# **Assignment of Functions (Module 5)

## Theory Questions

### Q1. What is the difference between a function and a method in Python?

Ans-> In Python, a function is a reusable block of code that performs a specific task, while a method is a function associated with an object or class, designed to operate on the data within that object.

- Example: def add(x, y): return x + y

### Q2. Explain the concept of function arguments and parameters in Python?

Ans-> In Python, parameters are placeholders defined in a function's definition, while arguments are the actual values passed to the function when it's called.

- Parameters:
- Parameters are variables listed inside the parentheses in a function definition.
- They act as placeholders for the values that will be passed into the function when it's called.
- For example, in the function definition def greet(name, greeting):, name and greeting are parameters.
    - Arguments:
    - Arguments are the values that are actually passed to the function when it's called.
    - They are used to provide the values for the parameters defined in the function.
    - For example, in the function call greet("Alice", "Hello"), "Alice" and "Hello" are arguments.

### Q3. What are the different ways to define and call a function in Python?

Ans-> In Python, you define a function using the def keyword, followed by the function name, parentheses for parameters, a colon, and an indented code block. You call a function by its name followed by parentheses, optionally including arguments.

Defining a Function:

- Keyword: Use def to signal the start of a function definition. Function Name: Choose a descriptive name for your function.
- Parameters: Placeholders for values that will be passed into the function when it's called, enclosed in parentheses.
- Colon: End the function definition line with a colon.
- Indentation: The code within the function (the function body) must be indented.
- Example: def greet(name): # Define a function named 'greet' with a parameter 'name' print(f"Hello, {name}!") # Function body (indented)

  Calling a Function:
- Function Name: Use the name of the function you want to execute.
- Parentheses: Follow the function name with parentheses.
- Arguments: If the function takes parameters, provide the values (arguments) to be passed to the function, separated by commas within the parentheses.

  Example: greet("Alice") # Call the 'greet' function with the argument "Alice"

### Q4. What is the purpose of the 'return' statement in Python function?

Ans-> The return statement in Python functions is used to send a value back to the caller, terminating the function's execution and making the returned value available for use elsewhere in the program.

Example: def add_numbers(a, b): """ This function adds two numbers and returns the result. """ sum = a + b return sum # Return the calculated sum

## Call the function and store the returned value

result = add_numbers(5, 3) print(result) # Output: 8

### Q5. What are iterators in Python and how do they differ from iterables?

Ans-> Iterable is an object, that one can iterate over. It generates an Iterator when passed to iter() method. An iterator is an object, which is used to iterate over an iterable object using the **next**() method. Iterators have the **next**() method, which returns the next item of the object.

Note: Every iterator is also an iterable, but not every iterable is an iterator in Python.

```
# code
s="GFG"
s=iter(s)
print(s)
print(next(s))
print(next(s))
print(next(s))

# Function to check object
```

## is iterable or not

def it(ob): try: iter(ob) return True except TypeError: return False

## Driver Code

for i in [34, [4, 5], (4, 5), {"a":4}, "dfsdf", 4.5]:

```
print(i,"is iterable :",it(i))
```

### Q6. Explain the concept of generators in Python and how they are defined?

Ans-> A generator function is a special type of function that returns an iterator object. Instead of using return to send back a single value, generator functions use yield to produce a series of results over time. This allows the function to generate values and pause its execution after each yield, maintaining its state between iterations.

def fun(max): cnt = 1 while cnt <= max: yield cnt cnt += 1

ctr = fun(5) for n in ctr: print(n)

```
* Explanation: This generator function fun yields numbers from 1 up to a specified max. Each call to next() on the generator object resumes exe
```

### Q7. What are the advantages of using generators over regular functions?

Ans-> Generators offer advantages over regular functions, particularly for memory efficiency and lazy evaluation, allowing them to handle large or infinite sequences without consuming excessive memory, and simplifying code for iterators and asynchronous workflows.

** Here's a some of the key advantages:

- Memory Efficiency: Generators produce values on-the-fly, meaning they don't store the entire sequence in memory at once, which is crucial for large or infinite datasets.
- Lazy Evaluation: Generators evaluate expressions only when needed, unlike regular functions that execute all code at once.
- Simplified Iterators: Generators provide a concise and readable way to create custom iterators, eliminating the need for explicit state management.
- Code Simplicity: Generators often lead to more readable and maintainable code, especially when dealing with complex iteration logic or asynchronous processes.
- Asynchronous Workflows: Generators can simplify asynchronous workflows by pausing and resuming execution, making it easier to handle events and manage state.

