Name : Parth Nandedkar
Date : 31 Jan 2024
Topics : Python (Unique values, sorting, JSON)
Batch : Data Engineering Batch-1

## Getting Unique Values :

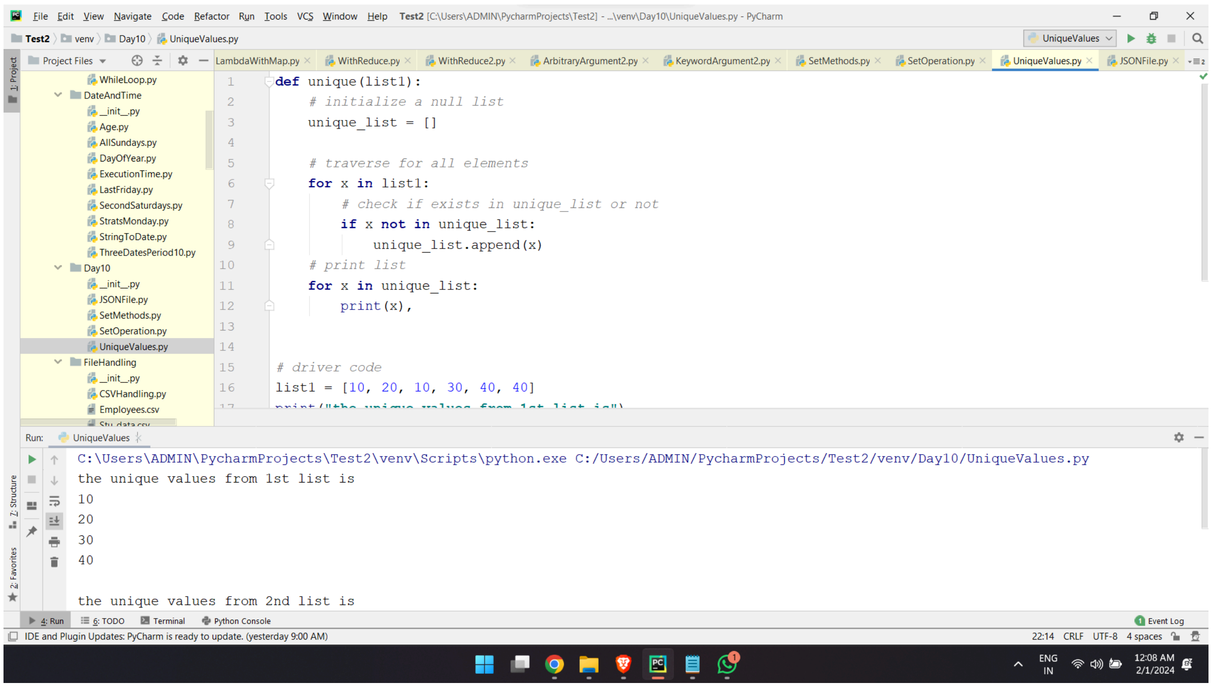
Get Unique Values from a List Using Set Method :

To get unique values from a list in Python, one common approach is to use the set() method. Here's a brief explanation:

Using Set Method to Get Unique Values from a List:
Set Data Structure:

In Python, a set is an unordered collection of unique elements. It does not allow duplicate values.
Sets are defined using curly braces {} or the set() constructor.
Converting List to Set:

To obtain unique values from a list, you can convert the list to a set using the set() method.
This process automatically removes any duplicate elements from the list, leaving only unique values in the set.



```python
def unique(list1):
    # initialize a null list
    unique_list = []

    # traverse for all elements
    for x in list1:
        # check if exists in unique_list or not
        if x not in unique_list:
            unique_list.append(x)
    # print list
    for x in unique_list:
        print(x),


# driver code
list1 = [10, 20, 10, 30, 40, 40]
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UniqueValues.py
the unique values from 1st list is
10
20
30
40

the unique values from 2nd list is
```

