functions

July 15, 2024

[ ]:
```
#Q - 1 -  What is the difference between a function and a method in Python?

#Answer
#Function -1--- A set of instructions that perform a task.
#--2--Functions are independent blocks of code that can be called from anywhere.
#--3--Use the keyword def to declare the function and follow this up with the␣
 ↪function name.

#Example-
def add(a,b):
    print(a+b)
add(2,3)
```

5

[ ]:
```
#Method - 1---A set of instructions that are associated with an object.
#---2---Methods aThey are called using dot notation, with the object name␣
 ↪followed by a period and the method namere tied to objects or classes and␣
 ↪need an object or class instance to be invoked.
#---3---They are called using dot notation, with the object name followed by a␣
 ↪period and the method name.

# Example-
#count() #These methods count the elements.
#extend()#Adds each element of an iterable to the end of the List
#index()        #Returns the lowest index where the element appears.
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

1

[ ]:
```
# Q-2 --- Explain the concept of function arguments and parameters in Python.

#Answer
# A parameter is the variable listed inside the parentheses in the function␣
 ↪definition. An argument is the value that is sent to the function when it is␣
 ↪called.
```

```
#Function parameters are the names listed in the function's definition.␣
 ↪Function arguments are the real values passed to the function. Parameters␣
 ↪are initialized to the values of the arguments supplied.

#Example
# Here a,b are the parameters
def sum(a,b):
  print(a+b)
# Here the values 1,2 are arguments
sum(1,2)
```

3

[ ]:
```
#Q-3---What are the different ways to define and call a function in Python?

# Answer
#To call a built-in Python function, simply use the function name followed by␣
 ↪parentheses and any required arguments. For example, to call the print()␣
 ↪function, we would write print("Hello, world!").
def greet(name):
    print("Hello, " + name + "! How are you?")
#In this example, we define a function called greet that takes one argument␣
 ↪called name. The function then prints out a greeting message to the console␣
 ↪that includes the name argument.

#To call a function in Python, you simply type the name of the function␣
 ↪followed by parentheses (). If the function takes any arguments, they are␣
 ↪included within the parentheses.
greet("John")
#In this example, we call the greet function with the argument "John".
```

Hello, John! How are you?

[ ]:
```
# Example -2 -
# default arguments
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)
myFun(2)
```

x: 2
y: 50

[ ]:
```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)
```

```
# Keyword arguments
student(firstname='Ashu', lastname='Mishra')
student(lastname='Mishra', firstname='Ashu')
```

```
Ashu Mishra
Ashu Mishra
```

[ ]:
```
# *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print(arg,end=" ")


myFun('Hello', 'Welcome', 'to', 'office')
```

```
Hello Welcome to office
```

[ ]:
```
#-Q-4--What is the purpose of the `return` statement in a Python function?
# Answer
#Return statement marks the end of a function and specifies the value or values␣
 ↪to pass back from the function. Return statements can return data of any␣
 ↪type, including integers, floats, strings, lists, dictionaries, and even␣
 ↪other functions.
#A return statement is overall used to invoke a function so that the passed␣
 ↪statements can be executed.
#Example
def cube(x):
   r=x**3
   return r
cube(4)
```

[ ]:
```
64
```

[ ]:
```
#-Q--5--What are iterators in Python and how do they differ from iterables?
#Answer
#An iterator is an object that allows you to iterate over collections of data,␣
 ↪such as lists, tuples, dictionaries, and sets. Python iterators implement␣
 ↪the iterator design pattern, which allows you to traverse a container and␣
 ↪access its elements.

#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.
#Examples--
```

```
# list of cities
cities = ["Berlin", "Vienna", "Zurich"]

# initialize the object
iterator_obj = iter(cities)

print(next(iterator_obj))
print(next(iterator_obj))
print(next(iterator_obj))
```

```
Berlin
Vienna
Zurich
```

[24]:
```
#--Q-6- Explain the concept of generators in Python and how they are defined.
#Answer
#A generator function in Python is defined like a normal function, but whenever␣
 ↪it needs to generate a value, it does so with the yield keyword rather than␣
 ↪return. If the body of a def contains yield, the function automatically␣
 ↪becomes a Python generator function.
#A Python generator function allows you to declare a function that behaves like␣
 ↪an iterator, providing a faster and easier way to create iterators.
#To define

def function_name():
   yield statement
#fibonacci series
def fib(x):
    a=0
    b=1
    for i in range(x):
        yield a
        a,b=b,a+b
x=fib(5)
print(next(x))
print(next(x))
print(next(x))
print(next(x))
```

