# function assignment

September 2, 2024

## *1 function assignment*

Name:-Yash solanki

Reg email id:-yash87015@gmail.com

Course name:-Data analytic

Assignment name:-function

Submission date:-not declare

Git link:-https://github.com/Yash87015/YASH (in git hub use assignment folder )

## 2 Theory questions

### 2.1 question 1 What is the difference between a function and a method in Python?

#### 2.1.1 Ans

→ function:-function is a block of code which performs some specific task/computation

→ Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

→ Types of Functions: 1)Built-in Function: Built-in functions are the pre-defined function in Python that can be directly used. 2)User-Defined Function: They are not pre-defined functions. The user creates their own function to fulfill their specific needs.

syntax of function

def function_name(arguments):

“ “ “

functions logics

” ” ”

return values

***Function is block of code that is also called by its name. (independent) The function can have different parameters or may not have any at all. If any data (parameters)***

***are passed, they are passed explicitly. It may or may not return any data. Function does not deal with Class and its instance concept.***

[1]:
```
def func_name(a,b):
    return a+b

func_name(5,8)
```

[1]: 13

[3]:
```
#some inbuid fuction,when we use inbit function no need to use def,always write a umber in list other wise it give you error
a=sum([5,8])
print(a)
```

13

→ method:- As Python is an Object-Oriented Programming language, so it contains objects, and these objects have different properties and behavior. Methods in Python are used to define the behavior of the python objects.

→ It facilitates code reusability by creating various methods or functions in the program. It also improves readability and accessibility to a particular code block. Methods creation makes it easy to debug for the programmers.

→ Types of Methods in Python: 1)Instance Method 2)Class Method 3)Static Method

syntax of method

class class_name: def method_name () :

```
    ------
    # method body
    ------
```

***Method is called by its name, but it is associated to an object (dependent). A method definition always includes 'self' as its first parameter. A method is implicitly passed to the object on which it is invoked. It may or may not return any data. A method can operate on the data (instance variables) that is contained by the corresponding class***

[4]:
```
class Alphabet:
    def Capital_Letters(self):
        print("A,B,C,D..... ")

    def Small_Letters(self):
        print("a,b,c,d..... ")
```

[5]:
```
A=Alphabet()
```

[6]:
```
A.Capital_Letters()
```

A,B,C,D...

[7]:
```
A.Small_Letters()
```

a,b,c,d...

**Key Difference Between Methods and Function**

→ Method definition is always present inside the class, while the class is not required to define the function.

→ Functions can have a zero parameter, whereas the method should have a default parameter, either self or cls, to get the object.

→ The method operates the data in the class, while a function is used to return or pass the data.

→ A function can be directly called by its name, while a method can't be called by its name.

→ The method lies under Object-Oriented Programming, while a function is an independent functionality.

→ Functions can be called only by its name, as it is defined independently. But methods can't be called by its name only, we need to invoke the class by a reference of that class in which it is defined, i.e. method is defined within a class and hence they are dependent on that class.

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

### 2.2.1 Ans

→ parameter:-In programming, a parameter refers to a named variable within a function or method definition. The parameter serves as a placeholder for a value that will be passed to the function or method at the time of its call. Essentially, parameters define the type of data a function can accept and also dictate how many pieces of data it expects. They act as the blueprint for the arguments that will be provided during the function's execution.

→ Arguments:-An argument in programming is the value passed to a function or a method when it is called. The function uses these values to perform operations or calculations.Arguments are the real, concrete data that replace the function's parameters when the function executes.

**Fundamentally, parameters are the variables inside a function's parentheses. Arguments provide values for those parameters.

