OVERVIEW

This project report consists of the following two modules:

1.Classes:

It shows the implementation of classes and objects in python.Classes and objects are used to model real world problems through programming in python.

It includes the source code for solving the following problem:

Managing a company which consists of various employees and buiding a hierarchy of various types of employees using classes and objects.

2.Data Structures:

This module implements an inverted index which is a data structure used to create full text search.

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents.

In simple words, it is a hashmap like data structure that directs us from a word to a document.. .

INTRODUCTION TO PYTHON

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

What can Python do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a function

PROJECT

1.Problem-

Company Manager:

Create a hierarchy of classes-abstract class Employee and subclasses Hourly Employee,Salaried Employee,manager and executive.Their salaries are calculated.After establishing an employee hierarchy,create a Company class that allows to manage(hire,fire and raise salary ) the employees.

SOURCE CODE:

from abc import ABC,abstractmethod

class Employee(ABC):

@abstractmethod

def calculatesalary(self):

pass

class HourlyEmployee(Employee):

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

self.name=name

self.perhrsalary=perhrsalary

self.totalhours=totalhours

def calculatesalary(self):

cal=self.perhrsalary\*self.totalhours

print("total calculated salary of "+self.name+" is "+str(cal))

class SalariedEmployee(Employee):

def \_\_init\_\_(self,name,monthly\_salary,allowance):

self.name=name

self.monthly\_salary=monthly\_salary

self.allowance=allowance

def calculatesalary(self):

cal=self.monthly\_salary+self.allowance

print("total calculated salary of "+self.name+" is "+str(cal))

class Manager(Employee):

def \_\_init\_\_(self,name,base\_salary,bonus):

self.name=name

self.base\_salary=base\_salary

self.bonus=bonus

def return\_salary(self):

return self.base\_salary

def calculatesalary(self):

cal=self.base\_salary+2\*self.bonus

print("total calculated salary of "+self.name+" is "+str(cal))

class Executive(Employee):

def \_\_init\_\_(self,name,base\_salary,experience):

self.name=name

self.base\_salary=base\_salary

self.experience=experience

def calculatesalary(self):

cal=self.base\_salary+4\*self.experience

print("total calculated salary of "+self.name+" is "+str(cal))

class Company():

def \_\_init\_\_(self,base\_salary):

self.base\_salary=base\_salary

def HireEmp(self):

eh\_no=input("enter no of employees to be hired:")

print("enter details of employee to be hired:")

for i in range(int(eh\_no)):

e\_name=input('enter emp name:')

e\_desig=input("enter emp desig:")

fh=open(r"C:\Users\user\Desktop\abc.txt",'a')

fh.write(e\_name+"\t ")

fh.write(e\_desig)

fh.write("\n")

fh.close()

print(eh\_no+" employee(s) is/are hired")

def FireEmp(self):

ef\_no=int(input('enter no of employees to be fired:'))

for i in range(int(ef\_no)):

ef\_name=input('enter name of emp to be fired:')

fh=open(r"C:\Users\user\Desktop\abc.txt","r")

lines=fh.readlines()

fh=open(r"C:\Users\user\Desktop\abc.txt","w")

for line in lines:

if ef\_name not in line:

fh.write(line)

fh.close()

print(str(ef\_no)+" employee(s) is/are fired")

def raise\_emp\_sal(self,object1):

#object1 here want to take object of manager class salary = object1.return\_salary()

name=input('enter name of employee whose salary has to be raised:')

salary=self.base\_salary+0.1\*self.base\_salary

print('salary of employee '+name+' is raised to '+str(salary))

H\_E=HourlyEmployee("rahul",25,360)

H\_E.calculatesalary()

S\_E=SalariedEmployee("vishal",30000,250)

S\_E.calculatesalary()

man=Manager("abc",40000,500)

man.calculatesalary()

man.return\_salary()

ex=Executive("pqr",50000,4)

ex.calculatesalary()

com= Company(30000)

print("enter 1 to hire emp")

print("enter 2 to fire emp")

print("enter 3 to raise emp salary")

while True:

choice=int(input("enter your choice:"))

if choice==1:

com.HireEmp()

elif choice==2:

com.FireEmp()

