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

Ans: Both def statement and lambda expressions are used for defining function, in def statement we need to specify function, but in lambda we don’t need. In def statements number of small operation can be done inside function, but lambda it is very limited almost 1 to 2 operations.

1. What is the benefit of lambda?

Ans: Lambda used for small operations, it reduces multiple lines of code.

They are generally used when a function is needed temporarily for a short period of time, often to be used inside another function such as filter, map and reduce.

Using lambda function, you can define a function and call it immediately at the end of definition. This can’t be done with def functions.

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

Ans: All three functions to apply on iterator as input

MAP function to each element of iterator and collects result.

FILTER applies function to each element of iterator and collect those elements for which function returns true. For this function need to return Boolean value.

REDUCE applies rolling computation to sequential pair of elements in iterator. Initially it takes two elements from iterator in sequence, applies function, collect result, then take next element in sequence from iterator, applies function and repeat this until list has single value.

fruit =['Apples','Bananas','Graphes','Amul']

print(list(map(lambda a:a[0]=="A",fruit)))🡪 [True, False, False, True]

print(list(filter(lambda a:a[0]=="A",fruit)))🡪 ['Apples', 'Amul']

from functools import reduce

a=[1,2,3,4]

print(reduce(lambda x,y:x+y,a))🡪10

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

Ans: Function annotations adds a feature that allows you to add arbitrary metadata to function parameters and return value. Provide a way of associating various parts of a function with arbitrary python expressions at compile time.

import math

def circ(radius:float)->float:

return 2\*math.pi\*radius

circ.\_\_annotations\_\_

result: {'radius': float, 'return': float}

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

Ans: recursive functions, calling a function itself.

def tri\_recurse(k):

if k>0:

result= k+tri\_recurse(k-1)

else:result=0

return result

tri\_recurse(6)

result:21

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

Ans:

* Use 4 spaces instead of tabs
* Maximum line length is 120 symbols
* 2 blank lines between classes and functions
* 1 blank line within class between class methods
* No blank line following a def line.
* No whitespace inside parenthesis, brackets, braces.
* Surround operators with single whitespace on either side.
* Never use whitespaces around = when passing keyword arguments or defining a default parameter value.
* Use blank lines for logic separation of functionally within functions/methods whenever it is justified
* Move function arguments to a new line with an indentation, if they do not fit into the specified line length

# Good

def long\_function\_name(var\_one, var\_two, var\_three,

var\_four):

print(var\_one)

# Good

def long\_function\_name(

var\_one,

var\_two,

var\_three,

var\_four

):

print(var\_one)

* Move the logical conditions to the new line if the line does not fit the maximum line size. This will help in understand the condition by looking from top to bottom. Poor formatting makes it difficult to read and understand complex predicates.

# Good

if (this\_is\_one\_thing

and that\_is\_another\_thing

or that\_is\_third\_thing

or that\_is\_yet\_another\_thing

and one\_more\_thing

):

do\_something()

* Use multiline strings, not \\ since it gets much more readable.
* Place a class’ \_\_init\_\_ at the beginning of each class.
* Use named arguments to improve readability and avoid dummy mistakes in the future.
* Never end your lines with a semicolon, and do not use a semicolon to put two statements on the same line.
* Chaining methods should be broken up on multiple lines for better readability

(df.write \

.format('jdbc')

.option('url', 'jdbc:postgresql:dbserver')

.option('dbtable', 'schema.tablename')

.option('user', 'username')

.option('password', 'password')

.save()

)

* Names should be clear about what a variable, class, function contains/do. If a developer cannot come up with one clear name, then something is wrong with the implementation
* If the code needs comments to clarity its work, you should think about refactoring or rewriting it. The best comments to code are the code itself.
* Minimize the amount of code in a try/except block, The larger the body of the try, the more likely of code that you didn’t expect to raise an exception.
* Never use \* in imports. Always be explicit about what you’re importing. Namespaces make code easier to read so please use them.
* Break long imports using parenthesis and indent by 4 space. Include the training comma after the last import and place the closing bracket on a separate line
* It’s a bad idea to use constructions like, global variables, iterators where they can be replaced by vectorized operations, lambda where it is not required, map and lambda where it can be replaced by a simple list comprehension, multiple nested maps and lambdas, nested functions.

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

Ans: ways to calling a function

function(),function(\*args),print(function())