Assignment

Functions (Theory)

Question. 1) what is defference between a function and method in Python with example ?

Answer 1)

Aspect	Function	Method
Definition	An independent block of code that perfooms a specific task	A function associated within a class
Usage	Called directly using its name	Called on an instance of a class
Scope	Can exist independently of classes	Exist within the context of a class and its instances
Example	def greet (name): return F" Hello, {name}!"	class Person: def greet (self): return F" Hello, {self.name}!"

Question 2) Explain the concept of function arguments and parameters Python ?

1) Parameters are the variables listed inside the parentheses. They are placeholders that indicate what kind of input the function expects.

2) Arguments are the actual values you pass to the function when you call it. These values fill in the placeholders defined by the parameters of

Date

2) Function with default Parameters

Default parameters allow you to specify default values for parameters, making them optimal when calling the function

```
        def greet(name="Guest"):
            return F"Hello, {name}!"

    print (greet())          #Output Hello, Guest!
    print (greet("Ritik"))   #Output Hello, Ritik!
```

3) Function with Variable-length Arguments (*args)

This lets you pass a variable numbers of argument to function. These arguments are stored in a tuple.

```
    def greet(*name):
        return F"Hello, {', '.join(name)}!"
print (greet("Ritik", "Ankit", "Vishal")) ~~#Hello~~;
```

Output: Hello, ~~Alice~~ Ritik, Ankit, Vishal!

4) Function with Keyword Arguments (**Kwargs)

This allow you to pass a variable number of Keyword argument, These arguments are stored in a dictionary.

```
def greet(**details):
    return F"Hello, {details['name']}! ~~Age:~~
            ~~Age~~ Age: {details['age']}"

  Print (greet(name="Ritik", age=20))
```

Output: ~~Hel~~ Hello, Alice! ~~A~~ Age: 30

Parameters

```
        def greet (name, age): # 'name' & 'age' are
                                     Parameters
          return f'Hello, my name is {name} and
          I am {age} year old."
```

Function Call (~~Arg~~ Arguments)

~~In this example(s):~~
~~• name and age are parameters~~

~~Print Cog~~

```
Print (greet ("Ritik", 20)) # 'Ritik' and '30' are arguments
# output: Hello, my name is Ritik and I am 20 year old
```

• name and age are parameters
• "Ritik" and 30 are ~~ag~~ arguments passed to those parameters when the function is Called

Question 3) What are the different ways to define and call a function in Python?

1) Regular Function: This function is defined using the def Keyword followed by the Function name and parameters in parathesses

Example

```
            def greet (name):
                return f"Hello, {name}!"
        print (greet ("Ritik")) # Output: Hello,
                                          Ritik!
```

5.) Anonymous Function (Lambda Function)

A Lambda function is a small anonymous function defined with the Lambda Keyword it used for short & simple function.

```
great = Lambda name: F"Hello, {name}!"
print (great ("Alice")) # Output: Hello Alice!
```

Question 4.) what is the purpose of the 'return' statement in Python function?

The return statement in a python function serves to ~~exit~~ exit the function and pass back a value to the caller. It the way functions send back results to part of the program that called them.

use it.

Send back a value or result: After processing, a function can provide its result using 'return'.

· Exit the function: it terminates the function's execution ~~[illegible]~~ immediatly

· Make function reusable: Function can be used in different contexts and combined in complex operation if they return results.

```
def add (a, b):
    return a + b
return = add (3, 5)
Print (result) #= Output: 8.
```

Question5) what are iterators in Python and How do they differ from iterables?

Answers5) Iterators in Python are objects that allows you to traverse through all the elements of a Collection (like a list or tuple) one at a time. They maintain a state to keep track of where they are during iteration.

Iterables, on the other hand, are objects that can return an iterator. These includes Collections like lists, tuples and strings. You can use the iter() function to get an ~~it~~ iterator from an iterable.

Difference between

Aspet	Iterable	Iterator
Delination	Any object that can return an iterator	An object with a state that can iterate over elements
implemention	Implements the iter_() memod	Implements both the _iter_ and _next_() methods
Usage	Can be looped over direct by using a for loop.	Needs to be explicity fetched using next().
State	Does not maintain State	Maintains State during iteration

```
# Iternable
my_list = [1, 2, 3]
for item in my_list:
    print (item) # Outputs: 1,2,3
```

```
# Iterator
my_iterator = iter (my_list)
print (next (my_iterator)) # Output: 1
print (next (my_iterator)) #Output: 2
print (next (my_iterator)) #Output: 3
```

Question 6 Explain the Concept of generators in Python and how they are defined?

Answer 6) Generators are special type of iterator of iterator in Python, used to iterate over a sequence of values without creating the entire sequence in memory at once. This make them more memory-efficient for large datasets.

Concept: A generator function is defined using the def keyword, but instead of returning a single value, it yield a series of values using the yield keyword. Each call to generator function resumes where it left off and continues until the sequence is exhausted.

Defining a Generator:

```
def countdown (n):
    while n > 0:
        yield n
        n -= 1
```

In this example, countdown is generator that yield a sequence of numbers from n down to 1. Each time yield n is executed, the function's state is preserved, allowing it to resume from the same point on the next call.