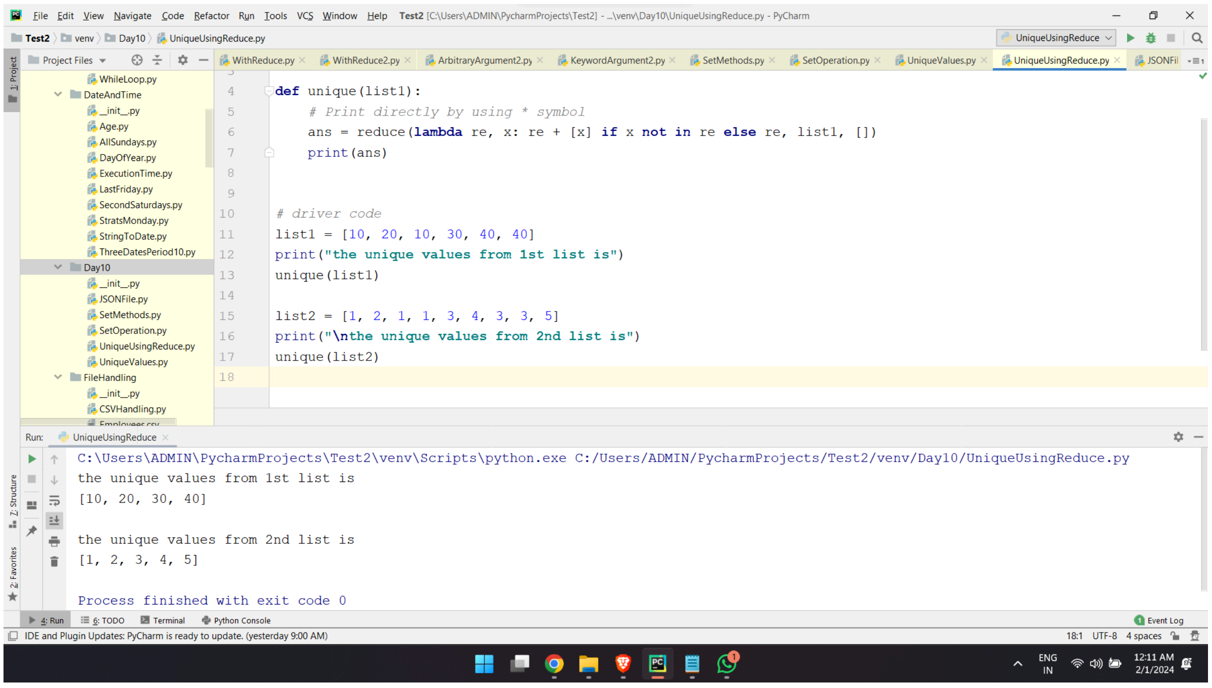
Using reduce() :

The reduce() function will iteratively combine elements while keeping only unique values.

The lambda function takes two parameters: unique_set (the accumulated unique values) and value (the current value from the list).

unique_set | {value} performs a union operation, adding the current value to the set of unique values.

The reduce() function is used to successively apply the lambda function to each element of the list, starting with an empty set (set()).


Test2 [C:\Users\ADMIN\PycharmProjects\Test2] - ...\venv\Day10\UniqueUsingReduce.py - PyCharm
Test2 › venv › Day10 › UniqueUsingReduce.py
Project Files

```
def unique(list1):
    # Print directly by using * symbol
    ans = reduce(lambda re, x: re + [x] if x not in re else re, list1, [])
    print(ans)


# driver code
list1 = [10, 20, 10, 30, 40, 40]
print("the unique values from 1st list is")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique values from 2nd list is")
unique(list2)
```

