# Assignment 1

1. ***COLLEGE LIBRARY***
   * **LIBRARY**
     1. List of Books
   * **Book**
     1. Book ID
     2. Title
     3. Author
     4. Publisher
   * **Librarian** 
     1. Name
     2. Employee ID
   * **Issued Book**
     1. Date of issue
     2. Book ID
     3. Borrower
   * **Borrower**
     1. Type
     2. Borrower’s ID
2. ***CLASSROOM***
   * **Classroom**
     1. Whiteboard
     2. Desk
     3. Bench
   * **Student**
     1. Roll no.
     2. Name
   * **Teacher**
     1. Name
     2. Department
     3. Expertise

# Assignment 2

* Facebook Account
  + Attributes
    - Name
    - Password
    - Email id
    - User name
    - Profile Picture
    - Cover Photo
    - Date of Birth
    - Friends
    - Relationship Status
    - Liked Pages
    - Following
    - Followers
  + Behaviours
    - Upload Photo
    - Upload Status
    - Send Friend Request
    - Accept Friend Request
    - Message Someone
    - View timeline
    - Delete Post/Photo
    - Report
    - Block
* Bank Account
  + Attributes
    - Name
    - Address
    - PAN number
    - Aadhar number
    - Balance
    - Deposit history
    - Withdrawal history
    - Loan
    - Credit score
  + Behaviours
    - Update Account
    - Deposit
    - Withdraw
    - Close Account
* Employee
  + Attributes
    - Name
    - Address
    - Job Description
    - Title
    - Salary
    - Phone number
  + Behaviours
    - Check in
    - Give Appraisals
    - Do work
    - Attend meetings
    - Take salary
    - Report to HR
    - Check out

# Assignment 3:

Customer: id, name, mobile number, email id

Product: id, name, price, description, image path (since image will be separately stored)

Shopping Cart:

List of Ordered\_item

Ordered\_item:Product id, product name, price, quantity

Order

Order id

Date of purchase

Shipping address

Customer id or Customer depending on design/logic

List of Ordered\_item

# Assignment 4:

**CODE:**

class Employee:

def \_\_init\_\_(self,first\_name,last\_name,pay):

self.first\_name=first\_name

self.last\_name=last\_name

self.pay=pay

self.email=first\_name+"."+last\_name+"@company.com"

def print(self):

print(emp1.first\_name + '\n' + emp1.last\_name + '\n' + emp1.email + '\n' + str(emp1.pay))

emp1=Employee("Mohandas","Gandhi",5000)

emp1.print()

**OUTPUT:**

Mohandas

Gandhi

Mohandas.Gandhi@company.com

5000

# Assignment 5:

**Code:**

class Employee:

def \_\_init\_\_(self,first\_name,last\_name,pay):

#self creates a variable for that specific instance

self.first\_name=first\_name

self.last\_name=last\_name

self.pay=pay

self.email=first\_name+"."+last\_name+"@company.com"

def getEmail(self):

return self.email

def getFullName(self):

return self.first\_name+" "+self.last\_name

def getPay(self):

return self.pay

emp1=Employee("Mohandas","Gandhi",5000)

print(emp1.getEmail())

print(emp1.getFullName())

print(emp1.getPay())

**Output:**

Mohandas.Gandhi@company.com

Mohandas Gandhi

5000

# Assignment 6:

The balance can be set to very high/low value accidentally. (#stmt1)

The balance can be accessed or changed by user of the class.

The balance can be set to non-permitted value (#stmt2)

Make balance a private variable ( \_\_balance)

def \_\_init\_\_(self, initial\_amount):

self.\_\_balance = initial\_amount

# Assignment 7:

**Code:**

class Dog:

def \_\_init\_\_(self,name):

self.name=name

self.tricks = []

def add\_trick(self,trick):

self.tricks.append(trick)

d=Dog('Fido')

e=Dog('Buddy')

d.add\_trick('roll over')

e.add\_trick('play dead')

print(d.tricks)

print(e.tricks)

**Output:**

['roll over']

['play dead']

Assignment 8

class Employee:

@classmethod

def from\_string(cls,emp\_str):

cls.fname,cls.lname,cls.pay=emp\_str.split('-')

return cls.fname,cls.lname,cls.pay

def \_\_init\_\_(self,first,last,pay):

self.firstname = first

self.lastname = last

self.pay = pay

def print(self):

print(fname,lname,pay)

emp\_1\_str = 'John-Abraham-50000'

emp\_1 = Employee.from\_string(emp\_1\_str)

emp\_1 = Employee.from\_string(emp\_1\_str)

print(emp\_1)



Assignment 9

class Store:

\_\_item\_count = 100

#adds to count to \_\_item\_count

def addItem(self,count):

self.\_\_item\_count = self.\_\_item\_count + count

#subtracts count from \_\_item\_count

def issueItem(self,count):

self.\_\_item\_count = self.\_\_item\_count - count

#returns \_\_item\_count

def getItemCount(self):

return self.\_\_item\_count

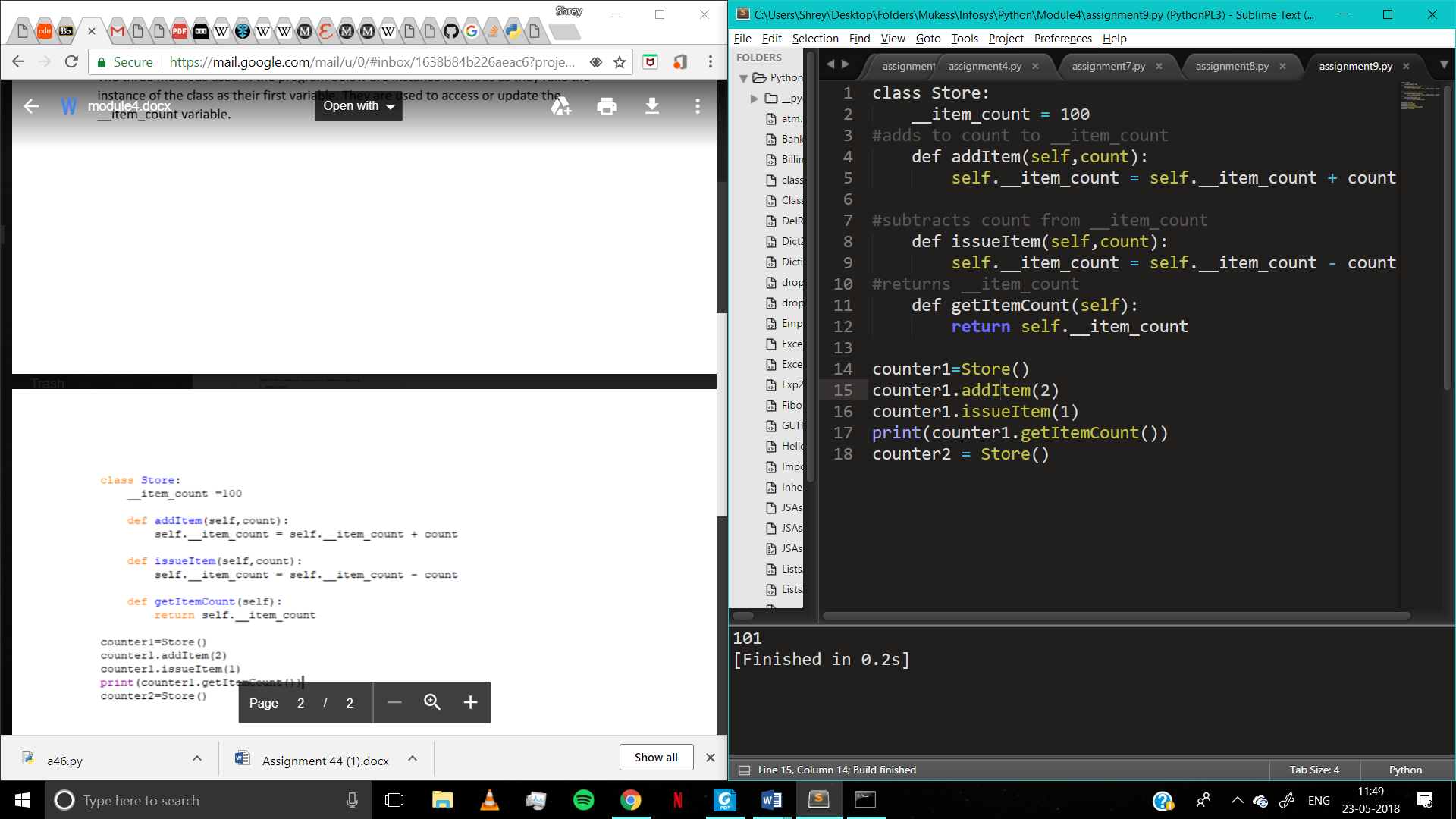
counter1=Store()

counter1.addItem(2)

counter1.issueItem(1)

print(counter1.getItemCount())

counter2 = Store()



Assignment 10

The objects have knowledge about class from which it is instantiated but the class has no knowledge of the objects that have been created. Python runtime creates only one copy of static and class members and all instances share the same; whereas individual copies of instance members are created with respective objects. The instance handle, self is not available inside class and static methods.

Assignment 11

class Book:

title = ""

author = ""

publisherInfo = ""

def \_\_init\_\_(self,title,author,publisherInfo):

self.title = title

self.author = author

self.publisherInfo = publisherInfo

def print(self,book):

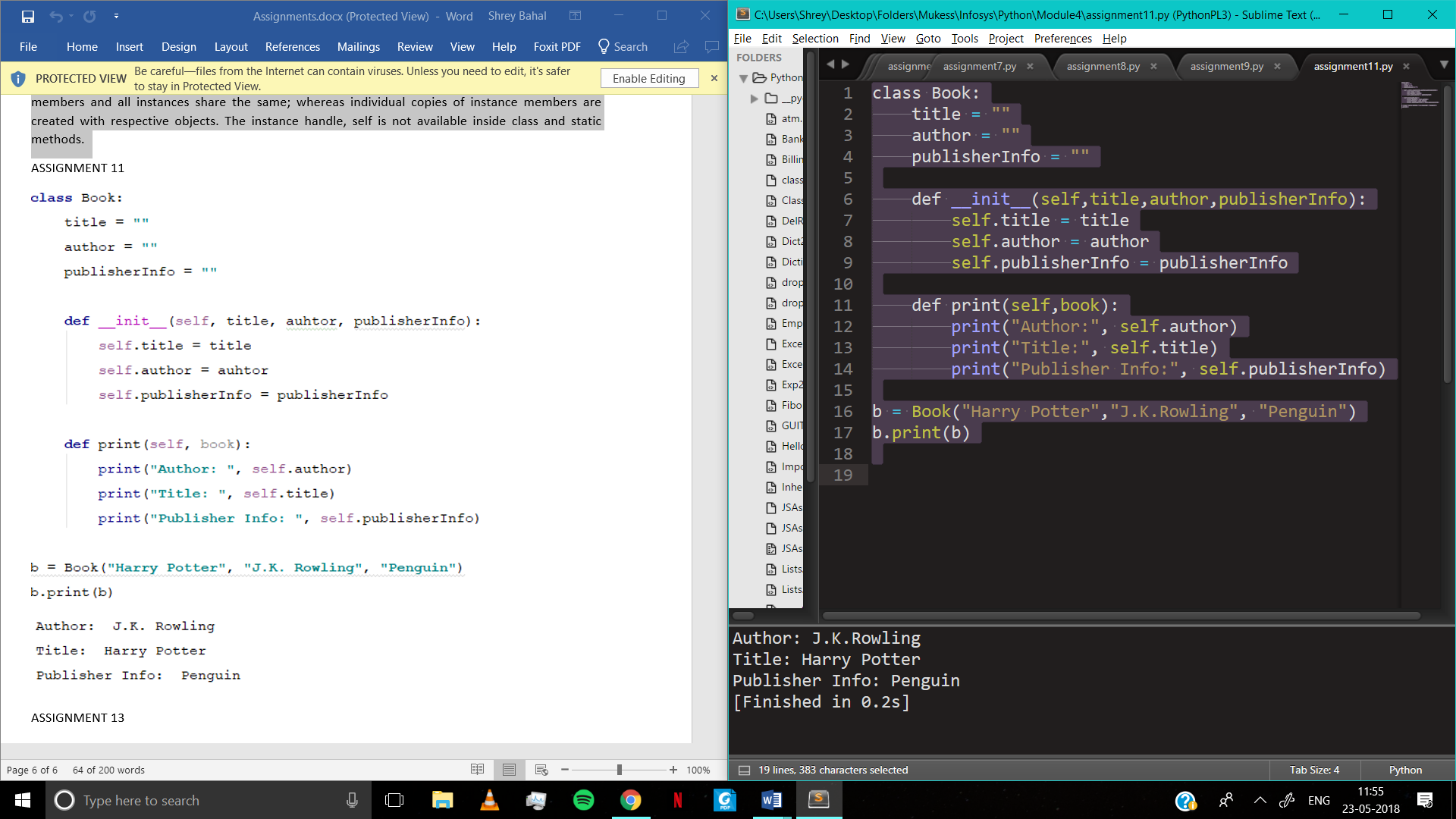
print("Author:", self.author)

