Question -1 what is the diff between a function and method in python

This is a function

def greet(name):

  """This is a function that greets a person."""

  print(f"Hello, {name}!")

This is a method

class Dog:

  def bark(self):

    """This is a method that makes the dog bark."""

    print("Woof!")

Calling the function

greet("Alice")

Creating an instance of the class and calling the method

my\_dog = Dog()

my\_dog.bark()

Question 2 -explain the concept of function argument and parameter in python

Function definition with a parameter

def greet(name):

  """

  This function greets the person passed in as a parameter.

  """

  print(f"Hello, {name}!")

Function call with an argument

greet("Alice")

Question 3- explain the different ways to define and call a function in python

Function definition with no arguments

def greet():

  """This function prints a greeting."""

  print("Hello!")

Calling the function

greet()

Function definition with arguments

def add\_numbers(x, y):

  """This function adds two numbers."""

  sum = x + y

  return sum

Calling the function with arguments

result = add\_numbers(5, 3)

print(result)

Function definition with default arguments

def greet\_person(name="World"):

  """This function greets a person with a default name."""

  print("Hello, " + name + "!")

Calling the function with and without arguments

greet\_person()  # Output: Hello, World!

greet\_person("Alice")  # Output: Hello, Alice!

Function definition with variable number of arguments

def sum\_all\_numbers(\*args):

  """This function sums all the numbers passed as arguments."""

  total = 0

  for num in args:

    total += num

  return total

Calling the function with multiple arguments

result = sum\_all\_numbers(1, 2, 3, 4, 5)

print(result)

Function definition with keyword arguments

def print\_info(\*\*kwargs):

  """This function prints the key-value pairs passed as arguments."""

  for key, value in kwargs.items():

    print(key + ": " + str(value))

Calling the function with keyword arguments

print\_info(name="Alice", age=30, city="New York")

explain the different ways to define and call a function in python

Function definition with no arguments

def greet():

  """This function prints a greeting."""

  print("Hello!")

Calling the function

greet()

Function definition with arguments

def add\_numbers(x, y):

  """This function adds two numbers."""

  sum = x + y

  return sum

Calling the function with arguments

result = add\_numbers(5, 3)

print(result)

Function definition with default arguments

def greet\_person(name="World"):

  """This function greets a person with a default name."""

  print("Hello, " + name + "!")

Calling the function with and without arguments

greet\_person()  # Output: Hello, World!

greet\_person("Alice")  # Output: Hello, Alice!

Function definition with variable number of arguments

def sum\_all\_numbers(\*args):

  """This function sums all the numbers passed as arguments."""

  total = 0

  for num in args:

    total += num

  return total

Calling the function with multiple arguments

result = sum\_all\_numbers(1, 2, 3, 4, 5)

print(result)

Function definition with keyword arguments

def print\_info(\*\*kwargs):

  """This function prints the key-value pairs passed as arguments."""

  for key, value in kwargs.items():

    print(key + ": " + str(value))

Calling the function with keyword arguments

print\_info(name="Alice", age=30, city="New York")

 question 4-what is the purpose of return statement in a python function

The return statement is used to end the execution of a function and return a value to the caller.

If no return statement is used, the function returns None.

def add\_numbers(x, y):

  """This function adds two numbers and returns the result."""

  result = x + y

  return result

Call the function and store the returned value

sum = add\_numbers(5, 3)

print(sum)  # Output: 8

Queation 5 -what are iterators in python and how do they differ from iterables

Iterables are objects that can be iterated over, but they don't necessarily store the entire sequence in memory.

Examples: lists, tuples, strings, dictionaries, files

Iterators are objects that allow you to traverse through a sequence of values.

They have a special method called \_\_next\_\_() that returns the next item in the sequence.

you can create an iterator from an iterable using the iter() function.

my\_list = [1, 2, 3]  # This is an iterable

my\_iterator = iter(my\_list)  # This creates an iterator from the iterable

print(next(my\_iterator))  # Output: 1

print(next(my\_iterator))  # Output: 2

print(next(my\_iterator))  # Output: 3

Key difference:

Iterables are the objects that you can loop over.

Iterators are the objects that allow you to loop over an iterable.

Question 6- explain the concept of generators in python and how they are defined

Generators are functions that return an iterable object.

They use the `yield` keyword instead of `return`.

This allows them to produce a sequence of values over time,

rather than computing them all at once and storing them in memory.

def my\_generator(n):

  """This is a simple generator function."""

  for i in range(n):

    yield i

To use the generator, you can iterate over it:

for value in my\_generator(5):

  print(value)

Generators are memory-efficient because they produce values on demand.

They are useful for large datasets or infinite sequences.

Question 7-what are the advantages of using generators over regular functions

Generators are more memory-efficient than regular functions because they produce values one at a time,

instead of storing the entire result in memory. This is especially useful when dealing with large datasets.

Generators are lazy, meaning they only produce values when needed. This can improve performance,

especially when processing large datasets or when the result is not needed all at once.

Generators can be used to create infinite sequences, which is not possible with regular functions.

Generators can be used to create pipelines, where the output of one generator is fed as input to another.

Question 8-what is lambda function in python and when it is typically used

Lambda functions are small, anonymous functions defined with the 'lambda' keyword.

They are typically used when you need a simple function for a short period of time,

often as an argument to another function.

Example:

double = lambda x: x \* 2

print(double(5))  # Output: 10

Here, we define a lambda function called 'double' that takes one argument 'x'

and returns 'x \* 2'. This is a concise way to define a simple function

without the need for a full 'def' statement.

Common use cases:

- Sorting lists with custom keys:

data = [("apple", 3), ("banana", 1), ("cherry", 2)]

data.sort(key=lambda x: x[1])  # Sort based on the second element of each tuple

print(data)

- Applying functions to iterables:

numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x\*\*2, numbers))  # Square each number in the list

print(squared)

Question 9-explain the purpose and usage of map()function in python

The map() function applies a given function to each item of an iterable (like a list, tuple, etc.) and returns an iterator.

Example:

numbers = [1, 2, 3, 4, 5]

Define a function to square a number

def square(x):

  return x \* x

Use map() to apply the square function to each number in the list

squared\_numbers = map(square, numbers)

Convert the map object to a list

result = list(squared\_numbers)

print(result)  # Output: [1, 4, 9, 16, 25]

Question 10-  what is the difference between map() , reduce(), and filter() function in python

Map applies a function to all items in an input list.

Example: Double each number in a list

numbers = [1, 2, 3, 4, 5]

doubled\_numbers = list(map(lambda x: x \* 2, numbers))

print(doubled\_numbers)  # Output: [2, 4, 6, 8, 10]

Reduce applies a rolling computation to sequential pairs of values in a list.

Example: Calculate the product of all numbers in a list

from functools import reduce

numbers = [1, 2, 3, 4, 5]

product = reduce(lambda x, y: x \* y, numbers)

print(product)  # Output: 120

Filter creates a list of elements for which a function returns true.

Example: Filter out even numbers from a list

numbers = [1, 2, 3, 4, 5]

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers)  # Output: [2, 4]

Question 11- internal mechanism for sum operation using reduce function on the list [47,11,42,13]

from functools import reduce

numbers = [47, 11, 42, 13]

def sum\_two\_numbers(x, y):

  return x + y

total = reduce(sum\_two\_numbers, numbers)

print(total)