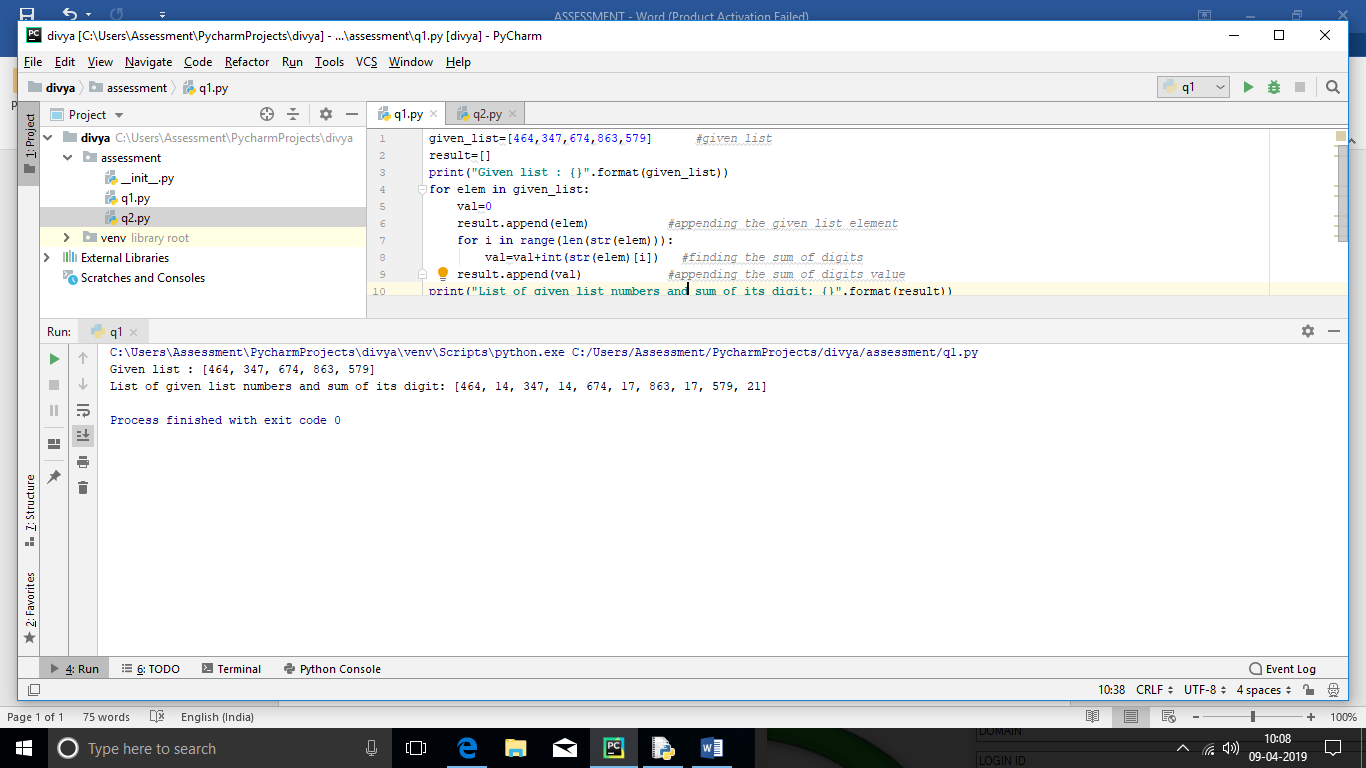
**ASSESSMENT**

**1. lst1 = [ 464,347,674,863,579].  Find the sum of digits of each number.  Store the number  and the sum of its digits in another list.**

**CODE:**

given\_list=[464,347,674,863,579] *#given list*result=[]  
print(**"Given list : {}"**.format(given\_list))  
**for** elem **in** given\_list:  
 val=0  
 result.append(elem) *#appending the given list element* **for** i **in** range(len(str(elem))):  
 val=val+int(str(elem)[i]) *#finding the sum of digits* result.append(val) *#appending the sum of digits value*print(**"List of given list numbers and sum of its digit: {}"**.format(result))

**OUTPUT:**



Given list : [464, 347, 674, 863, 579]