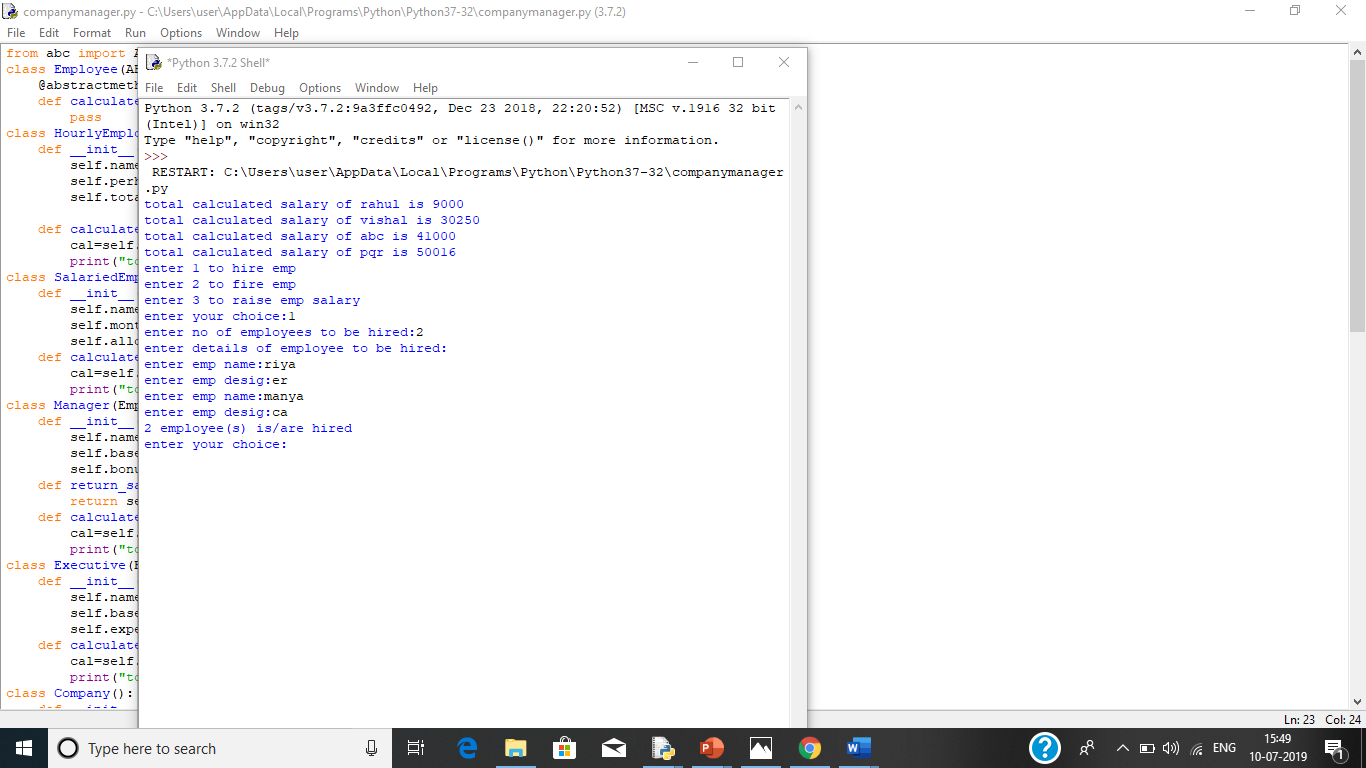
elif choice==3:

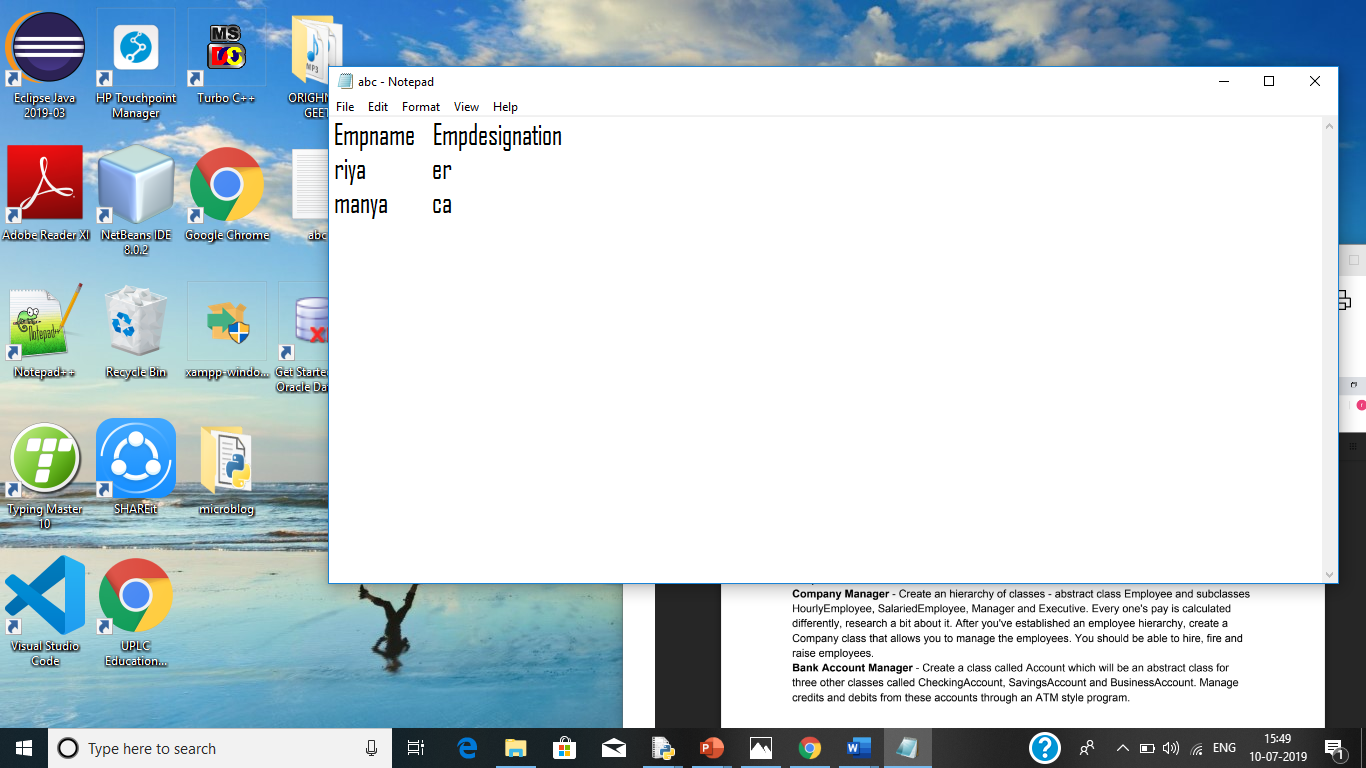
com.raise\_emp\_sal(man)