### Q8. What is a lamda function in Python and when is it typically used?

Ans-> In Python, a lambda function is a small, anonymous function defined using the lambda keyword, typically used for short, simple operations where a full function definition would be unnecessary, often as arguments to other functions. Here's a breakdown:

- Anonymous: Lambda functions don't have a name, unlike functions defined with def.
- Single Expression: They can only contain a single expression, which is automatically returned.
- Syntax: lambda arguments: expression.
- Common Use Cases: With Higher-Order Functions: map(), filter(), and sorted().
- Example:

## Regular function

def square(x):

```
 return x * x
```

## Lambda function (equivalent)

square_lambda = lambda x: x * x

print(square(5)) # Output: 25 print(square_lambda(5)) # Output: 25

## Q9. Explain the purpose and usage of the map () function in Python?

Ans-> The Python map() function applies a given function to each item in an iterable (like a list or tuple) and returns an iterator that yields the transformed results. Here's a more detailed explanation: Purpose:

- Transformation: The primary purpose of map() is to transform data by applying the same operation (defined by a function) to each element within an iterable.
- Concise Code: It allows you to process and transform elements in an iterable without needing explicit for loops, making your code more concise and readable.
- Lazy Evaluation: map() returns an iterator, not a list directly. This means the transformation is performed only when you iterate over the resulting iterator.

  Syntax: map(function, iterable, [iterable2, iterable3, ...])

## Q10. What is the difference between map(), reduce(), and filter() function in Python?

Ans-> In Python, map() applies a function to each item in an iterable, filter() selects elements based on a condition, and reduce() (from functools) cumulatively applies a function to pairs of elements, reducing the iterable to a single value. Here's a more detailed explanation:

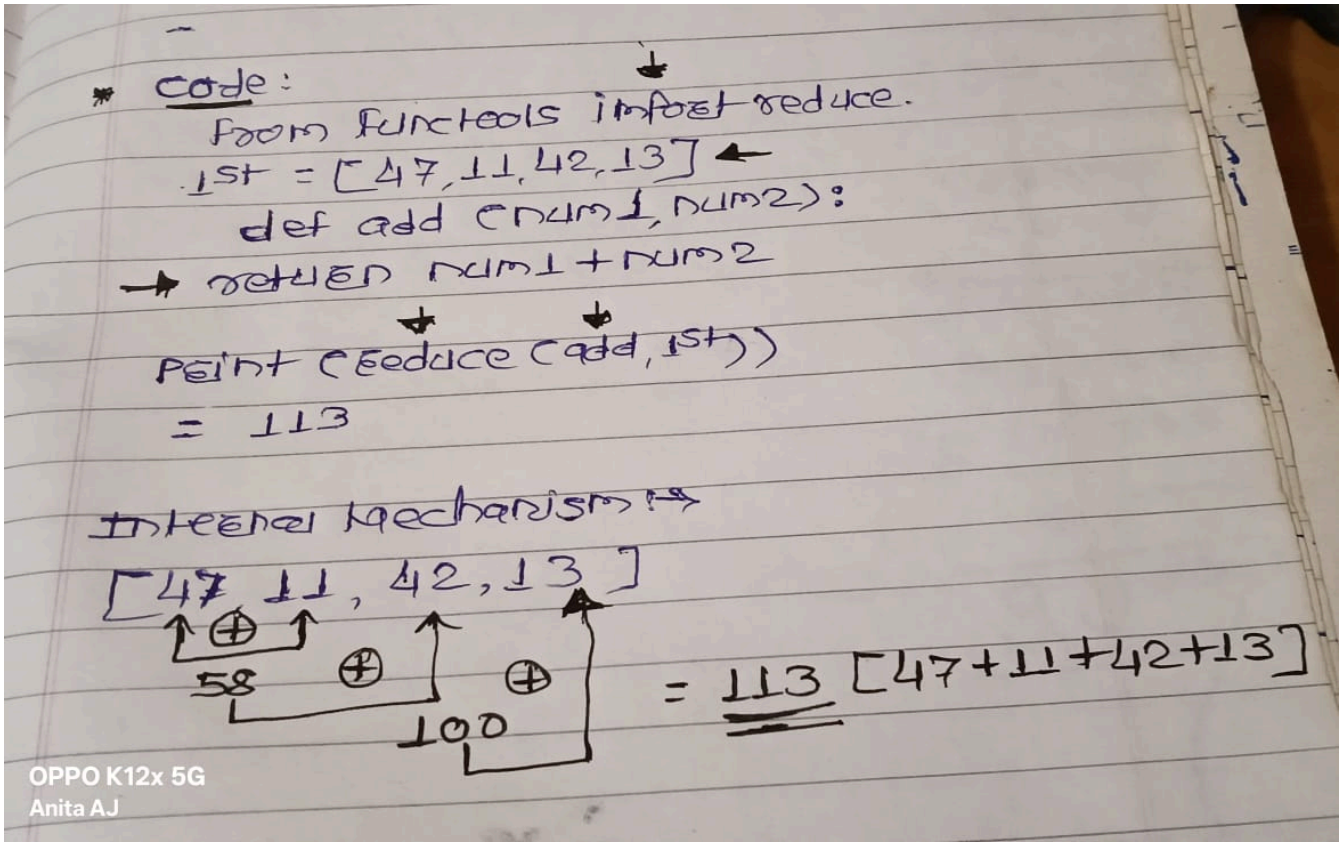
- map(): Applies a given function to each item in an iterable (like a list, tuple, or string). Returns a new iterable (usually a list) containing the transformed elements. Example: map(lambda x: x * 2, [1, 2, 3]) would return [2, 4, 6].
- filter(): Filters elements from an iterable based on a condition (a function that returns True or False). Returns a new iterable containing only the elements that satisfy the condition.
- Example: filter(lambda x: x % 2 == 0, [1, 2, 3, 4]) would return [2, 4]. reduce() (from functools): Applies a function cumulatively to pairs of elements in an iterable, reducing it to a single value. Requires importing from the functools module: from functools import reduce. Example: reduce(lambda x, y: x + y, [1, 2, 3]) would return 6 (1 + 2 + 3).