***A parameter is the variable listed inside the parentheses in the function definition.***

***An argument is the value that are sent to the function when it is called.***

[33]:
```
#The terms parameter and argument can be used for the same thing: information
 ↪that are passed into a function.
#From a function's perspective: 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.
def greet(name):  # 'name' is a parameter
    print("Hello,", name)
```

```
greet("jk") #"jk" is an argument
```

```
Hello, jk
```

[29]:
```
#In programming, we utilize functions to organize our code and make it simpler to reuse.
#Like a recipe , a function encompasses a set of instructions (code) that tells the computer what to do.
#However, like a chef , we may have to plan different types of dishes (execute the function)
#with diverse ingredients (arguments) and cooking strategies (parameters).
def make_pizza(pizza_type, toppings):
    # Code to make pizza
    print("Making a " + pizza_type + " pizza with " + toppings + " toppings!")

make_pizza("Margherita", "cheese and tomato")
```

```
Making a Margherita pizza with cheese and tomato toppings!
```

that the pizza_type and toppings are the arguments that are passed to the make_pizza function

***Think of a function as a recipe. The parameters are the ingredients listed in the recipe, and the arguments are the actual ingredients you use when you cook.***

(Similarities Between Argument and Parameter)(optioal for understanding perpose)

→ Function Association: Both parameters and arguments are intrinsically linked to functions. They are essential in the process of defining and calling functions.

→ Data Passing: They are both used to pass data to functions. Parameters receive the data, and arguments supply it.

→ Flexibility in Function Execution: Both contribute to the flexibility of functions, allowing the same function to operate on different data values.

→ Types of Data: Both can represent any type of data that the programming language supports, such as integers, strings, objects, etc.

→ Influence on Function Behavior: Parameters and arguments directly affect the function's behaviour. The function's output depends on the argument values passed to its parameters.

Types of Function Arguments in Python (optioal for understanding perpose)(its not include in question) Python has four built-in function argument types we can use to call functions and have them perform their intended tasks. These are:

1)Default Arguments:-Default arguments in Python have pre-set values if the user does not specify them when calling the function. The default value can be any data type like a list, dictionary, string, number, etc.Default arguments are defined by assigning values to arguments in the function definition.

2)Keyword Arguments:-Keyword arguments, or named arguments, are parameters passed to a Python function that specify the argument name and value. This makes the code more readable by eliminating the need to remember the order of parameters. Keyword arguments can also set

default values for unspecified parameters. Using keyword arguments enhances code readability and maintainability. Keyword arguments explicitly state which argument matches which parameter.

3) Arbitrary Arguments:-Arbitrary arguments in Python enable functions to accept an unlimited number of arguments. This allows for flexibility when creating functions that require an unknown number of arguments. Arbitrary arguments are denoted with an asterisk (*) before the argument name . These arguments can create a list or dictionary of the passed arguments

explaination of below example:-This code defines a function called sum_numbers which takes an arbitrary number of arguments. The asterisk before args indicates that this argument will take an arbitrary number of arguments. When the function is called, it takes in a series of numbers as arguments and adds them together, returning the sum. The function then iterates through each of the arguments, adding them together and storing the total in a variable called the result. Finally, the function returns the result.

4) Required Arguments:-Required arguments are when calling a function. They are position-dependent, meaning their matters. Required arguments are defined when the function is created and cannot when the function is called. Arguments must be given in the same as defined when invoking a function.

explaination of below example:-This code defines a function named my_function that accepts two arguments, arg1, and arg2. The code calls the function, passing 10 and 20 as the two arguments. The function adds arg1 and arg2, then returns the result. The result of the function is stored in the variable result, then printed to the screen, showing the value 30.

[38]:
```
# example 1 default argument
def suml(b, c, a=0):#if a is not(its not given eroor) its okay if give then␣
 ↪calculated #thats the define function
    return a+b+c
sum1(5,4) #thats how you call a function
```

[38]: 9

[34]:
```
#example 2 of keyword argument
def greeting(name, age, gender):
    print("Hello, %s. You are %d years old and you are a %s." % (name, age,␣
 ↪gender))

greeting("Bob", 25, "male") # Hello, Bob. You are 25 years old and you are a␣
 ↪male.
```

Hello, Bob. You are 25 years old and you are a male.

[35]:
```
#example 3 of arbitary argument # you can use number of argument as you want␣
 ↪#variable length arguement>>when you dont know the no of arguments
def sum_numbers(*args):#(*)thats called asterisk
    result = 0
    for i in args:
        result += i
```

```
    return result

print(sum_numbers(1, 2, 3, 4, 5))
```

15

[36]:
```
#example 4 of required argument
def my_function(arg1, arg2):
    # function code
    return arg1 + arg2

result = my_function(10, 20)
print(result) # prints 30
```

30

## 2.3 question 3 What are the different ways to define and call a function in Python?

### 2.3.1 Ans

***Define a function***

→ Use the keyword def to declare the function and follow this up with the function name.