```
0
1
1
2
```

[1]:
```
#--Q--6- What are the advantages of using generators over regular functions?
#Answer
#Unlike a regular function, a generator does not return its results all at once.
 ↪ Instead, it yields its values one by one, each time it is called. This␣
 ↪makes it possible to generate an infinite sequence of values, as long as␣
 ↪there is sufficient memory to store them
#The generator function contains one or more yield statements instead of a␣
 ↪return statement.
#As the methods like _next_() and _iter_() are implemented automatically, we␣
 ↪can iterate through the items using next().
#Example
#generators function
def function_name():
   yield statement#gives resukts only one at a time using next()
#regular function
def function_name():
    return statement#gives all at a time leads to memory consumption and larger␣
 ↪value takes time to execute.
```

[5]:
```
#--Q--8-What is a lambda function in Python and when is it typically used?
#Answer
#Python Lambda Functions are anonymous functions means that the function is␣
 ↪without a name. As we already know the def keyword is used to define a␣
 ↪normal function in Python. Similarly, the lambda keyword is used to define␣
 ↪an anonymous function .
#This function can have any number of arguments but only one expression, which␣
 ↪is evaluated and returned.
Syntax: lambda arguments : expression
#Example
str1 = 'my name is ashutosh!'

upper = lambda string: string.upper()
print(upper(str1))
```

```
MY NAME IS ASHUTOSH!
```

[4]:
```
upper = lambda string: string.capitalize()
print(upper(str1))
```

```
My name is ashutosh!
```

[7]:
```
length = lambda string: len(str1)
print(length(str1))
```

```
20
```

[9]:
```
#--Q--9- Explain the purpose and usage of the `map()` function in Python.
#Answer
#Map in Python is a function that works as an iterator to return a result after␣
 ↪applying a function to every item of an iterable (tuple, lists, etc.). It is␣
 ↪used when you want to apply a single transformation function to all the␣
 ↪iterable elements.
#Syntax : map(fun, iter)

#Parameters:

#fun: It is a function to which map passes each element of given iterable.
#iter: It is iterable which is to be mapped.
# Python program to demonstrate working
# of map.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

```
[2, 4, 6, 8]
```

[10]:
```
#map() with Lambda Expressions
numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))
```

```
[2, 4, 6, 8]
```

[13]:
```
l = ["sat"]
result=list(map(list,l))
print(result)
```

```
[['s', 'a', 't']]
```

[22]:
```
#--Q--10-What is the difference between `map()`, `reduce()`, and `filter()`␣
 ↪functions in Python?
#Answer
#The 'Map' operation in programming transforms data from one form into another.␣
 ↪The 'Reduce' operation combines the elements of a list into a single result.␣
 ↪The 'Filter' operation selects a subset of items based on a given condition.
#Examples
```

```python
# Example of using map, filter, and reduce in Python
data = [1, 2, 3, 4, 5]

# Using the map to apply a function to each element
# Lambda function returns the square of x
result1 = map(lambda x: x * 2, data)  # Result: [2, 4, 6, 8, 10]
print(list(result1))

# Using the filter to filter elements based on a condition
# Lambda function returns True for an even number
result2 = filter(lambda x: x % 2 == 0, data)  # Result: [2, 4]
print(tuple(result2))

# Using reduce to aggregate elements
# Lambda function returns product of x and y
from functools import reduce
result3 = reduce(lambda x, y: x * y, data)  # Result: 120
print(result3)
```

```
[2, 4, 6, 8, 10]
(2, 4)
120
```

[25]:
```python
#--Q--11- Using pen & Paper write the internal mechanism for sum operation␣
 ↪using  reduce function on this given
list1=[47,11,42,13]
#Answer
result=reduce(lambda x,y:x+y,list1)
print(result)
```

```
113
```

[ ]:
```python
#Find Below the attached doc for mechanism
```

Mechanism of Sum in reduce

List1 = [47, 11, 42, 13]

Working: $y = x + y$

1. First, two elements are picked
   [47, 11]
   Result is → 58

2. Now result obtained is picked along with next element.
   [58, 42]
   Result is → 100

3. Process continues till the list empty.
   Again,
   [100, 13]
   Result is → 113

4. Finally, the result obtained and printed at [113]
   Answer.

function coding

July 23, 2024

[1]:
```
#PRACTICAL QUESTIONS
```

[14]:
```
#Q--1. Write a Python function that takes a list of numbers as input and␣
 ↪returns the sum of all even numbers in the list.

# CODE
#Defining function even number:
def sum_even_number(list1):
    even_sum =0
    for num in list1:
        if num%2==0:
            even_sum=num+even_sum
    return f"Sum of even numbers from list:{even_sum}"
```

[15]:
```
#calling function by name
sum_even_number([345, 893, 1948, 34, 2346])
```

