the new function to the name in the header, instead of returning is as a result. lambda's body is a single expression, not a block of statements

2. What is the benefit of lambda?

Ans: The lambda keyword in Python provides a shortcut for declaring small anonymous functions. Lambda functions behave just like regular functions declared with the def keyword. They can be used whenever function objects are required.

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

Ans: Map operation takes a mapping function and a vector of data as arguments and returns a new vector, which is the result of applying the mapping function on each element of the vector independently. The returned value from map() (map object) then can be passed to functions like list() (to create a list), set() (to create a set) and so on.

The filter function operates on a list and returns a subset of that list after applying the filtering rule.

The reduce function will transform a given list into a single value by applying a given function continuously to all the elements. It basically keeps operating on pairs of elements until there are no more elements left.

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

Ans: 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 of function annotations for simple parameters and for excess parameters respectively:

def foobar(a: expression, b: expression = 5):

def foobar(*args: expression,* kwargs: expression):

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

Ans: A recursive function is a function defined in terms of itself via self-referential expressions. This means that the function will continue to call itself and repeat its behavior until some condition is met to return a result.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

```
def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
    print(result)
  else:
    result = 0
  return result

print("Recursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
21
```

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

Ans: Use 4-space indentation and no tabs.

Use docstrings

Wrap lines so that they don't exceed 79 characters

Use of regular and updated comments are valuable to both the coders and users

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

Ans: To call a function, we specify the function name with the round brackets.

Use of return keyword inside function which returns the results to the caller.

Use of print statement inside function which prints the results to the caller.