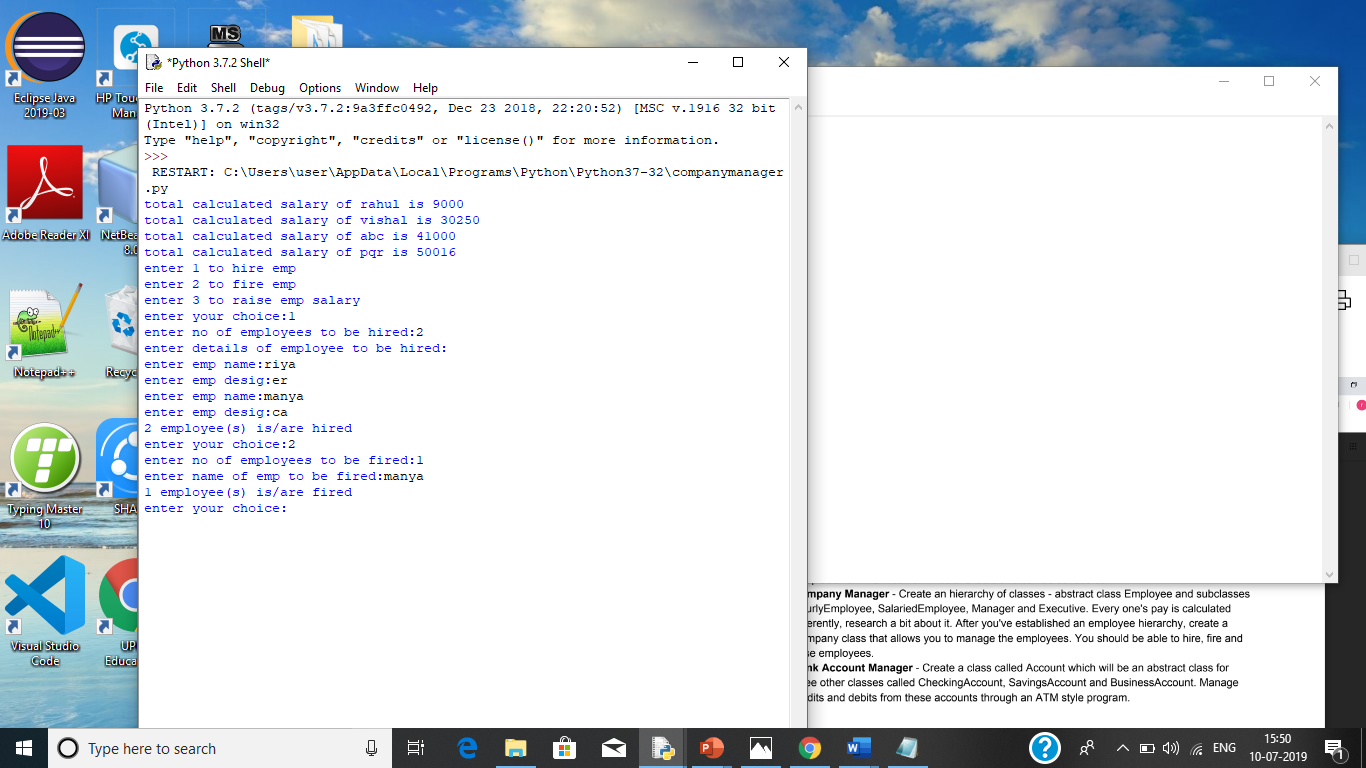
else:

