1. What is the relationship between def statements and lambda expressions ?\n",

Ans:`def` statement is used to create a normal function. where as `lamba` expressions are used to create `Anonymous functions`. which can be assigned to a variable and can be called using the variable later in function.\n",

"\n",

"Lambda's body is a single expression and not a block of statements like def statement. The lambda expression's body is similar to what we'd put in a def body's return statement. We simply type the result as an expression instead of explicitly returning it. Because it is limited to an expression, a lambda is less general than a def statement. "

2. What is the benefit of lambda?

Ans:The following are some of the benefits of lambda expressions:

1. Can be used to create Nameless/Anonymous functions inside some complex functions if we are planning to use it only once.

2. Moderate to small functions can be created in a single line.

3. Fuctions created using lambda expressions can be assigned to a variable and can be used by simply calling the variable

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

Ans: The differences between map, filter and reduce are:

1. map()`: The map() function is a type of higher-order. This function takes another function as a parameter. along with a sequence of iterables and returns an output after applying the function to each iterable present in the sequence.

2. `filter()`: The filter() function is used to create an output list consisting of values for which the function returns true.

3. `reduce()`: The reduce() function, as the name describes, applies a given function to the iterables and returns a single value

**from** functools **import** reduce

*# map function*

print('Map ->',list(map(**lambda** x:x**+**x, [1,2,3,4])))

*# fitler function*

print('Filter ->',list(filter(**lambda** x:x**%2** !=0, [1,2,3,4])))

*# reduce function*

print('Reduce ->',reduce(**lambda** x,y:x**+**y, [1,2,3,4,5,6]))

Reduce -> 21

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

Ans: Function annotations provide a way of associating various parts of a function with arbitrary pythoncexpressions at compile time.

Annotations of simple parameters `def func(x: expression, y: expression = 20):

Whereas the annotations for excess parameters are as − `def func (\*\*args: expression, \*\*kwargs: expression):`"

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

Ans: A recursive function is a function that calls itself during its execution. The process may repeat several times, "outputting the result and the end of each iteration. "

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

Ans:\*\* Some of the general design guidelines for coding functions are:

1. Always use a docstring to explain the functionality of the function

2. avoid using or limited use of global variables

3. Proper Identation to increase the code readability

4. try to follow a naming convention for function names (pascalCase or camelCase) and stick with the same convention throughout the application.

5. Avoid using digits while choosing a variable name

6. try to use a name for the function which conveys the purpose of the function

7. Local variables should be named using camelCase format `(ex: localVariable)` whereas Global variables names should be using PascalCase `(ex:GlobalVariable).

8. Constant should be represented in allcaps `(ex:CONSTANT)`.

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

Ans: Some of the ways in which a function can communicate with the calling function is

"1. print\n",

"2. return\n",

"3. yield"