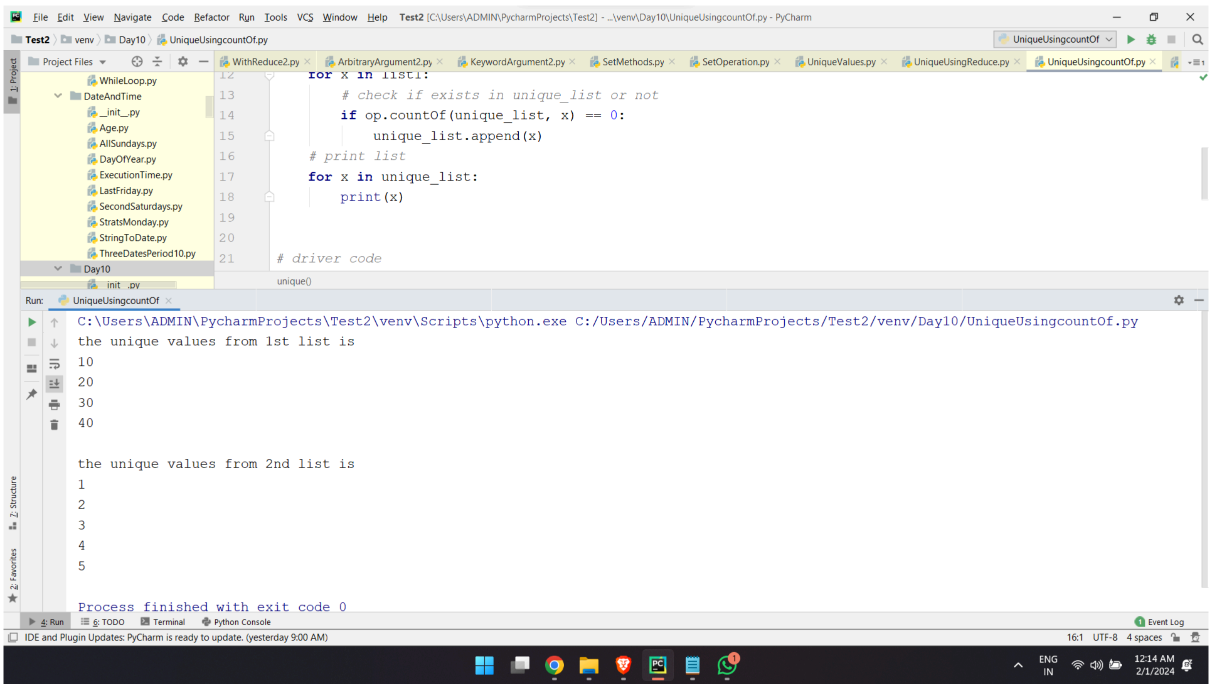
```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UniqueUsingReduce.py
the unique values from 1st list is
[10, 20, 30, 40]

the unique values from 2nd list is
[1, 2, 3, 4, 5]

Process finished with exit code 0
```

Getting Unique Values From a List in Python Using Operator.countOf() :

operator.countOf(a, b) returns the number of occurrences of b in the sequence a.

Usage:

It is part of the operator module in Python.
Commonly used when you want to count occurrences without manually iterating through the sequence.



```
    for x in list1:
        # check if exists in unique_list or not
        if op.countOf(unique_list, x) == 0:
            unique_list.append(x)
    # print list
    for x in unique_list:
        print(x)


# driver code
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UniqueUsingcountOf.py
the unique values from 1st list is
10
20
30
40

the unique values from 2nd list is
1
2
3
4
5

Process finished with exit code 0
```

IDE and Plugin Updates: PyCharm is ready to update. (yesterday 9:00 AM)

Get Unique Values From a List Using numpy.unique :