Using a Generator

```
For number in countdown (5):
    Print (number)

# Output:
# 5
# 4
# 3
# 2
# 1
```

When you iterate over the generator using a for loop, it keeps yielding the next ~~[illegible]~~ value in the sequence until there are no more values left. Generators are particularly useful for handling large data streams or infinite sequence, as they compute values on-the-fly.

Question 7) what are advantage of using generators over regulars functions?

i) Memory Efficiency: Unlike regular functions, which might generate and store all values at once generators produce values one at a time, only when needed. This minimize memory usage.

Lazy Evaluation: Generators: Generators Compute values on-the-fly. Regular function would compute all value upfront, but generators only do the work when you ask for the next item. This can lead to performance gains.

Improved Readability: Generators often you ~~for~~ Code ~~eastly~~ easier to read and maintain. Writing an iterator manually requires more boilerplate code to track state, whereas generators handle this elegantly with the yield Keyword.

Handling infinite Sequences: Generators can model infinite sequences or very large datasets without hitting memory limits, making them highly suitable for streams of data or continuous process.

Example

```
    def count_up():
        n = 0
    While True:
        yield n
        n += 1

    Counter = Count_up()
  for _ in range(5):

  Print (next (Counter))
```

Output: 0, 1, 2, 3, 4

Question 8.) What is a lambda function in Python and When is it typically used?

A lambda function in Python is a small, anonymous function defined using the lambda Keyword. Unlike a regular function, it has no name and is typically used for short, simple operation that are needed temporily.

Syntax: lambda arguments : expression

Example

```
# Regular function
~~def (~~ def add (a,b):
    return a+b

# lambda function
add = lambda a,b : a+b

Print (add (3,4)) # Output 7
```

Use Lambda Function

1.) Short, Simple Function: They're ideal for simple operation that can be written in a single line.

2.) Function Programming: Often used with function like map(), filter(), and sorted() to provide quick, inline processing

3.) Sorting or Key Functions: Handy when you need a small function to serve as a key for sorting or other function.

Example with sorted ():

```
Pairs = [(1, 'one'), (2, 'two'), (3, 'three')]
sorted_pairs = sorted(pairs, key=lambda
                      pair: pair[1])

print(sorted_pairs)

# Output: [(1, 'one'), (3, 'three'), (2, 'two')]
```

Question 9) Explain the purpose and usage of the 'map()' function in Python?

The map () function in Python applies a given function to all the items in an iterable (like a list or tuple) and return a map object (an iterator) with the results. It's convenient way to transform items in an iterable without using a 'for' loop

Purpose

The main purpose of (map()) is to apply a specific function to each item of an iterable, which can simplify and make your code more readable. It's especially useful for ~~pe~~performing operation on large datasets or when you need to apply a common operation to all items in a collection

Usage

Syntax:

```
map(function, iterable)
```

Example let's say you want to square each no. in list :

```
# Regular function :
        def Square(x):
            returns x ** 2
    number = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)

print(list(squared_numbers))
  # output : [1, 4, 9, 16 25)
```

You can use a lambda function to make it even more concise:

```
  numbers = [1, 2, 3, 4, 5)
squared_numbers = map(lambda x: x ** 2, no.)

print(list(squared_number)) # (1, 4, 9, 16, 25)
```

Question 10.) What is the difference between map(), reduce(), and filter() function in Python?

Answers 10: All three of these function - map(), reduce(), and filters () - are the Python's built-in functional programming toolkit. They are each serve different purpose for the processing iterables. Here's a comparison:

[illegible]	[illegible]	Filter()	Reduce()
[illegible]	[illegible] function [illegible] each item [illegible] the results	Applies a function that returns a boolean [illegible] keeps items only those where the function returns True	Applies a function to elements cumulatively to reduce iterable to a single value.
[illegible]	map(function, iterable)	filter(function, iterable)	reduce(function, iterable)
[illegible]	A map object [illegible] iterator	A filter object (which is iterator)	A single value
[illegible]	map(lambda x: x**2, [1,2,3]) - Squares each no. in the list and returns [1,4,9]	filter(lambda x: x % 2 == 0, [1,2,3,4]) - Filters the list, keeping only ~~[illegible]~~ even no.: [2,4]	reduce(lambda x,y: x*y, [1,2,3,4]) - Multiples all ~~[illegible]~~ returning [24]

All these functions are tools that give you different ways to handle collection of data, with map() [illegible] transforming items, filter() a Selecting items, and ~~[illegible]~~ reduces() [illegible] Combining item.

Question Write the internal mechism for sum operation reduce function on this given.
~~[list~~

list'. [47, 11, 42, 13] ;

Using [reduce] to Sum the list [47, 11, 42, 13]

i) list: [47, 11, 42, 13]

2 Initial Call: The first two element are passed the function.

· 47 + 11 = 58

3 Next Call: The result from the previous Call and next element are passed.

58 + 42 = 100

4) Final Call: The result from the previous Call and the next element are passed.

100 + 13 = 113

The reduce function essentially performs the Cumulative Summing operation until the list is exhausted.

```
from functools import reduce
number = [47, 11, 42, 13]
Sum_result = reduce(lambda x, y : x+y, no.])
Print (Sum_result) # Output 113
```