→ Add parameters to the function: they should be within the parentheses of the function.End your line with a colon.

→ Add statements that the functions should execute.

→ End your function with a return statement if the function should output something. Without the return statement, your function will return an object None.

→ function_name: A unique name to identify the function

→ function body: The code that executes when the function is called.

#parameters: (Optional) Input values for the function. #Docstring: It is an optional section that documents what the role of the function is. It enhances readability if a user looks at his code after a long duration. #Return statement: It is an optional statement. The return keyword can be used by the Python function to return a value.

1) Standard Function Definition: This is the most common way to define a function:
2) Lambda Function (Anonymous Function): Used for small, one-line functions:
3) Nested Functions: Functions defined inside other functions:

***Call a Function***

→ Calling a function means that you execute the function that you have defined - either directly from the Python prompt or through another function

→ To call a function, simply use its name followed by parentheses and pass any required arguments:

1) Direct Call: This is the most common way of calling a function. You simply write the function name followed by parentheses containing any arguments.
2) Using Keyword Arguments: You can pass arguments by explicitly specifying their names. This improves readability and allows you to change the order of arguments.
3) Using Default Arguments: You can define default values for function parameters. If an argument is not provided during the call, the default value is used.
4) Calling Functions as Methods: When a function is defined within a class, it becomes a method. You can call the method using an instance of the class.
5) Using Arbitrary Number of Keyword Arguments: You can use ** kwargs to pass an arbitrary number of keyword arguments to a function. The arguments are collected in a dictionary.

we see example as well and we also see some other types of example as well how to define and call a function in examples

see above example 1 to 4 in question 2 there you can see different example how to difine and call a function

[5]:
```
# example 5
def print_info(**kwargs): # here see how to define
    for key, value in kwargs.items():
        print(f"{key}:{value}")

print_info(name="jk",corse="data analytic") # here see how to call
```

```
name:jk
corse:data analytic
```

[ ]:
```

```

## 2.4 question 4 What is the purpose of the return statement in a Python function?

### 2.4.1 Ans

→ the return statement serves two primary purposes: 1) Exiting a Function: It marks the end of a function's execution and returns control back to the caller. 2) Returning a Value: It allows a function to pass data back to the caller, which can then be used for further operations or decisions in the program.

→ Using return , you can write a function that returns a value. You can assign the returned values to a variable which can be used in whatever manner you choose. This is useful when you need to use the results your function produces for something else

lets see with example

[18]:
```
#example 6
def sum1(x,y):
    total=x+y
```

```
    return total
sum1(5,8)
```

[18]: 13

[21]:
```
#example 7
def greeting(name, age, gender):
    return "Hello, %s. You are %d years old and you are a %s." % (name, age,␣
 ↪gender)
greeting("yk",45,"male")
```

[21]: 'Hello, yk. You are 45 years old and you are a male.'

[ ]:

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

### 2.5.1 Ans

Iterable:***An iterable is any python object/sequential structure/data structure that is capable of returning its members one at a time*** → permitting it to be iterated over in a for loop → example» list, tuples → To be iterable, an object must implement the **iter**() method, which returns an iterator. → Examples include lists, tuples, strings, dictionaries, sets, and files.
.__iter__():- Called to initialize the iterator. It must return an iterator object.

iterators:- ***An iterator is an object representing a stream of data and return the data one by one*** → an iterator is an object that allows you to iterate over collections of data, such as lists, tuples, dictionaries, and sets. ***Iterators take responsibility for two main actions: 1) Returning the data from a stream or container one item at a time 2) Keeping track of the current and visited items***

A Python object is considered an iterator when it implements two special methods collectively known as the iterator protocol. These two methods make Python iterators work. So, if you want to create custom iterator classes, then you must implement the following methods:

→ Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods is iter and next

.__next__():- Called to iterate over the iterator. It must return the next value in the data stream.

Process of iterator its called iteration(iteration is a process of looping through the elements fof an iterable (list, string).using a loop The process of returning element one by one iteration) → An object that represents a stream of data. → It keeps track of the current position within the iterable and provides the next element using the **next**() method. → When there are no more elements, the **next**() method raises a StopIteration exception.

***Analogy potato(iterable) » it can not be cooked in original form chop/wash the potato » ready for cooking»iterator the process of cooking»iteration***

In simple terms: Iterable: Think of it as a container holding data that can be iterated over. Iterator: Think of it as a pointer that moves through the elements of an iterable.

***Key Differences between iterable and iterator***

***Iterable:-You can iterate over an iterable multiple times. Each time you use iter() on it, you get a new iterator.***

***Iterator:-You can iterate over an iterator only once. Once you reach the end, it's exhausted.***

[10]:
```
#example 8
iterable=([1,2,3])#you can also use string or dict or set ant data type
iterator=iter(iterable)
next(iterator)
```

[10]: 1

[11]:
```
next(iterator)
```

[11]: 2

[12]:
```
next(iterator)
```

[12]: 3

[13]:
```
next(iterator)#see that exhausted and also throut the whole process called␣
↪iteration
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Cell In[13], line 1
----> 1 next(iterator)

StopIteration:
```

[ ]:
```

```

## 2.6 question 6 Explain the concept of generators in Python and how they are defined.?

### 2.6.1 Ans

***generator:-A generator in Python is a function that uses the yield keyword to return a sequence of values one at a time, instead of returning them all at once like a regular function. This makes generators memory-efficient and suitable for working with large datasets.***

→ Generator functions in Python are a way to create iterators in a more concise and memory-efficient manner

→ These functions allow you to iterate over a potentially large sequence of data without loading the entire sequence into memory.

syntax of generator function:

def function_name():

yield statement

Use of yield: Instead of using return to send a value back to the caller, a generator function uses 'yield'. When the generator function encounters a 'yield' statement, it returns the value specified in the yield andpauses its execution state. The next time the generator is called, it resumes from where it left off.

[5]:
```
#example 9
def Feb_generator():
    a , b=0 , 1
    while True:
        yield a    #yield is creating a generator similar to iterator object
        a,b=b,a+b
f_g=Feb_generator()
for _ in range(10):
    print(next(f_g,))
```

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

[7]:
```
type(f_g)
```

[7]: generator

## 2.7 question 7 What are the advantages of using generators over regular functions?

### 2.7.1 Ans

→ The main difference between a regular function and a generator function is that the latter uses the yield keyword to produce a series of values over time, one at a time, rather than returning a single result.

advantages of using generators: 1) Lazy Evaluation: Generator functions follow the concept of lazy evaluation. This means that the values are generated on-the-fly and only when requested. This is particularly useful for working with large datasets or infinite sequences. 2) Memory Efficiency:

Generators are memory-efficient because they don't store all values in memory at once.Each value is generated and consumed one at a time, making them suitable for working with large datasets or when memory usage is a concern. 3) Convenience: Generators simplify code for iterating over large datasets or streams of data without needing to manage state explicitly. 4) Maintain State: Generators automatically maintain their state between yield statements, making them suitable for iterative processes.

regular function calculates the square of the values and return in one go,since its returning in one go then memory usage will be more and it will a lot of execution time if there is lot of computation as all the computations will be done first and then the calculation will be returned in one go but in generator result one by one instead of go lets see with example

[11]:
```
def square_number_gen(n):
    for i in range(n):
        yield (i ** 2)
square_number_gen(4)
gen=square_number_gen(4)
```

[12]:
```
next(gen)
```

[12]: 0

[13]:
```
next(gen)
```

[13]: 1

[14]:
```
next(gen)
```

[14]: 4

[15]:
```
next(gen)
```

[15]: 9

[ ]:
```

```

## 2.8 question 8 What is a lambda function in Python and when is it typically used?

### 2.8.1 Ans

***lambda:-A lambda function is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression.***

→ A lambda function in Python is a concise way to create anonymous functions, also known as lambda expressionsp → Unlike regular functions defined using the def keyword, lambda functions are often used for short-term operations and are defined in a single line.