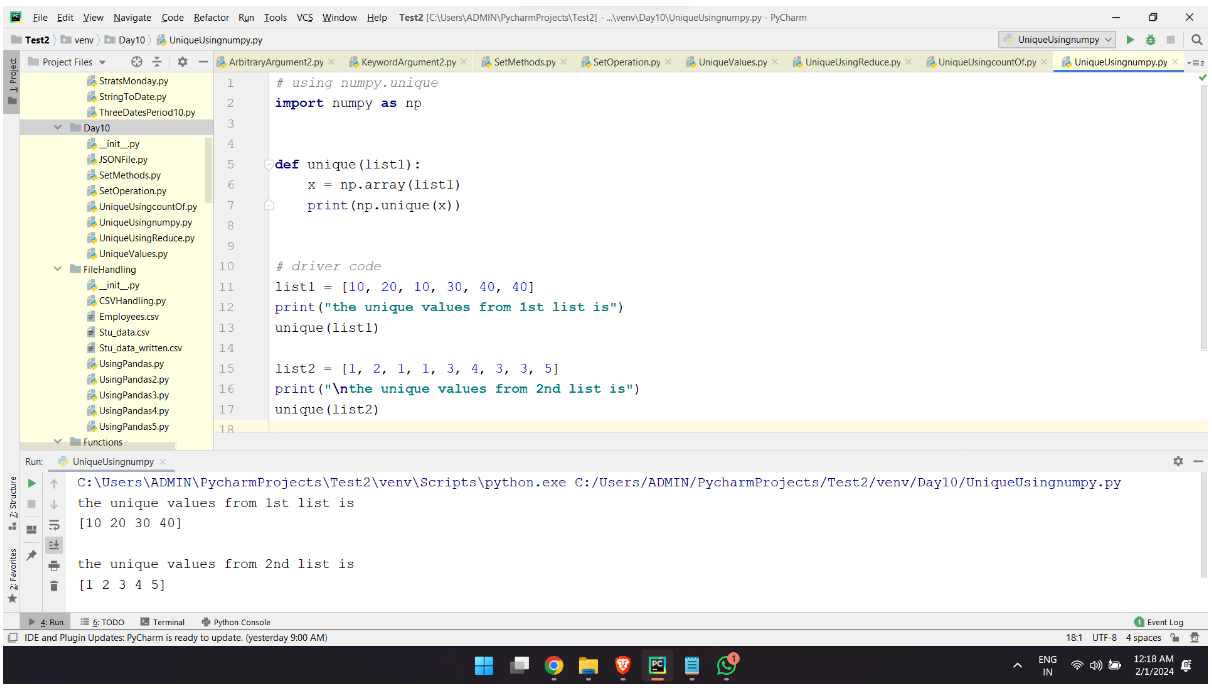
numpy.unique()

Definition:

numpy.unique(arr, return_index=False, return_inverse=False, return_counts=False, axis=None) returns the unique elements of an array or a list-like object.

Parameters:
arr: Input array or list.
return_index: If True, returns the indices of the first occurrences of unique values.
return_inverse: If True, returns the indices to reconstruct the original array from the unique array.
return_counts: If True, returns the number of times each unique value appears.
axis: The axis or axes along which unique values should be found.

Usage:
The primary purpose is to find unique elements in an array or list.



Test2 [C:\Users\ADMIN\PycharmProjects\Test2] - ...\venv\Day10\UniqueUsingnumpy.py - PyCharm
Test2 › venv › Day10 › UniqueUsingnumpy.py
UniqueUsingnumpy

```
# using numpy.unique
import numpy as np


def unique(list1):
    x = np.array(list1)
    print(np.unique(x))


# driver code
list1 = [10, 20, 10, 30, 40, 40]
print("the unique values from 1st list is")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique values from 2nd list is")
unique(list2)
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UniqueUsingnumpy.py
the unique values from 1st list is
[10 20 30 40]

the unique values from 2nd list is
[1 2 3 4 5]
```

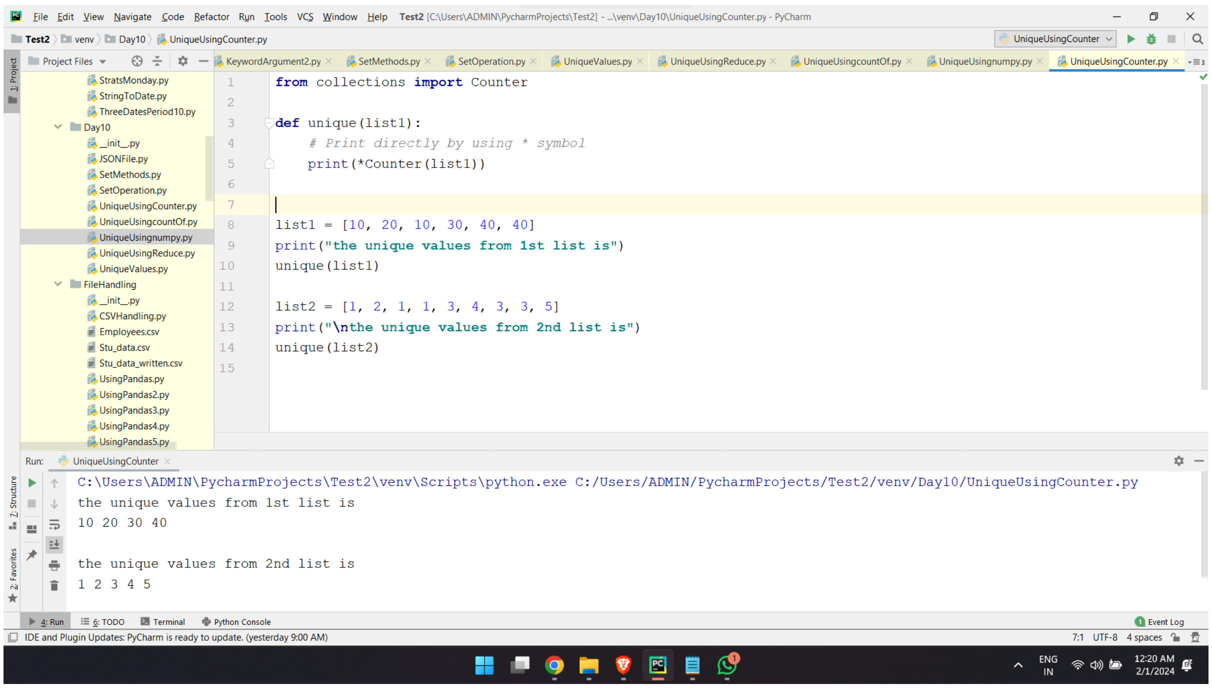
Get Unique Values From a List in Python Using collections.Counter() :