## Q11. 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?

```
from functools import reduce
lst = [47,11,42,13]
def add (num1, num2):
  return num1 + num2

print(reduce(add,lst))
```

```
113
```




## Practical Questions

Q1. Write a Python function that takes a list of numbers as input and returns the sum of all even numbers in list?

```
# prompt: list of numbers as input and returns the sum of an even numbers

from functools import reduce

def sum_even_numbers(numbers):
  """
  This function takes a list of numbers and returns the sum of all even numbers.
  """
  even_numbers = [num for num in numbers if num % 2 == 0]
  return reduce(lambda x, y: x + y, even_numbers, 0)

# Example usage
numbers = [1, 2, 3, 4, 5, 6]
sum_of_evens = sum_even_numbers(numbers)
print("Sum of even numbers:", sum_of_evens)
```

```
Sum of even numbers: 12
```

Q2. Create a Python function that accepts a string and returns the reverse of that string?

```
s = "GanpatiPule"
rev = s[::-1]
print(rev)
```

```
eluPitapnaG
```

Q3. Implement a Python function that takes a list of integers and returns a new list containing the squares of each number?

```
numbers = [1,2,3,4,5]
squared = map(lambda x: x*x, numbers)
```

```
print(list(squared))
```

```
[1, 4, 9, 16, 25]
```

## Q4. Write a Python function that checks if a given number is prime or not from 1 to 200?

```
# prompt: python function given number is prime or not from 1 to 200

def is_prime(number):
  """
  This function checks if a given number is prime or not.
  """
  if number <= 1:
    return False
  for i in range(2, int(number**0.5) + 1):
    if number % i == 0:
      return False
  return True

# Check prime numbers from 1 to 200
for i in range(1, 201):
  if is_prime(i):
    print(i, "is a prime number")
```

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
53 is a prime number
59 is a prime number
61 is a prime number
67 is a prime number
71 is a prime number
73 is a prime number
79 is a prime number
83 is a prime number
89 is a prime number
97 is a prime number
101 is a prime number
103 is a prime number
107 is a prime number
109 is a prime number
113 is a prime number
127 is a prime number
131 is a prime number
137 is a prime number
139 is a prime number
149 is a prime number
151 is a prime number
157 is a prime number
163 is a prime number
167 is a prime number
173 is a prime number
179 is a prime number
181 is a prime number
191 is a prime number
193 is a prime number
197 is a prime number
199 is a prime number
```

## Q5. Create an iterator class in Python that generates the Fibonacci sequence up to a specified number of terms?