print("Title:", self.title)

print("Publisher Info:", self.publisherInfo)

b = Book("Harry Potter","J.K.Rowling", "Penguin")

b.print(b)



Assignment 12

class calculator:

def \_\_init\_\_(self, num):

self.lastprime = num

def nextPrime(self):

self.num = self.lastprime + 1

v=0

if (self.num > 1):

for i in range(2, self.num):

if((self.num % i) != 0):

v+=1

else:

break

if(v==self.num-2):

print(self.num, " is a prime number")

else:

# print(self.num, " is not a prime number")

pass

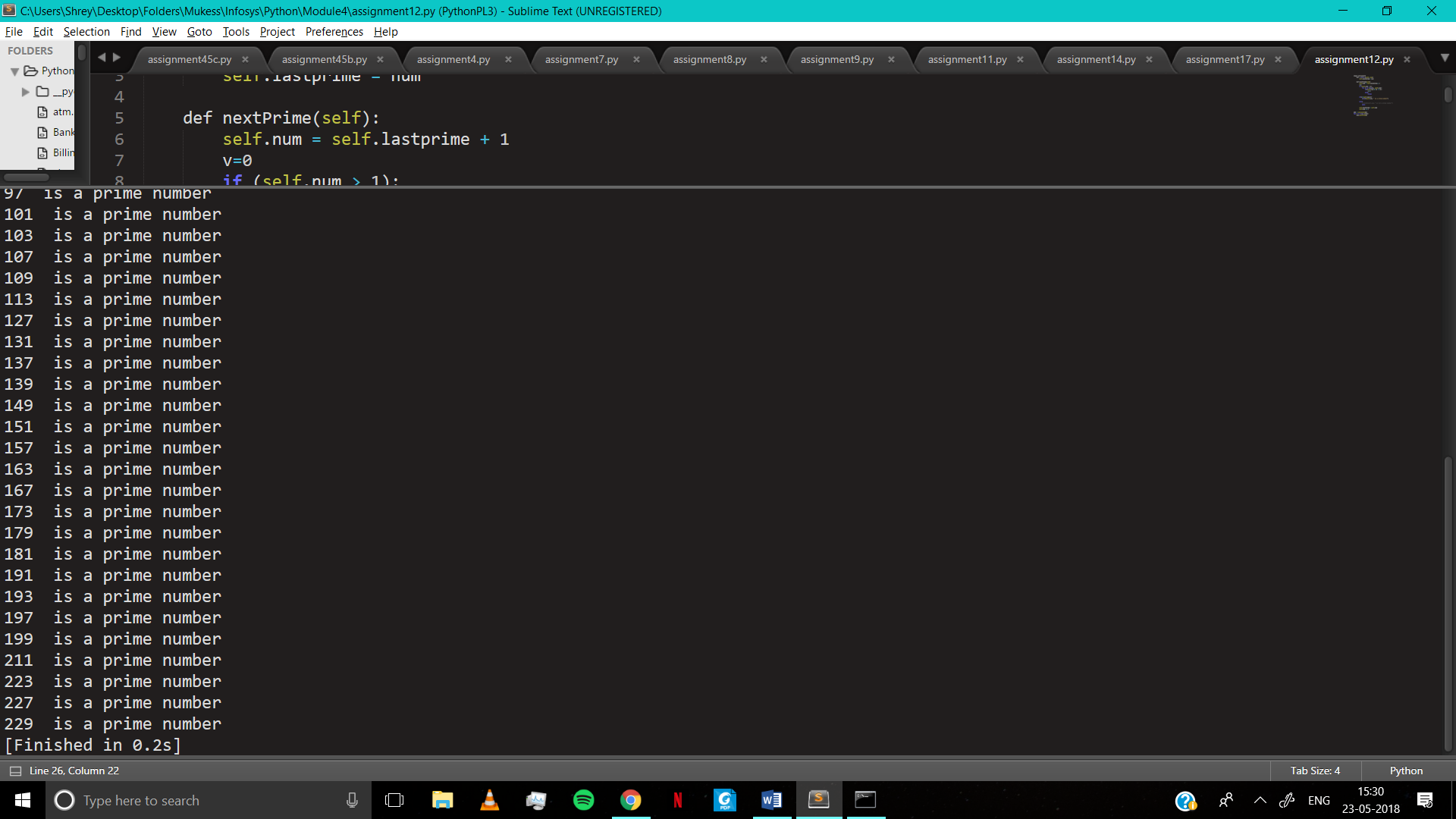
self.lastprime = self.num

self.num += 1

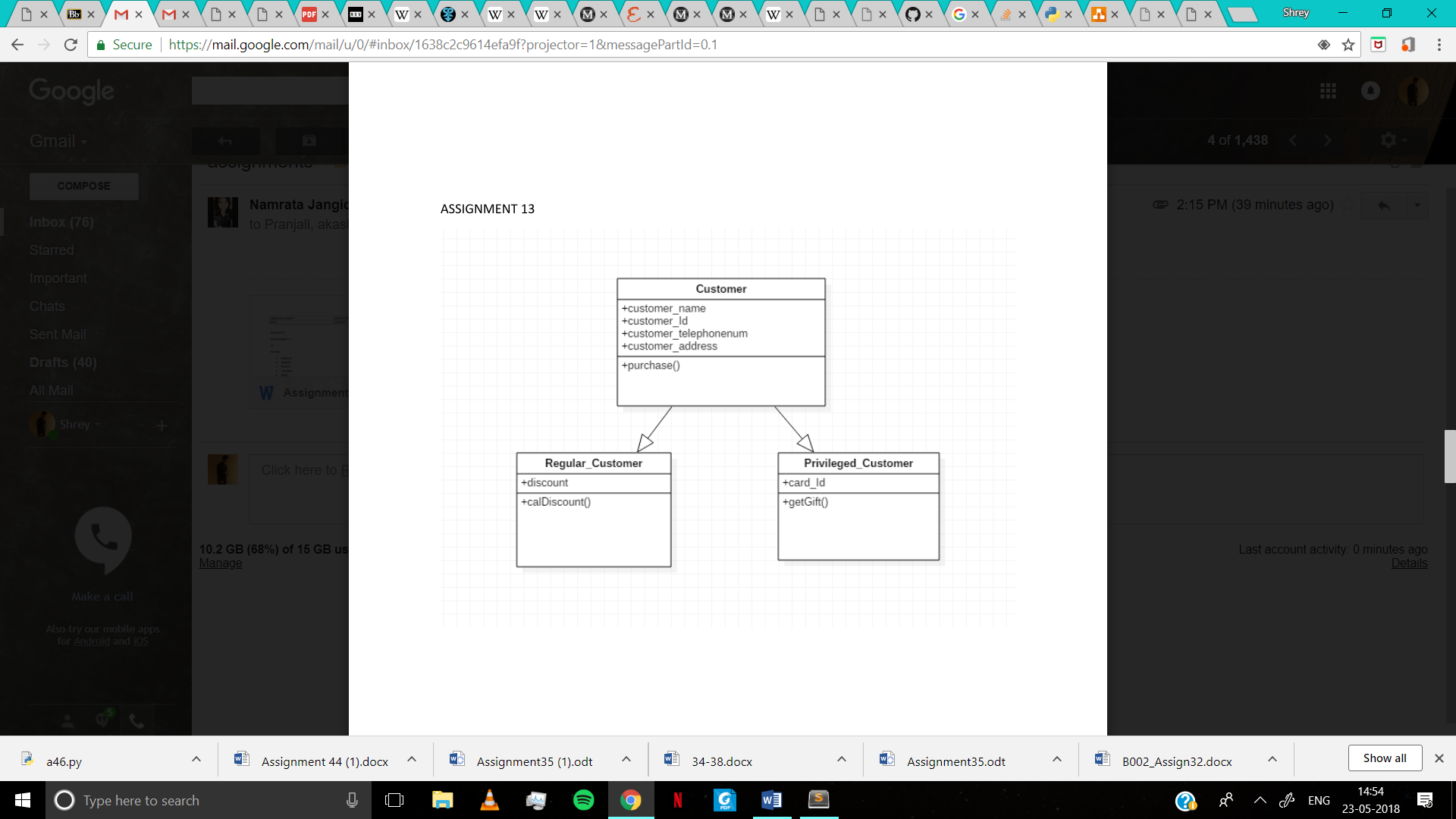
obj = calculator(2)

for i in range (230):

obj.nextPrime()



Assignment 13



Assignment 14

class box:

def \_\_init\_\_(self, length,breadth,height):

self.length = length

self.breadth=breadth

self.height=height

def getVolume(self):

vol = self.length\*self.breadth\*self.height

return vol

class BigBox(box):

def \_\_init\_\_(self, length,breadth,height):

box.\_\_init\_\_(self, length, breadth, height)

self.capacity=box.getVolume(self)

def getCapacity(self,sbox):

self.capacity=self.capacity/sbox.getVolume()