collections.Counter() Overview:

The collections.Counter class is a dictionary subclass in the collections module.
It is designed for counting hashable objects. It returns a dictionary where keys are unique elements, and values are their respective counts.

Using collections.Counter() for Unique Values:

You can utilize collections.Counter() to count occurrences of each element in the list.
Extract the keys from the resulting Counter dictionary to obtain unique values.



```python
from collections import Counter

def unique(list1):
    # Print directly by using * symbol
    print(*Counter(list1))

list1 = [10, 20, 10, 30, 40, 40]
print("the unique values from 1st list is")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique values from 2nd list is")
unique(list2)
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UniqueUsingCounter.py
the unique values from 1st list is
10 20 30 40

the unique values from 2nd list is
1 2 3 4 5
```

## Sorting Lists in Python :

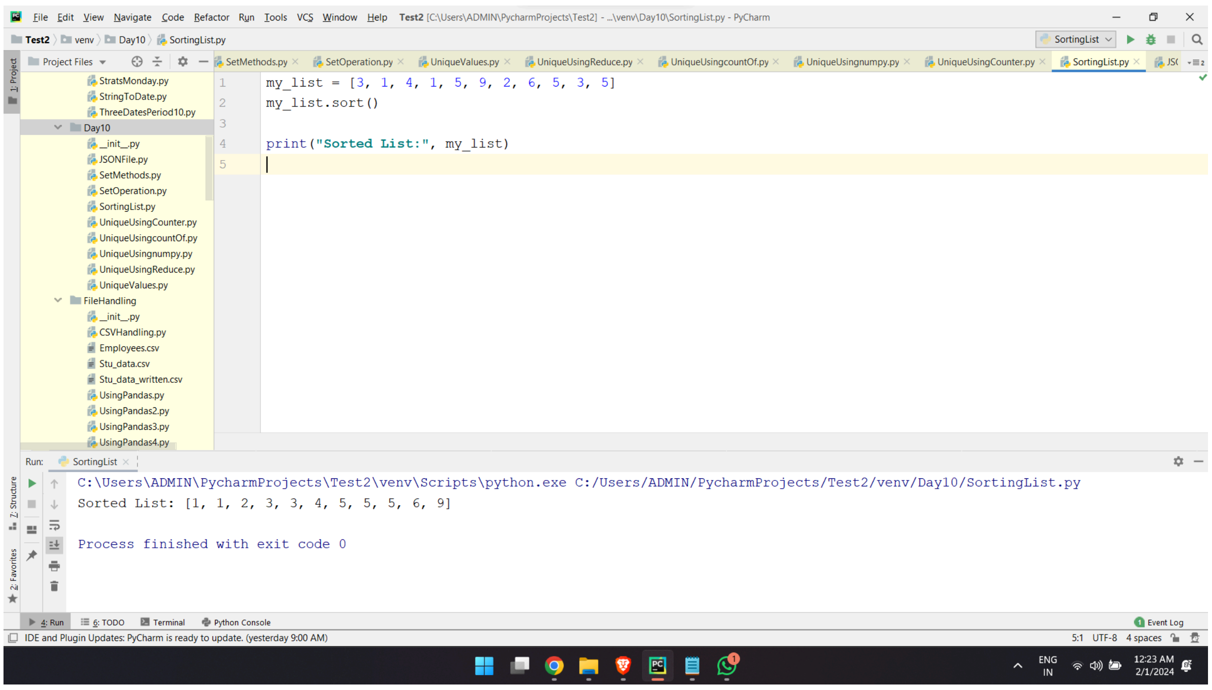
The list.sort() method in Python is used to modify a list in-place by sorting its elements. Here's an explanation of its key characteristics:

