Function

1. What is the difference between a function and a method in Python? Answer: In Python, a function is a block of organized, reusable code that is used to perform a single, related action. It is defined outside of a class. A method, on the other hand, is a function that belongs to an object or a class and is defined within a class. Methods implicitly receive the instance of the class (or the class itself) as their first argument (conventionally named self for instance methods and cls for class methods).
2. Explain the concept of function arguments and parameters in Python. Answer: Parameters are the names listed in the function definition, serving as placeholders for the values the function expects to receive. Arguments are the actual values passed to the function when it is called.
3. What are the different ways to define and call a function in Python? Answer: Defining a function: Functions are defined using the def keyword, followed by the function name, parentheses for parameters, and a colon. The function body is indented. Calling a function: Functions are called by using their name followed by parentheses containing the arguments. Positional arguments: Arguments are passed in the order Keyword arguments: Arguments are passed by explicitly naming the parameter.
4. What is the purpose of the 'return' statement in a Python function? Answer: The return statement in a Python function is used to exit the function and send a value back to the caller. If no return statement is present, or if return is used without an expression, the function implicitly returns None. Example:
5. What are iterators in Python and how do they differ from iterables? Answer: An iterable is an object that can be iterated over (e.g., lists, tuples, strings, dictionaries). It has an **iter**() method that returns an iterator. An iterator is an object that represents a stream of data and provides the **next**() method, which returns the next item in the sequence. When there are no more items, it raises a StopIteration exception. Difference: An iterable is something you can loop over, while an iterator is the object that actually performs the iteration (keeping track of the current position and providing the next item). You can get an iterator from an iterable using iter().
6. Explain the concept of generators in Python and how they are defined. Answer: Generators are a special type of iterator in Python that allow you to declare a function that behaves like an iterator, i.e., it can be iterated over. They are memory-efficient because they generate values on the fly, one at a time, instead of storing all values in memory simultaneously. Definition: Generators are defined like regular functions but use the yield keyword instead of return to produce a sequence of results. Each time yield is encountered, the state of the generator is saved, and the yielded value is returned. When the generator is called again, execution resumes from where it left off.
7. What are the advantages of using generators over regular functions? Answer: Memory Efficiency: Generators produce items one at a time and only when requested, making them ideal for handling large datasets where storing all data in memory would be unfeasible. Performance: They can be more efficient for certain tasks as they don't build the entire sequence in memory upfront. Lazy Evaluation: Values are computed only when needed, which can save computation time if not all values are ultimately used. Infinite Sequences: Generators can represent infinite sequences as they don't need to store the entire sequence. Clean Code: They often lead to more readable and concise code for implementing iterators.
8. What is a lambda function in Python and when is it typically used? Answer: A lambda function (also called an anonymous function) in Python is a small, single-expression function that doesn't have a name. It is defined using the lambda keyword. Typical Usage: Lambda functions are typically used for short, simple operations where a full function definition with def would be overly verbose. They are commonly used as arguments to higher-order functions like map(), filter(), sorted(), or reduce(), where a function is expected as an argument.
9. Explain the purpose and usage of the map() function in Python. Answer: The map() function in Python applies a given function to each item in an iterable (like a list or tuple) and returns a map object (an iterator) that yields the results. Usage: The syntax is map(function, iterable, ...). The function is applied to each item of the iterable.
10. What is the difference between 'map()', 'reduce()', and 'filter()' functions in Python? Answer: map(function, iterable): Applies a function to every item of an iterable and returns a new iterable (map object) containing the results. It transforms each item individually. filter(function, iterable): Constructs an iterator from elements of an iterable for which a function returns True. It selects items based on a condition. reduce(function, iterable, initializer) (from functools module): Applies a function of two arguments cumulatively to the items of an iterable, from left to right, so as to reduce the iterable to a single value.
11. Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:; Answer: Mechanism of reduce() for sum operation on The reduce() function, with a sum operation (e.g., lambda x, y: x + y), iteratively applies the function to pairs of elements in the list. Step 1: Initial call The reduce() function takes the first two elements of the list, 47 and 11, and applies the sum function. Result: 47 + 11 = 58 Step 2: Second iteration The result from Step 1 (58) is taken as the first argument, and the next element from the list, 42, is taken as the second argument. Result: 58 + 42 = 100 Step 3: Third iteration The result from Step 2 (100) is taken as the first argument, and the next (and final) element from the list, 13, is taken as the second argument. Result: 100 + 13 = 113 Answer: The final result of the sum operation using reduce() on the list [47, 11, 42, 13] is 113. he parameters.

[ ]

```
1. Sum of all even numbers

def sum_even(numbers):
    return sum(n for n in numbers if n % 2 == 0)

    print(sum_even([1,2,3,4,5,6]))  # 12

    2. Reverse a string

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

        print(reverse_string("hello"))  # olleh

        3. Squares of list elements

        def square_list(lst):
            return [x**2 for x in lst]

            print(square_list([1,2,3]))  # [1,4,9]

            4. Check prime (1-200)

            def is_prime(n):
                if n < 2: return False
                    for i in range(2, int(n**0.5)+1):
                            if n % i == 0: return False
                                return True

                                print([x for x in range(1,201) if is_prime(x)])

                                5. Fibonacci iterator

                                class Fibonacci:
                                    def __init__(self, n): self.n, self.a, self.b, self.count = n
                                        def __iter__(self): return self
                                            def __next__(self):
                                                    if self.count >= self.n: raise StopIteration
                                                            self.a, self.b, self.count = self.b,
                                                                    return self.a

                                                                    print(list(Fibonacci(10
```

```
t(Fibonacci(10
        6. Generator powers of 2

        def powers_of_two(exp):
            for i in range(exp+1):
                    yield 2**i

                    print(list(powers_of_two(5)))  # [1,2,4,8,16,32]


        7. File line generator

        def read_file(filename):
            with open(filename) as f:
                    for line in f:
                                yield line.stri

        8. Sort tuples by 2nd element

        tuples = [(1,3),(2,1),(3,2)]
        tuples.sort(key=lambda x: x[1])
        print(tuples)  # [(2,1),(3,2),(1,3)]

        9. Celsius → Fahrenheit using map

        celsius = [0,10,20,30]
        fahrenheit = list(map(lambda c: (c*9/5)+32, celsius))
        print(fahrenheit)  # [32,50,68,86]

        10. Remove vowels with filter

        s = "Hello World"
        result = "".join(filter(lambda x: x.lower() not in "aeiou", s))
        print(result)  # Hll Wrld

        11. Accounting routine (lambda + map)

        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 Python3, Bernd Klein", 3, 24.99]
                        ]

                        result = list(map(lambda x: (x[0], x[2]*x[3] + (10 if x[2]
                        print(result)
                        # [(34587, 173.8), (98762, 284.0), (77226, 108.85), (88112
```