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

Both lambda and def create the same kind of function – they have the same kind of metadata and capabilities. Their technical difference is syntactical: A lambda is an expression producing a function. A def is a statement producing a function.

2. What is the benefit of lambda?

* The limited operation can be performed using lambda functions
* No need of using the return statement
* Execution time of the program is fast for the same operation
* Fewer Lines of Code

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

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])

4. 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. They take life when interpreted by third party libraries, for example, mypy.

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.

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

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of constructs are termed as recursive functions. The function will continue to call itself and repeat its behaviour until some condition is met to return a result

**Example Code:**

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))

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

* Use 4-space indentation and no tabs.
* Use docstrings
* Wrap line that they don’t exceed 79 characters
* Use of regular and updated comments are valuable to both the coders and users
* Use of trailing commas : in case of tuple -> ('good',)
* Use Python’s default UTF-8 or ASCII encodings and not any fancy encodings
* Naming Conventions
* Characters that should not be used for identifiers : ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), ‘I’ (uppercase letter eye) as single character variable names as these are similar to the numerals one and zero.
* Don’t use non-ASCII characters in identifiers
* Name your classes and functions consistently
* While naming of function of methods always use self for the first argument

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

Function can return single value; Can return multiple values, tuple; can return list,dictionary; can return function object; can return class object