list.sort()
In-Place Sorting:

The list.sort() method sorts the elements of a list in ascending order by default. It modifies the original list in-place, meaning it does not create a new sorted list but rather rearranges the elements of the existing list.
Syntax:

The basic syntax of list.sort() is:

**list.sort(key=None, reverse=False)**



```python
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
my_list.sort()

print("Sorted List:", my_list)
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/SortingList.py
Sorted List: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

Process finished with exit code 0
```

**Introduction to JSON :**

JSON, or JavaScript Object Notation, is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is widely used as a data format for representing structured data and exchanging information between a server and a web application, as well as between different parts of an application.

Here are key aspects of the JSON data type:

Format and Structure:

JSON data is represented as key-value pairs, similar to Python dictionaries or JavaScript objects. It is a text format that uses human-readable syntax.
Data Types:

JSON supports several data types, including strings, numbers, objects, arrays, booleans, and null.
Syntax Rules:

JSON follows a strict syntax with well-defined rules.
Data is represented in name/value pairs.
Data is separated by commas.
Objects are enclosed in curly braces {}, and arrays are enclosed in square brackets [].

JSON example,

```
{
  "name": "John Doe",
  "age": 30,
  "city": "New York",
  "isStudent": false,
  "grades": [85, 90, 78],
  "address": {
    "street": "123 Main St",
    "zip": "10001"
  },
  "contact": null
}
```

json.load() :

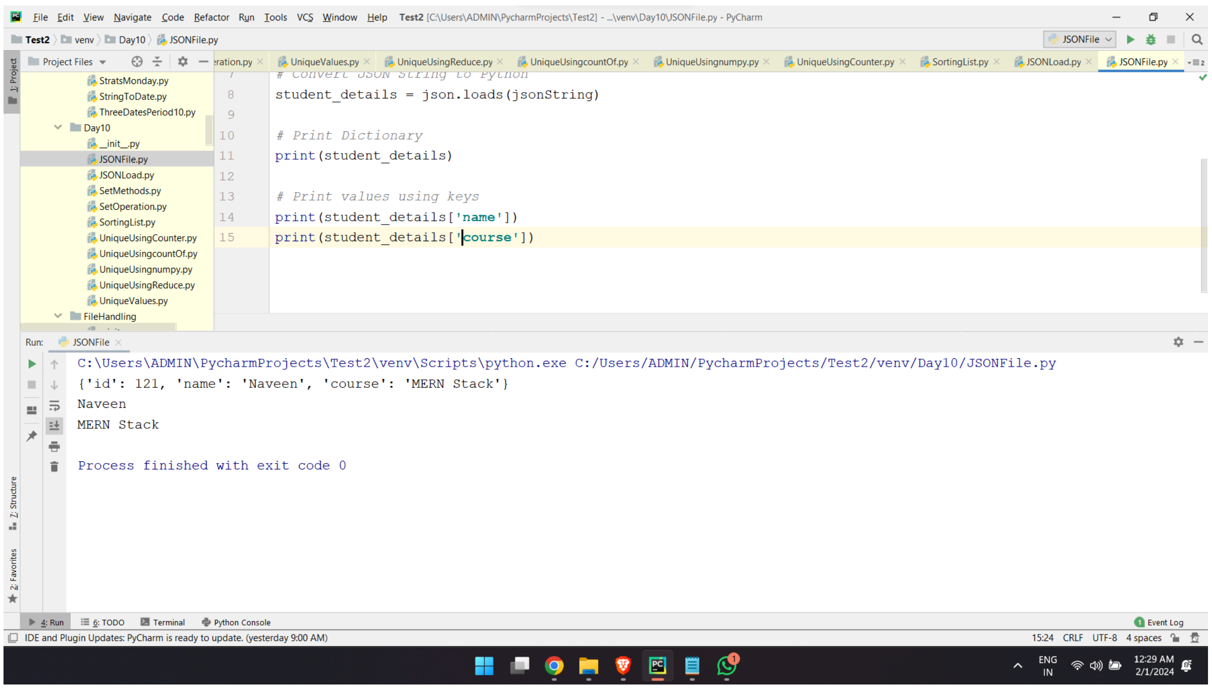
json.load() and json.loads() functions in Python are both used for parsing JSON data, but they are applied in slightly different contexts.

