Assignment No.3
Functions

Theory Questions:

1. What is the difference between a function and a method in Python?

1. Function

- Definition: A function is a block of reusable code that is not bound to any object.
- Defined with: def keyword at the top level or within another function.
- Called by: Using its name directly.

Example:

Input

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Jawan singh"))
```

Output

```
Hello, jawan singh!
```

Method

- Definition: A method is a function that is associated with an object (usually a class instance).
- Defined inside: A class.
- Called on: An object using dot (.) notation.

Example:

Input

```
class Greeter:
    def greet(self, name):
        return f"Hello, {name}!"
g = Greeter()
print(g.greet("Jawan singh"))
```

Output

```
Hello, jawan singh!
```

Main difference:

- Function = independent
- Method = tied to an object or class

2. Explain the concept of function arguments and parameters in Python.

2. Parameters

- Definition: Parameters are the variables that are listed in a function's definition.
- Purpose: They act as placeholders for the values (arguments) that will be passed to the function when it is called.

Example:

```
def greet(name):
    print("Hello,", name)
```

Arguments

- Definition: Arguments are the actual values you provide to the function when you call it.
- Purpose: They replace the parameters during the function call.

Example:
Input
```
greet("Jawan singh")
```

Output
```
Hello, Jawan singh
```

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

3. Defining Functions

- Using def – regular function

Example:

```
def greet(name):
    return f"Hello, {name}"
```

- Using lambda – one-liner anonymous function

Example:

```
greet = lambda name: f"Hello, {name}"
```

Calling Functions

- Regular call

Example:
Input
```
greet("Jawan singh")
```

Output
```
'Hello, Jawan singh'
```

- Lambda call

Example:
Input
```
(lambda x: x * 2)(5)
```

Output
```
10
```

- With unpacked arguments

Example:
Input
```
def add(a, b): return a + b
add(*[1, 2])
```

Output
```
3
```

4. What is the purpose of the `return` statement in a Python function?

4. The **return** statement in a Python function is used to **send a result back** to the caller. It ends the function and **outputs a value**, if provided.

Purpose of return:

- Ends the function execution.
- Sends data back to where the function was called.
- Allows the result to be stored or used elsewhere.

Example:

```
Input
def add(a, b):
    return a + b
result = add(3, 5)
print(result)

Output:
8
```

If you **don't use return**, the function returns None by default.

5. What are iterators in Python and how do they differ from iterables?

5. Iterable: An object you can loop over (like list, str).

```
Example:
Input:
for i in [1, 2, 3]: print(i)
Output:
1
2
3
```

Iterator: An object that gives one value at a time using next().
Made using iter().

```
Example:
Input:
it = iter([1, 2, 3])
print(next(it))
OutPut:
1
```

6. Explain the concept of generators in Python and how they are defined.

6. **Generators** are special functions that return an iterator, allowing you to generate values one at a time using the yield keyword.

How to define a generator:

```
Use yield instead of return in a function.
def count_up_to(n):
    i = 1
    while i <= n:
        yield i
        i += 1
How it works:
gen = count_up_to(3)
print(next(gen))
print(next(gen))
print(next(gen))

Output:
1
2
3
```

Why use generators?

- Saves memory (no need to store all items at once)
- Useful for large data or infinite sequences

7. What are the advantages of using generators over regular functions?

7. Advantages of Generators over Regular Functions

- ❖ Memory Efficient
  - ➢ Generates values one at a time — no need to store the whole list in memory.
- ❖ Faster with Large Data
  - ➢ Ideal for looping over large or infinite data sets.
- ❖ Lazy Evaluation
  - ➢ Values are created only when needed using `next()`.
- ❖ Simpler Code
  - ➢ Easier to write than using classes with `__iter__` and `__next__`.

Example:

```
def gen_numbers():
    for i in range(1000000):
        yield i
```

This uses less memory than returning a big list.

8. What is a lambda function in Python and when is it typically used?

8. A lambda function is a short, one-line function without a name.

Uses:

- For quick tasks
- Inside map(), filter(), or sorted()
- Use lambda when the function is simple and used only once.

Example:

```
add = lambda a, b: a + b
print(add(2, 3))
```

Output: 5

9. Explain the purpose and usage of the `map()` function in Python.

9.The `map()` function in Python applies a given function to each item of an iterable (like a list) and returns a map object (an iterator) with the results.

Example:

```
nums = [1, 2, 3]
squares = list(map(lambda x: x * x, nums))
print(squares)
```

Output: [1, 4, 9]

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

10. map():Changes each item using a function.
filter():Keeps only items that match a condition.
reduce():Combines all items into one value.
Example:

```
from functools import reduce
# map
print(list(map(lambda x: x * 2, [1, 2, 3])))  #Output [2, 4, 6]
# filter
print(list(filter(lambda x: x % 2 == 0, [1, 2, 3])))  #Output [2]
# reduce
print(reduce(lambda x, y: x + y, [1, 2, 3]))  #Output 6
```

11. Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:[47,11,42,13];
(Attach paper image for this answer) in doc or colab notebook.

Q11: Internal Mechanism for reduce() with Sum Operation

Given list: [47, 11, 42, 13]

We use Python's reduce() function as follows:

reduce(lambda x, y: x + y, [47, 11, 42, 13])

Step 1: $x = 47, y = 11 \rightarrow x + y = 58$

Step 2: $x = 58, y = 42 \rightarrow x + y = 100$

Finp 3: $x = 100, y = 13 \rightarrow x + y = 113$

Final Result: 113

Explanation:

The reduce() function applies the lambda function cumulatively to the items of the list, from left to right:

$$(((47 + 11) + 42) + 13) = 113$$

Practical Questions:

1. Write a Python function that takes a list of numbers as input and returns the sum of all even numbers in the list.

1.

```
def sum_of_even_numbers(numbers):
    # Using list comprehension to filter even numbers and then summing them
    return sum(num for num in numbers if num % 2 == 0)

# Example usage:
numbers = [1, 2, 3, 4, 5, 6]
result = sum_of_even_numbers(numbers)
print(result)

# Output: 12 (2 + 4 + 6)
```

Explanation:

- The function filters the even numbers using `num % 2 == 0`.
- Then, it uses the `sum()` function to add up all the even numbers in the list.

2. Create a Python function that accepts a string and returns the reverse of that string.

2.

```
def reverse_string(s):
    return s[::-1]

# Example usage:
input_string = "Hello, world!"
result = reverse_string(input_string)
print(result)

# Output: "!dlrow ,olleH"
```

Explanation:

- The `[::-1]` slicing technique reverses the string.
- The function then returns the reversed string.

3. Implement a Python function that takes a list of integers and returns a new list containing the squares of each number.

3.

```
def square_numbers(numbers):
    return [num ** 2 for num in numbers]

# Example usage:
input_list = [1, 2, 3, 4, 5]
result = square_numbers(input_list)
print(result)  # Output: [1, 4, 9, 16, 25]
```

Explanation:

- The function uses a list comprehension to iterate over each number in the list and squares it (`num ** 2`).
- The result is a new list containing the squares.

4. Write a Python function that checks if a given number is prime or not from 1 to 200.

4.

```
def is_prime(n):
    if n <= 1:
        return False  # 0 and 1 are not prime
    for i in range(2, int(n**0.5) + 1):  # Check up to the square root of n
        if n % i == 0:
            return False  # n is divisible by i, so it's not prime
    return True

# Checking prime numbers from 1 to 200
primes = [num for num in range(1, 201) if is_prime(num)]
print(primes)
# Output [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
```

Explanation:

- The function `is_prime()` checks if a number `n` is prime by testing divisibility from `2` to the square root of `n`.
- It returns `True` if the number is prime, otherwise `False`.
- We then generate a list of prime numbers between 1 and 200 using a list comprehension.

5. Create an iterator class in Python that generates the Fibonacci sequence up to a specified number of terms.

5.

```
class FibonacciIterator:
    def __init__(self, terms):
        self.terms = terms
        self.current = 0
        self.previous = 0
        self.next_term = 1
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count < self.terms:
            if self.count == 0:
                self.count += 1
                return 0
            elif self.count == 1:
                self.count += 1
                return 1
            else:
```

```
                fib_num = self.previous + self.next_term
                self.previous, self.next_term = self.next_term, fib_num
                self.count += 1
                return fib_num
        else:
            raise StopIteration

# Example usage:
fibonacci = FibonacciIterator(10)  # Generate first 10 Fibonacci numbers
for num in fibonacci:
    print(num)

# Output

0
1
1
2
3
5
8
13
21
34
```

Explanation:

1. **__init__(self, terms)**: Initializes the iterator with the number of terms (**terms**) you want to generate.

2. **__iter__(self)**: Returns the iterator object itself.

3. **__next__(self)**: Generates the next Fibonacci number:

    - Starts with 0 and 1.
    - For each subsequent term, it adds the previous two terms to get the next number in the sequence.
    - Stops when the specified number of terms (**self.terms**) is reached, raising a **StopIteration** exception when done.

6. Write a generator function in Python that yields the powers of 2 up to a given exponent.

```
6.
def powers_of_two(exponent):
    yield from (2 ** i for i in range(exponent + 1))

# Example usage:
for power in powers_of_two(5):
    print(power)

# Output
1
2
4
8
16
32
```

Explanation:

- The `yield from` statement is used to yield each value from the generator expression `(2 ** i for i in range(exponent + 1))`, which generates powers of 2.
- This approach makes the function more concise and efficient.

7. Implement a generator function that reads a file line by line and yields each

7.
line as a string.

```
def read_file_lines(file_path):
    with open(file_path, 'r') as file:
        for line in file:
            yield line.strip()  # strip() removes any leading/trailing whitespace

# Example usage:
file_path = 'example.txt'  # Replace with your file path
for line in read_file_lines(file_path):
    print(line)
```

Explanation:

1. `open(file_path, 'r')`: Opens the file in read mode.
2. `for line in file`: Iterates over the file object, which gives each line.
3. `yield line.strip()`: Yields each line one by one after stripping any leading/trailing whitespace.

8. Use a lambda function in Python to sort a list of tuples based on the second element of each tuple.

8.

```
# List of tuples
tuples = [(1, 3), (4, 1), (2, 2), (5, 0)]

# Sort by the second element of each tuple
sorted_tuples = sorted(tuples, key=lambda x: x[1])

# Print the sorted list
print(sorted_tuples)

# Output [(5, 0), (4, 1), (2, 2), (1, 3)]
```

Explanation:

- `lambda x: x[1]`: This is a lambda function that returns the second element (`x[1]`) of each tuple.
- `sorted(tuples, key=lambda x: x[1])`: This sorts the list of tuples based on the second element.

9. Write a Python program that uses `map()` to convert a list of temperatures from Celsius to Fahrenheit.

9.

```
# List of temperatures in Celsius
celsius_temperatures = [0, 20, 25, 30, 35]

# Function to convert Celsius to Fahrenheit
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

# Use map() to apply the conversion function to each temperature
fahrenheit_temperatures = list(map(celsius_to_fahrenheit, celsius_temperatures))

# Print the result
print(fahrenheit_temperatures)

# Output [32.0, 68.0, 77.0, 86.0, 95.0]
```

Explanation:

- The `map()` function applies the `celsius_to_fahrenheit` function to each element in the `celsius_temperatures` list.
- The result of `map()` is converted into a list using `list()`.
- The conversion formula is: `Fahrenheit = (Celsius * 9/5) + 32`.

10. Create a Python program that uses `filter()` to remove all the vowels from a given string.

10.

```
# Function to check if a character is a vowel
def is_not_vowel(char):
    return char.lower() not in 'aeiou'

# Given string
input_string = "Hello, World!"

# Use filter() to remove vowels
filtered_string = ''.join(filter(is_not_vowel, input_string))

# Print the result
print(filtered_string)

#Output "Hll, Wrld!"
```

Explanation:

- `is_not_vowel(char)`: This function returns `True` if the character is not a vowel (it checks both uppercase and lowercase vowels).
- `filter(is_not_vowel, input_string)`: Filters out the vowels from the string using the `is_not_vowel` function.
- `''.join()`: Joins the remaining characters into a new string.
- 11) Imagine an accounting routine used in a book shop. It works on a list with sublists, which look like this:

Order Number	Book Title and Author	Quantity	Price per Item

34587	Learning Python, Mark Lutz	4	40.95
98762	Programming Python, Mark Lutz	5	56.80
77226	Head First Python, Paul Barry	3	32.95
88112	Einführung in Python 3, Bernd Klein	3	24.99

Write a Python program, which returns a list with 2-tuples. Each tuple consists of the order number and the product of the price per item and the quantity. The product should be increased by 10,- € if the value of the order is smaller than 100,00 €.

Write a Python program using lambda and map.

11.

```
# List of orders (Order Number, Book Title and Author, Quantity, Price per Item)
orders = [
    [34587, "Learning Python, Mark Lutz", 4, 40.95],
    [98762, "Programming Python, Mark Lutz", 5, 56.80],
    [77226, "Head First Python, Paul Barry", 3, 32.95],
    [88112, "Einführung in Python 3, Bernd Klein", 3, 24.99]
]

# Function to calculate order value and apply the bonus if value is smaller than 100.00 €
def calculate_order_total(order):
    order_number, title, quantity, price = order
    total = quantity * price
    if total < 100.00:
        total += 10  # Adding 10€ if the total is smaller than 100
    return (order_number, total)

# Using map() and lambda to apply the calculation on each order
order_totals = list(map(lambda order: calculate_order_total(order), orders))

# Print the result
print(order_totals)

# Output [(34587, 163.8), (98762, 284.0), (77226, 113.84999999999999), (88112, 84.97)]
```

Explanation:

1. List of Orders: The **orders** list contains sublists with order details such as order number, book title, quantity, and price.

2. **calculate_order_total()**: This function computes the total price for an order. It multiplies the quantity by the price per item, and if the total is less than 100, it adds a 10€ bonus.

3. **map()**: We use the **map()** function to apply the **calculate_order_total** function to each sublist (order) in the **orders** list.

4. **lambda**: In this case, the lambda function simply passes each **order** to the **calculate_order_total()** function.