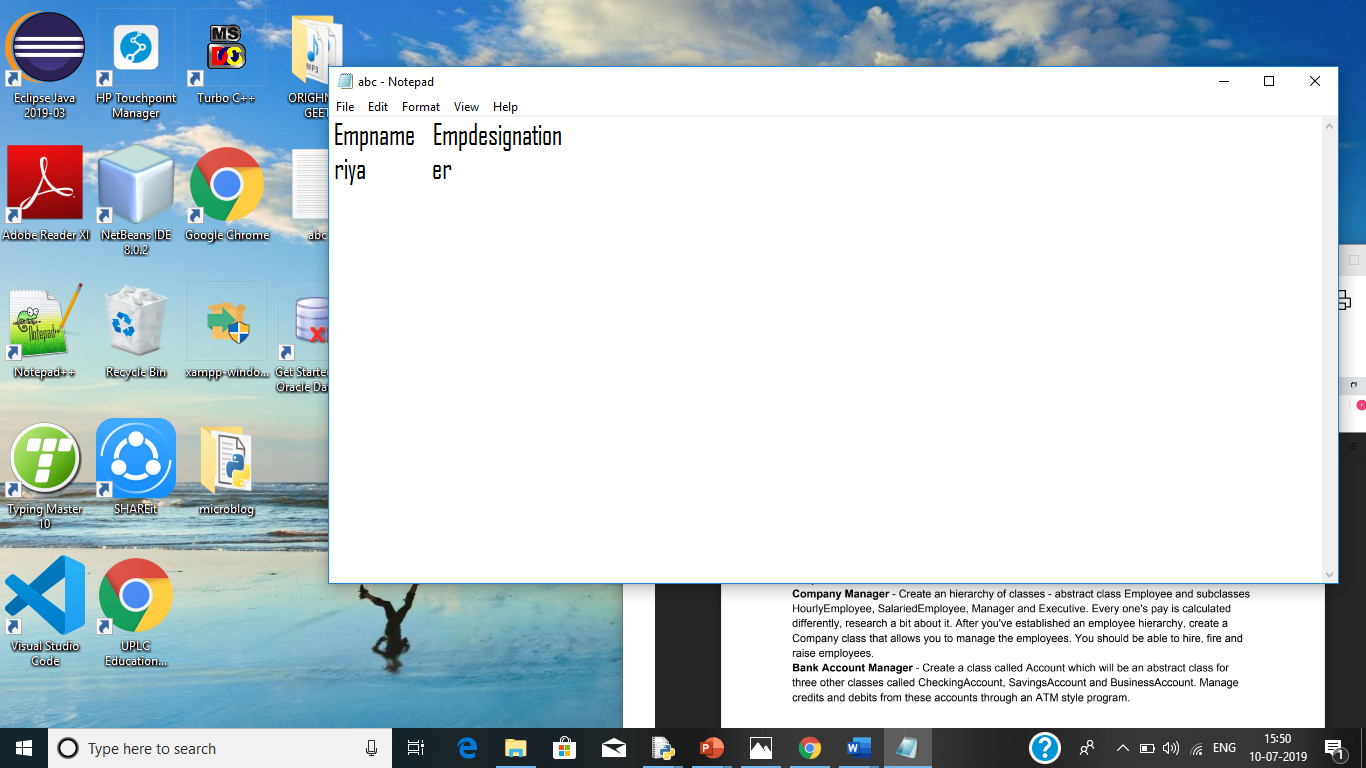
break

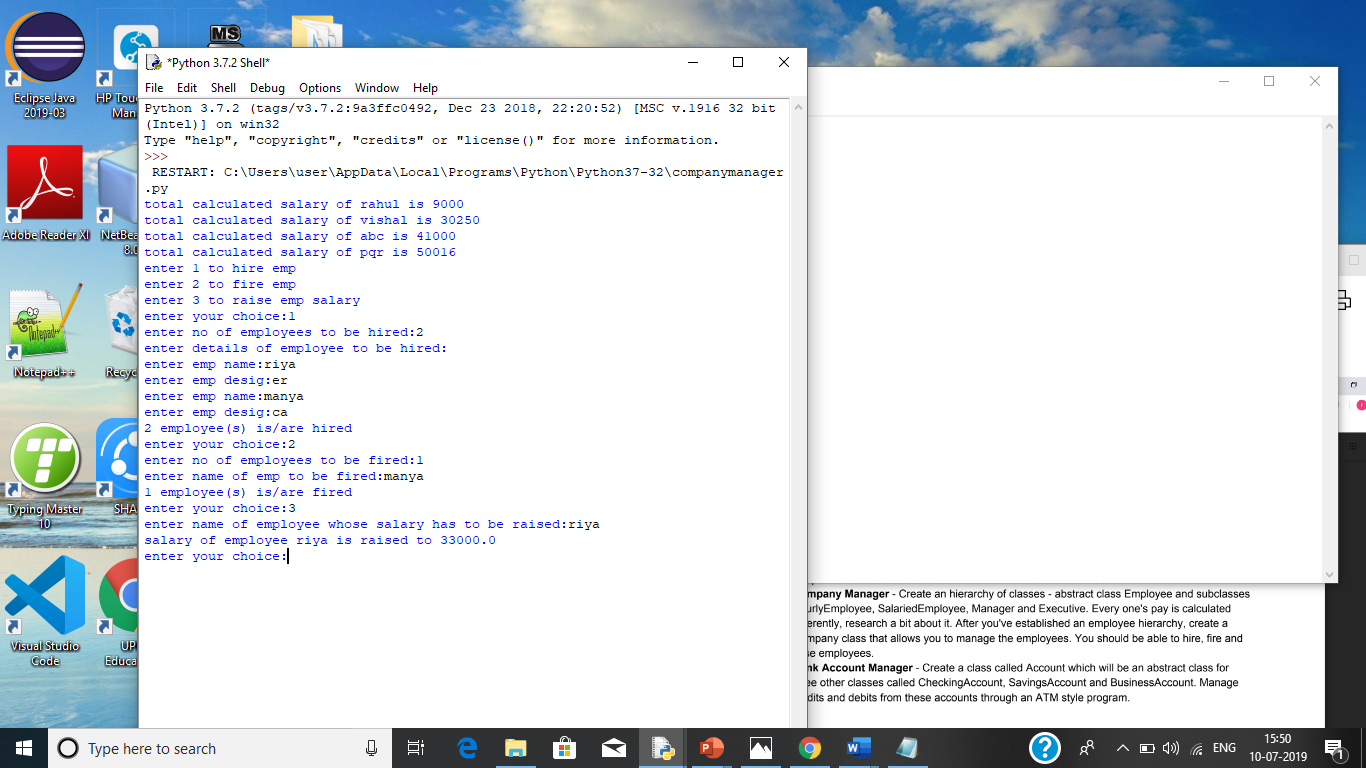
OUTPUT











2.Problem-

Inverted Index:

An inverted index is a datastructure is a data structure used to create a full text search.Given a set of text files,implement a program to create an inverte index.Also create a user interface to do a search using that inverted index which returns a list of files that contain the query term.

SOURCE CODE:

import unicodedata

import functools

# List Of English Stop Words

\_WORD\_MIN\_LENGTH = 3

#Words that are not indexed

\_STOP\_WORDS = frozenset([

'a', 'about', 'above', 'above', 'across', 'after', 'afterwards', 'again',

'against', 'all', 'almost', 'alone', 'along', 'already', 'also','although',

'always','am','among', 'amongst', 'amoungst', 'amount', 'an', 'and', 'another',

'any','anyhow','anyone','anything','anyway', 'anywhere', 'are', 'around', 'as',

'at', 'back','be','became', 'because','become','becomes', 'becoming', 'been',

'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides',

'between', 'beyond', 'bill', 'both', 'bottom','but', 'by', 'call', 'can',

'cannot', 'cant', 'co', 'con', 'could', 'couldnt', 'cry', 'de', 'describe',

'detail', 'do', 'done', 'down', 'due', 'during', 'each', 'eg', 'eight',

'either', 'eleven','else', 'elsewhere', 'empty', 'enough', 'etc', 'even',

'ever', 'every', 'everyone', 'everything', 'everywhere', 'except', 'few',

'fifteen', 'fify', 'fill', 'find', 'fire', 'first', 'five', 'for', 'former',

'formerly', 'forty', 'found', 'four', 'from', 'front', 'full', 'further', 'get',

'give', 'go', 'had', 'has', 'hasnt', 'have', 'he', 'hence', 'her', 'here',

'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him',

'himself', 'his', 'how', 'however', 'hundred', 'ie', 'if', 'in', 'inc',

'indeed', 'interest', 'into', 'is', 'it', 'its', 'itself', 'keep', 'last',

'latter', 'latterly', 'least', 'less', 'ltd', 'made', 'many', 'may', 'me',

'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most', 'mostly',

'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never',

'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not',

'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one', 'only',

'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out',

'over', 'own','part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 'same',

'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she',

'should', 'show', 'side', 'since', 'sincere', 'six', 'sixty', 'so', 'some',

'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere',

'still', 'such', 'system', 'take', 'ten', 'than', 'that', 'the', 'their',

'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby',

'therefore', 'therein', 'thereupon', 'these', 'they', 'thickv', 'thin', 'third',

'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus',

'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two',

'un', 'under', 'until', 'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well',

'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where', 'whereafter',

'whereas', 'whereby', 'wherein', 'whereupon', 'wherever', 'whether', 'which',

'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will',

'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself',

'yourselves', 'the'])

def word\_split(text):

"""

Split a text in words. Returns a list of tuple that contains

(word, location) location is the starting byte position of the word.

"""

word\_list = []

wcurrent = []

windex = None

for i, c in enumerate(text):

if c.isalnum():

wcurrent.append(c)

windex = i

elif wcurrent:

word = u''.join(wcurrent)

word\_list.append((windex - len(word) + 1, word))

wcurrent = []

if wcurrent:

word = u''.join(wcurrent)

word\_list.append((windex - len(word) + 1, word))

return word\_list

def words\_cleanup(words):

"""

Remove words with length less then a minimum and stopwords.

"""

cleaned\_words = []

for index, word in words:

if len(word) < \_WORD\_MIN\_LENGTH or word in \_STOP\_WORDS:

continue

cleaned\_words.append((index, word))

return cleaned\_words

def words\_normalize(words):

"""

Do a normalization process on words. In this case it is just to lower()"""

normalized\_words = []

for index, word in words:

wnormalized = word.lower()

normalized\_words.append((index, wnormalized))

return normalized\_words

def word\_index(text):

"""A helper method to process a text.It calls word split, normalize and cleanup."""

words = word\_split(text)

words = words\_normalize(words)

words = words\_cleanup(words)

return words

def inverted\_index(text):

"""

Create an Inverted-Index of the specified text document-{word:[location]}"""

inverted = {}

for index, word in word\_index(text):

locations = inverted.setdefault(word, [])

locations.append(index)

return inverted

def inverted\_index\_add(inverted, doc\_id, doc\_index):

"""

Add Invertd-Index doc\_index of the document doc\_id to the

Multi-Document Inverted-Index (inverted),

using doc\_id as document identifier.

{word:{doc\_id:[locations]}}

"""

for word, locations in doc\_index.items():

indices = inverted.setdefault(word, {})

indices[doc\_id] = locations

return inverted

def search(inverted, query):

"""

Returns a set of documents id that contains all the words in the query.

"""

words = [word for \_, word in word\_index(query) if word in inverted]

results = [set(inverted[word].keys()) for word in words]

return functools.reduce(lambda x, y: x & y, results) if results else []

if \_\_name\_\_ == '\_\_main\_\_':

doc1 = """

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike

Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working

version. This language was initially called “Oak,” but was renamed “Java” in 1995. Between

the initial implementation of Oak in the fall of 1992 and the public announcement of Java in

the spring of 1995, many more people contributed to the design and evolution of the language.

Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key

contributors to the maturing of the original prototype.

Somewhat surprisingly, the original impetus for Java was not the Internet! Instead, the

primary motivation was the need for a platform-independent (that is, architecture-neutral)

language that could be used to create software to be embedded in various consumer electronic

devices, such as microwave ovens and remote controls. As you can probably guess, many

different types of CPUs are used as controllers. The trouble with C and C++ (and most other

languages) is that they are designed to be compiled for a specific target. Although it is possible

to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler

targeted for that CPU. The problem is that compilers are expensive and time-consuming to

create. An easier—and more cost-efficient—solution was needed. In an attempt to find such a

solution, Gosling and others began work on a portable, platform-independent language that

could be used to produce code that would run on a variety of CPUs under differing

environments. This effort ultimately led to the creation of Java.

"""

doc2 = """

The fifth edition of West Coast Green, a conference focusing on "green" home

innovations and products, rolled into San Francisco's Fort Mason last week

intent, per usual, on making our living spaces more environmentally friendly

- one used-tire house at a time.

To that end, there were presentations on topics such as water efficiency and

the burgeoning future of Net Zero-rated buildings that consume no energy and

produce no carbon emissions.

"""

# Build Inverted-Index for documents

inverted = {}

documents = {'doc1':doc1, 'doc2':doc2}

for doc\_id, text in documents.items():

doc\_index = inverted\_index(text)

inverted\_index\_add(inverted, doc\_id, doc\_index)

# Print Inverted-Index

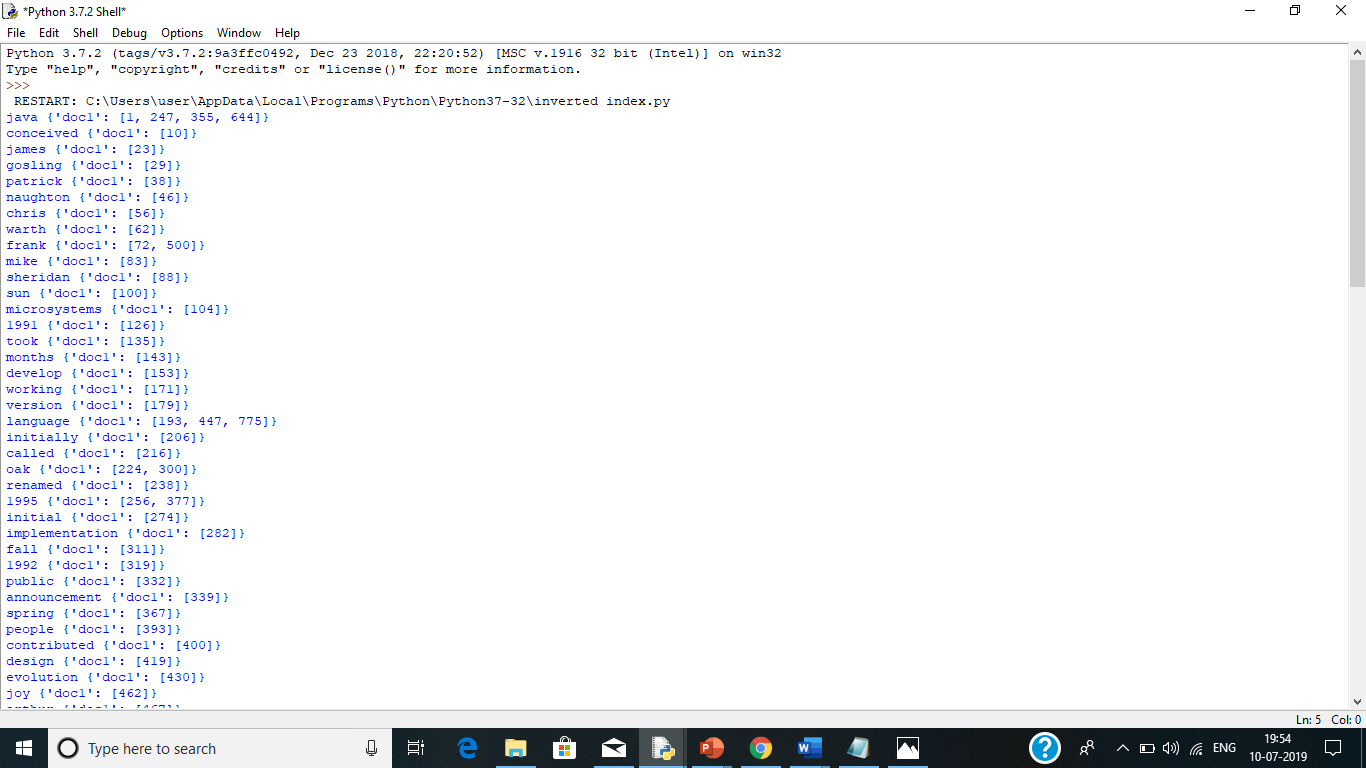
for word, doc\_locations in inverted.items():

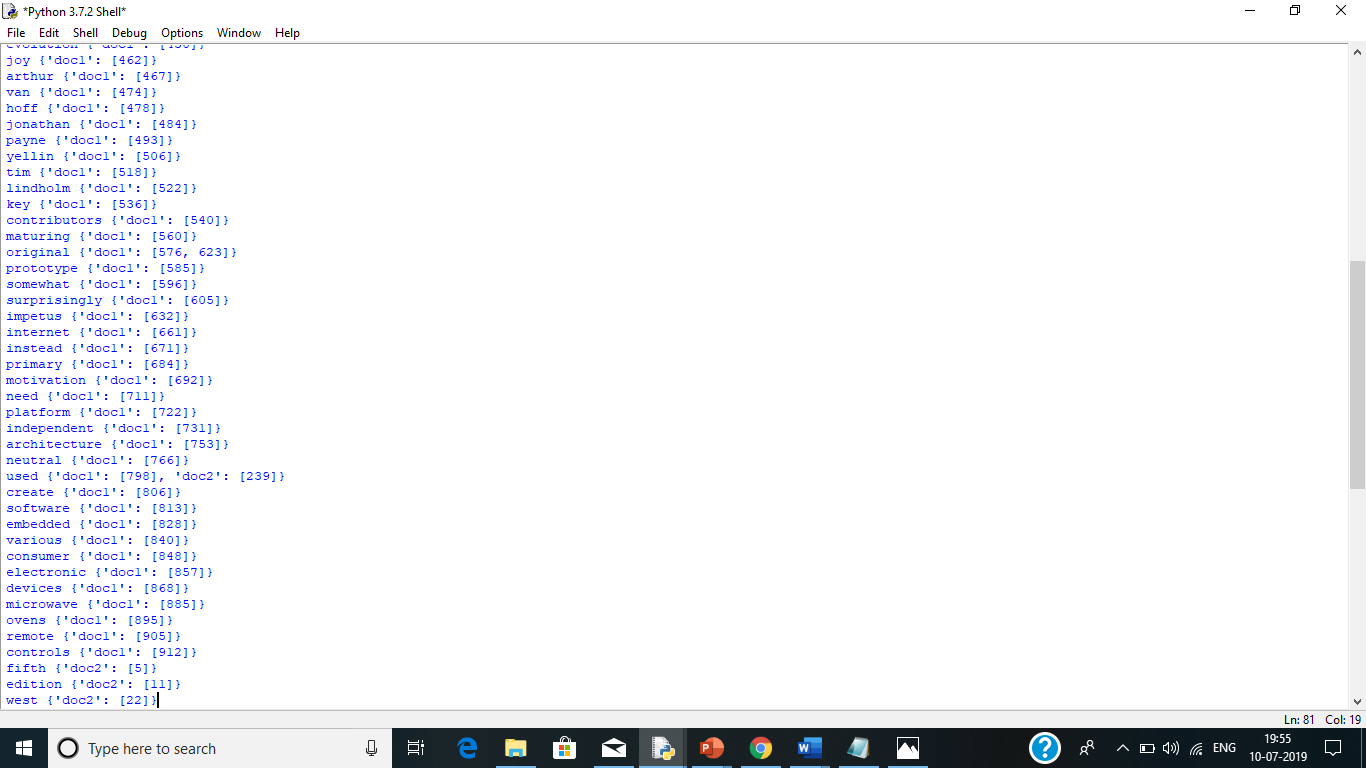
print (word, doc\_locations)

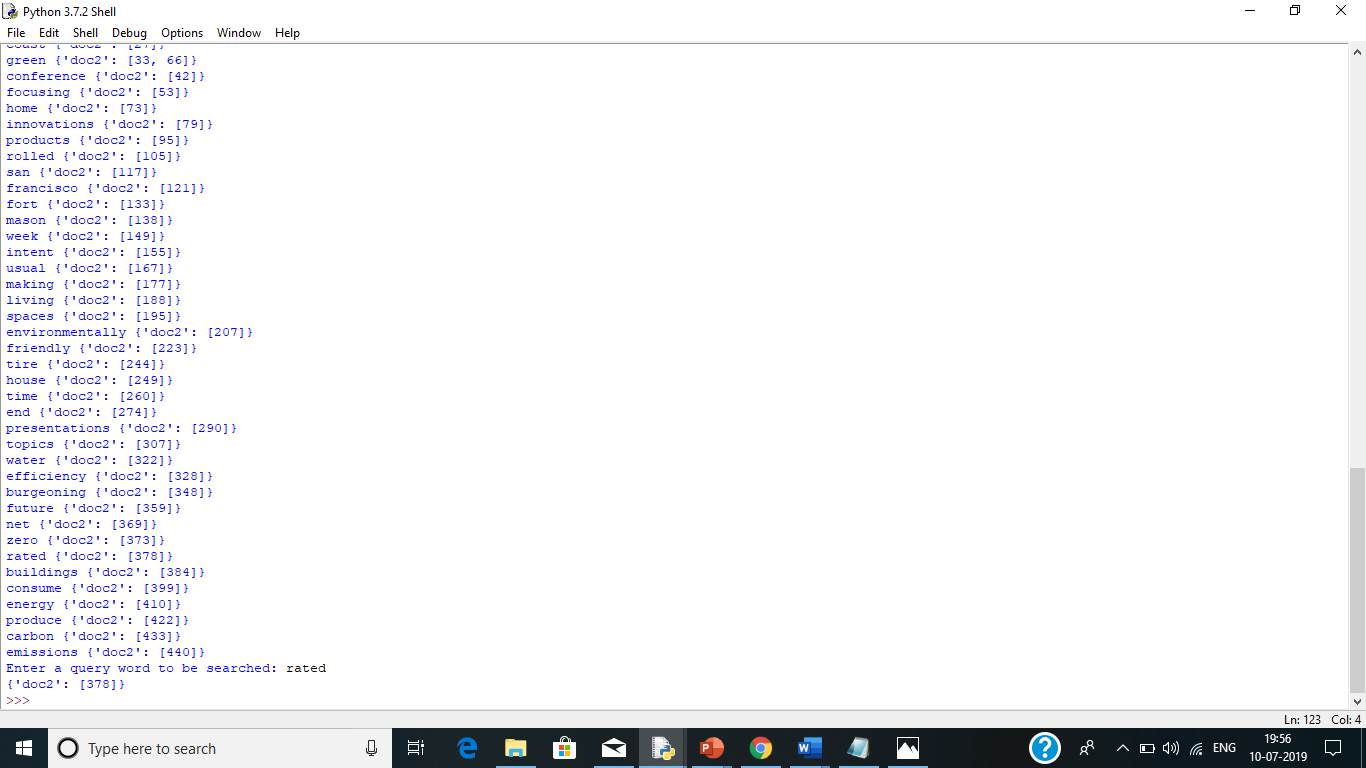
# Search something and print results

query = input("Enter a query word to be searched: ")

print(inverted[query])

OUTPUT





REFERENCES

1. <https://www.w3schools.com/python/>
2. <https://www.wikipedia.org/>
3. <https://www.geeksforgeeks.org/python-programming-language/>