[15]:
```
'Sum of even numbers from list:4328'
```

[34]:
```
#-Q--2-. Create a Python function that accepts a string and returns the reverse␣
 ↪of that string.

#CODE
def reverse(s):
    str = ""
    for i in s:
        str = i + str
    return str

s = "Ashutosh Kumar Mishra"

print(f"The original string is :{s} ")
```

```
print(f"The reversed string is : {reverse(s)}")
```

```
The original string is :Ashutosh Kumar Mishra
The reversed string is : arhsiM ramuK hsotuhsA
```

[3]:
```
#-Q--3- Implement a Python function that takes a list of integers and returns a␣
 ↪new list containing the squares ofeach number.

#CODE
def square_of_list(list1):
    square_list =[]
    for num in list1:
        square_list.append(num**2)

    return f"Square of list : {square_list} "
list1=[345, 893, 1948, 34, 2346]
```

[4]:
```
square_of_list(list1)
```

[4]:
```
'Square of list : [119025, 797449, 3794704, 1156, 5503716] '
```

[29]:
```
#-Q--4. Write a Python function that checks if a given number is prime or not␣
 ↪from 1 to 200

#CODE

def check_prime_number(number):

    if number<2:
        print(f"{numbers} is not a Prime")
    else:
        for num in range(2,number):
            if number%num ==0:
                print(f"{number} is not a Prime")
                break

        return f"This {number} is a prime number"
```

[30]:
```
numbers=3
check_prime_number(numbers)
```

[30]:
```
'This 3 is a prime number'
```

[31]:
```
#-Q--5. Create an iterator class in Python that generates the Fibonacci␣
 ↪sequence up to a specified number of terms.
```

```
#CODE

def fibonacci(n):
    a=0
    b=1
    for num in range(n):
        yield a #Iterator to generate one by one using yield
        a,b=b,a+b
d=fibonacci(10)

print(next(d))#printing one by one using next()
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
```

```
0
1
1
2
3
5
8
```

[37]:
```
#-Q--6. Write a generator function in Python that yields the powers of 2 up to␣
 ↪a given exponent.

#code

def power_of_exponent(exponent,power=2):

    for num in range(exponent):
        result_of_exponent =num**power

        yield result_of_exponent

power= power_of_exponent(5)
```

[39]:
```
print(next(power))
print(next(power))
print(next(power))
print(next(power))
```

```
1
```

```
4
9
16
```

[7]:
```python
#-Q--7. Implement a generator function that reads a file line by line and␣
 ↪yields each line as a string

# CODE

def read_file_line_by_line(file):
    with open(file_path, 'r') as file:
        for line in file:
            yield line.strip()
print(next(file_path))
```

[6]:
```python
#-Q---8. Use a lambda function in Python to sort a list of tuples based on the␣
 ↪second element of each tuple

#CODE

# Python program to sort a list of tuples by the second Item

# Function to sort the list of tuples by its second item
def sort_tuple(lst):

        lst.sort(key=lambda x: x[1])

        return lst

lst= [('rishav', 10), ('akash', 5), ('ram', 20), ('gaurav', 15)]

sort_tuple(lst)
```

[6]:
```
[('akash', 5), ('rishav', 10), ('gaurav', 15), ('ram', 20)]
```

[16]:
```python
#-Q-9. Write a Python program that uses `map()` to convert a list of␣
 ↪temperatures from Celsius to Fahrenheit.

#CODE
list_of_temp=[30,40,50,20]

fahrenheit = map(lambda x: (9/5)*x + 32 , list_of_temp)
print(f"The temperature in fahrenheit is {list_of_temp} F")
```

```
The temperature in fahrenheit is [30, 40, 50, 20] F
```

[ ]:
```python

```

[80]:
```
#--Q--10. Create a Python program that uses `filter()` to remove all the vowels␣
 ↪from a given string

#CODE
string1="My Name is Ashutosh Kumar Mishra"



def remove_vowels(string1):
    vowel_letters=["a","e","i","o","u"]
    for i in range(len(string1)):
        if string1[i].lower() not in vowel_letters:
            return True
        else:
            return False

#using filter function
string_without_vowel=filter(remove_vowels,string1)
```

[81]:
```
for s in string_without_vowel:
    print(s,end="" )
```

```
My Nm s shtsh Kmr Mshr
```

[ ]:
```

```

[11]:
```
'''
#-Q--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 Python3, Bernd Klein        3        24.99


Write a Python program, which returns a list with 2-tuples. Each tuple consists␣
 ↪of a the order number and the product of the price per items and the␣
 ↪quantity. The product should be increased by 10,- € if the value of the␣
 ↪order is less than 100,00 €.
Write a Python program using lambda and map.
'''
```

```
#CODE
#In this first we will convert in form of list

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]]

invoice_totals = list(map(lambda x: x if x[1] >= 100 else (x[0], x[1] +␣
 ↪10),map(lambda x: (x[0],x[2] * x[3]), orders)))
```

[12]:
```
print(invoice_totals)
```

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

[ ]:

[ ]: