AUDISANKARA COLLEGE OF ENGINEERING & TECHNOLOGY
UGC - AUTONOMOUS
GUDUR, TIRUPATI(DT), A.P, INDIA
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PYTHON LAB MANUAL

R20 - Regulation

Name : _____

Roll No.: _____

Mr. D.V. VARAPRASAD, M.Tech., (Ph.D.)
Associate Professor
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AUDISANKARA COLLEGE OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS)
(Accredited by NAAC A+)
Approved by AICTE, Affiliated to JNTUA Anantapur,
Tirupati (DT), Andhra Pradesh

EXPERIMENT 1(a)

Perfect Number: perfect number, a positive integer that is equal to the sum of its proper divisors. The smallest perfect number is 6, which is the sum of 1, 2, and 3. Other perfect numbers are 28, 496, and 8,128.

Example:

Input: n = 15

Output: false

Divisors of 15 are 1, 3 and 5. Sum of

divisors is 9 which is not equal to 15.

Input: n = 6

Output: true

Divisors of 6 are 1, 2 and 3. Sum of

divisors is 6.

AIM: Write a python program to find the perfect numbers between 1 to 1000, use the function name perfect in the program.

Problem Solution

Given an integer lower and upper values to find the perfect numbers between them. Call the function perfect with lower and upper values. Start for loop that iterates from 1 to 1000. Use a for loop and an if statement to add the proper divisors of the integer to the sum variable. Check if the sum of the proper divisors of the number is equal to the variable. Print the final result.

Source code:

```
lower=int(input("enter lower value:"))
upper=int(input("enter upper value:"))
def perfect(lower,upper):
    for num in range(lower,upper+1):
        result=0
        for i in range(1,num):
            if num%i == 0:
                result=result+i
        if num==result:
            print(num)

perfect(lower,upper)
```

Output:

enter lower value:1

enter upper value:1000

6

28

496

Conclusion: The above experiment has been executed successfully

EXPERIMENT1(b)

Odd Factors: find the factors of a number, in that search for factors which are odd. Odd numbers are **those numbers which are not divisible by 2**.

Example:

Input: n = 30

Output: 24

Odd dividers sum 1 + 3 + 5 + 15 = 24

Input : 18

Output : 13

Odd dividers sum 1 + 3 + 9 = 13

AIM: Python Program for Find sum of odd factors of a number.

Problem statement

Given a number input n, the task is to Find the sum of odd factors of a number. Here we first need to eliminate all the even factors. To remove all even factors, we repeatedly divide n till it is divisible by 2. After this step, we only get the odd factors of the number. Add the values to the variable odd. Print the final result.

Source Code:

```
x = int(input("Enter any number:"))
odd = 0
for i in range(1, x + 1):
    if x % i == 0:
        if i % 2 != 0:
            odd = odd + i
            print('odd factor is:', i)

print('sum of odd factors of a number is', odd)
```

Output:

Enter any number:30

odd factor is: 1

odd factor is: 3

odd factor is: 5

odd factor is: 15

sum of odd factors of a number is 24

Conclusion: The above experiment has been executed successfully

EXPERIMENT 2(a)

Vowels and Consonants: Words are built from vowels (a, e, i, o, u) and consonants (the rest of the alphabet)

AIM: Program to Count Vowels and Consonants in a string

Problem Statement:

Define a string. Convert the string to lower case so that comparisons can be reduced. Else we need to compare with capital **(A, E, I, O, U)**. If any character in string matches with vowels **(a, e, i, o, u)** then increment the vcount by 1. If any character lies between 'a' and 'z' except vowels, then increment the count for ccount by 1. Print both the counts.

Source Code:

```
sentence = input("Enter the string:")
string = sentence.lower()
print(string)
vcount = 0
ccount=0
list = ['a','e','i','o','u']
for char in string:
    if char in list:
        vcount = vcount+1
    else:
        ccount=ccount+1

print("Number of vowels in a given string is:", vcount)
print("Number of consonants in a given string is:",
ccount)
```

Output:

Enter the string: hello world

hello world

Number of vowels in a given string is: 3

Number of consonants in a given string is: 8

Note: The white spaces will be counted in consonants.

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 2(b)

Example of the sum of digits in a string:

String: 5Py8thon3

Sum of digits = 16

AIM: Write a Python program to compute sum of digits of a given string.

Problem Statement:

We will take a string while declaring the variables. Initialize the sum_total to zero. Then, compute the sum of digits in a given string using the for loop and if statement. The **isdigit()** is a built-in method used for string handling. The isdigit() method returns True if the characters are digits, otherwise False. We can check if one character is a digit or not. If it is a digit, we will add its value to the sum_digit variable. The final result can be displayed with the variable sum_total after the for loop execution.

Source Code:

```
inputstr = input("Enter your string: ")
sum_total = 0
for x in inputstr:
    if x.isdigit():
        sum_total = sum_total+int(x)
print("Total:- ", sum_total)
```

Output:

Enter your string: ab1v36

Total:- 10

Enter your string: 121

Total:- 4

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 3(a)

Given an array **arr[]** of non-negative integers and an integer **sum**, find a subarray that adds to a given **sum**.

AIM: Program for Sub array with given sum A[] = {1,2,3,7,5}

Problem statement:

Traverse the array from start to end. From every index start another loop from i to the end of the array to get all subarrays starting from i, and keep a variable currentSum to calculate the sum of every subarray. For every index in inner loop update currentSum = currentSum + arr[j]. If the currentSum is equal to the given sum then print the subarray.

Source Code:

```
def subArraySum(arr, n, sum):
    for i in range(n):
        curr_sum = arr[i]
        j = i + 1
        while j <= n:
            if curr_sum == sum:
                print("Sum found between indexes %d and
%d"%(i,j-1))
                return 1
            if curr_sum > sum or j == n:
                break
            curr_sum = curr_sum + arr[j]
            j += 1
    print("No subarray found")
    return 0
arr = [15, 2, 4, 8, 9, 5, 10, 23]
n = len(arr)
sum = 24
subArraySum(arr, n, sum)
```

Output:

Sum found between indexes 1 and 4

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 3(b)

AIM: Write a Python program to find the list of words that are longer than n from a given list of words.

Problem Statement:

Define a function that will find words with length greater than 'k'. To get words from the string, split the string wherever space comes. Run a loop to check words in the string. If the length of the word is greater than a given value k. Print the word. Else, move on to the next iteration. Declare a string and value of k. Call function and give string and k as parameters.

Source Code:

```
def word_k(k, s):
    word = s.split(" ")
    for x in word:
        if len(x)>k:
          print(x)
k = 3
s ="Python is good"
word_k(k, s)
```

Output:

Python

good

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 4(a)

AIM: Write a Python program to calculate the average value of the numbers in a given tuple of tuples. Original Tuple: ((10, 10, 10, 12), (30, 45, 56, 45), (81, 80, 39, 32), (1, 2, 3, 4)) Average value of the numbers of the said tuple of tuples: [30.5, 34.25, 27.0, 23.25]

Example:

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")
x = zip(a, b)
```

Output:

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")
x = zip(a, b)
```

Output:

Problem Statement: Define a function that will find the average values of numbers given in a tuple of tuple. Return the result. The tuple of tuple values are defined and the original tuple is printed. The function is called with the tuple values to find the average values. The result is returned and displayed.

Source Code:

```
def average_tuple(nums):
    result = [sum(x) / len(x) for x in zip(*nums)]
    return result

nums = ((10, 10, 10, 12), (30, 45, 56, 45), (81, 80,
39, 32), (1, 2, 3, 4))
print ("Original Tuple: ")
print(nums)
print("\nAverage value of the numbers of the said tuple
of tuples:\n",average_tuple(nums))
nums = ((1, 1, -5), (30, -15, 56), (81, -60, -39), (-
10, 2, 3))
print ("\nOriginal Tuple: ")
print(nums)
print("\nAverage value of the numbers of the said tuple
of tuples:\n",average_tuple(nums))
```

Output:

Original Tuple:

((10, 10, 10, 12), (30, 45, 56, 45), (81, 80, 39, 32), (1, 2, 3, 4))

Average value of the numbers of the said tuple of tuples:

[30.5, 34.25, 27.0, 23.25]

Original Tuple:

((1, 1, -5), (30, -15, 56), (81, -60, -39), (-10, 2, 3))

Average value of the numbers of the said tuple of tuples:

[25.5, -18.0, 3.75]

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 4(b)

AIM: Write a Python program calculate the product, multiplying all the numbers of a given tuple. Original Tuple: (4, 3, 2, 2, -1, 18) Product - multiplying all the numbers of the said tuple: -864

Problem Statement:

Define a function to multiply numbers. Declare a variable product and set it to 1. Run a loop for all elements in the list. Multiply all elements to the product. Return product. Declare a list. Pass list in our function. Print value returned by the function.

Source Code:

```
def mutiple_tuple(nums):
    temp = list(nums)
    product = 1
    for x in temp:
        product = product*x
    return product

nums = (4, 3, 2, 2, -1, 18)
print ("Original Tuple: ", nums)
print("Product - of the said
tuple:",mutiple_tuple(nums))

nums = (2, 4, 8, 8, 3, 2, 9)
print ("\nOriginal Tuple: ", nums)
print("Product - of the said
tuple:",mutiple_tuple(nums))
```

Output:

Original Tuple: (4, 3, 2, 2, -1, 18)

Product - of the said tuple: -864

Original Tuple: (2, 4, 8, 8, 3, 2, 9)

Product - of the said tuple: 27648

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 5(a)

AIM: Write A program to to sort a dictionary by value

Problem Statement: pass the dictionary to the sorted() method as the first value. use the items() method on the dictionary to retrieve its keys and values. write a lambda function to get the values retrieved with the item() method.

Source Code:

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
sorted_d = sorted(d.items(),
key=operator.itemgetter(1))
print('Dictionary in ascending order by value :
',sorted_d)
sorted_d = dict( sorted(d.items(),
key=operator.itemgetter(1),reverse=True))
print('Dictionary in descending order by value :
',sorted_d)
```

Output:

Original dictionary : {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}

Dictionary in ascending order by value : [(0, 0), (2, 1), (1, 2), (4, 3), (3, 4)]

Dictionary in descending order by value : {3: 4, 4: 3, 1: 2, 2: 1, 0: 0}

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 5(b)

AIM: Write a program to combined two dictionaries adding values for common keys

Problem Statement:

Imported the Counter function from the collections module. Then, assign the two dictionaries and called the **Counter()** function on dict1 and dict2. This method finds the common keys and sums their values. In the dict1, the key a has a value of 100, and in dict2, a has a value of 300. So the **Counter()** method added both values and returned a new value of 400. Hence we created a new dictionary.

Source Code:

```
from collections import Counter
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
d = Counter(d1) + Counter(d2)
print(d)
```

Output:

Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 6(a)

AIM: Write a program for set methods

Problem Statement: Define set with values. Use the remove method to remove the specified value. Use the add method to add the new value. Use the clear method to clear the values.

Source Code:

```
#Write a program for set methods
languages = {'Python', 'Java', 'English'}
# remove English from the set
languages.remove('English')
print(languages)
prime_numbers = {2, 3, 5, 7}
# add 11 to prime_numbers
prime_numbers.add(11)
print(prime_numbers)
numbers = {1, 2, 3, 4}
new_numbers = numbers
new_numbers.add(5)
print('numbers: ', numbers)
print('new_numbers: ', new_numbers)
#set of vowels
vowels = {'a', 'e', 'i', 'o', 'u'}
print('Vowels (before clear):', vowels)

# clearing vowels
vowels.clear()
print('Vowels (after clear):', vowels)
```

Output:

{'Python', 'Java'}

{2, 3, 5, 7, 11}

numbers: {1, 2, 3, 4, 5}

new_numbers: {1, 2, 3, 4, 5}

Vowels (before clear): {'e', 'i', 'u', 'a', 'o'}

Vowels (after clear): set()

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 6(b)

AIM: Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).

Problem Statement: It creates an empty list called "l" and then uses a for loop to iterate through the range of numbers from 1 to 31. For each number in the range, the program raises it to the power of 2 and appends the result to the "l" list. After the for loop completes, the program then prints out the first 5 elements of the "l" list using list slicing. The syntax "l[:5]" means to return the first 5 elements of the list. Then the program prints the last 5 elements of the "l" list using list slicing. The syntax "l[-5:]" means to return the last 5 elements of the list.

Source Code:

```
def printValues():
    l = list()
    for i in range(1, 31):
        l.append(i ** 2)
    print(l[:5])
    print(l[-5:])

printValues()
```

Output:

[1, 4, 9, 16, 25]

[676, 729, 784, 841, 900]

Conclusion: The above experiment has been executed successfully

EXPERIMENT 7(a)

AIM: Write a python program to listify the list of given strings individually using python map

Problem Statement: Define a list and display the list. The map() function takes two inputs as a function and an iterable object. The function that is given to map() is a normal function, and it will iterate over all the values present in the iterable object given.

Source Code:

```
color = ['Red', 'Blue', 'Black', 'White', 'Pink']
print("Original list: ")
print(color)
print("\nAfter listify the list of strings are:")
result = list(map(list, color))
print(result)
```

Output:

Original list:

['Red', 'Blue', 'Black', 'White', 'Pink']

After listify the list of strings are:

[['R', 'e', 'd'], ['B', 'l', 'u', 'e'], ['B', 'l', 'a', 'c', 'k'], ['W', 'h', 'i', 't', 'e'], ['P', 'i', 'n', 'k']]

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 7(b)

AIM: Write a python program to add two given lists and find the difference between lists using map function.

Problem Statement: define the lists and display the lists. Using the map function with lambda expression add the lists and print the values.

Source Code:

```
nums1 = [1, 2, 3]
nums2 = [4, 5, 6]
nums3 = [7, 8, 9]
print("Original list: ")
print(nums1)
print(nums2)
print(nums3)
result = map(lambda x, y, z: x + y + z, nums1, nums2, nums3)
print("\nNew list after adding above three lists:")
print(list(result))
```

Output:

Original list:

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

New list after adding above three lists:

[12, 15, 18]

Conclusion: The above experiment has been executed successfully

EXPERIMENT 8(a)

AIM: Write a Python program that takes a text file as input and returns the number of words of a given text file.

Problem Statement: Open a text file in read mode. Read each line by for loop. Count each words in a line using split() & len(). Print the output.

Source Code:

```
file = open("myfile.txt","r")
count=0
for l in file:
    words=l.split(" ")
    count=count+len(words)
file.close()
print("Number of words in a text file:", count)
```

Expected Output:

Number of words in a text file: 12

Conclusion:

The above experiment has been executed successfully

Write a Python program that takes a text file as input and returns the number of words, lines and characters of a given text file.

```
f=open('myfile.txt',mode='r')
nwords=0
nchars=0
nlines=0
for l in f:
    nlines=nlines+1
    l=l.strip("\n")
    nchars=nchars+len(l)
    l1=l.split()
    nwords=nwords+len(l1)
f.close()
print("Number of lines:",nlines)
print("Number of words:",nwords)
print("Number of characters", nchars)
```

EXPERIMENT 8(b)

AIM: Write a Python program to read a file line by line store it into a variable's

Souce Code:

```
f=open('myfile.txt','r')
l=f.readlines()
for a in l:
   print(a)
f.close()
```

Output:

hello

audisankara

I love Python

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 9(a)

AIM: How to count frequency of unique values in a NumPy array

Source Code:

```
import numpy as np
a = np.array( [10,10,20,10,20,20,20,30, 30,50,40,40] )
print("Original array:")
print(a)
unique_elements, counts_elements = np.unique(a,
return_counts=True)
print("Frequency of unique values of the said array:")
print(np.asarray((unique_elements, counts_elements)))
```

Expected Output:

Original array:

[10 10 20 10 20 20 20 30 30 50 40 40]

Frequency of unique values of the said array:

[[10 20 30 40 50]

[3 4 2 2 1]]

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 9(b)

AIM: how to calculate the sum of every row in a NumPy array in a python?

Source Code:

```
import numpy as np
x = np.array([[0,1],[2,3]])
print("Original array:")
print(x)
print("Sum of all elements:")
print(np.sum(x))
print("Sum of each column:")
print(np.sum(x, axis=0))
print("Sum of each row:")
print(np.sum(x, axis=1))
```

Expected Output:

Original array:

[[0 1]

[2 3]]

Sum of all elements:

6

Sum of each column:

[2 4]

Sum of each row:

[1 5]

Conclusion: The above experiment has been executed successfully

EXPERIMENT 10(a)

AIM: Create a class ATM and define ATM operations to create account, deposit, check balance, withdraw and delete account. Use constructor to initialize members.

Program:

```
class Bank_Account:
    def __init__(self):
        self.balance=0
        print("Hello!!! Welcome to the Deposit &
Withdrawal Machine")
    def deposit(self):
        amount=float(input("Enter amount to be
Deposited: "))
        self.balance += amount
        print("\n Amount Deposited:",amount)
    def withdraw(self):
        amount = float(input("Enter amount to be
Withdrawn: "))
        if self.balance>=amount:
            self.balance-=amount
            print("\n You Withdrew:", amount)
        else:
            print("\n Insufficient balance ")

    def display(self):
        print("\n Net Available Balance=",self.balance)
s = Bank_Account()
s.deposit()
s.withdraw()
s.display()
```

Expected Output:

Hello!!! Welcome to the Deposit & Withdrawal Machine

Enter amount to be Deposited: 5000

Amount Deposited: 5000.0

Enter amount to be Withdrawn: 2000

You Withdrew: 2000.0

Net Available Balance= 3000.0

Conclusion:

The above experiment has been executed successfully

EXPERIMENT 10(b)

AIM: Write a Python class Employee with attributes like emp_id, emp_name, emp_salary, and emp_department and methods like calculate_emp_salary, emp_assign_department, and rint_employee_details.

Sample Employee Data:
"ADAMS", "E7876", 50000, "ACCOUNTING"
"JONES", "E7499", 45000, "RESEARCH"
"MARTIN", "E7900", 50000, "SALES"
"SMITH", "E7698", 55000, "OPERATIONS"

- Use 'assign_department' method to change the department of an employee.
- Use 'print_employee_details' method to print the details of an employee.
- Use 'calculate_emp_salary' method takes two arguments: salary and hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula:

  overtime = hours_worked – 50

  Overtime amount = (overtime * (salary / 50))

Source Code:

```
class Employee:
    def __init__(self, name, emp_id,
salary, department):
        self.name = name
        self.id = emp_id
        self.salary = salary
        self.department = department
```

```
def calculate_salary(self, salary,
hours_worked):
        overtime = 0
        if hours_worked > 50:
            overtime = hours_worked - 50
        self.salary = self.salary +
(overtime * (self.salary / 50))

    def assign_department(self,
emp_department):
        self.department = emp_department

    def print_employee_details(self):
        print("\nName: ", self.name)
        print("ID: ", self.id)
        print("Salary: ", self.salary)
        print("Department: ",
self.department)
        print("----------------------")

employee1 = Employee("ADAMS", "E7876",
50000, "ACCOUNTING")
employee2 = Employee("JONES", "E7499",
45000, "RESEARCH")
employee3 = Employee("MARTIN", "E7900",
50000, "SALES")
employee4 = Employee("SMITH", "E7698",
55000, "OPERATIONS")

print("Original Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()
```

```
# Change the departments of employee1 and
employee4
employee1.assign_department("OPERATIONS")
employee4.assign_department("SALES")
# Now calculate the overtime of the
employees who are eligible:
employee2.calculate_salary(45000, 52)
employee4.calculate_salary(45000, 60)

print("Updated Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()
```

Expected Output:

Original Employee Details:

Name: ADAMS

ID: E7876

Salary: 50000

Department: ACCOUNTING

Name: JONES

ID: E7499

Salary: 45000

Department: RESEARCH

Name: MARTIN

ID: E7900

Salary: 50000

Department: SALES

Name: SMITH

ID: E7698

Salary: 55000

Department: OPERATIONS

Updated Employee Details:

Name: ADAMS

ID: E7876

Salary: 50000

Department: OPERATIONS

Name: JONES

ID: E7499

Salary: 46800.0

Department: RESEARCH

Name: MARTIN

ID: E7900

Salary: 50000

Department: SALES

Name: SMITH

ID: E7698

Salary: 66000.0

Department: SALES

Conclusion: The above experiment has been executed successfully

EXPERIMENT 11(a)

AIM: Wrte a program for pie charts, bar charts using pandas data frames

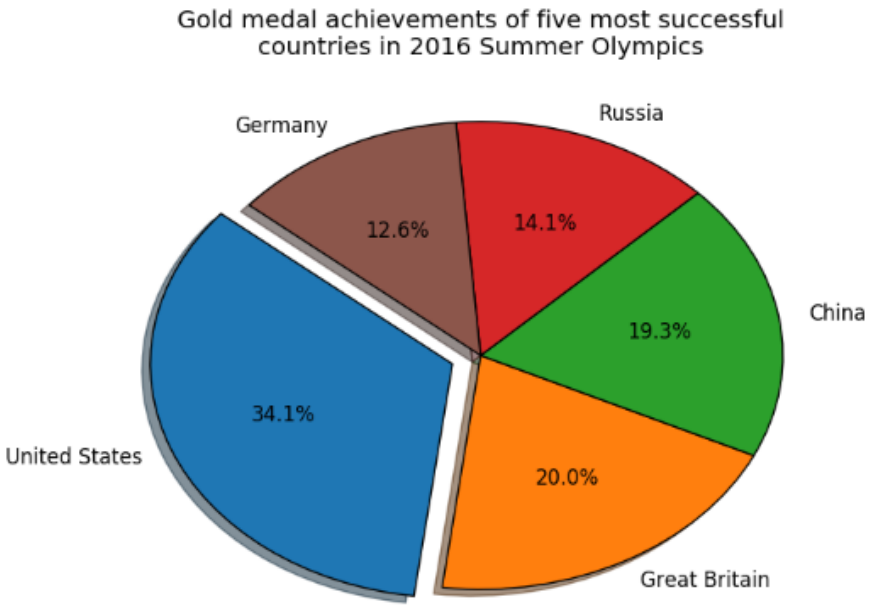
Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt

# creating dataframe
dataFrame = pd.DataFrame({
   "Car": ['BMW', 'Lexus', 'Tesla',
'Mustang', 'Mercedes',
'Jaguar'],"Reg_Price": [7000, 1500, 5000,
8000, 9000, 6000]
})

# plot a Pie Chart for Registration Price
column with label Car column
plt.pie(dataFrame["Reg_Price"], labels =
dataFrame["Car"])
plt.show()
```

Expected Output:



Conclusion:

The above experiment has been executed successfully

EXPERIMENT 11(b)

AIM: Write a Pandas program to create a Pivot table and find the total sale amount region wise, manager wise, sales man wise

Source Code:

```
import pandas as pd
df = pd.read_excel('E:\SaleData.xlsx')
table =
pd.pivot_table(df,index=["Region","Manager"
,"SalesMan"], values="Sale_amt")
print(table.query('Manager ==
["Douglas"]'))
```

Expected Output:

```
Region  Manager SalesMan  Sale_amt
Central Douglas John      41338.666667
East    Douglas Karen     16068.000000
West    Douglas Michael   33418.000000
```

Conclusion: The above experiment has been executed successfully