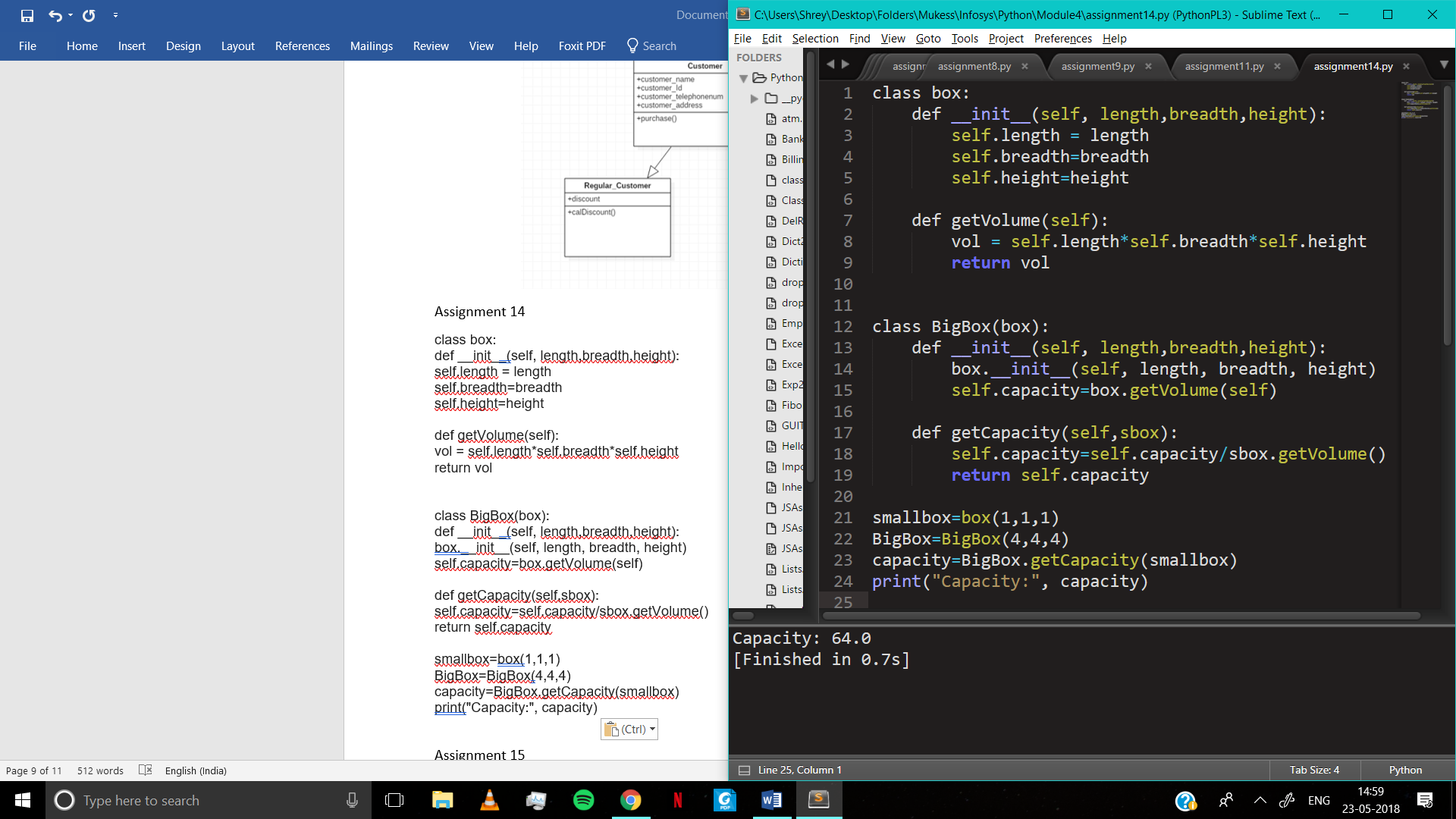
return self.capacity

smallbox=box(1,1,1)

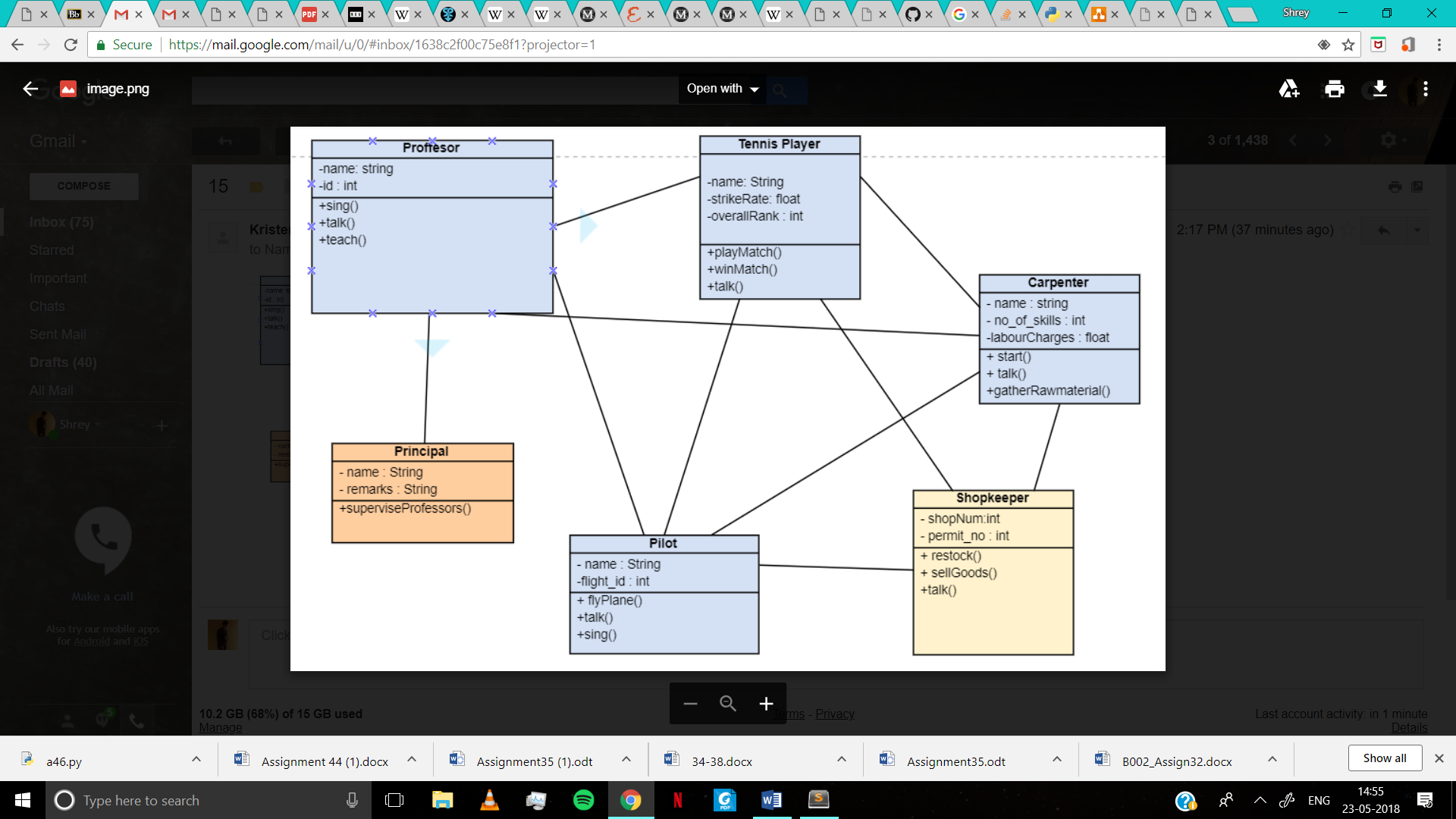
BigBox=BigBox(4,4,4)

capacity=BigBox.getCapacity(smallbox)

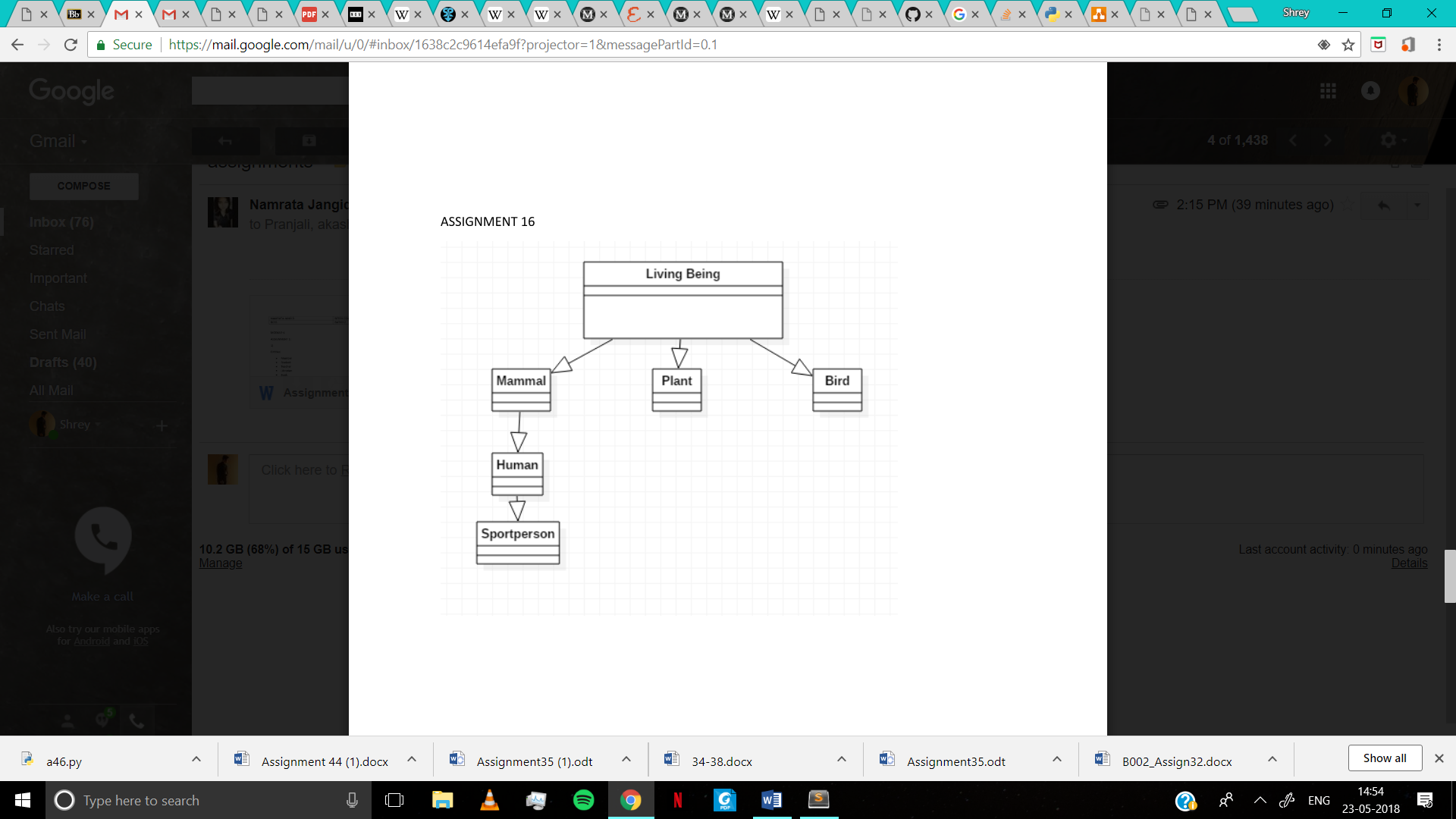
print("Capacity:", capacity)



Assignment 15



Assignment 16



Assignment 17

class Employee:

def \_\_init\_\_(self,salary):

self.salary = salary

def getSalary(self,payday,lop):

pass

class Faculty:

def \_\_init(self,remuneration):

self.remuneration = remuneration

def teach(self,course):

pass

class Visitor:

def \_\_init\_\_(self,remuneration):

self.remuneration = remuneration

def calculateFees(self,startTime, endTime):

pass

class Clerk(Employee):

def prepareBalanceSheet(self):

pass

class RegularFaculty(Faculty):

