Functions

What is a generator function in Python?

A generator function in Python is a type of function that, instead of returning a single value, returns an iterator that can be used to generate a sequence of values over time.

Generator functions are defined using the same def keyword as normal functions, but they use the yield statement to yield values one at a time. This allows for generating values lazily, meaning they are generated on-the-fly and consumed only as needed, which can be more memory-efficient for large datasets or complex computations.

Example:

```
# fibonacci series


# 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610


def fib(n):
  a, b = 0,1
  for i in range(n):
    yield a # yield means = try to give me a record 1 by 1
    a,b = b, a+b
```

Explain the role of the 'yield' keyword in a generator function.

The primary purpose of the 'yield' keyword in Python is "To enable a function to yield values over time." When used within

a function, the yield keyword allows the function to return a value and pause its execution, which can then be resumed from the same point in subsequent calls. This makes the function a generator function, allowing it to generate a sequence of values over time rather than computing and returning them all at once.

How is a lambda function defined in Python?

A lambda function in Python is defined using the lambda keyword, followed by a list of parameters, a colon, and the expression it evaluates and returns.

Example:

```
# lambda function to find maximum of two numbers

max = lambda x,y : x if x > y else y

max(10,20)
```

Answer : 20

What are the advantages of using lambda functions?

Lambda functions in Python offer several advantages, particularly in terms of brevity, simplicity, and flexibility. Here are some of the key benefits:

Conciseness: Lambda functions are concise, allowing you to define a function in a single line of code. This can make your code more readable and straightforward when the function is simple.

Anonymity: Since lambda functions are anonymous (they don't need a name), they are useful when you

need a function for a short period, especially as an argument to higher-order functions that expect a function as input, such as map(), filter(), and sorted().

Immediate Use: Lambda functions are defined and used on the spot, without the formal def keyword. This is particularly handy in functional programming styles or when defining simple callback functions for user interface actions, etc.

Simplicity: They are useful for simple operations that can be expressed in a single expression. This makes them syntactically simpler than using full function definitions for small tasks.

Inline Definition: Lambda functions allow for defining a function inline, which can make it easier to understand the flow of the program by keeping the function logic close to where it is used.

Functionality: They support all the operations available to traditional functions, including immediate execution, assignment to variables, and passing as arguments to functions.

How does the 'map' function work in Python?

The map() function in Python applies a specified function to each item of an iterable (like a list, tuple, etc.) and returns a map object (which is an iterator) of the results.

How it works:

Apply Function: The map() function takes each item from the iterable(s) and passes it as an argument to the function.

Return Results: It collects the results of applying the function to each item and stores them in a map object.

Iterator: The map object can then be converted to a list or another iterable type, or iterated over to access the transformed elements.

Example in next question answer.

Provide an example of using the 'map' function in Python.

Example :

```
l = [1,2,3,4,5,6]

def sq(x):
  return x**2

list(map(sq,l))  # map function returns a map object, so we need to convert it into list. map(function, argument)
```

Answer : [1, 4, 9, 16, 25, 36]

Explain the 'reduce' function in Python with an example.

The reduce() function in Python, which is part of the functools module, is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along. This function reduces a list to a single value by combining elements via a supplied function. The

reduce() function takes two arguments: a function and a sequence.

Example :

```
l3 = [250,2,3,4,55,6,7,8,88,123]

reduce(lambda x,y : x if x>y else y, l3) # reduce function can also be used to find the maximum number in the list
```

Answer : 250

What is the purpose of the 'filter' function in Python?

The purpose of the filter() function in Python is to construct an iterator from those elements of an iterable (like a list, tuple, etc.) for which a function returns true. In other words, filter() takes a function and an iterable and filters out those elements in the iterable that do not match the criteria specified by the function, returning an iterator with the elements that do match.

Differentiate between map and filter functions in Python.

The map() and filter() functions in Python are higher-order functions that apply a given function to each item of an iterable (like a list or tuple), but they serve different purposes and operate in distinct ways. Here's a comparison:

Purpose

map(): Applies a function to all the items in an input list (or iterable) and returns an iterator that produces the results of applying the function to each item.

filter(): Filters items out of an iterable by applying a function to each item and returning an iterator with the items for which the function returns True.

Function Requirement

map(): The function applied can transform each item into a new value; the return value of the function is directly included in the map object.

filter(): The function applied must return a Boolean value, True or False. Items for which the function returns True are included in the filter object.