```
fib = lambda n: n if n <=1 else fib(n-1) + fib(n-2)
[fib(i) for i in range(10)]
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Start coding or generate with AI.

## Q6. Write a generator function in Python that yields the powers of 2 up to a given exponent?

```
# prompt: generator function in python that yields the powers of 2

def powers_of_2(exponent):
  """
  This generator function yields the powers of 2 up to a given exponent.
  """
  power = 0
  while power <= exponent:
    yield 2 ** power
    power += 1

# Example usage
for power in powers_of_2(5):
    power # Indented this line to include it in the loop's execution.


print (powers_of_2(5))
```

```
<generator object powers_of_2 at 0x7cee1046bb90>
```

## Q7. Implement a generator function that reads a file line by line and yields each line as string?

```
# prompt: generator function that reads a file line by line and yields each line as string

def read_file_line_by_line(file_path):
  """
  This generator function reads a file line by line and yields each line as a string.
  """
  with open(file_path, 'r') as file:
    for line in file:
      yield line

# Example usage
# for line in read_file_line_by_line('your_file.txt'):
#     print(line)



def square_number_generators (n):
    for i in range (n):
        yield i**2


square_number_generators
```

```
square_number_generators
def square_number_generators(n)

<no docstring>
```

```
gen = square_number_generators(10)
gen
```

```
<generator object square_number_generators at 0x7cedd846e260>
```

```
next(gen)
```

```
0
```

```
next(gen)
```

```
1
```

```
next(gen)
```

```
4
```

```
next(gen)
```

```
9
```

```
next(gen)
```

```
16
```

```
next(gen)
```

```
25
```

## Q8. use a lambda function in Python to sort a list of tuples based on the second element of each tuple?

```
# input list of tuples
a = [(1, 'Python'), (2, 'with'), (3, 'PWskills')]

# define the specific order
order = ['with', 'Python', 'PWskills']

# sort the list of tuples based on the specific order
res = sorted(a, key=lambda x: order.index(x[1]))
print(res)
```

```
[(2, 'with'), (1, 'Python'), (3, 'PWskills')]
```

## Q9. Write a Python program uses 'map()' to convert a list of temeratures from celcius to Fahrenit?

```
# Temperature in celsius degree
celsius = 47

# Converting the temperature to
# fehrenheit using the formula
fahrenheit = (celsius * 1.8) + 32

# printing the result
print('%.2f Celsius is equivalent to: %.2f Fahrenheit'
      % (celsius, fahrenheit))
```

```
47.00 Celsius is equivalent to: 116.60 Fahrenheit
```

## Q10. Create a Python program uses 'filter()' to remove all the vowels from a given string?

```
# prompt: uses filter to remove all the vowels for a given string

def remove_vowels(string):
  """
  This function removes all vowels from a given string using filter.
  """
  vowels = "aeiouAEIOU"
  return "".join(filter(lambda char: char not in vowels, string))

# Example usage
string = "Hello, World! welcome to 'PWskills'"
string_without_vowels = remove_vowels(string)
print("String without vowels:", string_without_vowels)
```

```
String without vowels: Hll, Wrld! wlcm t 'PWsklls
```

## Q11. Imagine an accounting routine used in a book shop. It works on a list with sublists, which look like this:

order num 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 Einfuhrung in Python3, 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, -e if the value of the order is smaller than 100,00 e.

Write a Python program using lambda and map.?

```
# prompt: Imagine an accounting routine used in a book shop. It works on a list with sublists, which look like this:
#   order num  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    Einfuhrung in Python3, 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
#   Write a Python program using lambda and 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, "Einfuhrung in Python3, Bernd Klein", 3, 24.99],
]


def calculate_total_with_discount(order):
    order_number = order[0]
    quantity = order[2]
    price_per_item = order[3]
    total = quantity * price_per_item
    if total < 100:
        total += 10
    return (order_number, total)


result = list(map(calculate_total_with_discount, orders))
result
```

```
[(34587, 163.8), (98762, 284.0), (77226, 108.85000000000001), (88112, 84.97)]
```

```
order = [[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]
        ]

#with using lambda and map
print("Order Summary: ",list(map(lambda x: (x[0],x[-1]) if x[-1]*x[-2] > 100 else x[-1]*x[-2]+10, order)))
```

```
Order Summary:  [(34587, 40.95), (98762, 56.8), 108.85000000000001, 84.97]
```

Start coding or generate with AI.