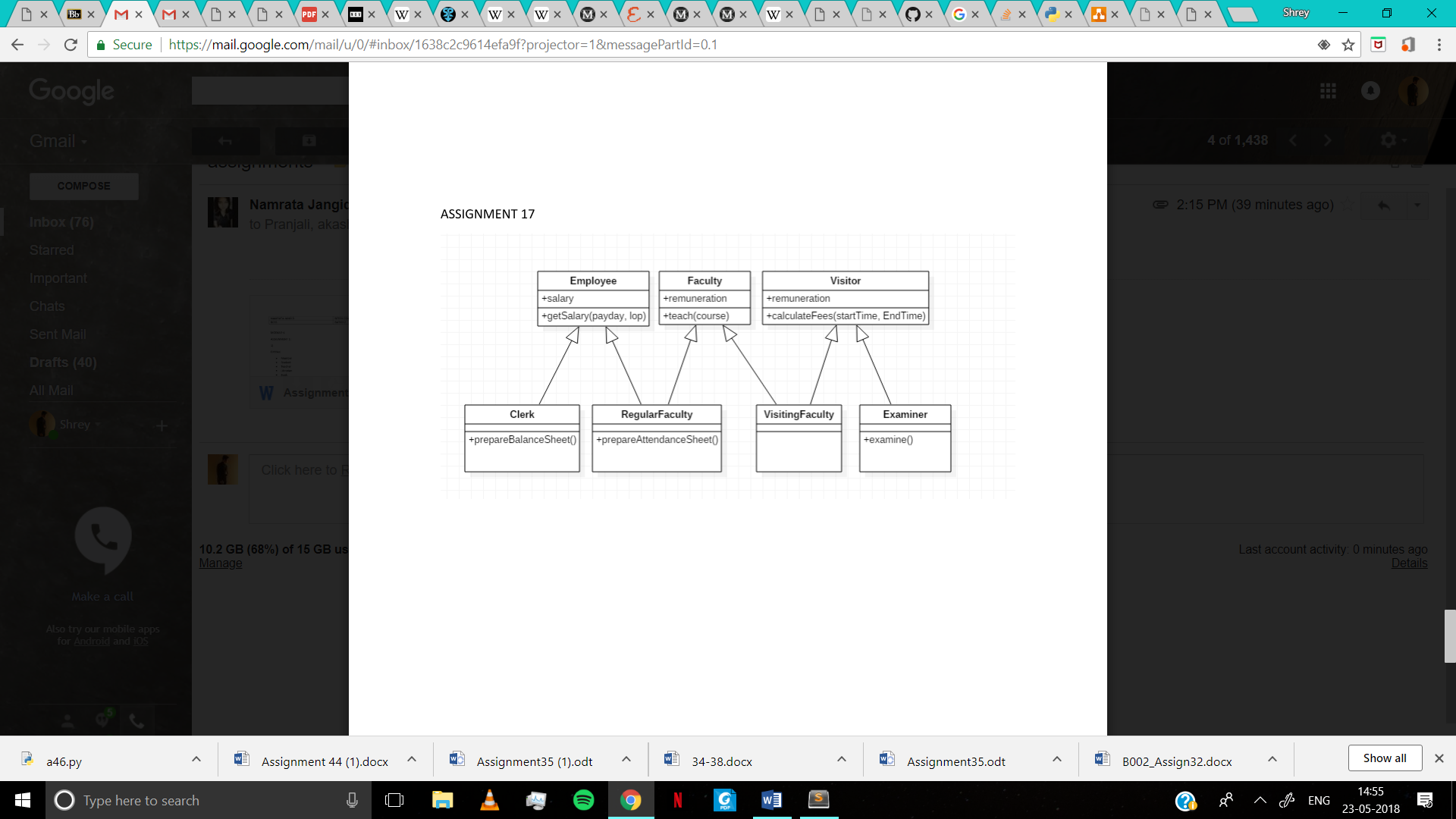
def prepareAttendanceReport(self):

pass

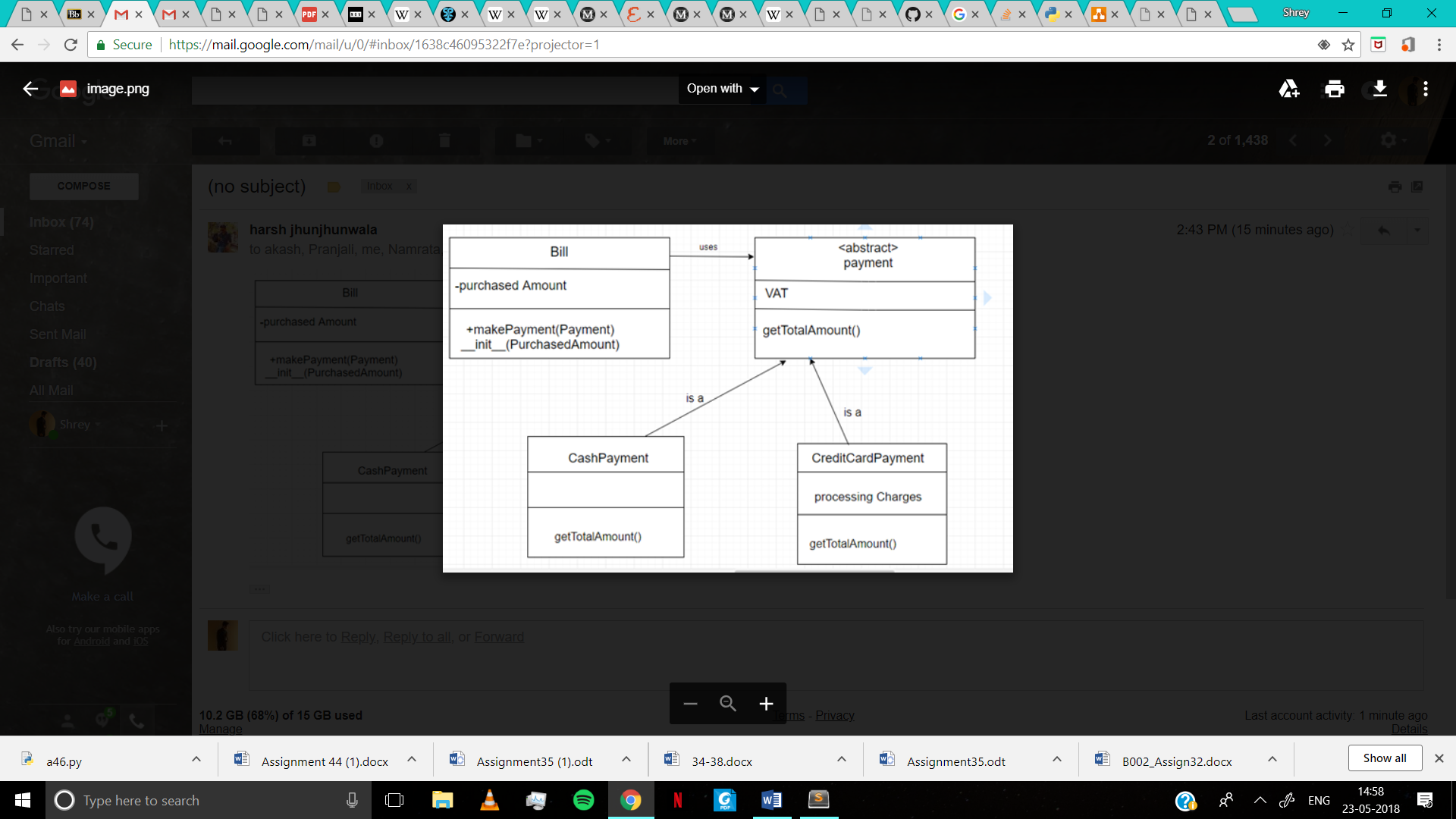
class VisitingFacult(Visitor):

def examine(self):

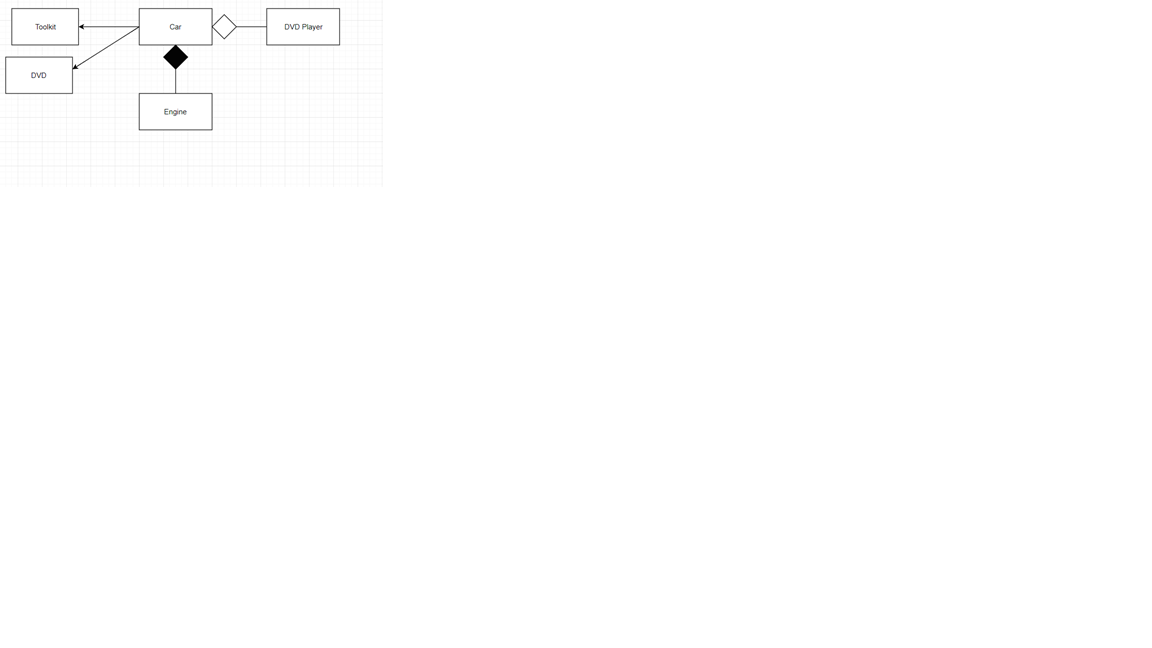
pass



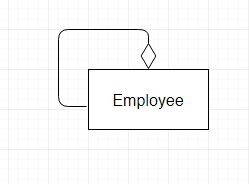
Assignment 18



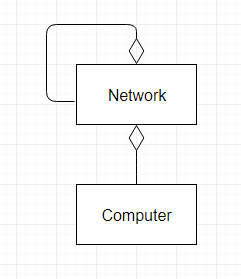
Assignment 19



Assignment20

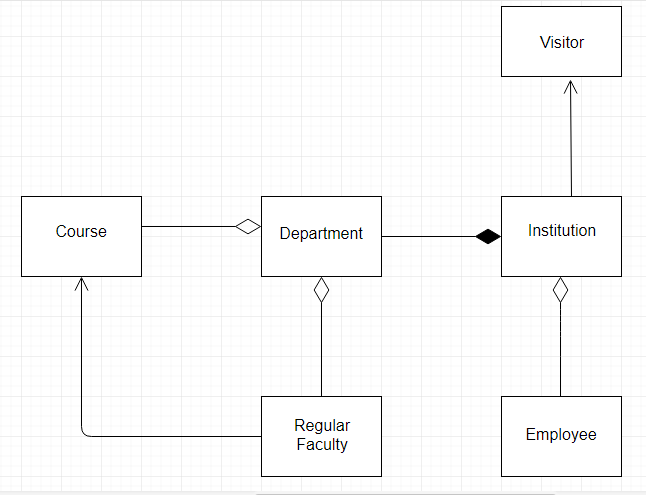


Assignment21



Assignment22

1.



2.



Assignment 23

class Faculty:

def addExpertise(self, course):

self.\_\_coursesExpertise.append(course)

def \_\_init\_\_(self, name):

self.\_\_name = name

self.\_\_coursesExpertise = []

class Student:

def enroll(self, course):

self.\_\_enrolledCourses.append(course)

def \_\_init\_\_(self, student\_id):

self.\_\_id = student\_id

self.\_\_enrolledCourses = []

class Course:

def registerStudent(self, student):

self.\_\_registeredStudents.append(student)

def registerExpert(self, faculty):

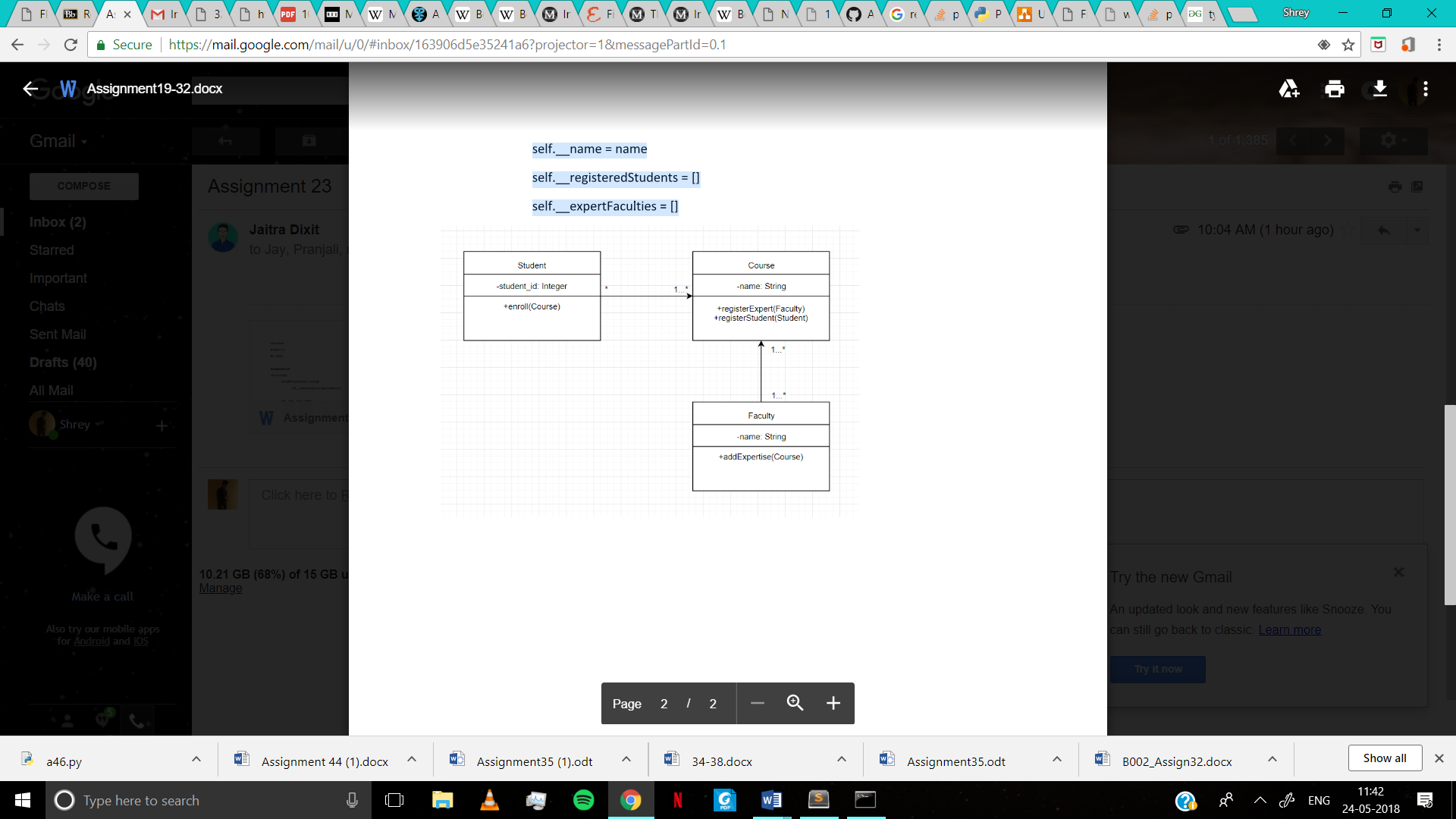
self.\_\_expertFaculties.append(faculty)

def \_\_init\_\_(self, name):

self.\_\_name = name

self.\_\_registeredStudents = []

self.\_\_expertFaculties = []



Assignment24



Assignment 25

class Employee():

# def \_\_init\_\_(self,FirstName,LastName,Pay):

def \_\_init\_\_(self,EmpId,FirstName,Designation):

self.EmpId = EmpId

self.first = FirstName

self.desig = Designation

# self.last = LastName

# self.salary = Pay

# self.email = self.first + '.' + self.last + '@company.com'

def details(self):

print(self.EmpId,self.first,self.desig)

class Department():

def \_\_init\_\_(self,DeptName):

self.dept = DeptName

def getDept(self):

print(self.dept)

class RegularFaculty(Employee,Department):

def \_\_init\_\_(self,EmpId,first,desig,dept):

Employee.\_\_init\_\_(self,EmpId,first,desig)

Department.\_\_init\_\_(self,dept)

# self.id = EmpId

# self.first = FirstName

# self.desig = Designation

# self.dept = DeptName

# class RegularFaculty(Department,Employee):

# def \_\_init\_\_(self,EmpId,FirstName,Designation,DeptName):

def getRegularFaculty(self):

print(self.EmpId,self.first,self.dept,self.desig)

class Visitor():

# def \_\_init\_\_(self,FirstName,LastName,Pay):

def \_\_init\_\_(self,VName):

self.VName = VName

def getVName(self):

print(self.VName)

class Institution(Employee,Department):

def \_\_init\_\_(self,EmpId,first,desig,dept):

Employee.\_\_init\_\_(self,EmpId,first,desig)

Department.\_\_init\_\_(self,dept)

# def \_\_init\_\_(self,FirstName,LastName,Pay):

def \_\_init\_\_(self,InsName):

self.InsName = InsName

def getName(self):

print(self.VName)

def addDepartment(self,d1):

self.d1 = d1

print(self.d1,"is added")

d1 = Department('Computer')

d2 = d1.getDept()

e1 = Employee('1','John','Clerk')

e1.details()

e2 = Employee('2','Jack','Professor')

e3 = RegularFaculty('2','Jack','Computer','Professor')

e3.getRegularFaculty()

v1 = Visitor('Bill Gates')

inst=Institution('Institution of technology')

inst.addDepartment(d1.getDept())

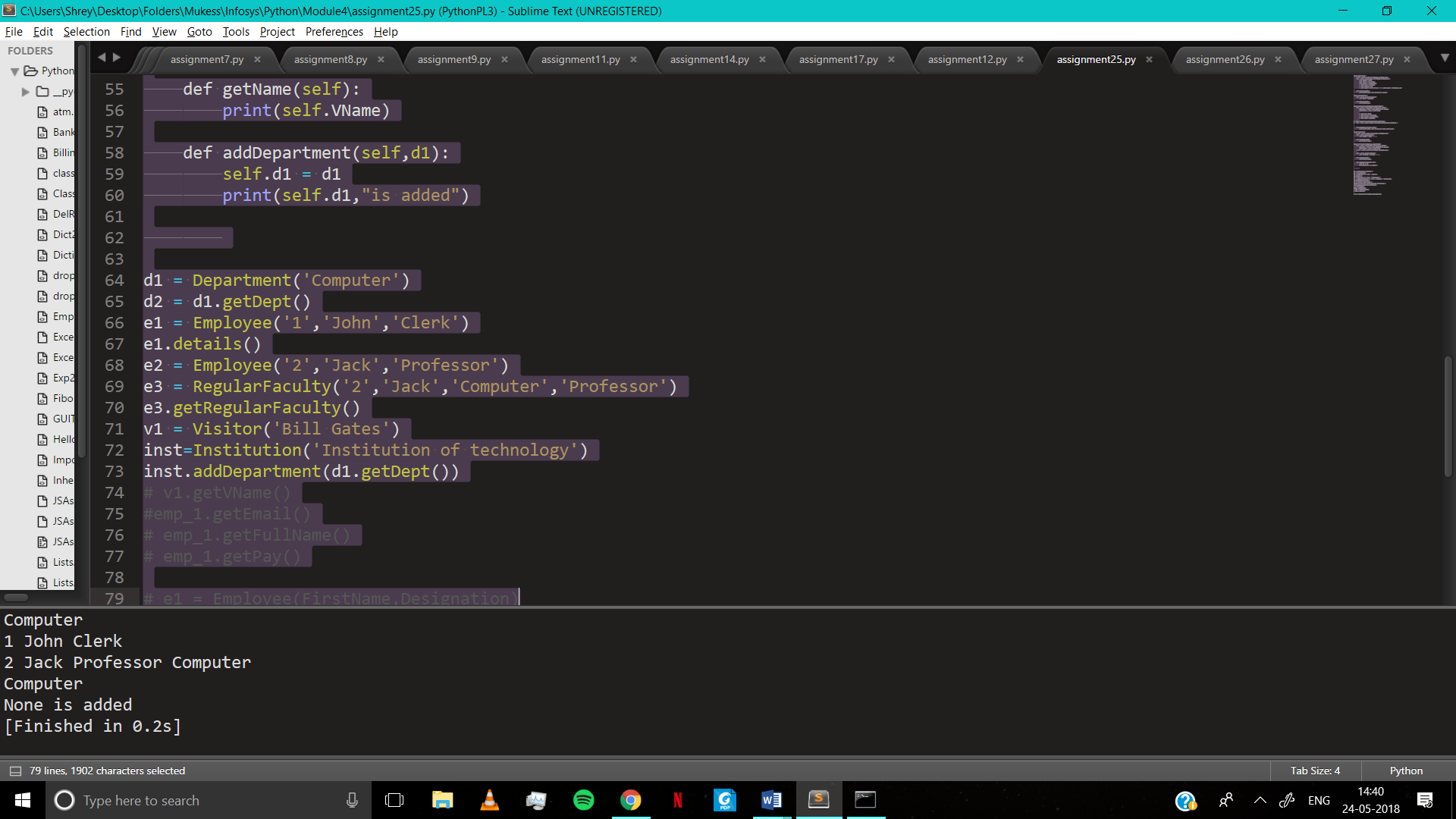
# v1.getVName()

#emp\_1.getEmail()

# emp\_1.getFullName()

# emp\_1.getPay()

# e1 = Employee(FirstName,Designation)



Assignment 26

class Error(Exception):

pass

class InvalidString(Error):

def \_\_init\_\_(self):

self.msg = "Found Aserisk"

def asteriskChecker(myString):

for i in myString:

if (i=="\*"):

raise InvalidString

mymessage = "abcde\*fz"

try:

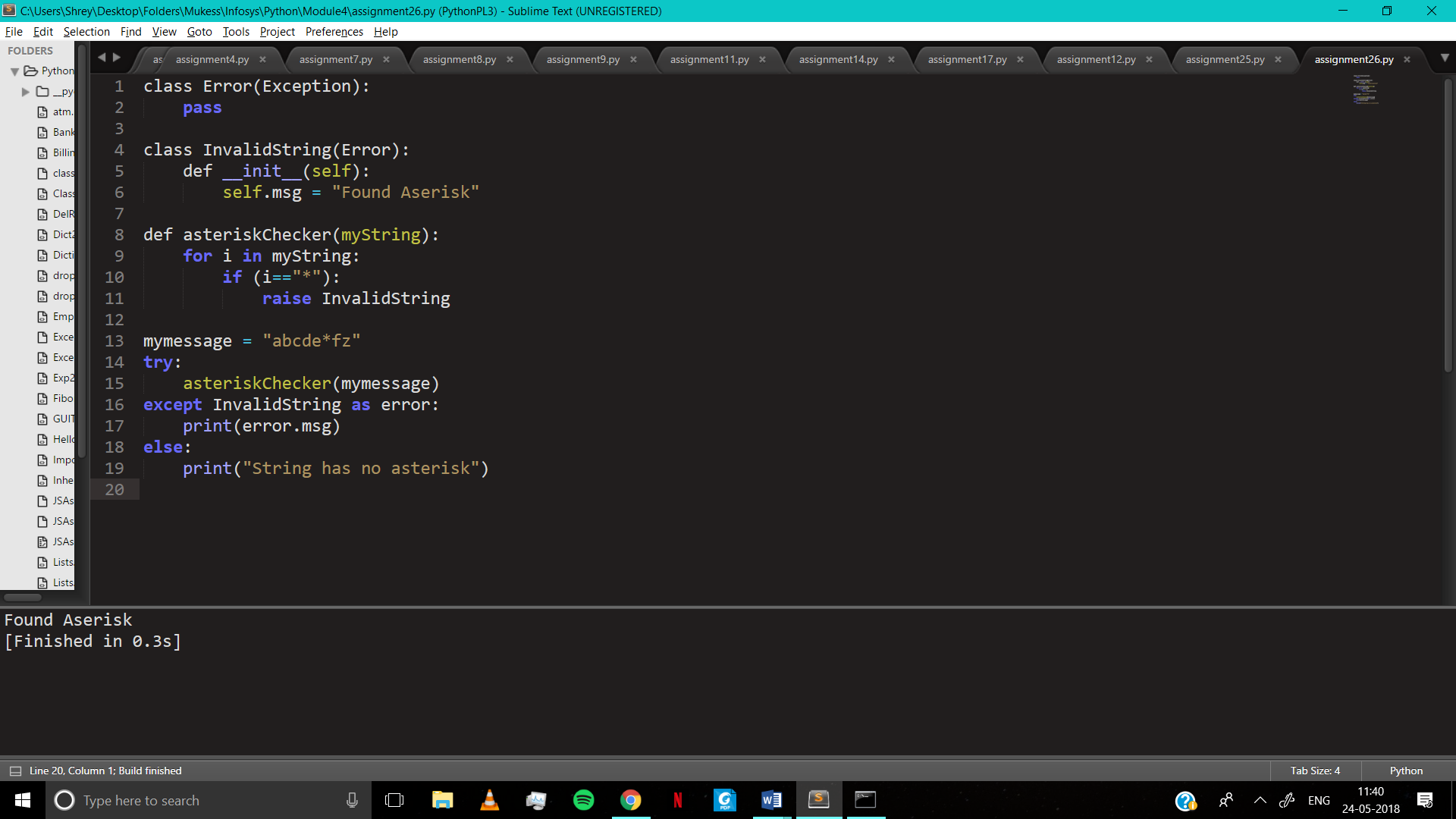
asteriskChecker(mymessage)

except InvalidString as error:

print(error.msg)

else:

print("String has no asterisk")

`

Assignment 27

1.

try:

file = open("C:\\Users\\jaitr\\Desktop\\College\\Sem 6\\Infosys Connect\\Module 4 OO Concepts\\a27\_file.txt", "r")

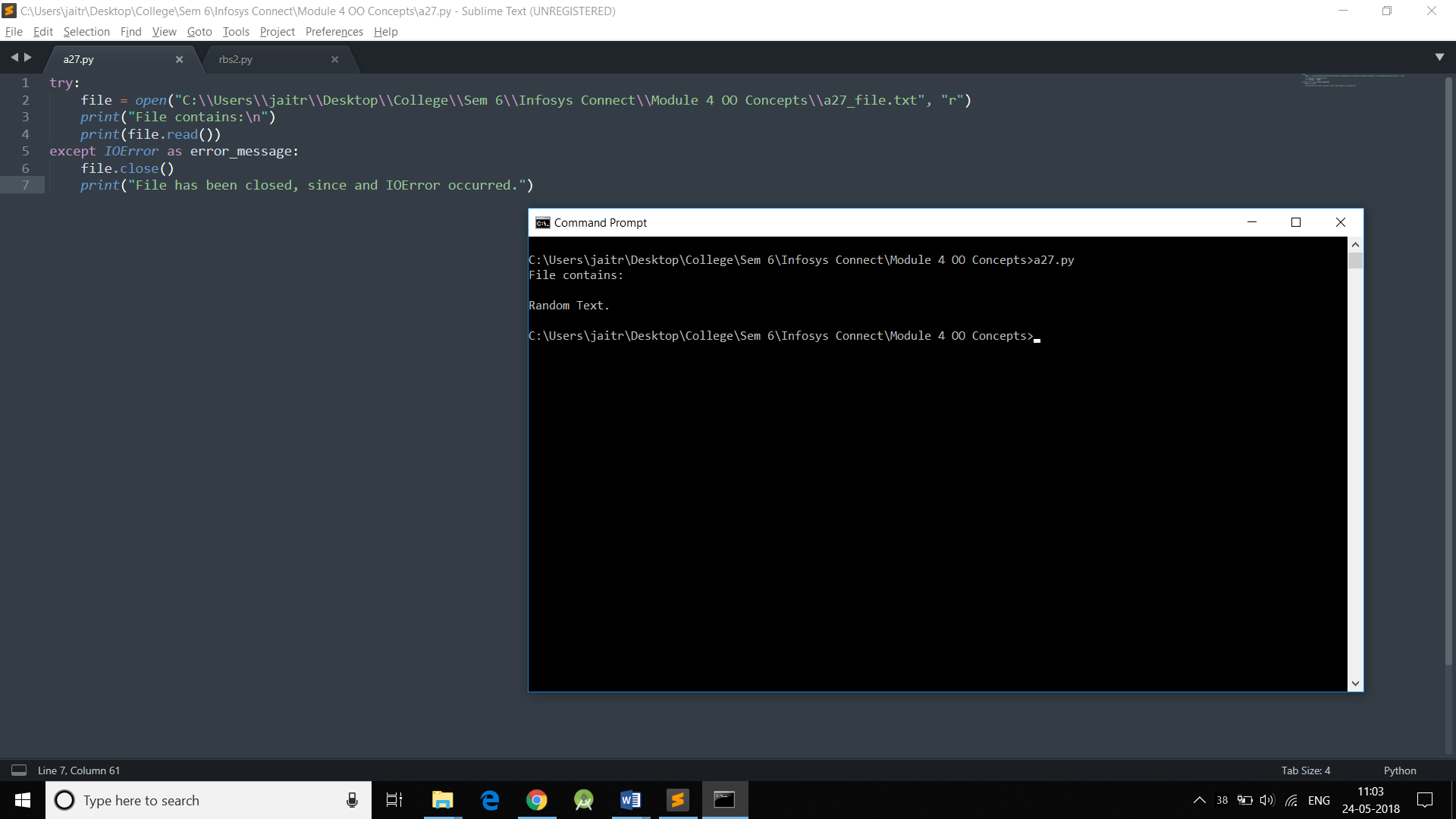
print("File contains:\n")

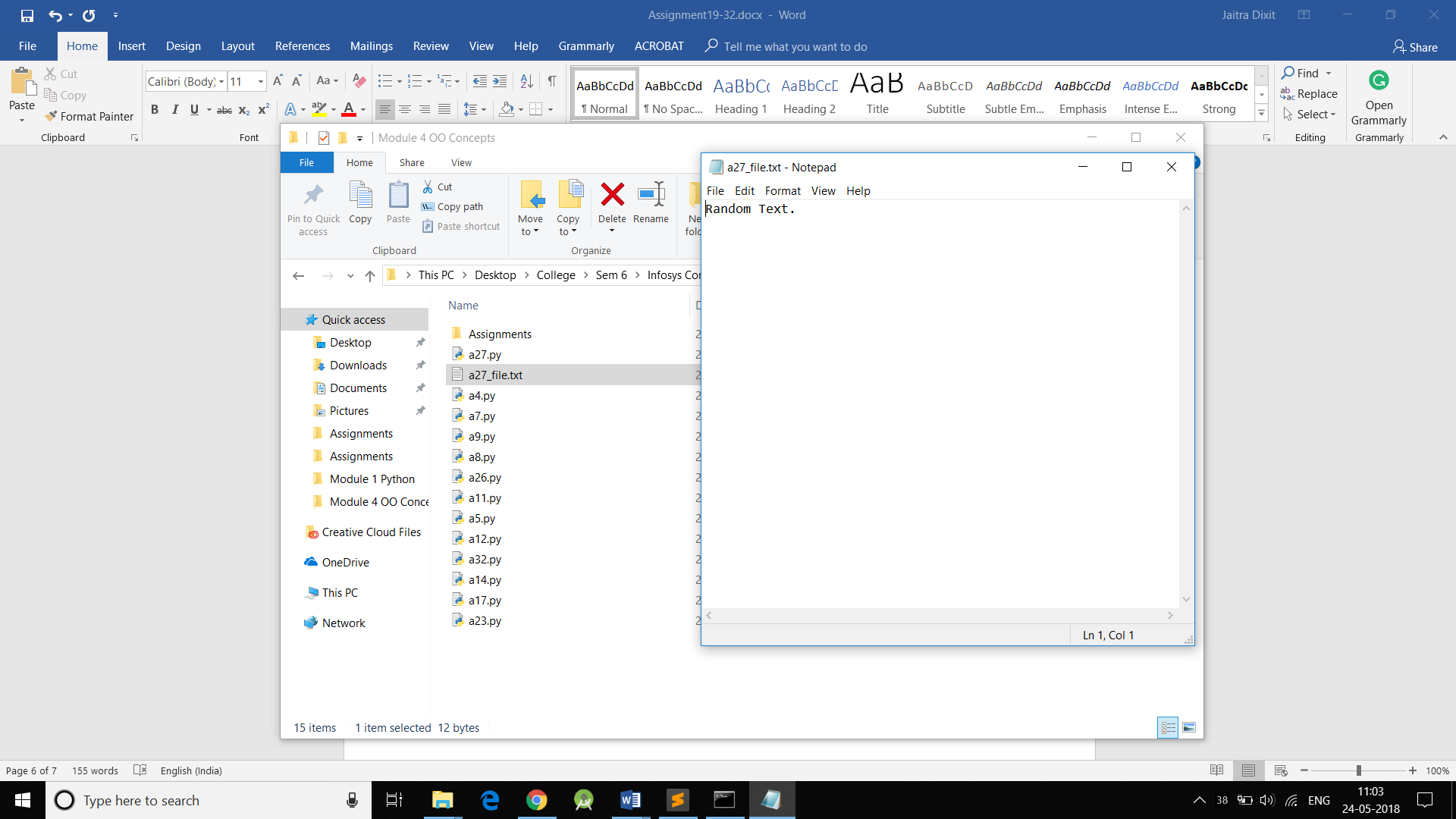
print(file.read())

except IOError as error\_message:

file.close()

print("File has been closed, since and IOError occurred.")





2.

try:

file = open("C:\\Users\\jaitr\\Desktop\\College\\Sem 6\\Infosys Connect\\Module 4 OO Concepts\\a27\_file.txt", "r")

print("File contains:\n")

print(file.read())

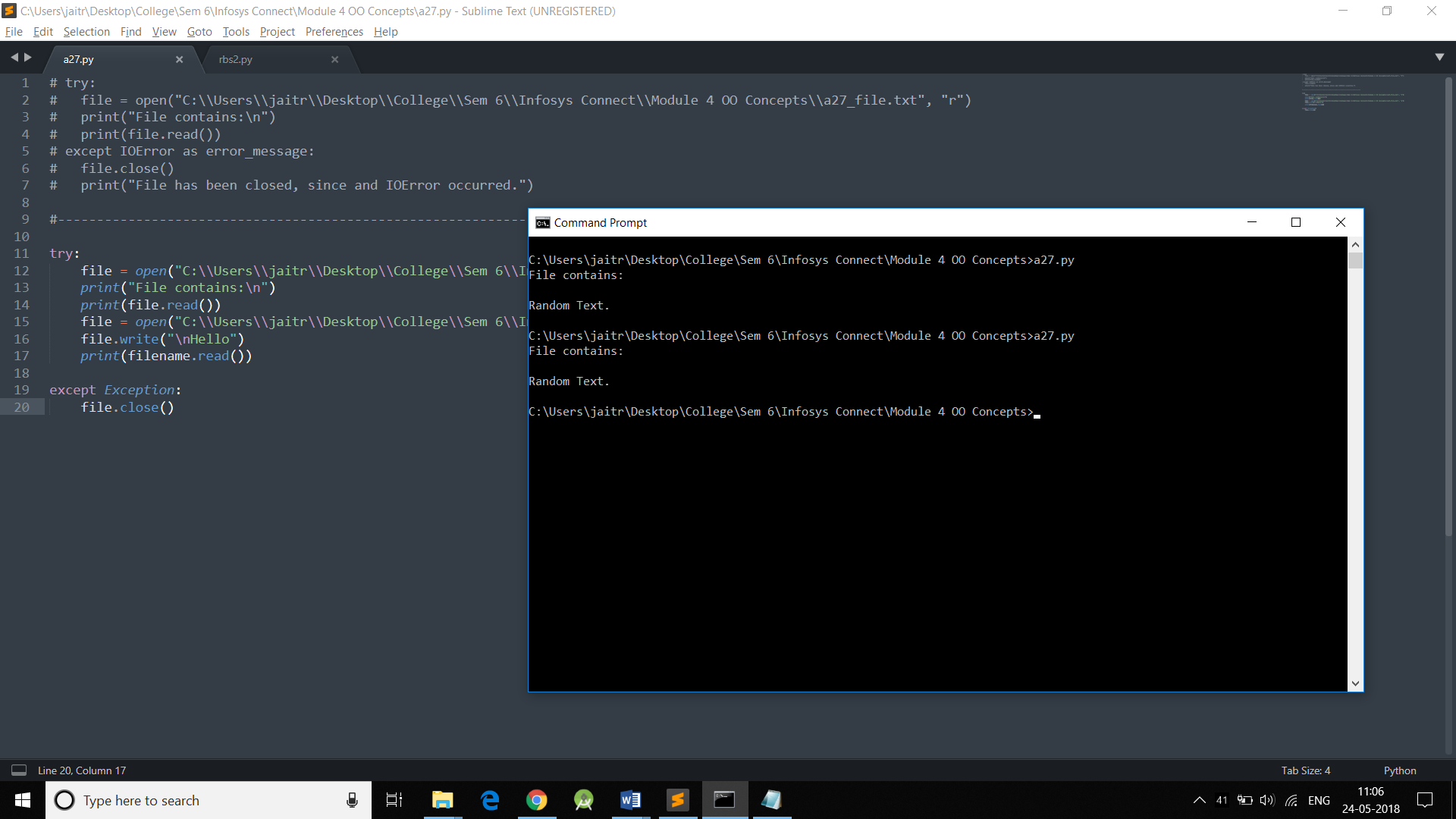
file = open("C:\\Users\\jaitr\\Desktop\\College\\Sem 6\\Infosys Connect\\Module 4 OO Concepts\\a27\_file.txt", "a")

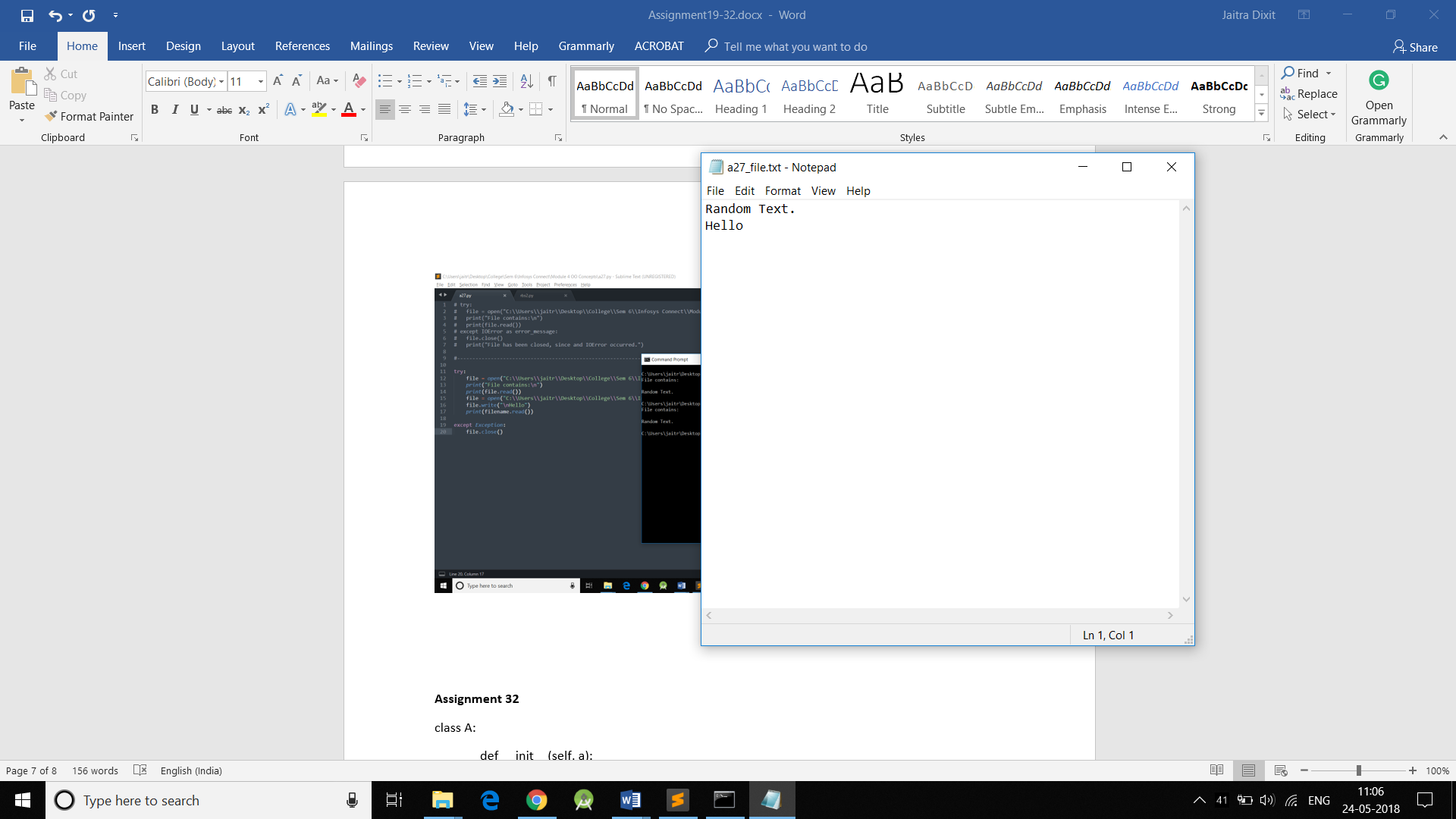
file.write("\nHello")

print(filename.read())

except Exception:

file.close()





Assignment 28

1.

class Error(Exception):

pass

class invalidEmail(Error):

pass

class invalidAge(Error):

pass

class invalidNumber(Error):

pass

try:

email = input("Enter the email ID: ")

mobile\_no = input("Enter the mobile number: ")

age = int(input("Enter the age: "))

if(age<0 or age>101):

raise invalidAge

else:

print("\nAge: ", age)

dot\_count = 0

at\_count = 0

for ch in email:

if(ch == "."):

dot\_count = dot\_count + 1

elif(ch == "@"):

at\_count = at\_count + 1

if(dot\_count>=1 and at\_count == 1):

print("Email ID: ", email)

else:

raise invalidEmail

if(len(mobile\_no) == 10 or len(mobile\_no) == 11):

if(mobile\_no.isdigit() == True):

print("Mobile Number: ", mobile\_no)

elif(len(mobile\_no) == 11 and mobile\_no[0] == "+"):

print("Mobile Number:", mobile\_no)

else:

raise invalidNumber

else:

raise invalidNumber

except invalidEmail:

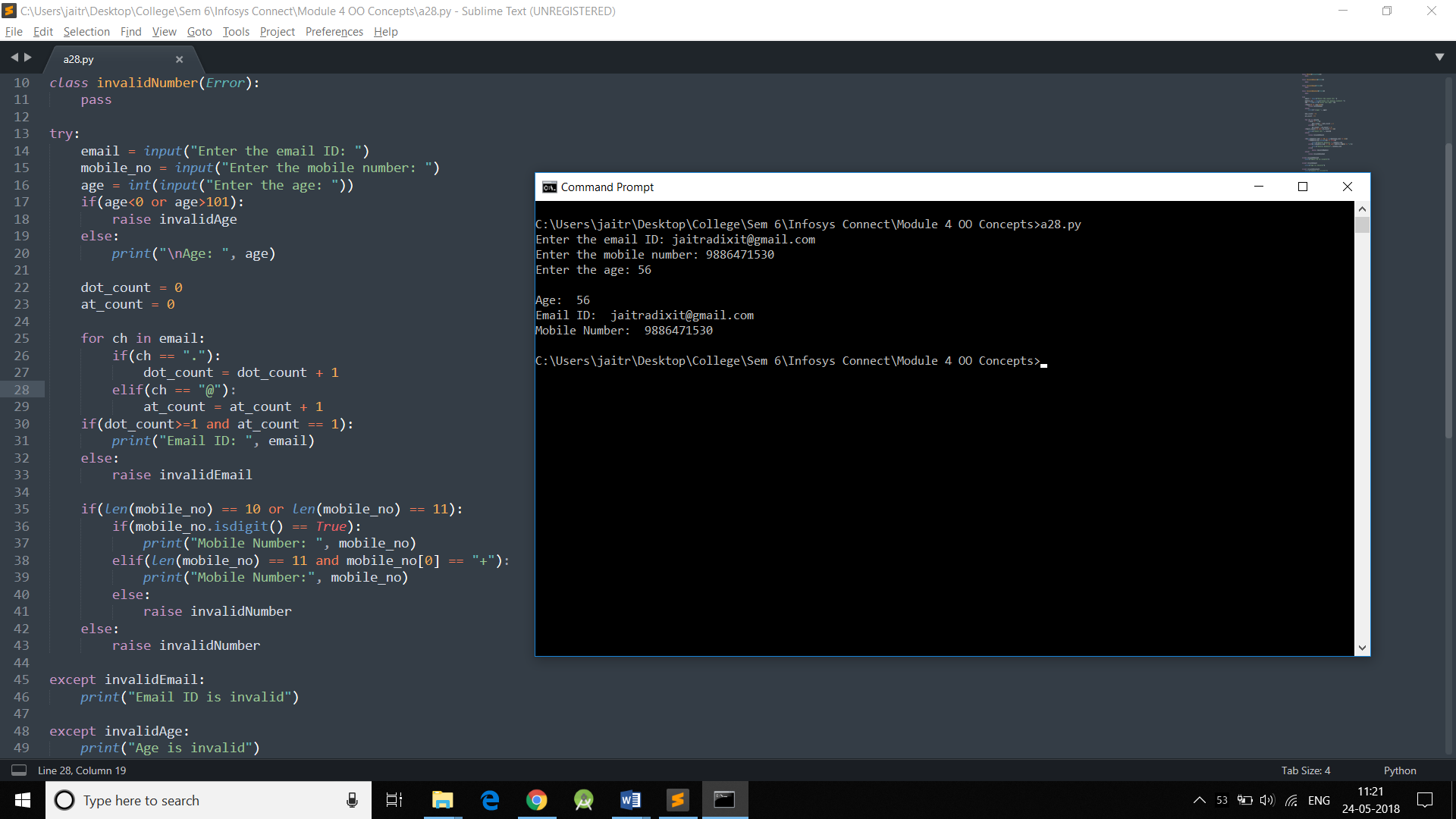
print("Email ID is invalid")

except invalidAge:

print("Age is invalid")

except invalidNumber:

print("Number is invalid")



Assignment 29

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Description** | **Expected Discount** |
| 1 | Customer age<60 : 59 | 0 |
| 2 | Customer age=60 :60 | 15% |
| 3 | Customer age>60 :61 | 15% |
| 4 | Customer age<70 :69 | 15% |
| 5 | Customer age=70 :70 | 30% |
| 6 | Customer age>70 :71 | 30% |

Assignment 30

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Description** | **Expected Discount** | **Actual Discount** | **Result**  **Pass/Fail** |
| 1 | Customer age<60 : 59 | 0 | 0 | Pass |
| 2 | Customer age=60 :60 | 15 | 0 | Fail |
| 3 | Customer age>60 :61 | 15 | 15 | Pass |
| 4 | Customer age<70 :69 | 15 | 15 | Pass |
| 5 | Customer age=70 :70 | 30 | 0 | Fail |
| 6 | Customer age>70 :71 | 30 | 30 | Pass |

a30.py

def getDiscount(age):

discount = 0

if (age>60 and age<70):

discount = 15

elif (age > 70):

discount = 30

return discount

test\_a30.py (For testing)

import a30

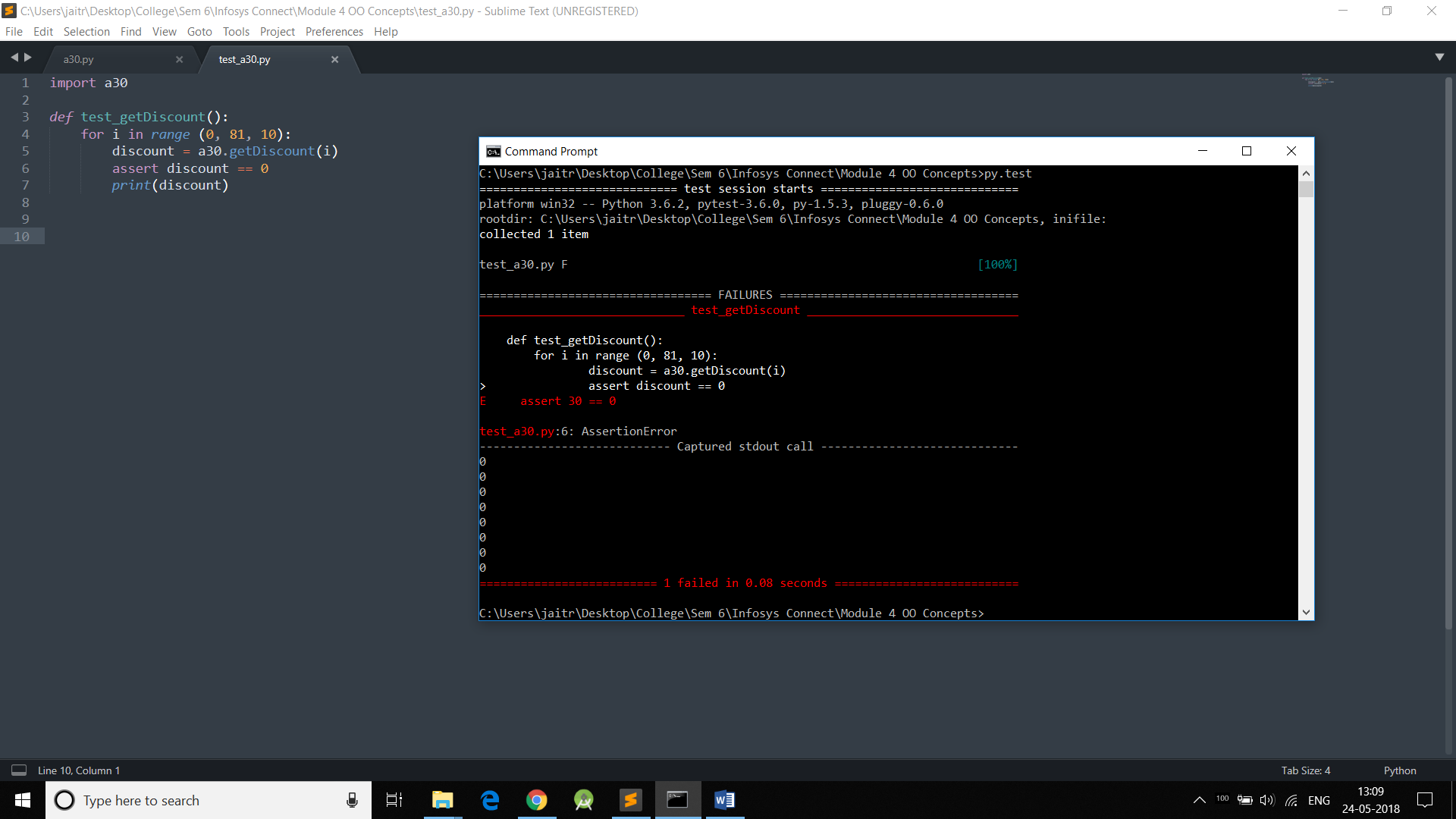
def test\_getDiscount():

for i in range (0, 81, 10):

discount = a30.getDiscount(i)

assert discount == 0

print(discount)



Assignment 31

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Description** | **Expected Discount** | **Actual Discount** | **Result**  **(Pass/Fail)** |
| 1 | age<60, gender=’F’ | 15 | 15 | Pass |
| 2 | age=60, gender=’F’ | 25 | 20 | Fail |
| 3 | age<60, gender=’M’ | 0 | 0 | Pass |
| 4 | age=60, gender=’M’ | 20 | 20 | Pass |

a31.py

def getDiscount(age, gender):

discount = 0

if age >= 60:

if (gender == "F"):

discount = 25

discount = 20

elif (gender == "F"):

discount = 15

return discount

test\_a31.py

import a31

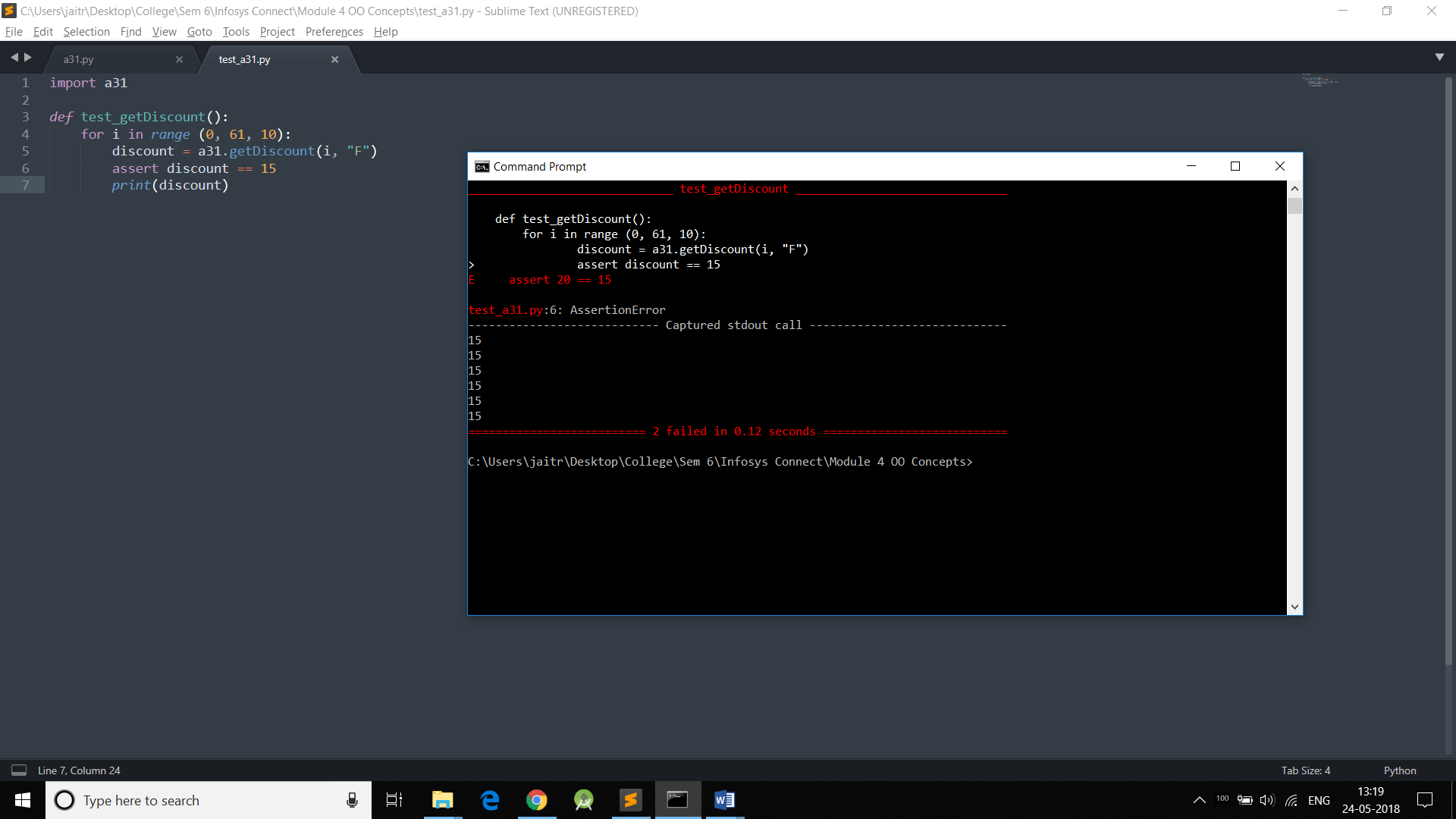
def test\_getDiscount():

for i in range (0, 61, 10):

discount = a31.getDiscount(i, "F")

assert discount == 15

print(discount)



Assignment 32

class A:

def \_\_init\_\_(self, a):

self.a = a

class B(A):

def \_\_init\_\_(self, a, b):

super().\_\_init\_\_(a)

self.b = b

class C(B):

def \_\_init\_\_(self, a, b, c):

super().\_\_init\_\_(a,b)

self.c = c

def printvalues(self):

print(self.a)

print(self.b)

print(self.c)

obj = C(1,2,3)

obj.printvalues()