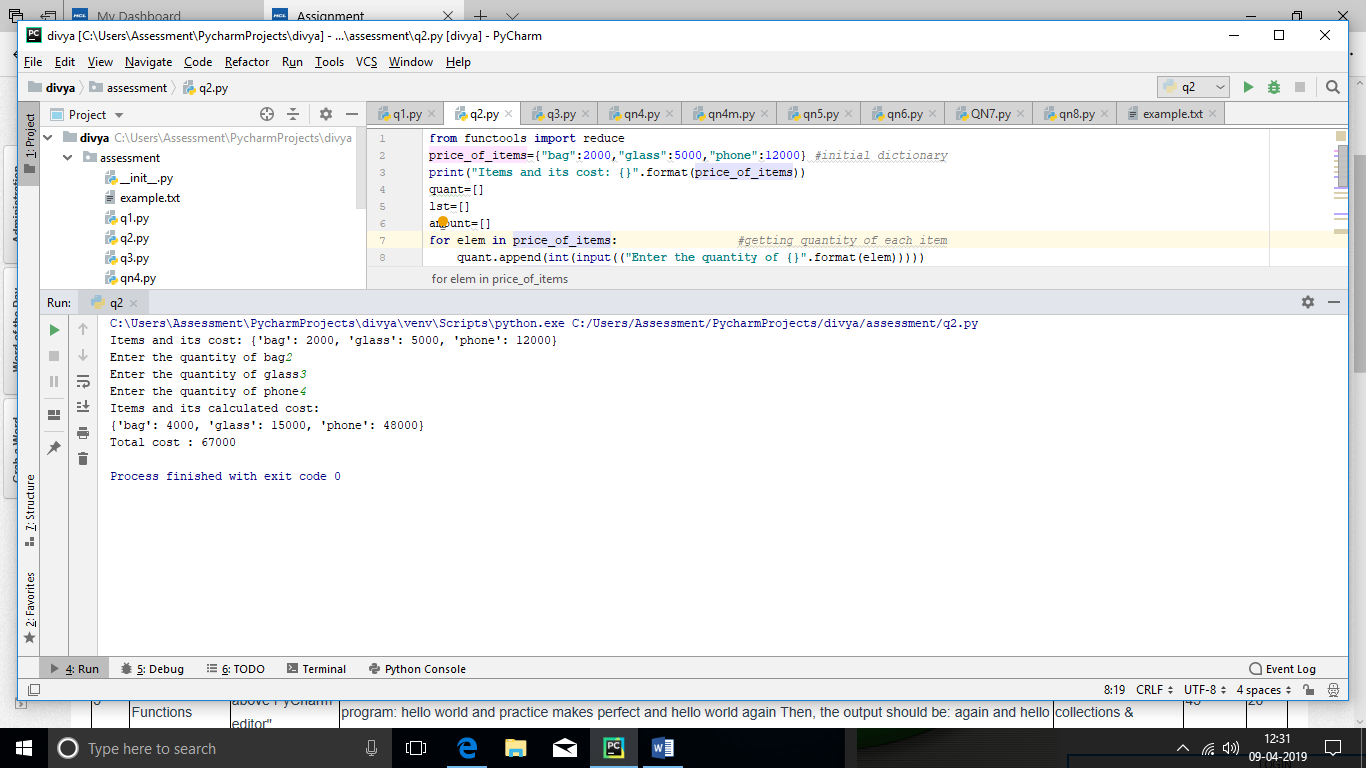
List of given list numbers and sum of its digit: [464, 14, 347, 14, 674, 17, 863, 17, 579, 21]

**2. "  Some of the items and its unit price are stored as below: price\_of\_items={""bag"":2000,""glass"":5000,""phone"":12000} Get the quantity as input  for each item and store the item name and amount (qty \* price) in another dictionary and display. Calculate the total amount and display.(dont store the total amount in dictionary)"**

**CODE:**

**from** functools **import** reduce  
price\_of\_items={**"bag"**:2000,**"glass"**:5000,**"phone"**:12000} *#initial dictionary*print(**"Items and its cost: {}"**.format(price\_of\_items))  
quant=[]  
lst=[]  
amount=[]  
**for** elem **in** price\_of\_items: *#getting quantity of each item* quant.append(int(input((**"Enter the quantity of {}"**.format(elem)))))  
**for** elem **in** price\_of\_items.values():  
 lst.append(elem) *#storing amount per 1 item in separate list***for** i **in** range(len(lst)): *#calculating amount per item \* quntity of items* amount.append(lst[i]\*quant[i])  
print(**"Items and its calculated cost:"**)  
print(dict(zip(price\_of\_items.keys(),amount))) *#calculated dictionary*total=reduce(**lambda** x,y:x+y,amount) *#calculating total*print(**"Total cost : {}"**.format(total))

**OUTPUT:**



Items and its cost: {'bag': 2000, 'glass': 5000, 'phone': 12000}

Enter the quantity of bag2

Enter the quantity of glass3

Enter the quantity of phone4

Items and its calculated cost:

{'bag': 4000, 'glass': 15000, 'phone': 48000}

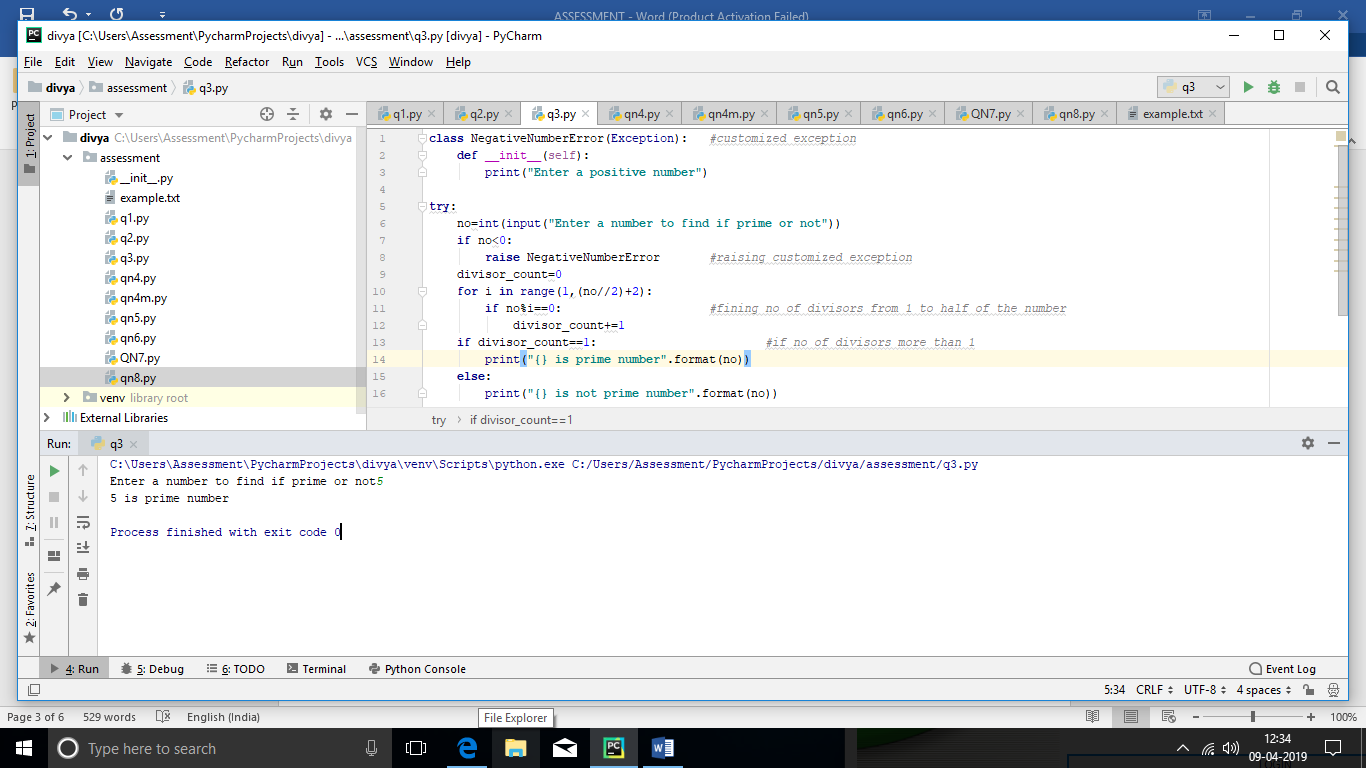
Total cost : 67000

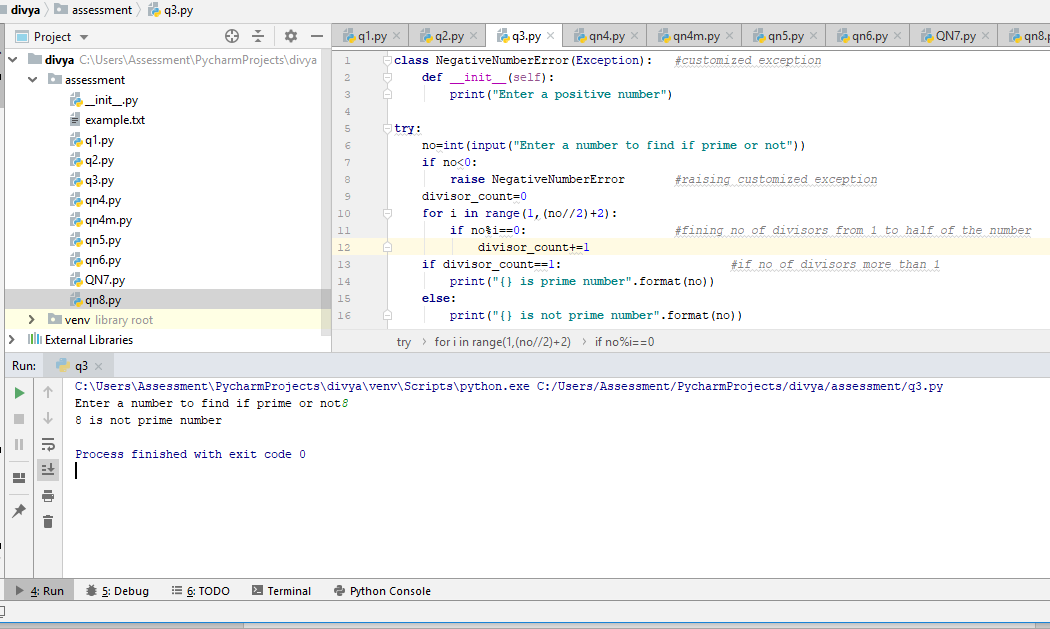
**3.Write a function to check whether a given number is prime or not .  If the input is negative, create a custom exception and handle it.**

**CODE:**

**class** NegativeNumberError(Exception): *#customized exception* **def** \_\_init\_\_(self):  
 print(**"Enter a positive number"**)  
  
**try**:  
 no=int(input(**"Enter a number to find if prime or not"**))  
 **if** no<0:  
 **raise** NegativeNumberError *#raising customized exception* divisor\_count=0  
 **for** i **in** range(1,(no//2)+2):  
 **if** no%i==0: *#fining no of divisors from 1 to half of the number* divisor\_count+=1  
 **if** divisor\_count==1: *#if no of divisors more than 1* print(**"{} is prime number"**.format(no))  
 **else**:  
 print(**"{} is not prime number"**.format(no))  
  
**except** NegativeNumberError **as** err:  
 print(err)

**OUTPUT:**





Enter a number to find if prime or not5

5 is prime number

Enter a number to find if prime or not8

8 is not prime number

**4. Create a  function to generate fibonacci series for a given number of terms.  Create  a new file, import fibonacci function and test it**

**CODE:**

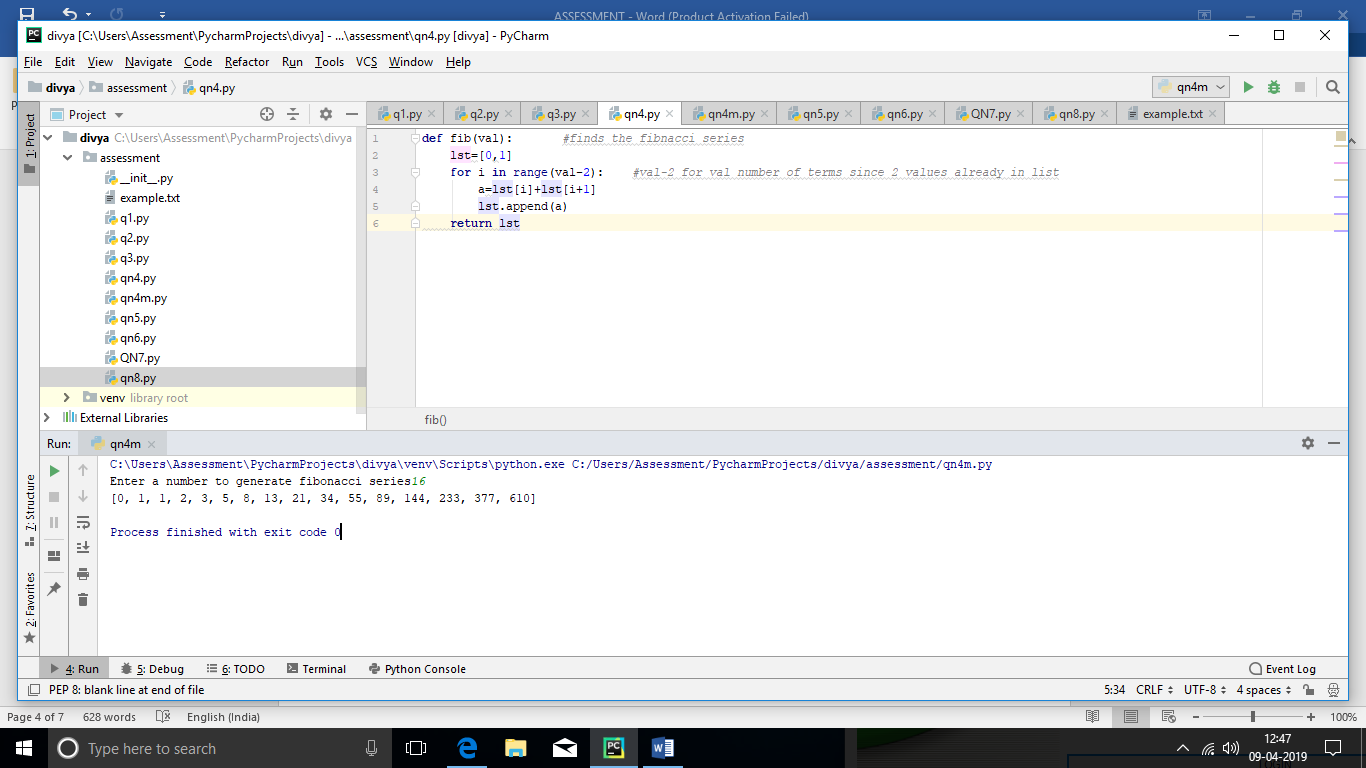
**IN ONE MODULE:**

**def** fib(val): *#finds the fibnacci series* lst=[0,1]  
 **for** i **in** range(val-2):*#val-2 for since 2 values already in list* a=lst[i]+lst[i+1]  
 lst.append(a)  
 **return** lst

ANOTHER MODULE:

**from** assessment **import** qn4*#importing function from assessment pkg and qn4 module*inp=int(input(**"Enter a number to generate fibonacci series"**))  
print(qn4.fib(inp))

**OUTPUT:**



Enter a number to generate fibonacci series 16

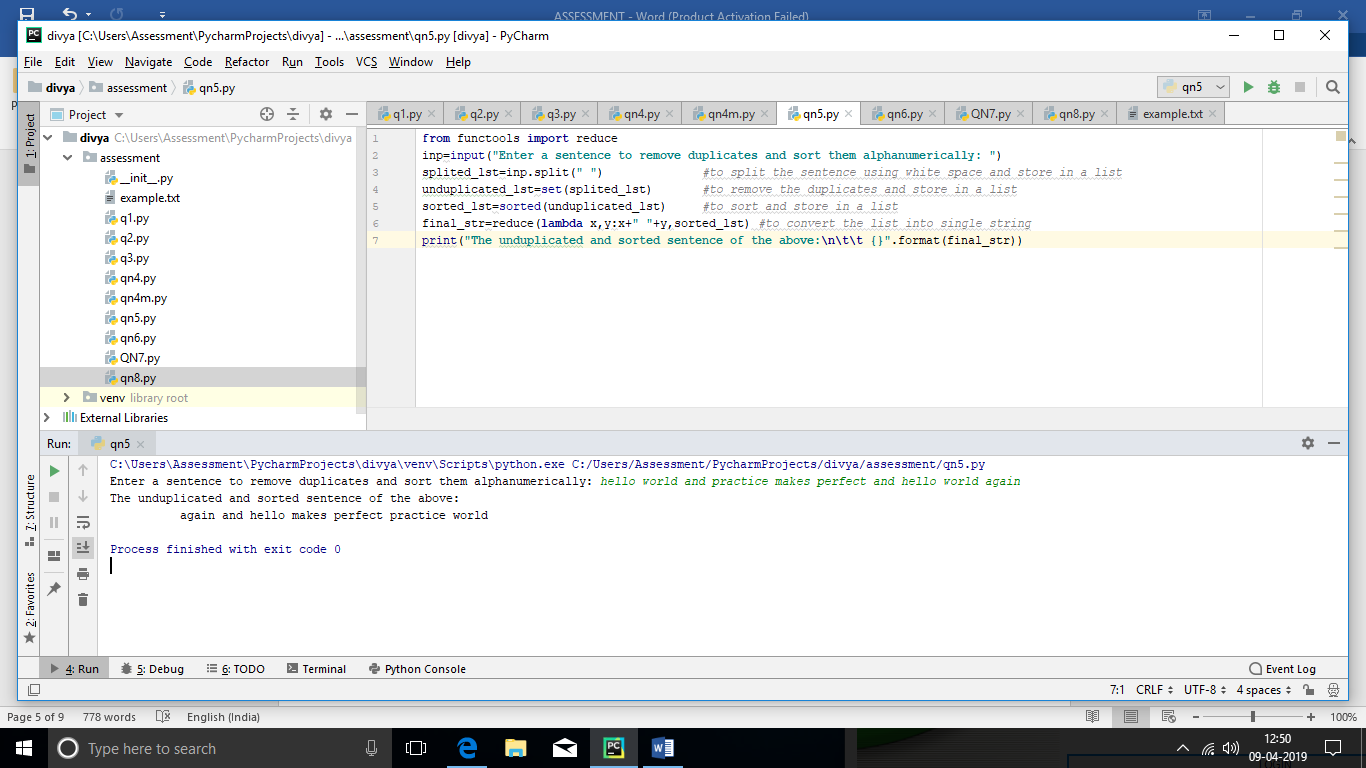
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]

**5. "Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically. Suppose the following input is supplied to the program: hello world and practice makes perfect and hello world again Then, the output should be: again and hello makes perfect practice world"**

**CODE:**

**from** functools **import** reduce  
inp=input(**"Enter a sentence to remove duplicates and sort them alphanumerically: "**)  
splited\_lst=inp.split(**" "**) *#to split the sentence using white space and store in a list*unduplicated\_lst=set(splited\_lst) *#to remove the duplicates and store in a list*sorted\_lst=sorted(unduplicated\_lst) *#to sort and store in a list*final\_str=reduce(**lambda** x,y:x+**" "**+y,sorted\_lst) *#to convert the list into single string*print(**"The unduplicated and sorted sentence of the above:\n\t\t {}"**.format(final\_str))

**OUTPUT:**



Enter a sentence to remove duplicates and sort them alphanumerically: hello world and practice makes perfect and hello world again

The unduplicated and sorted sentence of the above:

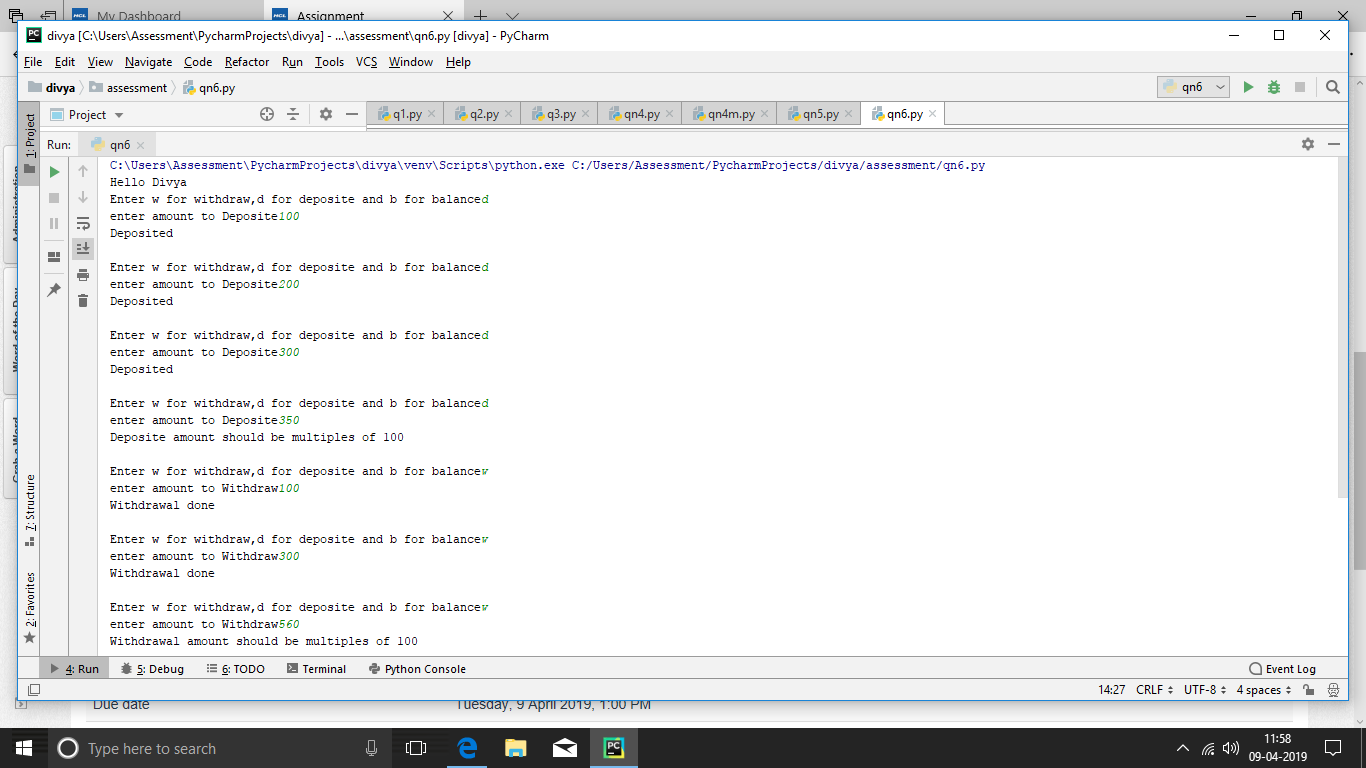
again and hello makes perfect practice world

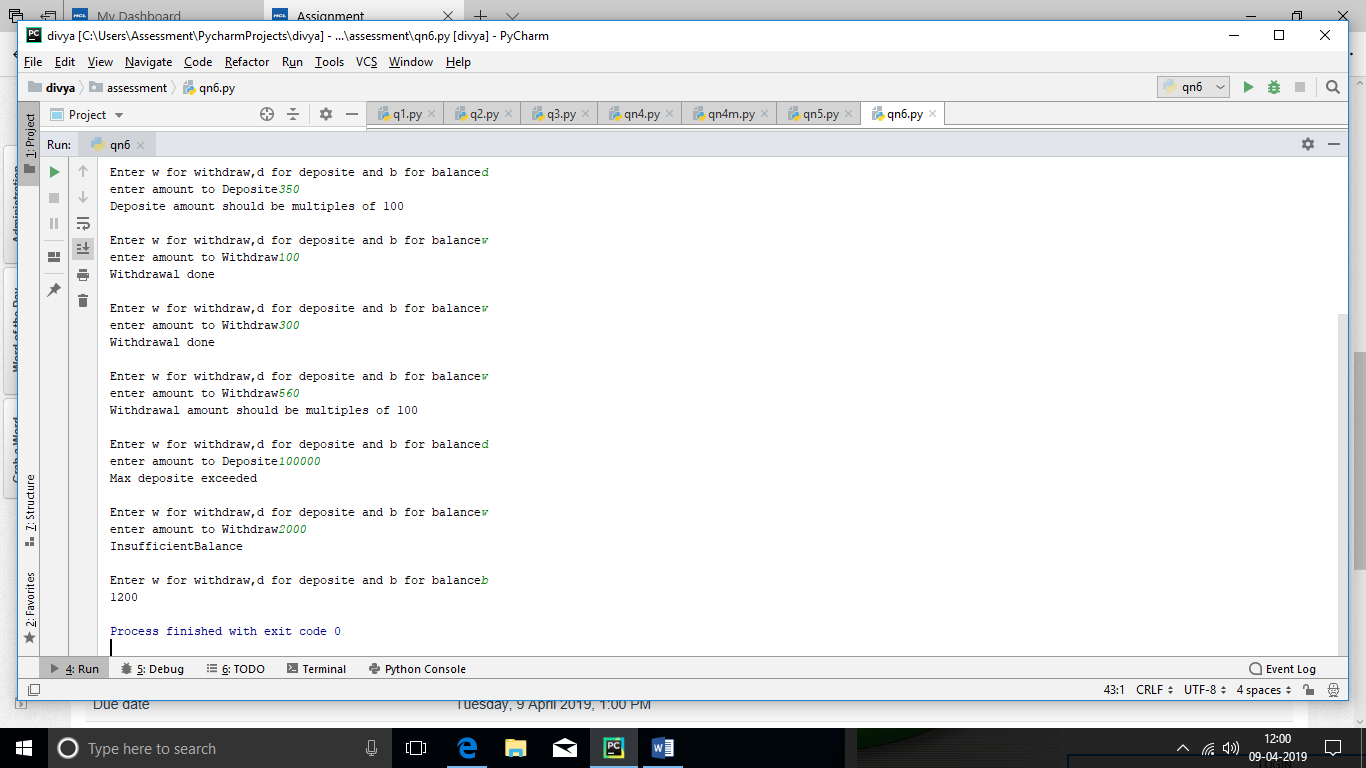
**6. Create a class ‘Account’ with a constructor that accepts ‘customer name’ and balance. Default balance amount should be Rs.1000/-. Create two methods one for deposit and another for withdraw. Allowed deposits/Withdraw should be multiples of 100’s. Customer should maintain minimum balance of Rs.500/- and maximum allowed deposit should be Rs.10000/-. Create an object of an Account class and verify both methods Note: Include exception handling wherever applicable"**

**CODE**:

**class** Bank:  
 **def** \_\_init\_\_(self,name,bal):  
 self.name=name  
 self.bal=bal  
 **def** withdrawal(self,amt,balance): *#calculate withdrawals* **if** amt%100==0: *#checking multiples of 100 condition* balance=balance-amt  
 print(**"Withdrawal done"**)  
 print()  
 **return** balance  
 **else**:  
 print(**"Withdrawal amount should be multiples of 100"**)  
 print()  
 **return** balance  
 **def** deposite(self,amt,balance): *#calculate deposites* **if** amt % 100 == 0: *#checking multiples of 100 condition* **if** amt<=10000: *#checking max deposite amount 10000* balance=balance+amt  
 print(**"Deposited"**)  
 print()  
 **return** balance  
 **else**:  
 print(**"Max deposite exceeded"**)  
 print()  
 **return** balance  
 **else**:  
 print(**"Deposite amount should be multiples of 100"**)  
 print()  
 **return** balance  
balance=1000 *#initial balance 1000*obj1=Bank(**"Divya"**,balance)  
inp=**" "**print(**"Hello {}"**.format(obj1.name))  
**while**(inp!=**"b"**.casefold()):  
 inp=input(**"Enter w for withdraw,d for deposite and b for balance"**)  
 **if** inp== **"w"**.casefold():  
 amount=int(input(**"enter amount to Withdraw"**))  
 **if** balance>=(500 + amount): *#maintaining balance 500* balance=obj1.withdrawal(amount,balance)  
 **else**:  
 print(**"InsufficientBalance"**)  
 print()  
 **elif** inp== **"d"**.casefold():  
 amount = int(input(**"enter amount to Deposite"**))  
 balance = obj1.deposite(amount,balance)  
 **else**:  
 **if** inp==**"b"**.casefold():  
 **pass  
 else**:  
 print(**"Unexpected input,Enter b,w or d"**)  
**else**:  
 print(balance)

**OUTPUT:**





**OUTPUT**:

Hello Divya

Enter w for withdraw,d for deposite and b for balanced

enter amount to Deposite100

Deposited

Enter w for withdraw,d for deposite and b for balanced

enter amount to Deposite200

Deposited

Enter w for withdraw,d for deposite and b for balanced

enter amount to Deposite300

Deposited

Enter w for withdraw,d for deposite and b for balanced

enter amount to Deposite350

Deposite amount should be multiples of 100

Enter w for withdraw,d for deposite and b for balancew

enter amount to Withdraw100

Withdrawal done

Enter w for withdraw,d for deposite and b for balancew

enter amount to Withdraw300

Withdrawal done

Enter w for withdraw,d for deposite and b for balancew

enter amount to Withdraw560

Withdrawal amount should be multiples of 100

Enter w for withdraw,d for deposite and b for balanced

enter amount to Deposite100000

Max deposite exceeded

Enter w for withdraw,d for deposite and b for balancew

enter amount to Withdraw2000

InsufficientBalance

Enter w for withdraw,d for deposite and b for balanceb

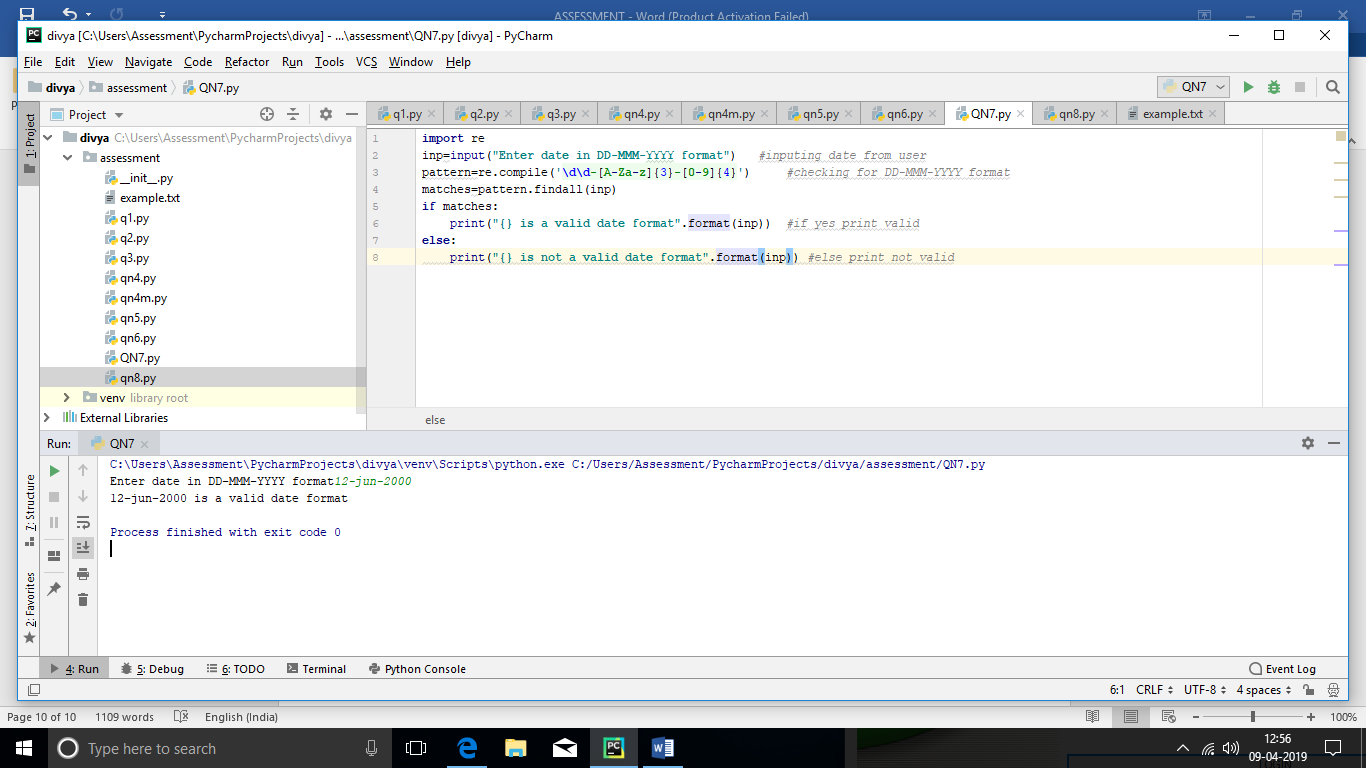
1200

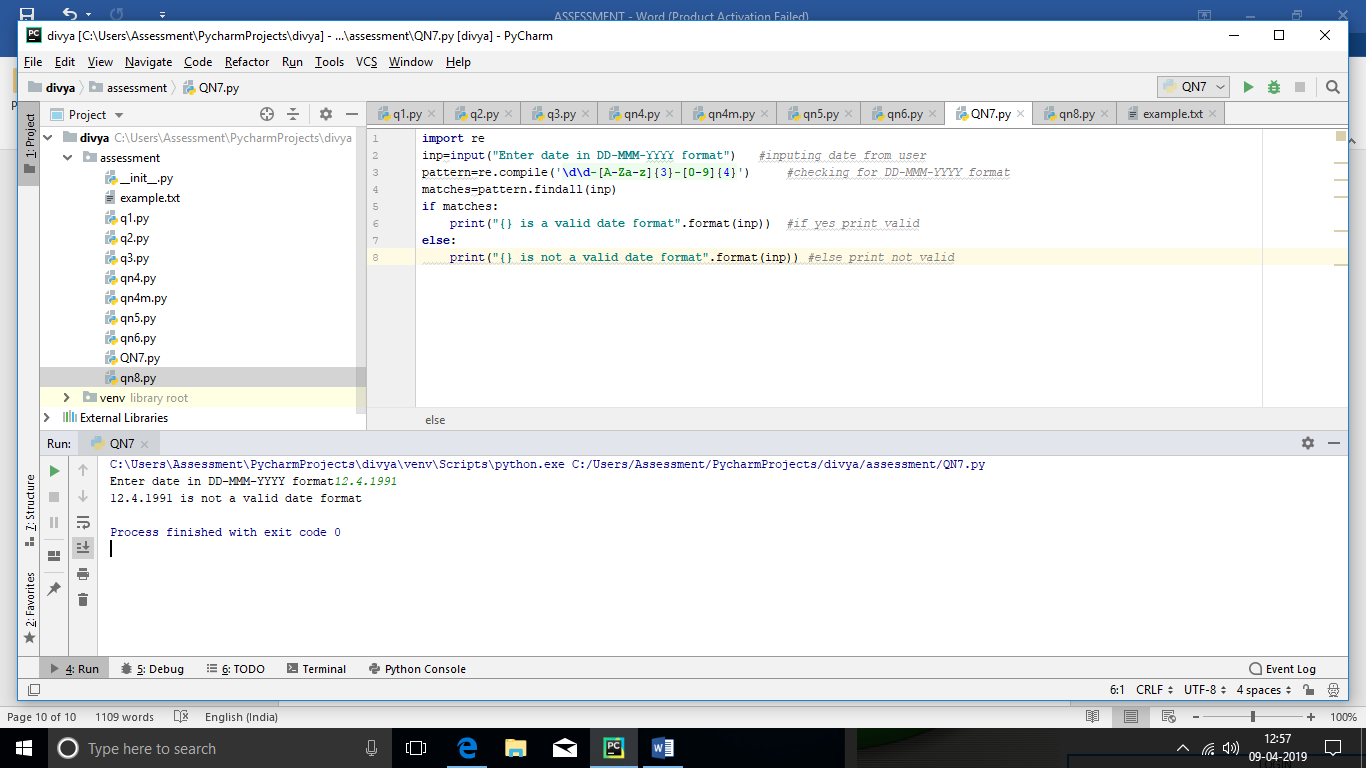
**7. "Write a regular expression pattern to input date as string and validate it is in the format ""DD-MMM-YYYY"". Eg. Input - 01-APR-2019, output - valid date format input - 1-APR-19, output - invalid date format input - 1-4-19, output - invalid date format"**

**CODE:**

**import** re  
inp=input(**"Enter date in DD-MMM-YYYY format"**) *#inputing date from user*pattern=re.compile(**'\d\d-[A-Za-z]{3}-[0-9]{4}'**) *#checking for DD-MMM-YYYY format*matches=pattern.findall(inp)  
**if** matches:  
 print(**"{} is a valid date format"**.format(inp)) *#if yes print valid***else**:  
 print(**"{} is not a valid date format"**.format(inp)) *#else print not valid*

**OUTPUT**



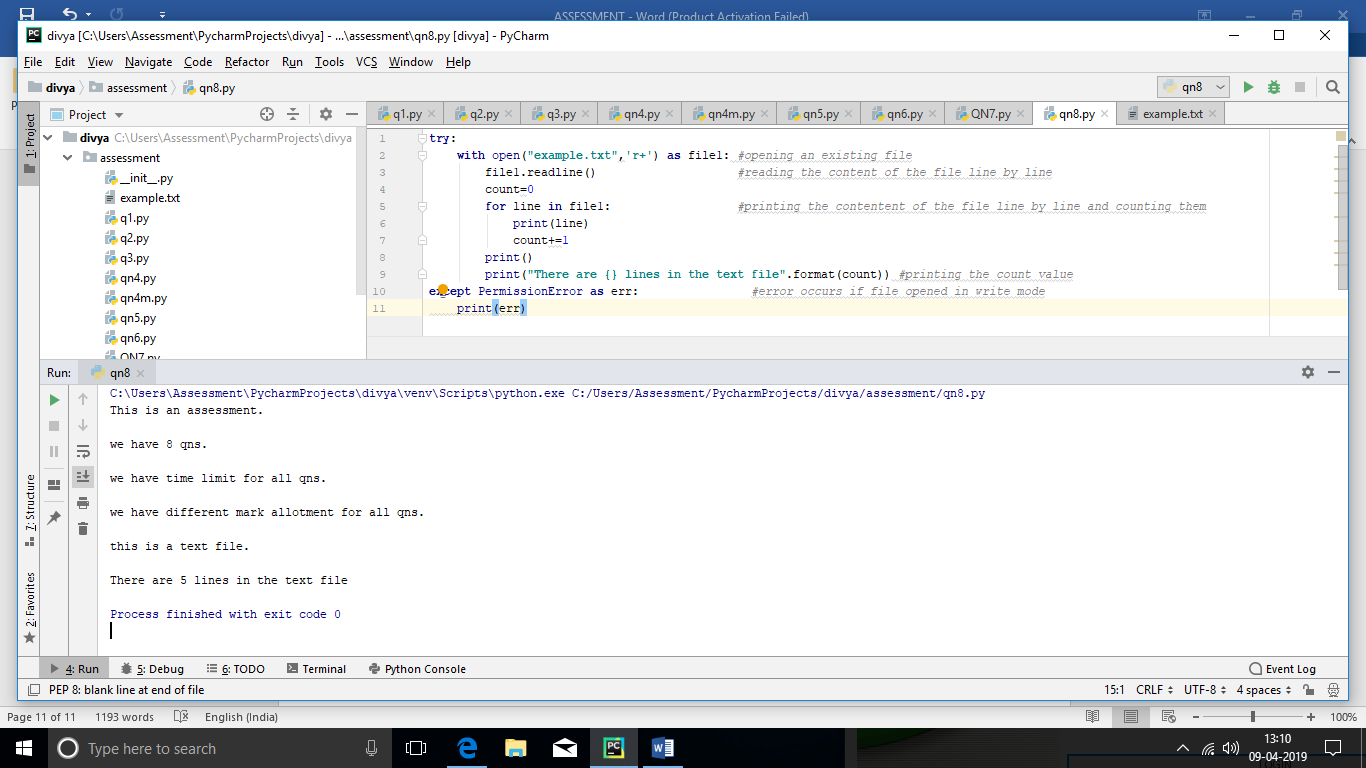


**8. "Write a program to read contents of a text file and print the no. of lines. Use exceptions & regular expressions as applicable."**

**CODE:**

**try**:  
 **with** open(**"example.txt"**,**'r+'**) **as** file1: *#opening an existing file* file1.readline() *#reading the content of the file line by line* count=0  
 **for** line **in** file1:*#print the content of the file line by line and counting* print(line)  
 count+=1  
 print()  
 print(**"There are {} lines in the text file"**.format(count)) **except** PermissionError **as** err: *#error occurs if file opened in write mode* print(err)

**OUTPUT:**



This is an assessment.

we have 8 qns.

we have time limit for all qns.

we have different mark allotment for all qns.

this is a text file.

There are 5 lines in the text file