The basic syntax of a lambda function is: lambda argument: expression

Here, arguments are the input parameters, and expression is the operation to be performed using those parameters. Lambda functions can have any number of input parameters but can only have one expression.

limitation: Simplicity: Lambda functions can only have one expression, so they can't handle larger tasks. Readability: If overused or made too complex, they can make code harder to understand.

use cases:-Lambda functions are ideal for situations where you need a small, anonymous function for a short period of time. They are not recommended for complex functions with multiple expressions and statements.

[23]:
```
#example 10
even_number=lambda x: x%2==0
```

[24]:
```
even_number(10)
```

[24] : True

[25]:
```
even_number(15)
```

[25] : False

[ ]:
```

```

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

### 2.9.1 Ans

→ The map() function in Python is a powerful tool for applying a function to every item in an iterable (like a list, tuple, etc.) and returning a new iterator with the results.

→ ***Map:- The map function executes a specified function for each item in an iterable. The item is sent to the function as a parameter***

→ map Make an iterator that computes the function using arguments from each of the iterables. Stops when the shortest iterable is exhauste

→ The map() function in Python is a built-in function that applies a given function to all the items in an iterable (e.g., a list) and returns an iterable map object (an iterator) of the results.

→ map() is a core concept in functional programming, promoting a declarative style of coding.

use cases of map:-Applying mathematical operations,Converting data types,Manipulating strings,Working with multiple iterables,Cleaning and filtering data.lets see one example

[7]:
```
#example 11
first_name = ["Ajay","Ramu"]
last_name  = ["Gupta","sinha"]
full_name=list(map(lambda first,last:f"{first} {last}",first_name,last_name))
```

[8]:
```
print(full_name)
```

['Ajay Gupta', 'Ramu sinha']

[ ]:

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

## 2.11 Ans

Map, Reduce, and filter are built-in functions that allow you to apply operations to iterables (like lists, tuples, etc.) in a functional programming style. Here's a breakdown of their differences below:

Map:-Applies a function to each item in an iterable and returns a new iterable with the transformed values.sy:-map(function, iterable) Filter:-Filters an iterable by keeping only the elements that meet a specific condition defined by a function.sy:-filter(function, iterable) Reduce:- Combines the elements of an iterable into a single value by repeatedly applying a function. Sy: reduce(function, iterable, initializer)

key difference: 1) Output:- map returns an iterable with the same length as the input, filter returns an iterable with potentially fewer elements, and reduce returns a single value. 2) Functionality:- map transforms elements, filter selects elements, and reduce aggregates elements. 3) Function Argument:-The function passed to map takes a single argument, the function passed to filter takes a single argument and returns a boolean value, and the function passed to reduce takes two arguments.

→'Map' applies a given function to each item of an iterable (like a list or set) and returns a list of the results. 'Filter' constructs a list from elements of an iterable for which a function returns true. 'Reduce' applies a rolling computation to sequential pairs of values in a list and returns a single result.

lets see with example

[23]:
```
#exaple 12
from functools import reduce
numbers=[1,2,3,4,5,6,7,8,9]
square_num=list(map(lambda x:x**2,numbers))
even_num=list(filter(lambda x:x%2==0,numbers))
addition=reduce(lambda x,y:x+y,numbers)
```

[17]:
```
print(square_num)
```

[1, 4, 9, 16, 25, 36, 49, 64, 81]

[20]:
```
print(even_num)
```

[2, 4, 6, 8]

[24]:
```
print(addition)
```

45

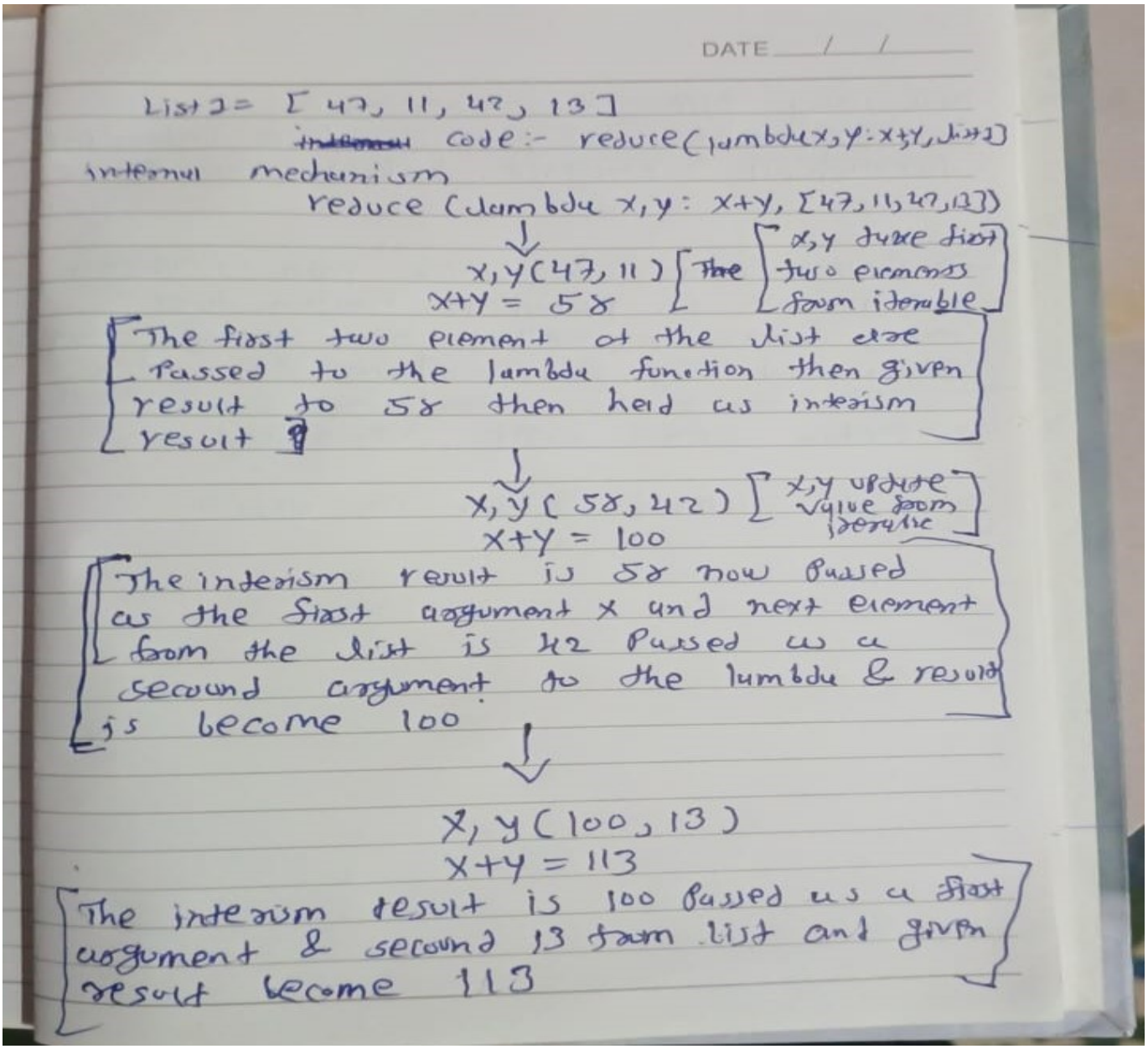
[ ]:

### 2.12 question 11 . Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:[47,11,42,13];

#### 2.12.1 Ans

[16]:
```
from functools import reduce
list1=[47,11,42,13]
total_sum=reduce(lambda x,y:x+y,list1)
print(total_sum)
```

113



[ ]:

# 3 Practical Questions:

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

### 3.1.1 ans

Here we see as a input we take a numbers as a list and return all the sum of even number using reduce function

[49]:
```
from functools import reduce
numbers=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1,7,18,19,20]
sum_even_number=reduce(lambda x,num:x + num if num%2==0 else x,numbers,0)
print(sum_even_number)
```

110

[ ]:

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

### 3.2.1 Ans

here ypu can see two example with .join()and reversed function and also see to how to use reduce functtion and create a reversed string

[1]:
```
#example 2
"".join(reversed(input(str())))
```

thats my first program

[1]: 'margorp tsrif ym staht'

[2]:
```
#example 3
from functools import reduce
def reversed_string(string_input):
    return reduce(lambda a, b:b+a,string_input)
reversed_string("hello world")
```

[2]: 'dlrow olleh'

[ ]:

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

#### 3.3.1 Ans

[2]:
```
x=[1,2,3,4,5,6,7,8]
squared_number = list(map(lambda x: x**2,x))
print(squared_number)
```

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

[ ]:

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

#### 3.4.1 Ans

[1]:
```
def is_prime(num,i=2):
    lis=[]
    if num < 2:
        return 0
    elif num <= 2:
        return num==2
    elif num % i == 0:
        return 0
    elif i * i > num:
        return True
    return is_prime(num,i+1)
# Print prime numbers between 1 and 200
for num in range(1, 201):
    if is_prime(num):
        print(num,end=" ")
```

```
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
```

[ ]:

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

### 3.5.1 Ans

[14]:
```
def fib(n):
    a = 0
    b = 1
    for i in range(n):
        yield a
        a, b = b, a+b
```

[15]:
```
f = fib(7)
```

[18]:
```
print(next(f))
```

```
0
```

[19]:
```
next(f)
```

[19]:
```
1
```

[20]:
```
next(f)
```

[20]:
```
1
```

[21]:
```
next(f)
```

[21]:
```
2
```

[22]:
```
next(f)
```

[22]:
```
3
```

[23]:
```
next(f)
```

[23]:
```
5
```

[24]:
```
next(f)
```

[24]:
```
8
```

[25]:
```
next(f)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Cell In[25], line 1
----> 1 next(f)
```

```
StopIteration:
```

[ ]:

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

#### 3.6.1 Ans

[55]:
```
def find_power(base,exponent):
    yield base**exponent
```

[56]:
```
ae=find_power(2,3)
```

[57]:
```
print(next(ae))
```

```
8
```

[ ]:

### 3.7 question 7. Implement a generator function that reads a file line by line and yields each line as a string.

#### 3.7.1 Ans

[65]:
```
import io

def read_line_by_line(L):
    file_object  =  io.StringIO("\n".join(L))
    for line in file_object:
        yield line.strip()


L = ["line 1:-First book in list", "line 2:-secoung file from list","line 3:
 ↪third file from book"]
for line in read_list_line_by_line(L):
    print(line)
```

```
line 1:-First book in list
line 2:-secoung file from list
line 3:third file from book
```

[ ]:

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

#### 3.8.1 Ans

[67]:
```
planet= [('Earth', 3), ('Jupiter', 5), ('Mercury', 1), ('Mars', 4), ('Neptune',␣
↪8), ('Saturn', 6), ('Uranus', 7), ('Venus', 2)]
new_planet_sorted_list = sorted(planet, key=lambda x: x[1])
```

[68]:
```
print(new_planet_sorted_list)
```

[('Mercury', 1), ('Venus', 2), ('Earth', 3), ('Mars', 4), ('Jupiter', 5), ('Saturn', 6), ('Uranus', 7), ('Neptune', 8)]

[ ]:

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

#### 3.9.1 Ans

[70]:
```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32
temperatures_celsius = [45,-20,5]
temperatures_fahrenheit = list(map(lambda x: (celsius_to_fahrenheit(x)),␣
↪temperatures_celsius))
print(temperatures_fahrenheit)
```

[113.0, -4.0, 41.0]

[ ]:

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

#### 3.10.1 Ans

[13]:
```
str1="hello world"
```

[14]:
```
new_str= ''.join(filter(lambda x:x.lower() not in "'a', 'e', 'i', 'o', 'u',␣
↪'A', 'E', 'I', 'O', 'U'",str1))
```

[15]:
```
new_str
```

[15]: 'hllwrld'

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

### 3.12

| 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 |

[94]:

### 3.13 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

#### 3.13.1 Ans

```
fr
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]]
min_order = 100
invoice_totals = list(map(lambda x: x if x[1] >= min_order else (x[0], x[1] +␣
 ↪10),
```

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

[ ]: