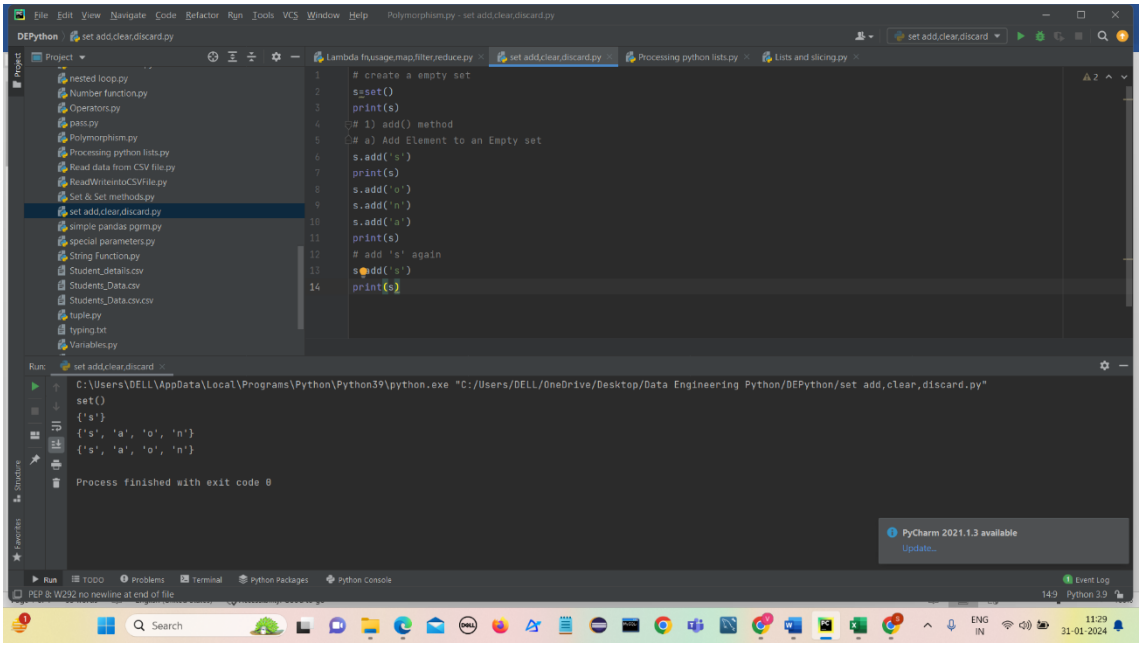
DAY-9

1) Set add() method:

It adds a given element to a set if the element is not present in the set

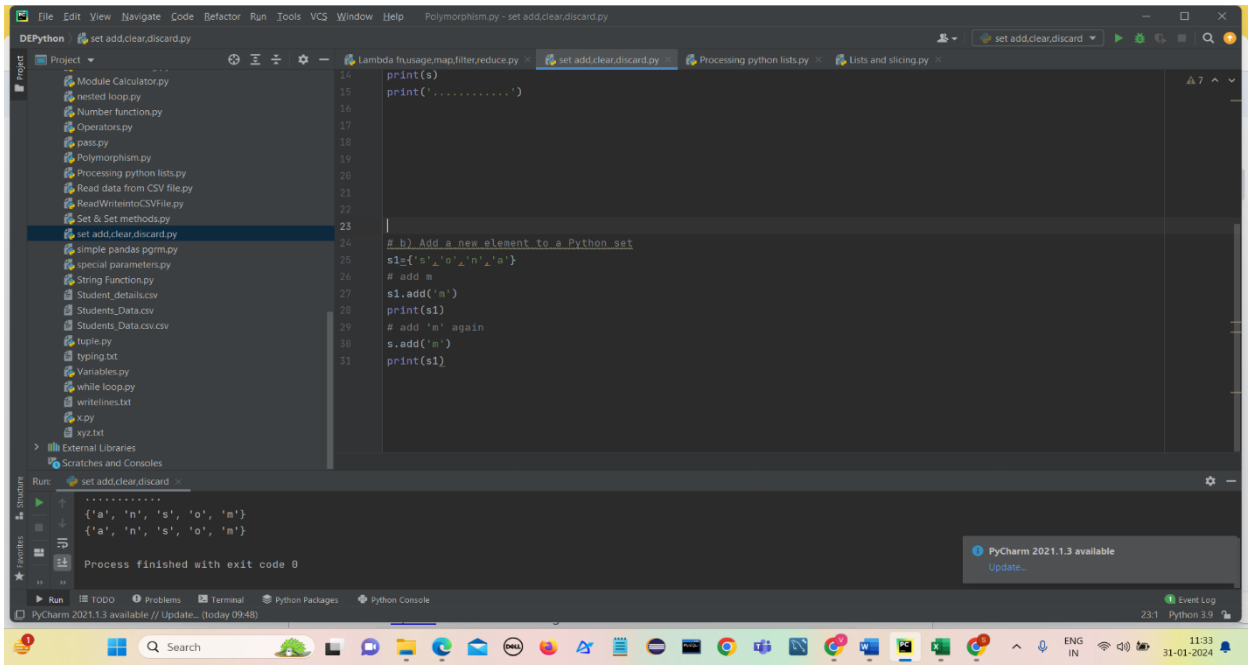
Syntax: set.add(element)

a) Add element to an empty set



b) Add a new element to a Python set

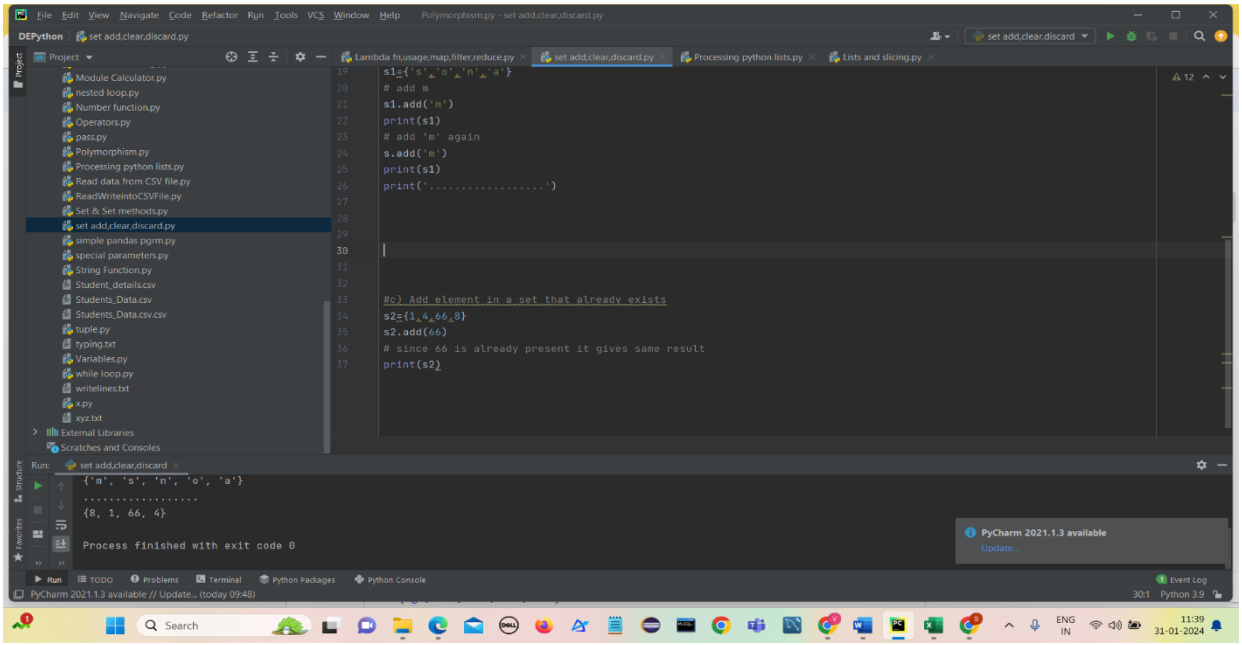
It is used to add a new element to the set if it is not existing in a set.



An element is added only if it is not present in the set else element doesn't get added since set doesn't allow duplicate elements.

c) Add element in a set that already exists

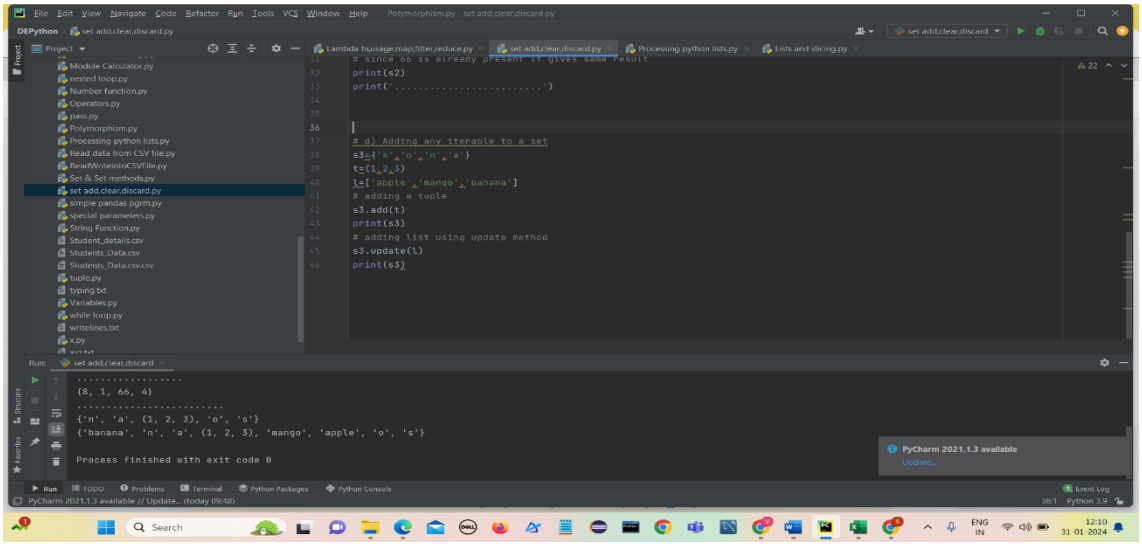
It is used to add an existing element to the set if it is existing in the Python set and check if it gets added or not.



d) Adding any iterable to set

We can add any Python iterable to a set using Python add or Python update function.

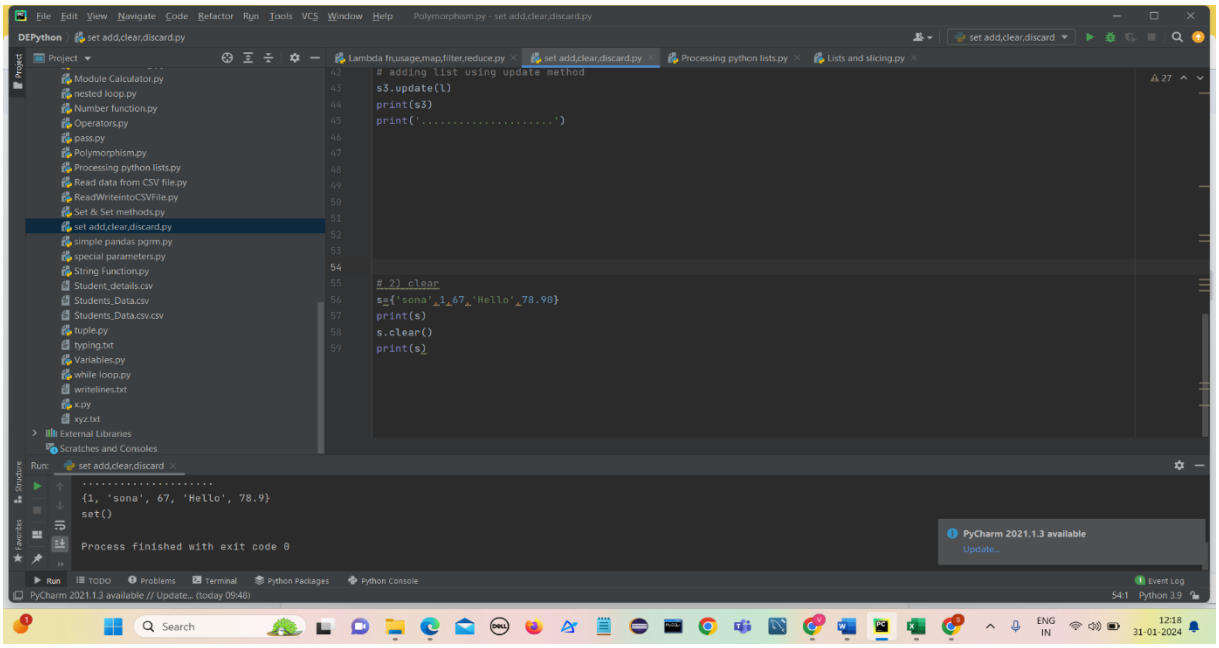
We can add tuple using add() method an list using update() method



2) Set clear() method:

Set clear() method removes all elements from the set and it doesnot take any parameters.

Syntax: set_name.clear()



3) Set copy() method:

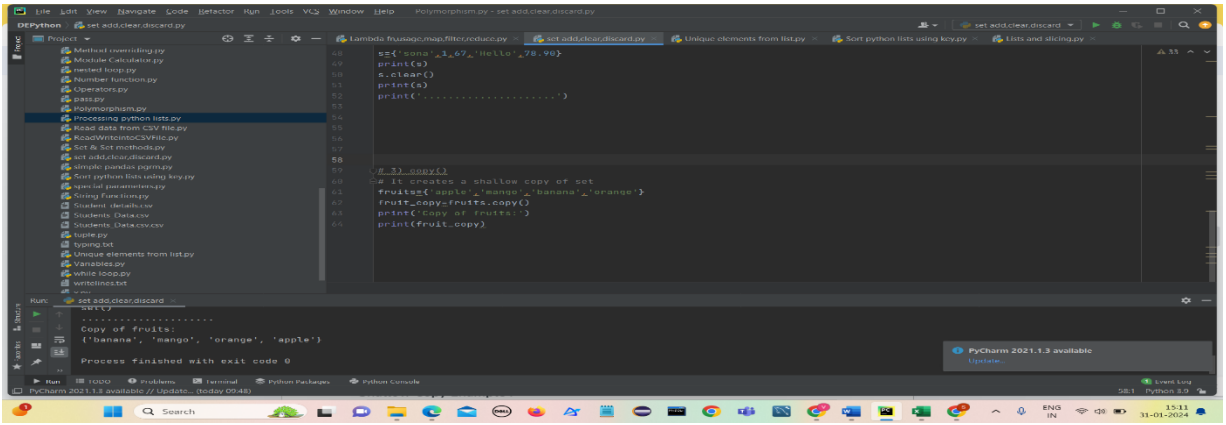
The copy() method returns a shallow copy of the set in python.

When we modify something in the copied set, changes are not reflected back in the original set.

Syntax: *set_name*.copy()

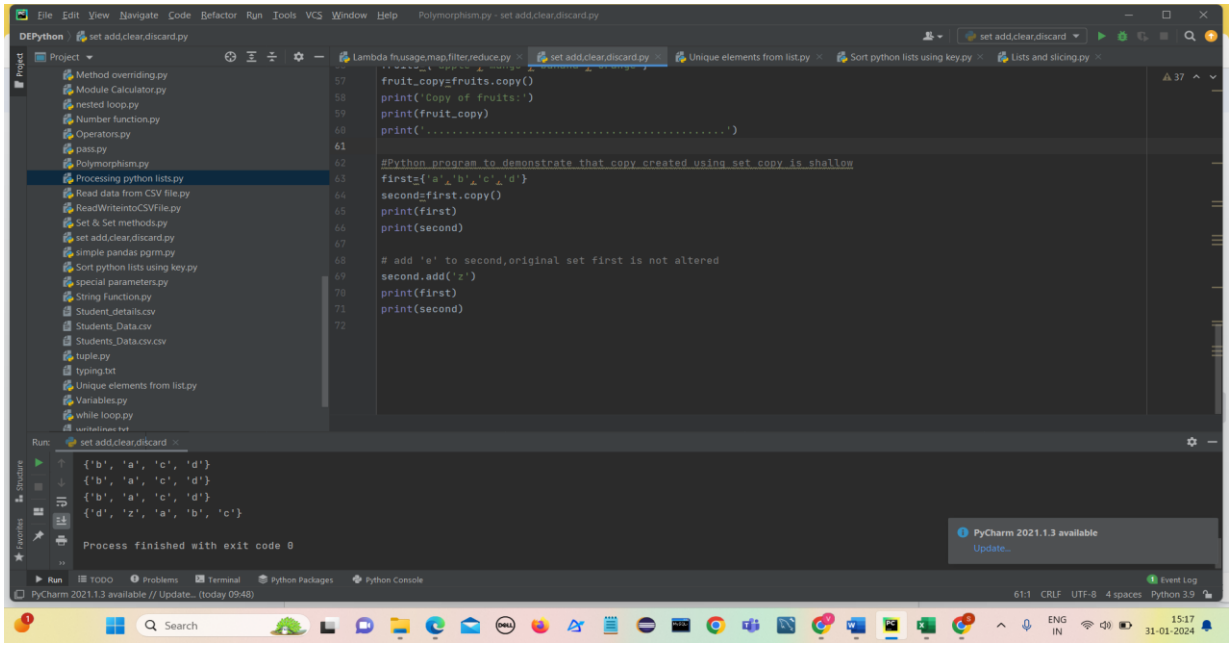
Parameters: The copy() method for sets doesn't take any parameters.

Return value: The function returns a shallow copy of the original set.



Shallow Copy Example :

Python program to demonstrate copy created is shallow



When we modify something in the copied set(second), changes are not reflected back in the original set(first).

4) Set discard method():

- Python **discard()** is a built-in method to remove elements from the set.
- The **discard()** method takes exactly one argument.
- This method does not return any value.

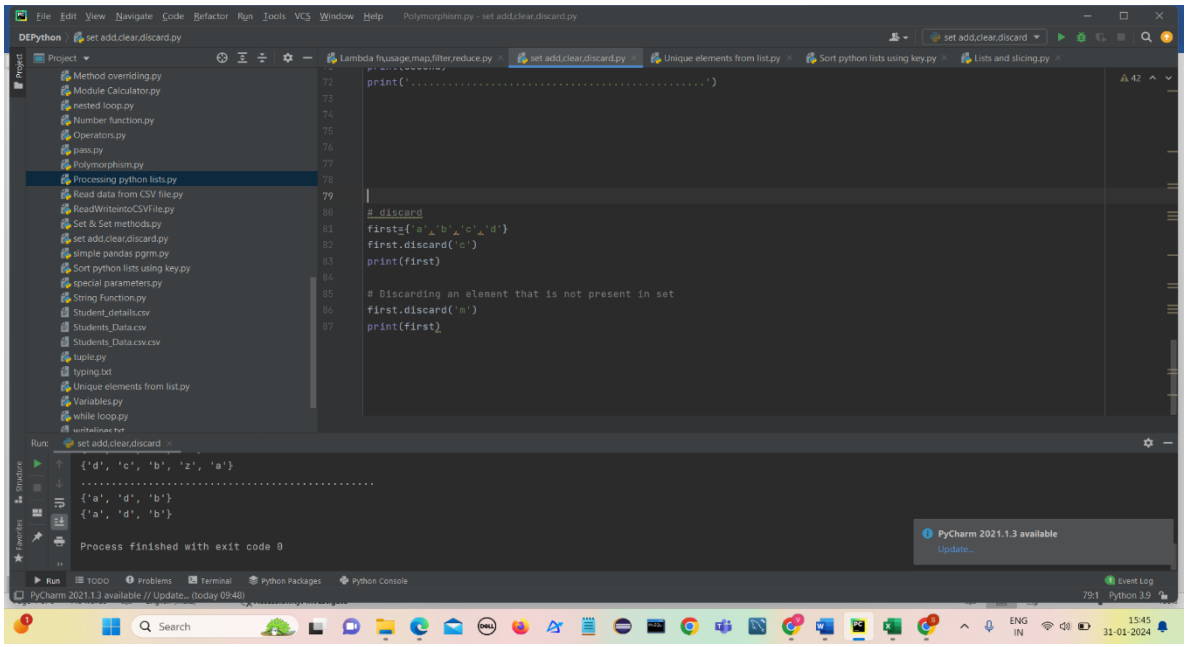
Syntax

set.discard(element)

Parameter: element – an item to remove from the set.

ReturnValue : return – discard() method doesn't return any value.

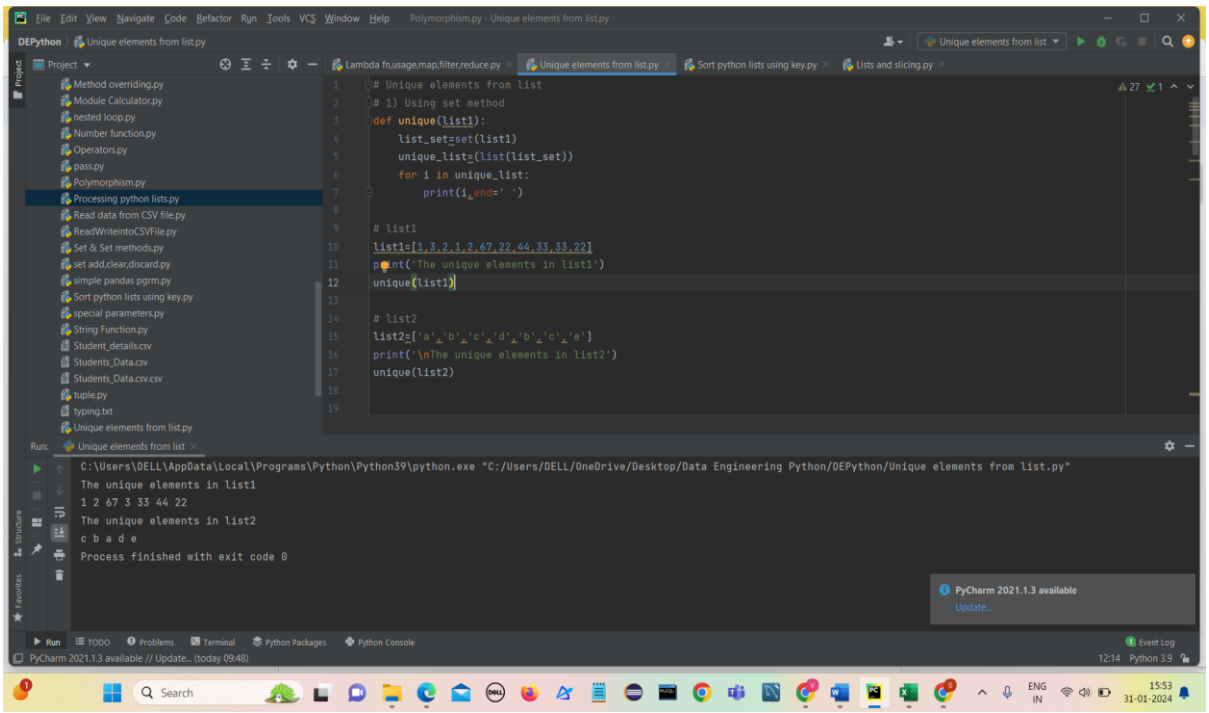
If we discard any element that is not existing in the set, it doesn't give any error.



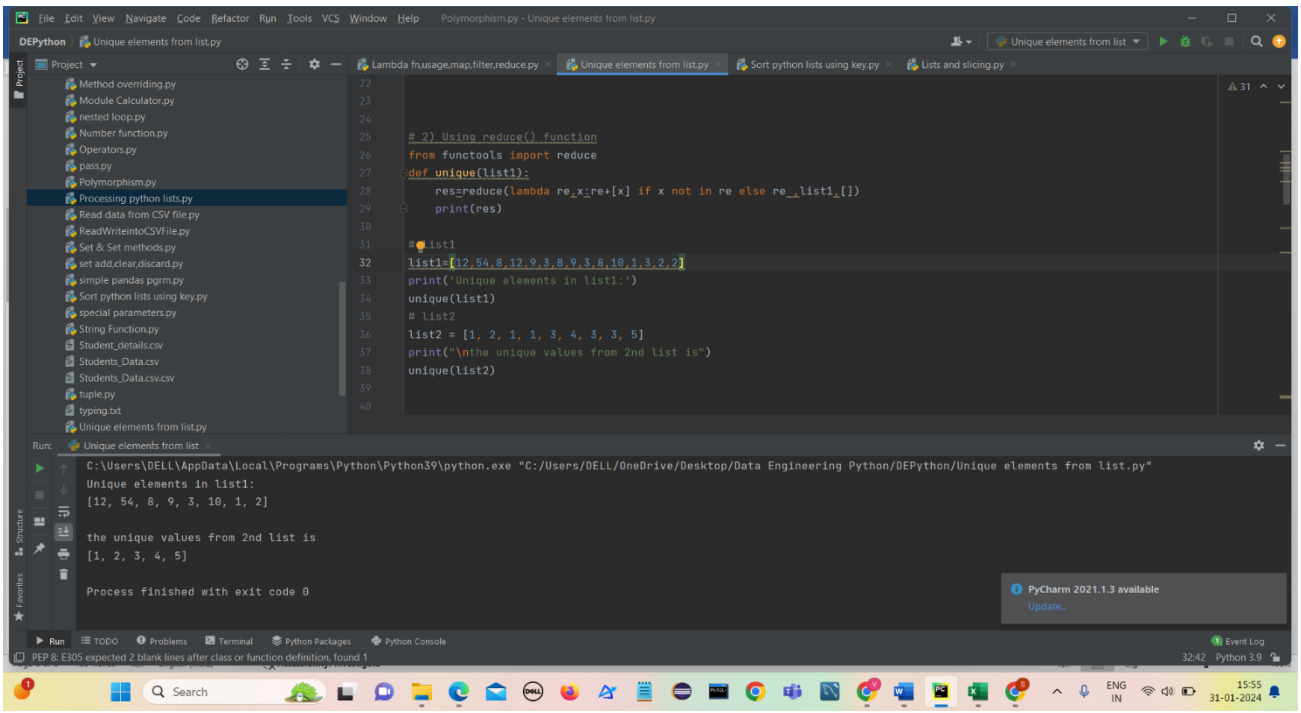
Get unique elements from the list

a) Using set Method:

- Insert the values of the list in a set.(convert the list into set)
- Set doesn’t allow duplicate values
- Now convert the set into list again and this resulted list contains all unique elements.



b) Using reduce() function

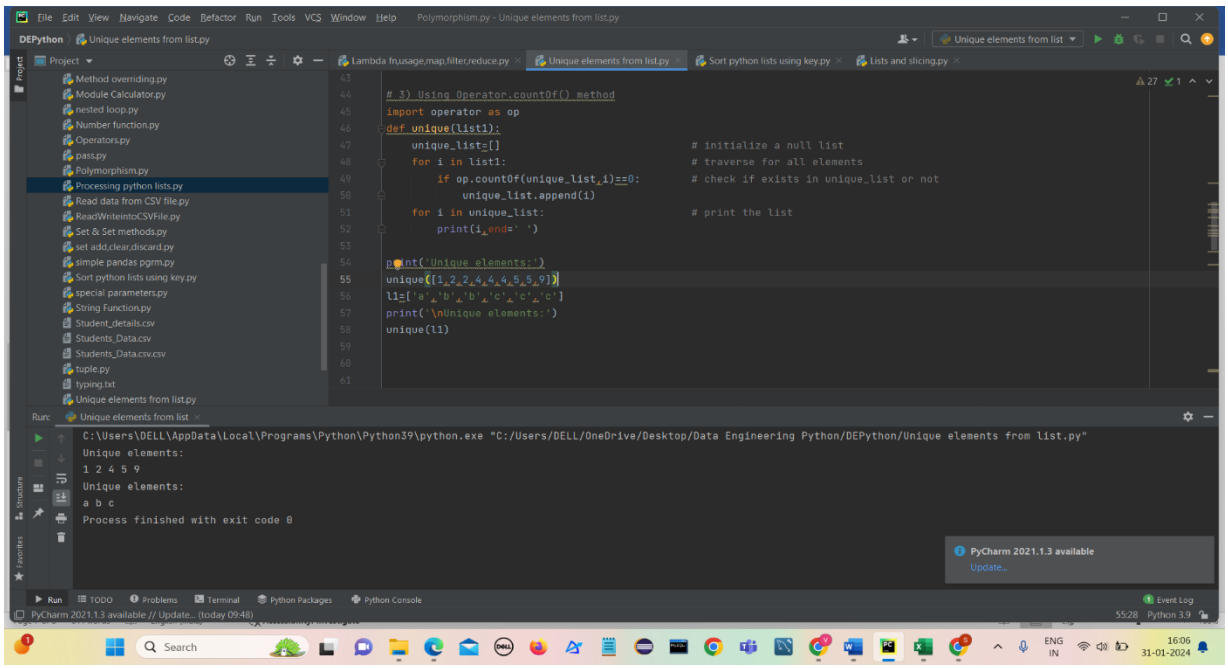


- **reduce()** takes three arguments: a function (lambda function), an iterable (**list1**), and an initial value (**[]**).
- The lambda function takes two arguments **re** (which accumulates the result) and **x** (the current element from the list).
- Inside the lambda function, **re+[x]** appends the current element to the accumulator list **re** if **x** is not already in **re**, effectively filtering out duplicates.
- If **x** is already in **re**, the lambda function returns **re** unchanged.
- **reduce()** iterates through the elements of **list1**, applying the lambda function to each element and accumulating the result.
- The initial value **[]** initializes the accumulator **re** as an empty list.
- The final result is the list of unique elements.

c) Using Operator.countOf() method

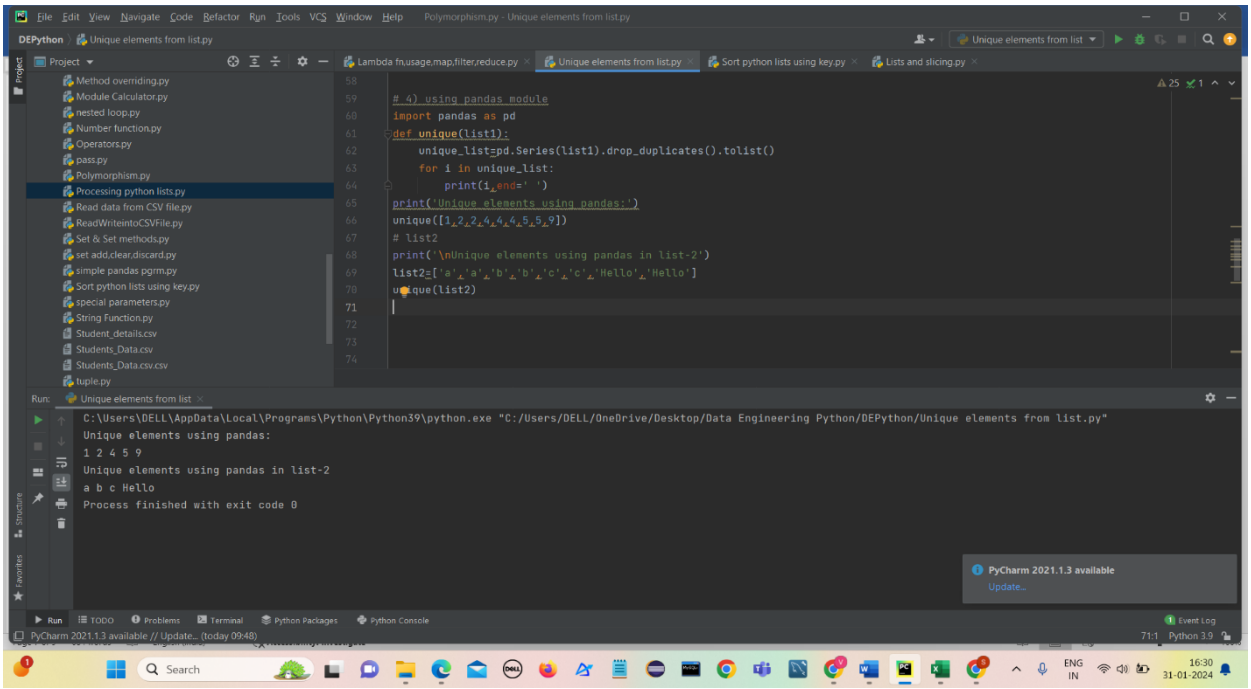
- import operator as op
- initialize a empty list(unique_list)
- traverse all elements
- using countOf() check if element is present in unique_list
- If countOf is 0 append that element to unique_list

- Print all the elements in the end which are all unique elements.



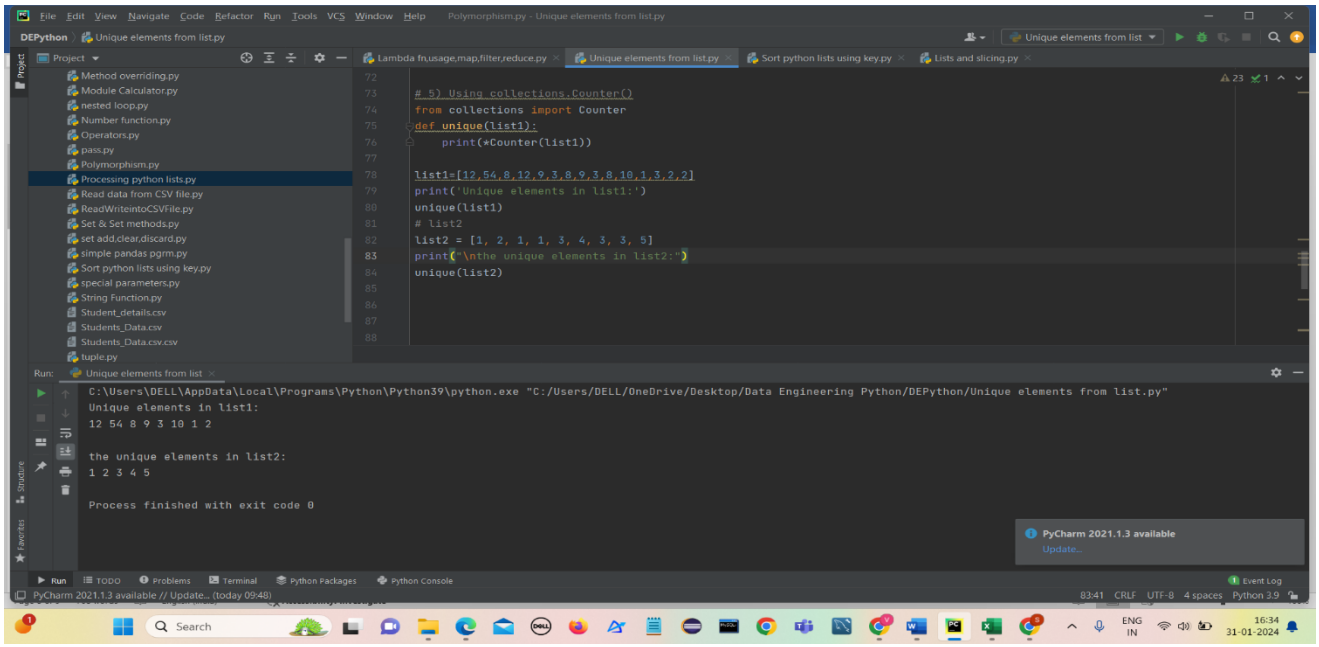
d) Using Pandas module

- The ‘unique’ function utilizes Pandas to create a Series from ‘list1’
- It then employs ‘drop_duplicates()’ to eliminate duplicates and obtain a list of unique values.
- Finally it iterates through the unique list and prints each element.



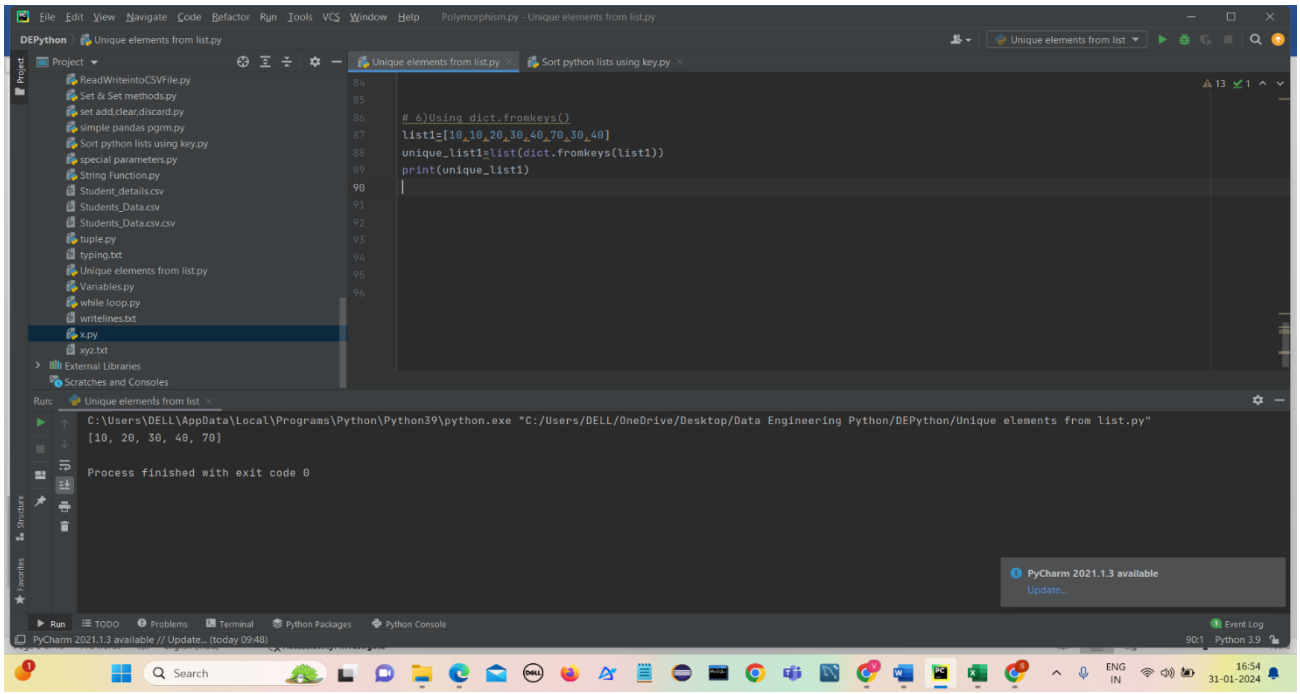
e) Using collections.Counter()

- import Counter() from collections
- print all the keys of Counter elements
  or
- we can print directly by using the * symbol.

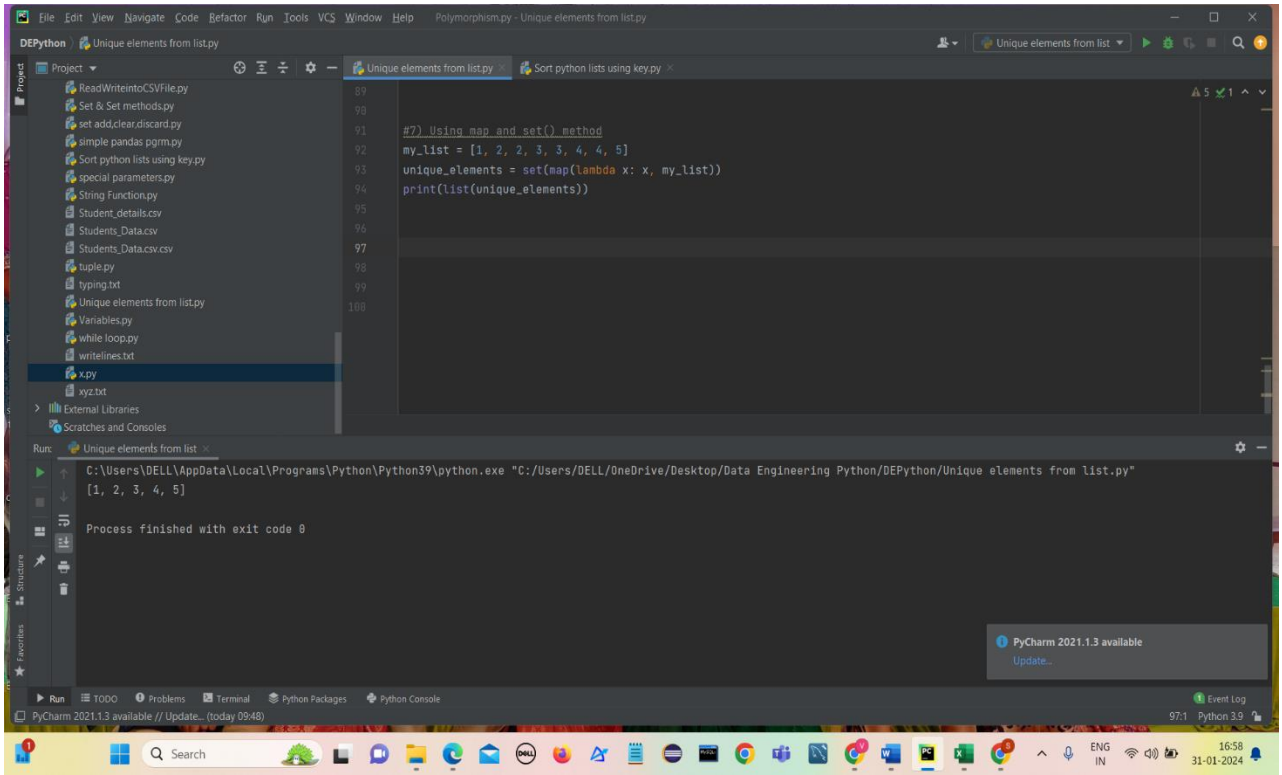


f) Using dict.fromkeys()

- First define a list that consists of duplicate elements.
- Then we need to use a variable in which we will store the result after using the fromkeys() method.
- We need to convert that result into a list
- fromkeys() method is part of the dictionary so by default it returns a dictionary with all the unique keys and None as their values.
- Keys in dictionary cannot have duplicate values.

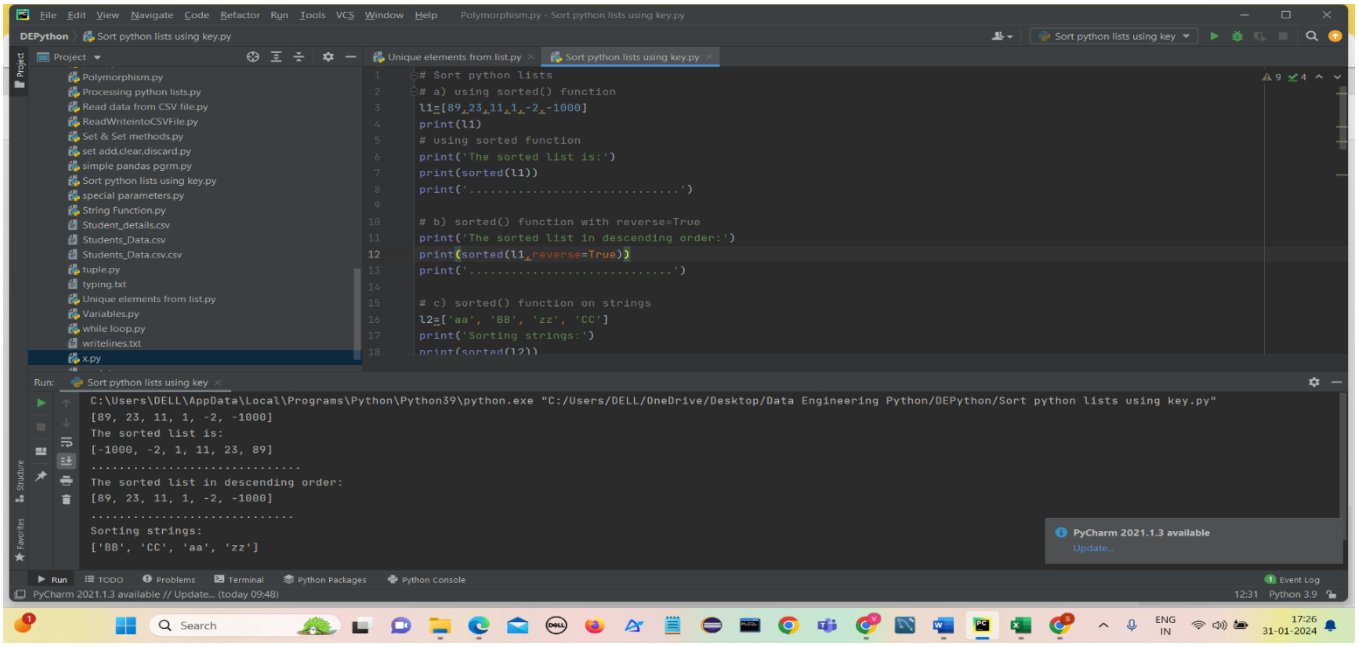


g) Using map and set() method



Sorting python Lists

- sorted(list) function takes a list and returns a new list with those elements in sorted order and the original list is not changed.
- The sorted() function can be customized through optional arguments.
- The sorted() optional argument reverse=True, e.g. sorted(list, reverse=True), makes it sort backwards.
    a) Using sorted() function
    b) Using sorted(reverse=True)
    c) Using sorted() function on strings



Using sorted(reverse=True) returns elements in Descending order.

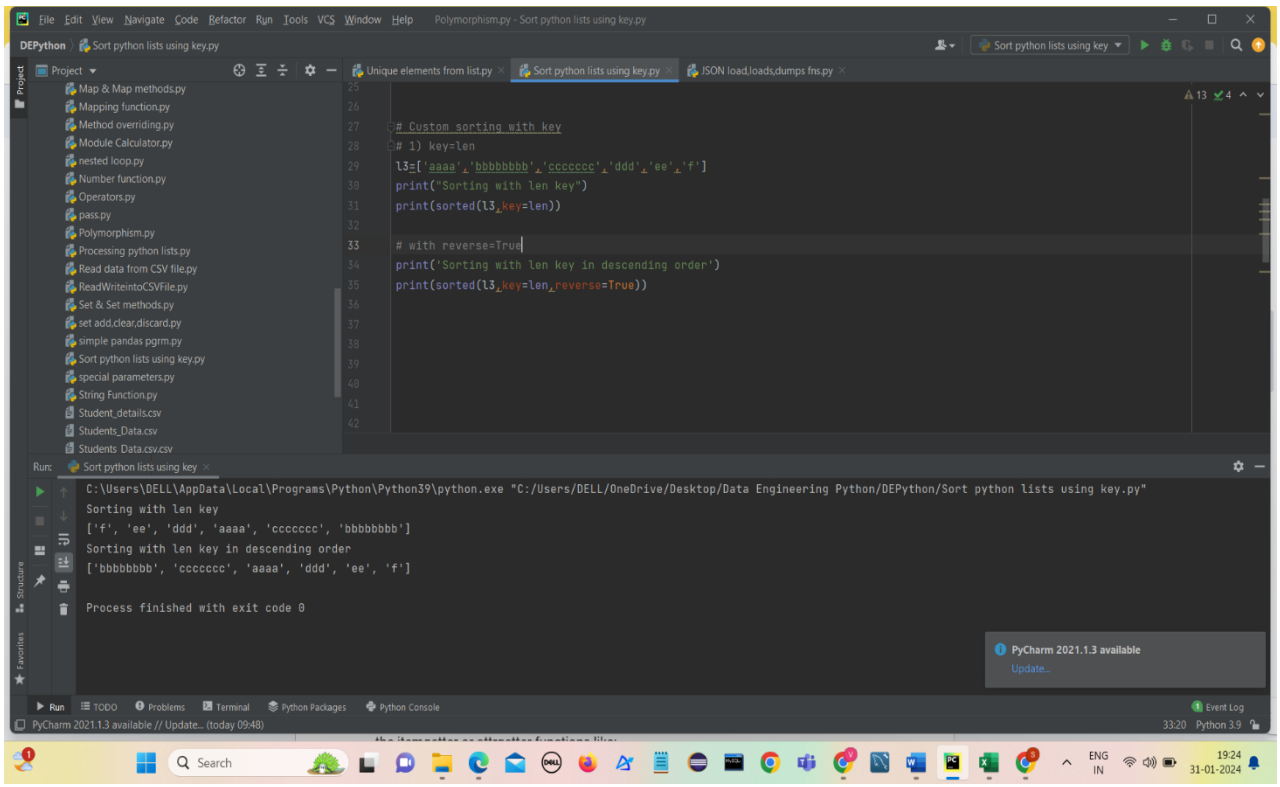
Custom Sorting With key=

- For more complex custom sorting, sorted() takes an optional "key=" specifying a "key" function .
- It transforms each element before comparison.
- The key function takes in 1 value and returns 1 value
- The returned "proxy" value is used for the comparisons within the sort.

a) key=len()

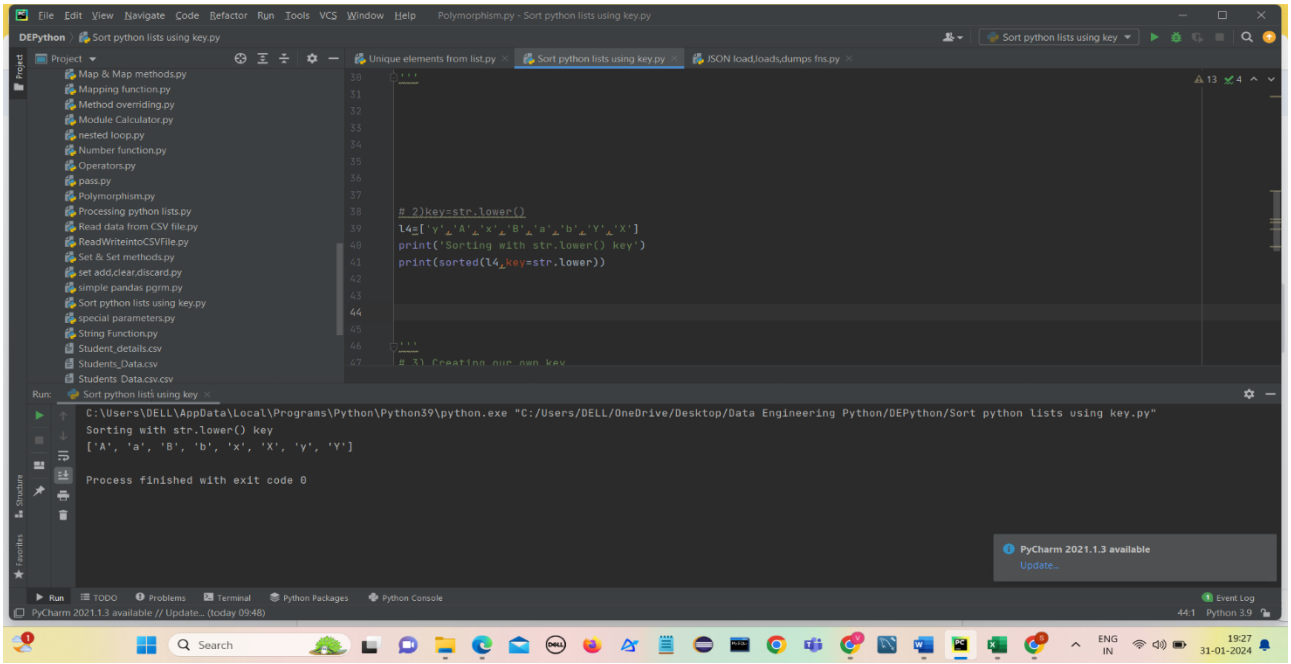
- key=len (the built in len() function) sorts the strings by length, from shortest to longest.

- The sort calls len() for each string to get the list of proxy length values, and then sorts with those proxy values.

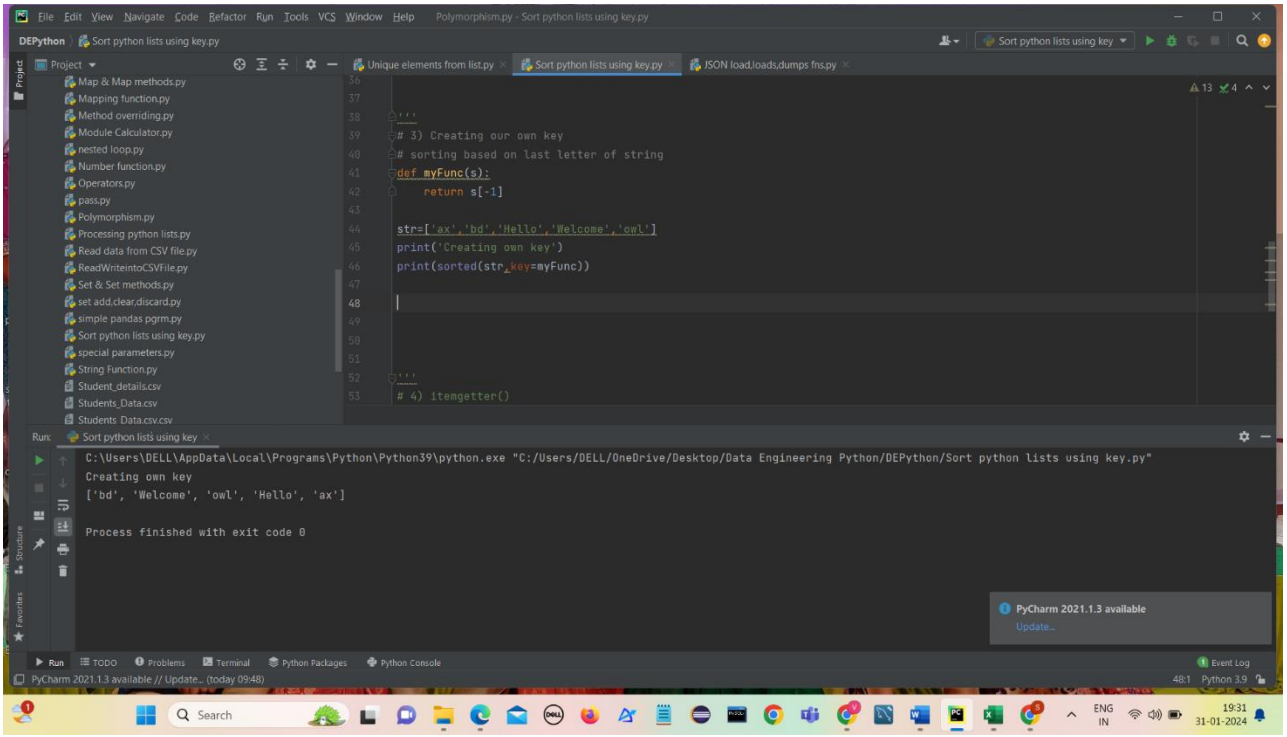


b) key=str.lower()

Key function is a way to force the sorting to treat uppercase and lowercase the same.

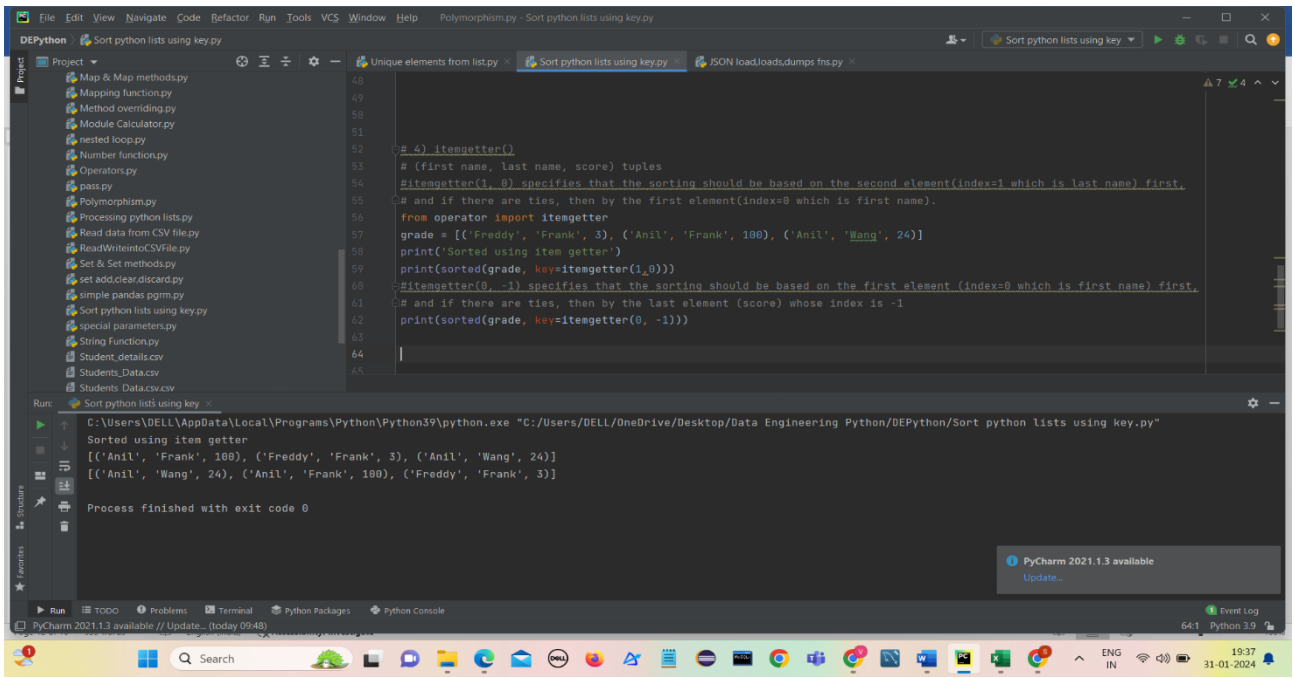


c) Creating our own key
Sorting based on last letter of string



itemgetter()

(firstname , lastname, score) tuples

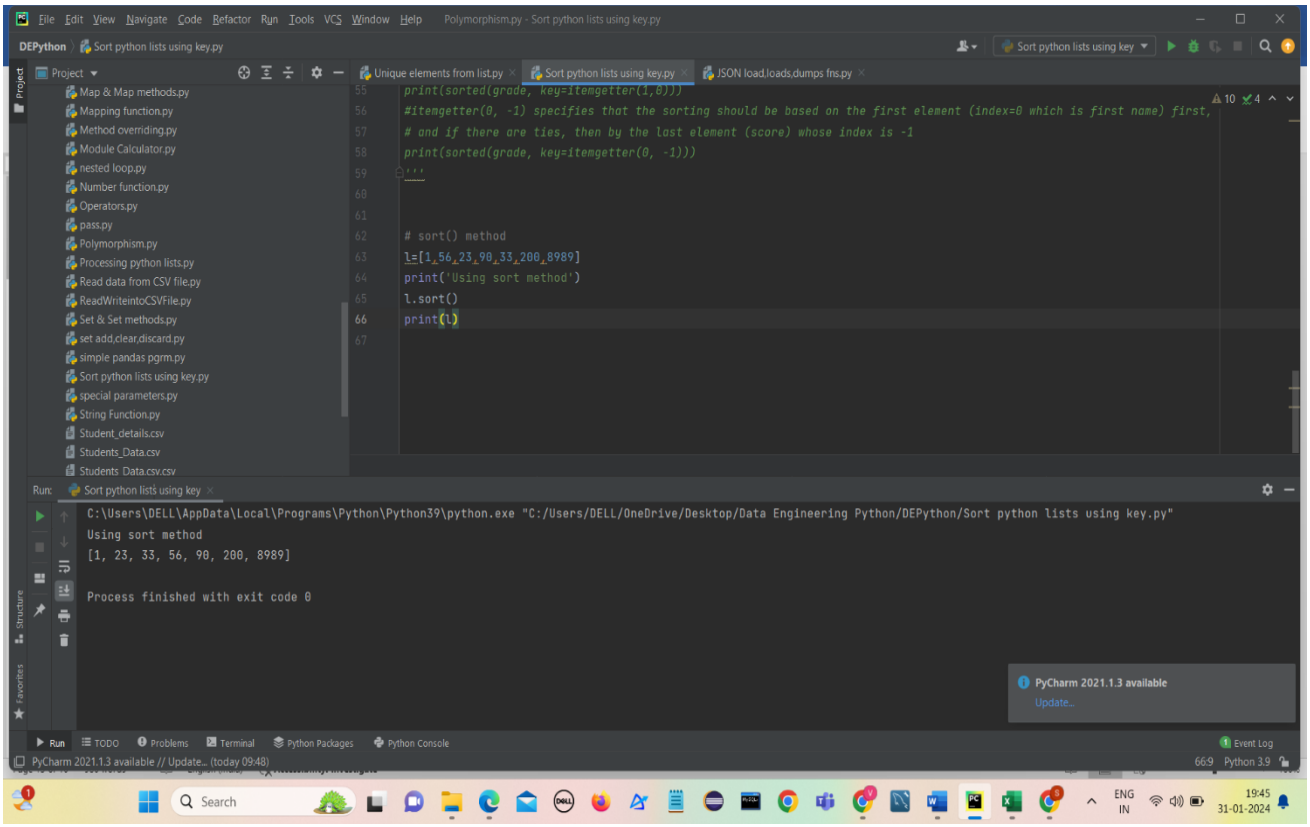


itemgetter (1,0) specifies that sorting should be based on the second element(index=1 which is last name) first, and if there are ties, then by the first element(index=0 which is first name)

itemgetter (0,-1) specifies that sorting should be based on thefirst element(index=0 which is first name) first, and if there are ties, then by the last element(index=-1 which is score)

sort():

- sort() method on a list sorts that list into ascending order, e.g. list.sort().
- The sort() method changes the underlying list and returns None.



JSON

- **JSON** stands for **J**ava**S**cript **O**bject **N**otation. It is a format for structuring data.
- This format is used by different web applications to communicate with each other.
- JSON is the replacement of the XML data exchange format in JSON.
- It is easy to struct the data compare to XML.
- It supports data structures like arrays and objects and the JSON documents that are rapidly executed on the server.
- It is also a Language-Independent format that is derived from JavaScript.

  Features of JSON:
  - Easy to understand
  - Format
  - Support
  - Dependency

Example of JSON String

s = '{"id":01, "name": "Emily", "language": ["C++", "Python"]}'

The syntax of JSON is considered a subset of the syntax of JavaScript including the following:

1) **Name/Value pairs:** Represents Data, the name is followed by a colon(**:**), and the Name/Value pairs are separated by a comma(**,**).
2) **Curly braces:** Holds objects.
3) **Square brackets:** Hold arrays with values separated by a comma (**,**).

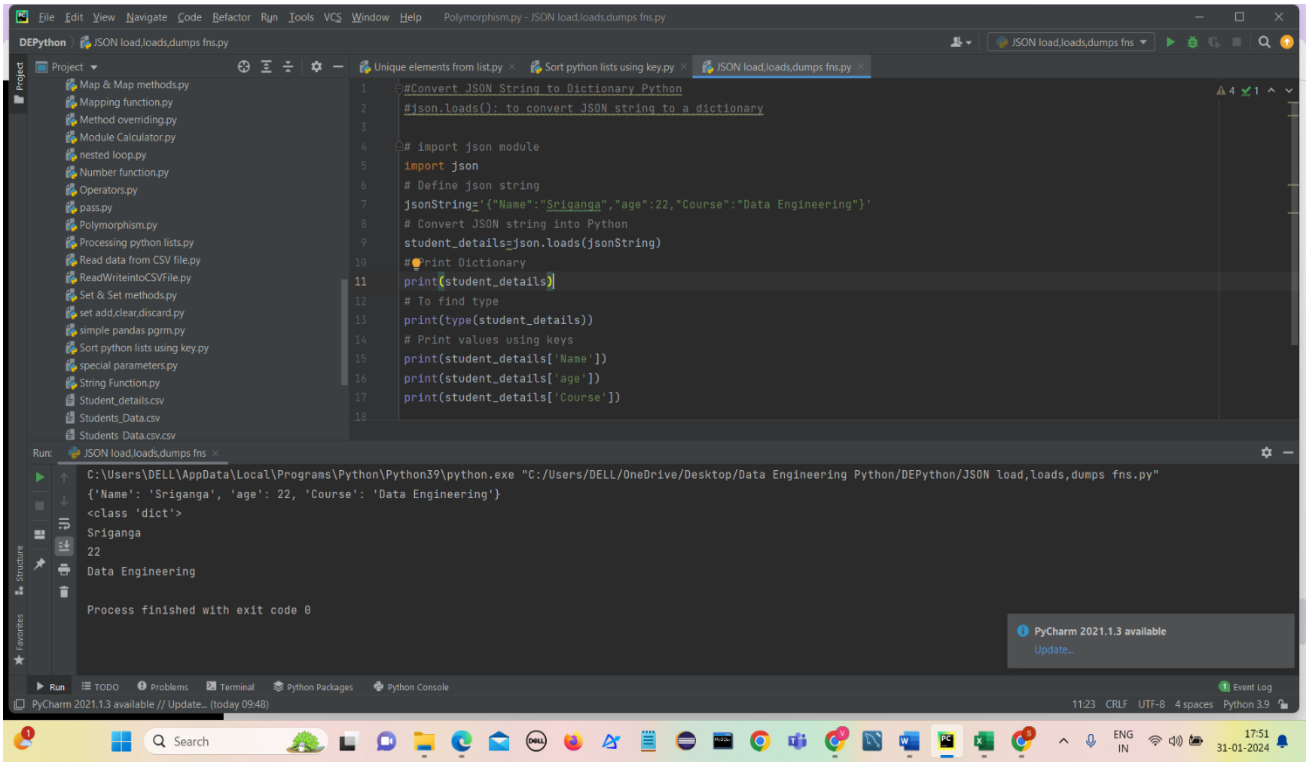
Example of JSON file:

```
{
   "employee": [
    {
          "id": "01",
          "name": "Amit",
         "department": "Sales"
    },
    {
          "id": "04",
          "name": "sunil",
         "department": "HR"
    }
   ]
}
```

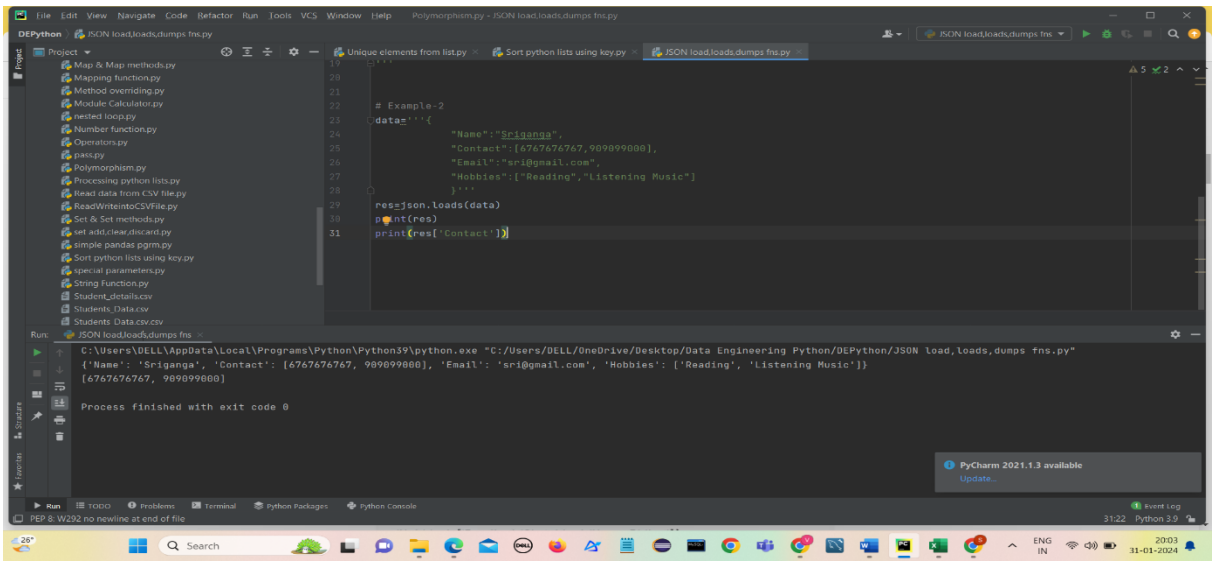
Convert JSON String to Dictionary Python

Json.loads(): converts JSON string into Python Dictionary

- import json module .
- Define JSON .
- Convert JSON string to Python dictionary by passing it to json.loads() in parameter.
- Print the dictionary and their values using the keys.



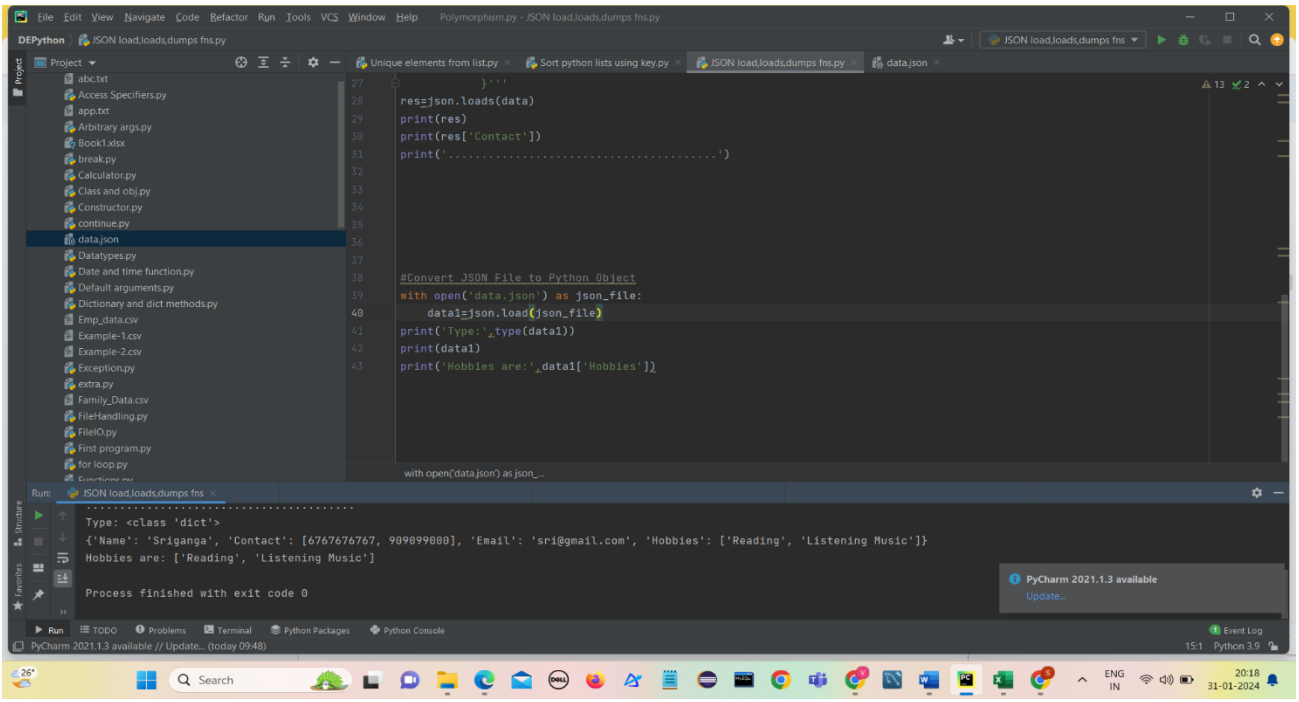
Example-2



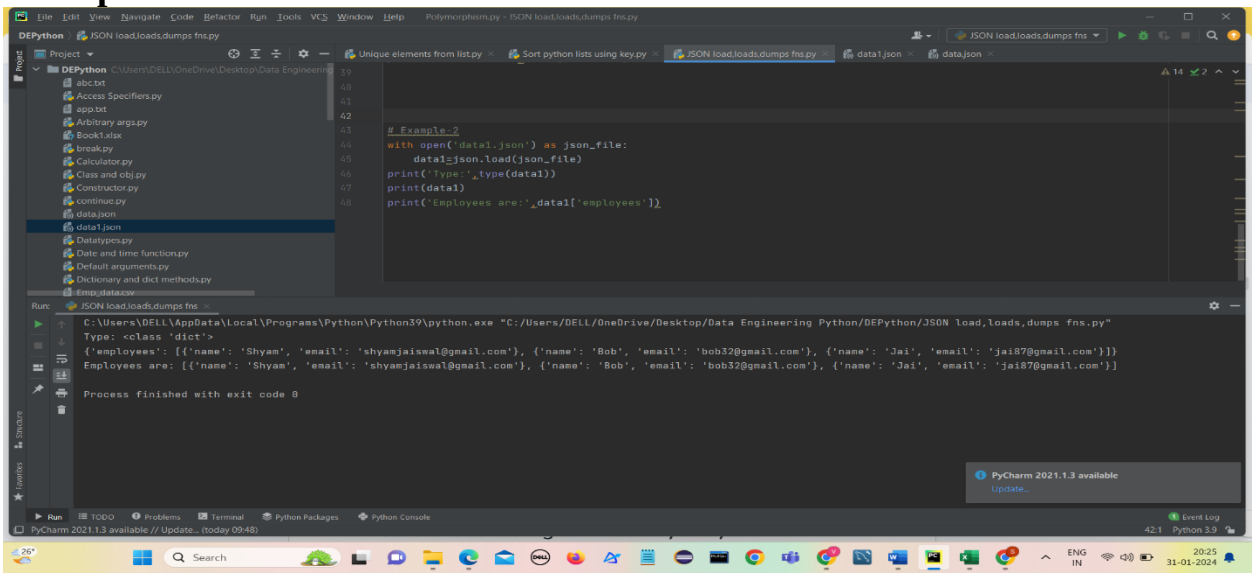
Convert JSON File to Python Object

Json.load()- It accepts a file object and read JSON data from a file and convert it into a dictionary.

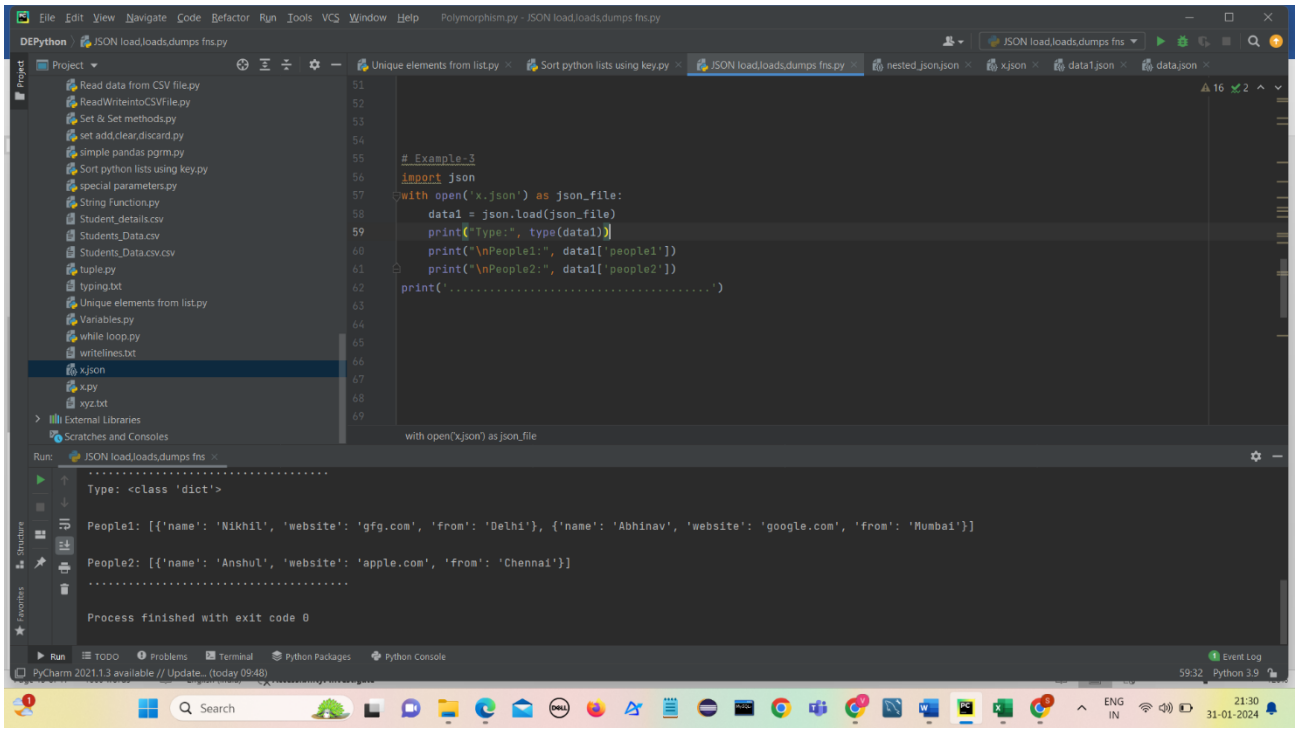
- open the “data.json” file
- convert the file to Python object using the json.load() method
- print the type of data after conversion
- print the dictionary.



Example-2

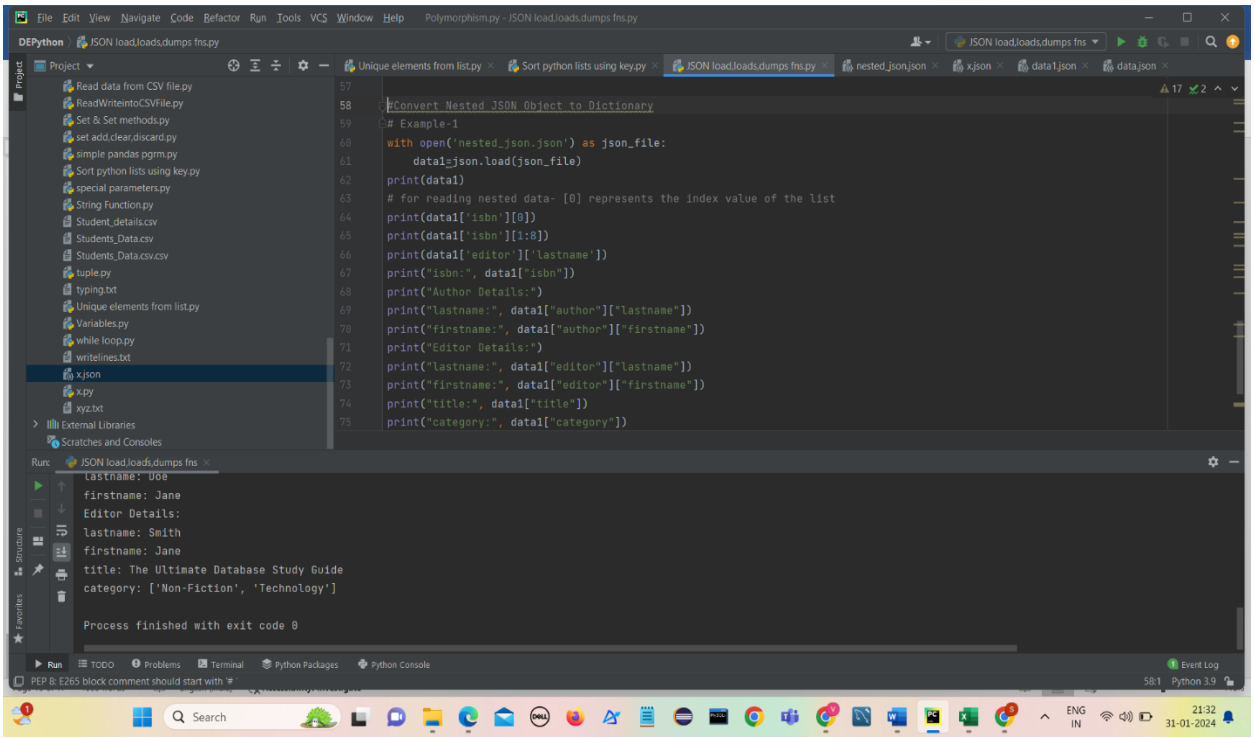


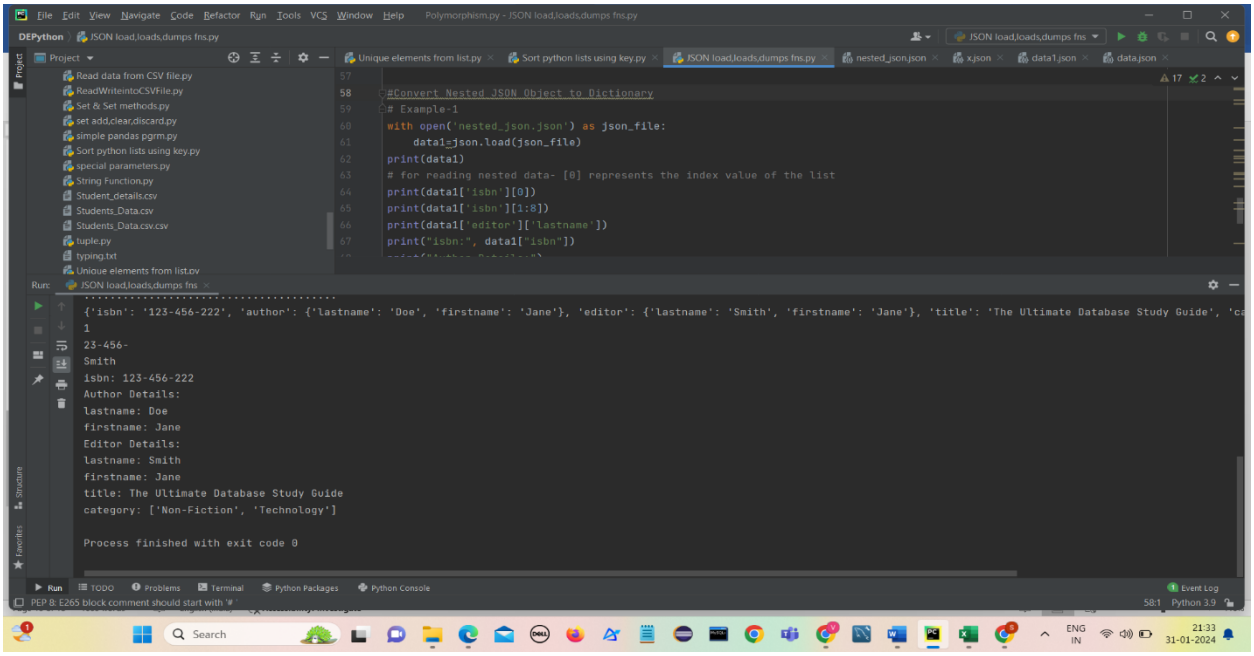
Example-3



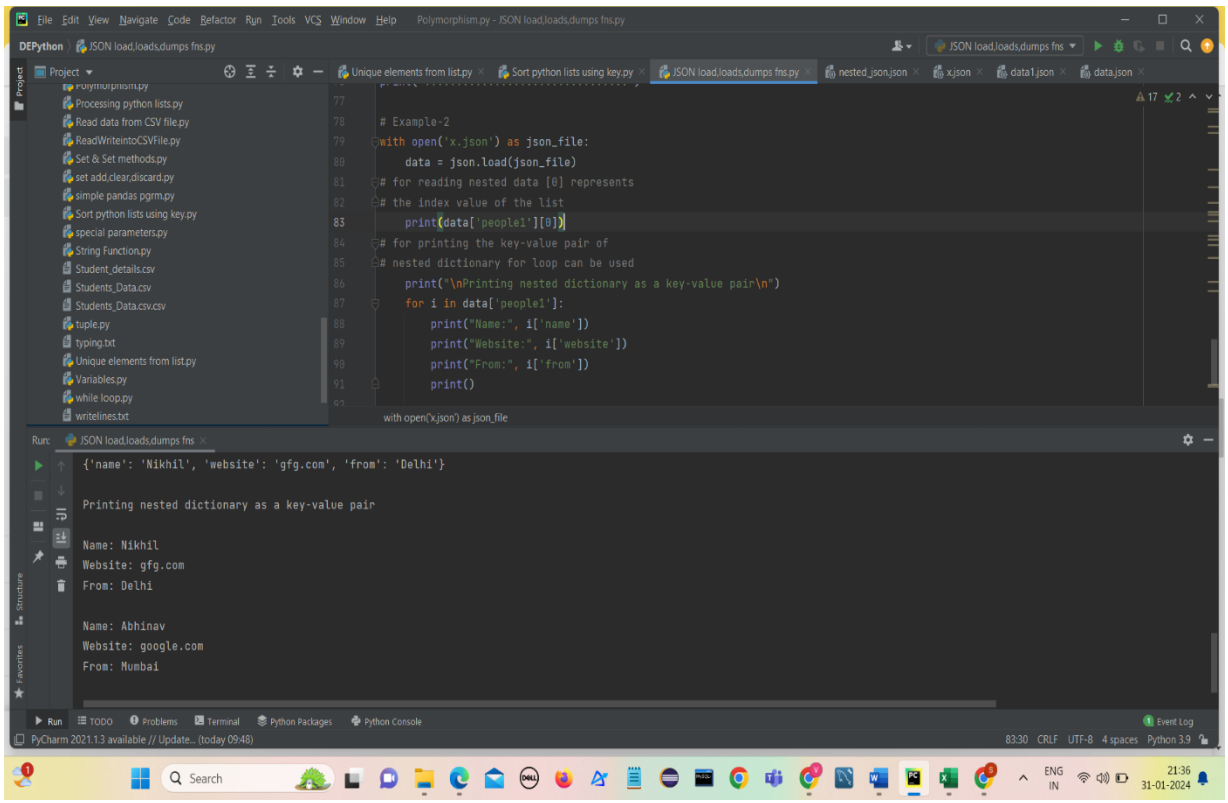
Convert Nested JSON Object to Dictionary

In this example, we will convert the nested JSON into a Python dictionary.




#Convert Nested JSON Object to Dictionary
Example-1
with open('nested_json.json') as json_file:
data1=json.load(json_file)
print(data1)
# for reading nested data- [0] represents the index value of the list
print(data1['isbn'][0])
print(data1['isbn'][1:8])
print(data1['editor']['lastname'])
print("isbn:", data1["isbn"])
{'isbn': '123-456-222', 'author': {'lastname': 'Doe', 'firstname': 'Jane'}, 'editor': {'lastname': 'Smith', 'firstname': 'Jane'}, 'title': 'The Ultimate Database Study Guide', 'c
1
23-456-
Smith
isbn: 123-456-222
Author Details:
lastname: Doe
firstname: Jane
Editor Details:
lastname: Smith
firstname: Jane
title: The Ultimate Database Study Guide
category: ['Non-Fiction', 'Technology']
Process finished with exit code 0

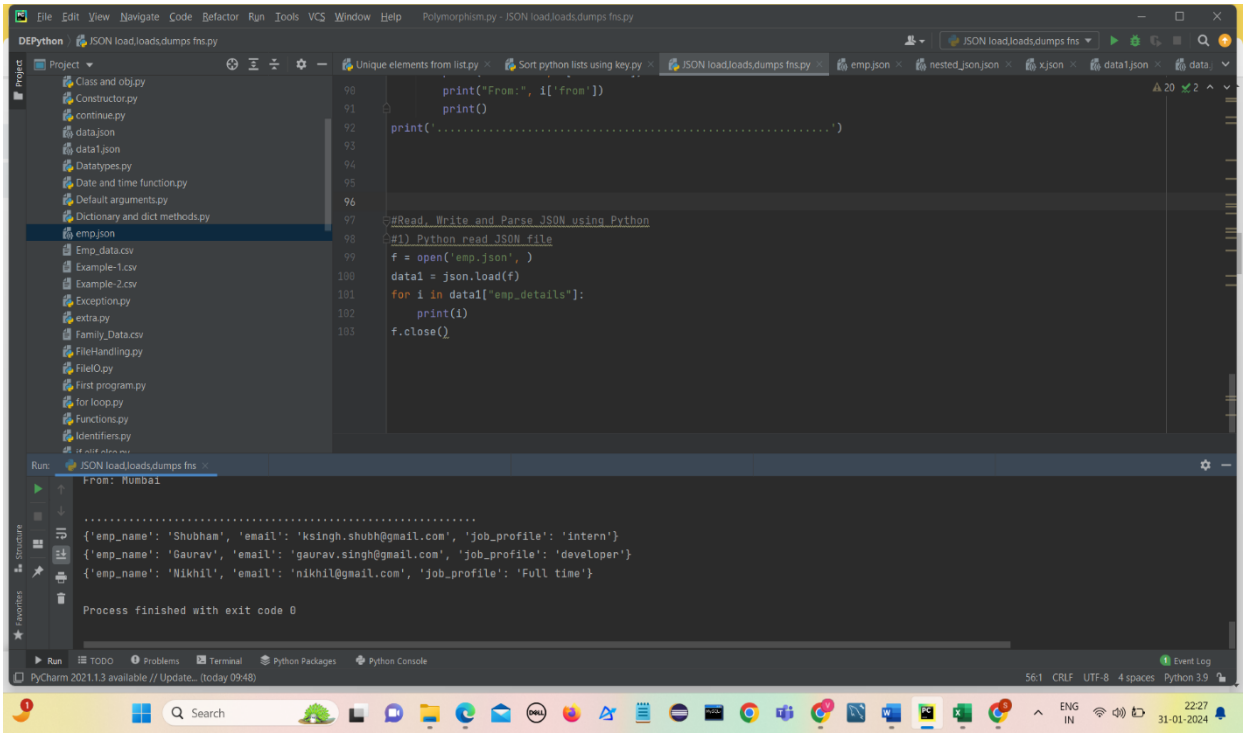
Example-2


Example-2
with open('x.json') as json_file:
data = json.load(json_file)
# for reading nested data [0] represents
# the index value of the list
print(data['people1'][0])
# for printing the key-value pair of
# nested dictionary for loop can be used
print("\nPrinting nested dictionary as a key-value pair\n")
for i in data['people1']:
print("Name:", i['name'])
print("Website:", i['website'])
print("From:", i['from'])
print()
{'name': 'Nikhil', 'website': 'gfg.com', 'from': 'Delhi'}
Printing nested dictionary as a key-value pair
Name: Nikhil
Website: gfg.com
From: Delhi
Name: Abhinav
Website: google.com
From: Mumbai

Read, Write and Parse JSON using Python

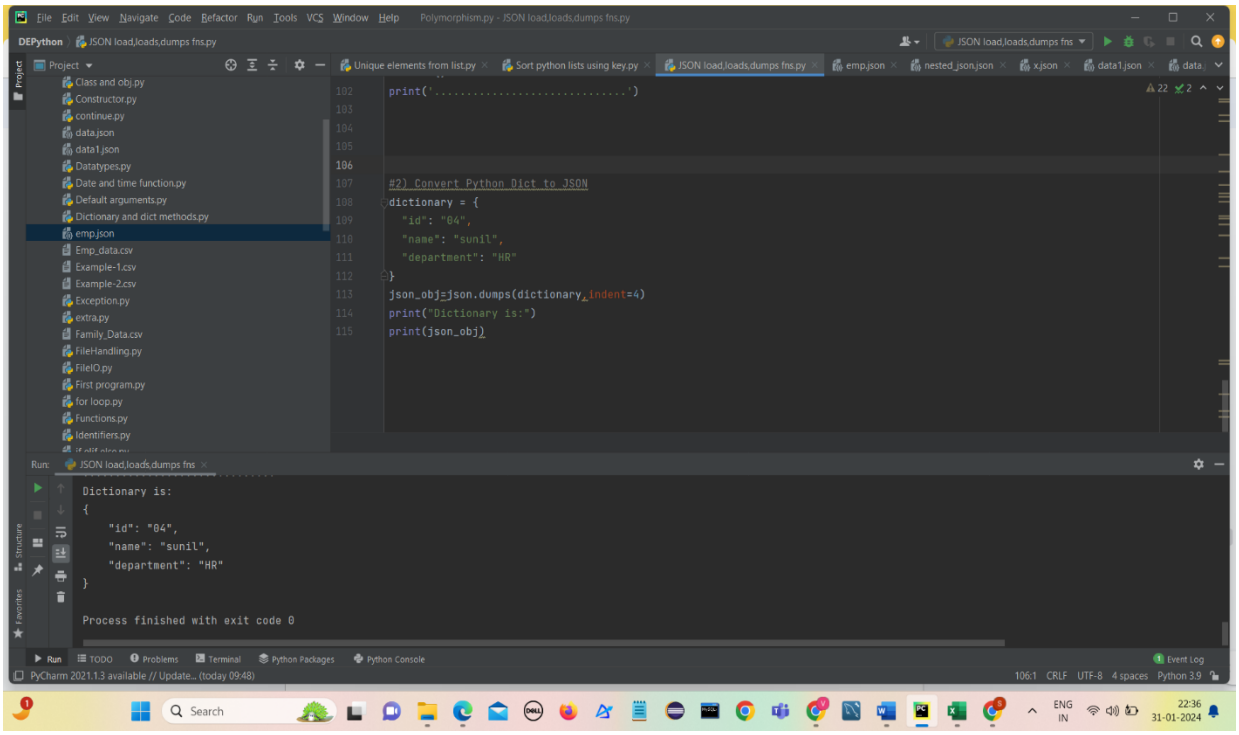
1) Python read JSON file

- We have used the open() function to read the JSON file.
- Then, the file is parsed using json.load() method which gives us a dictionary named data.



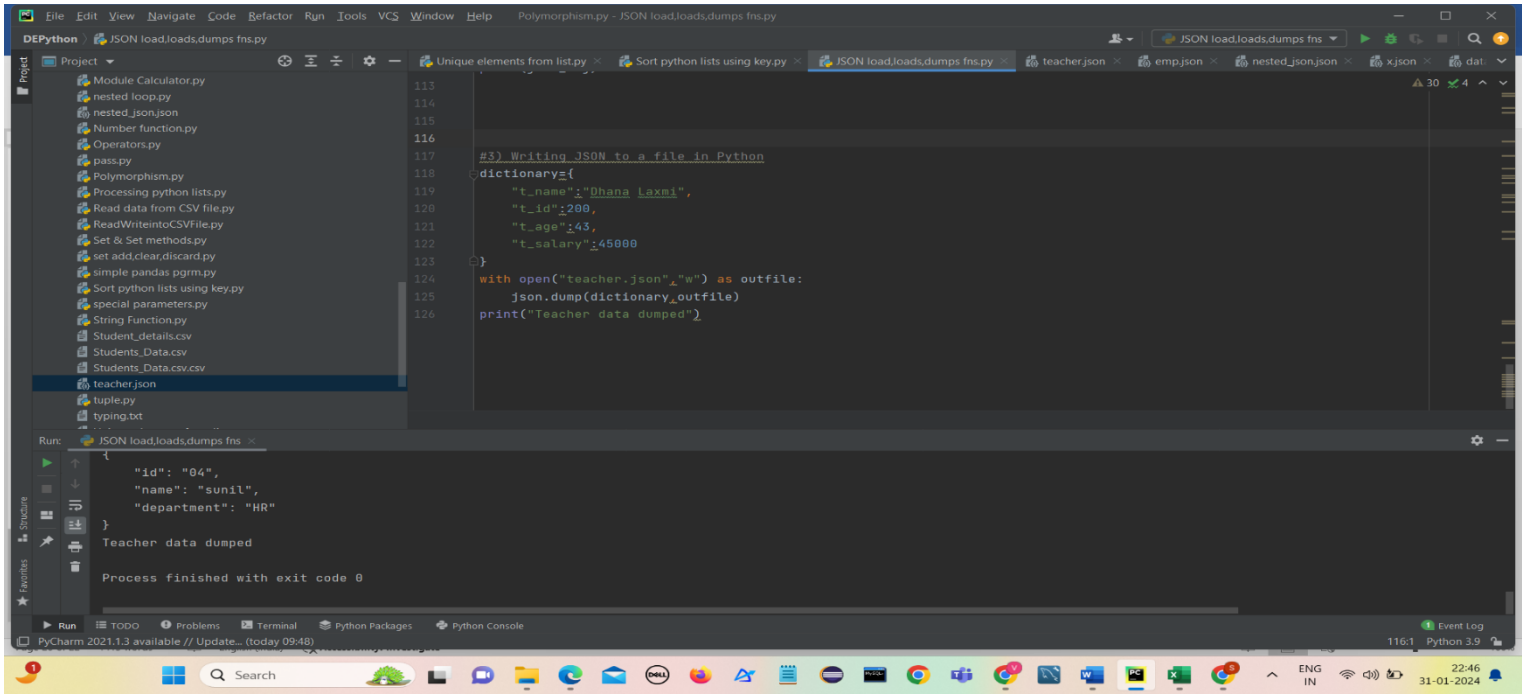
2) Convert Python Dict to JSON

- We are converting a Python dictionary to a JSON object using **json.dumps()** method
- make a small dictionary with some key-value pairs and then passed it into json.dumps() method with ‘indent=4’ to convert this Python dictionary into a JSON object.
- As we have given the value of indent to 4 there are four whitespaces before each data as seen in the output.

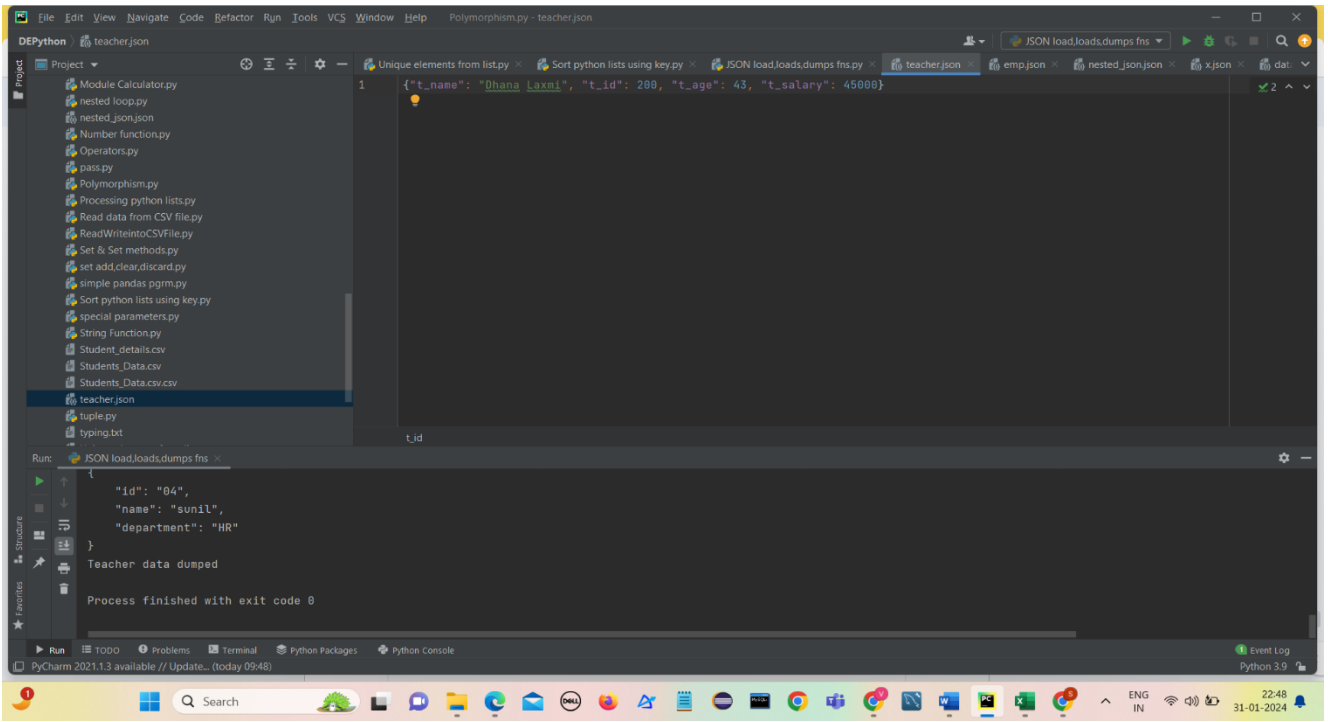


3) Writing JSON to a file in Python

- We can write JSON to file using json.dump() function of JSON module and file handling in Python.
- In the below program, we have opened a file named sample.json in writing mode using **'w'**.
- The file will be created if it does not exist.
- Json.dump() will transform the Python dictionary to a JSON string and it will be saved in the file sample.json.



A new file teacher.json is created and data is written into it



4) Python Pretty Print JSON

- When we convert a string to JSON the data is in a less readable format.
- To make it more readable we can use pretty printing by passing additional arguments in json.dumps() function such as **indent** and **sort_keys** as used