Return Value

map(): Returns an iterator that computes the function applied to each item of the iterable. It can produce an iterable of any type, not limited to Boolean values.

filter(): Returns an iterator that contains only the items from the original iterable for which the function returned True, effectively filtering out items that do not match the condition.

Use Case

map(): Used when you need to apply a transformation to each item in the iterable, such as converting all strings in a list to uppercase, or squaring all numbers in a list.

filter(): Used when you need to select a subset of items from an iterable based on a condition, such as finding all even numbers in a list.

What are the limitations of using lambda functions?

Lambda functions in Python, while useful for creating small, anonymous functions on the fly, come with certain limitations:

Single Expression: Lambda functions are limited to a single expression. This means you cannot write complex logic inside lambda functions, including statements like if...else, loops, or multiple expressions that need to be executed sequentially.

Readability: For complex operations, using lambda functions can make the code harder to read and understand, especially for those not familiar with the syntax or concept of lambda functions.

No Statements or Annotations: You cannot use statements (like pass, assert, or raise) or annotations inside a lambda function. This limitation restricts their use to simple operations.

Limited Functionality: Because of their single-expression nature, lambda functions are not suitable for tasks that require multiple operations or intermediate variables. For more complex functionalities, a standard function defined with def is more appropriate.

Debugging Difficulty: Debugging lambda functions can be more challenging than regular functions because they do not have a name. This means that the traceback does not include the name of the lambda function when an error occurs, making it harder to pinpoint the source of the error.

Scope Limitation: Like regular functions, lambda functions also have their own scope. However, because they are defined inline, it can sometimes be confusing to understand what variables are accessible within the lambda function, especially for beginners.

How can generator functions help in memory optimization?

Generator functions can significantly help in memory optimization due to their ability to yield items one at a time, instead of storing the entire collection of items in memory at once. This lazy evaluation approach offers several advantages for memory efficiency:

1. On-demand Value Generation

2. Reduced Memory Footprint

3. Lazy Evaluation

4. Efficient Pipeline Processing

5. Handling Infinite Sequences

Example:

Consider processing a large file. Using a generator function to read and process the file line by line can drastically reduce memory usage compared to reading the entire file into a list of lines first and then processing it.

```
def read_large_file(file_name):
 """A generator function to read a large file line by line."""
 with open(file_name, 'r') as f:
  for line in f:
   yield line.strip()
```

Hear, read_large_file reads one line at a time, allowing for the processing of files that are too large to fit into memory.

In summary, generator functions are a powerful tool for memory optimization in Python, enabling efficient processing of large or infinite data sets with minimal memory usage.

Write a Python code to demonstrate the use of a generator function.

```
# fibonacci series

# 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610

def fib(n):
 a, b = 0,1
 for i in range(n):
```

```
    yield a # yield means = try to give me a record 1 by 1
    a,b = b, a+b


for i in fib(10):
    print(i)
```

Answer :

0

1

1

2

3

5

8

13

21

34

Explain the concept of 'lazy evaluation' in the context of generator functions.

Generators do not compute the values they yield until they are specifically requested. This lazy evaluation model means that no unnecessary computations are performed, and no memory is allocated for the results until needed. This not only saves memory but can also save computational resources.

Discuss a real-world scenario where the use of map, reduce, or filter functions can be beneficial.

1. Calculating Total Sales per Product Category (Using reduce):

To calculate the total sales for each product category, the company could use the reduce function from the functools module. By grouping transactions by category and then applying reduce to sum up the sales amounts, they can efficiently calculate the total sales per category.

2. Identifying Transactions Above a Certain Value (Using filter):

The company might want to identify all transactions above a certain value to analyze high-value sales. Using filter, they can easily extract these transactions from the dataset.

3. Converting Transaction Amounts from One Currency to Another (Using map):

If the transactions are recorded in different currencies, and the company wants to analyze them in a single standard currency, map can be applied to convert all transaction amounts to this standard currency using a conversion function.

Benefits in the Real-World Scenario

Efficiency: These functional programming tools can efficiently process large datasets by applying operations in a concise and readable manner.
Memory Optimization: Especially with map and filter, the operations are lazy, meaning they don't load the entire dataset into memory at once, which is crucial for processing large datasets.
Readability: The code becomes more readable and concise, making it easier for other developers or analysts to understand and maintain it.
Reusability: The functions used with map, reduce, and filter can be easily reused for other analyses or datasets, improving code modularity.