json.load()

Functionality:
json.load() is used to read a JSON file and parse its content.

Syntax:

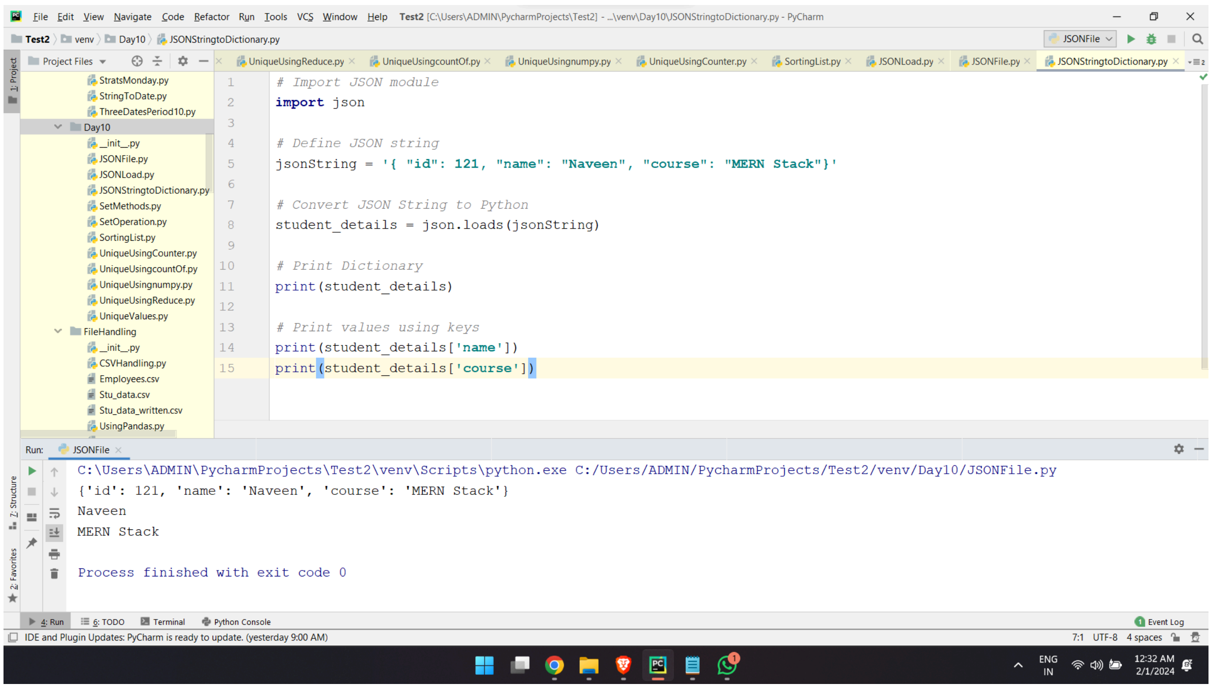
```
import json

with open('filename.json', 'r') as file:
    data = json.load(file)
```



```
# Convert JSON String to Python
student_details = json.loads(jsonString)

# Print Dictionary
print(student_details)

# Print values using keys
print(student_details['name'])
print(student_details['course'])
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/JSONFile.py
{'id': 121, 'name': 'Naveen', 'course': 'MERN Stack'}
Naveen
MERN Stack

Process finished with exit code 0
```

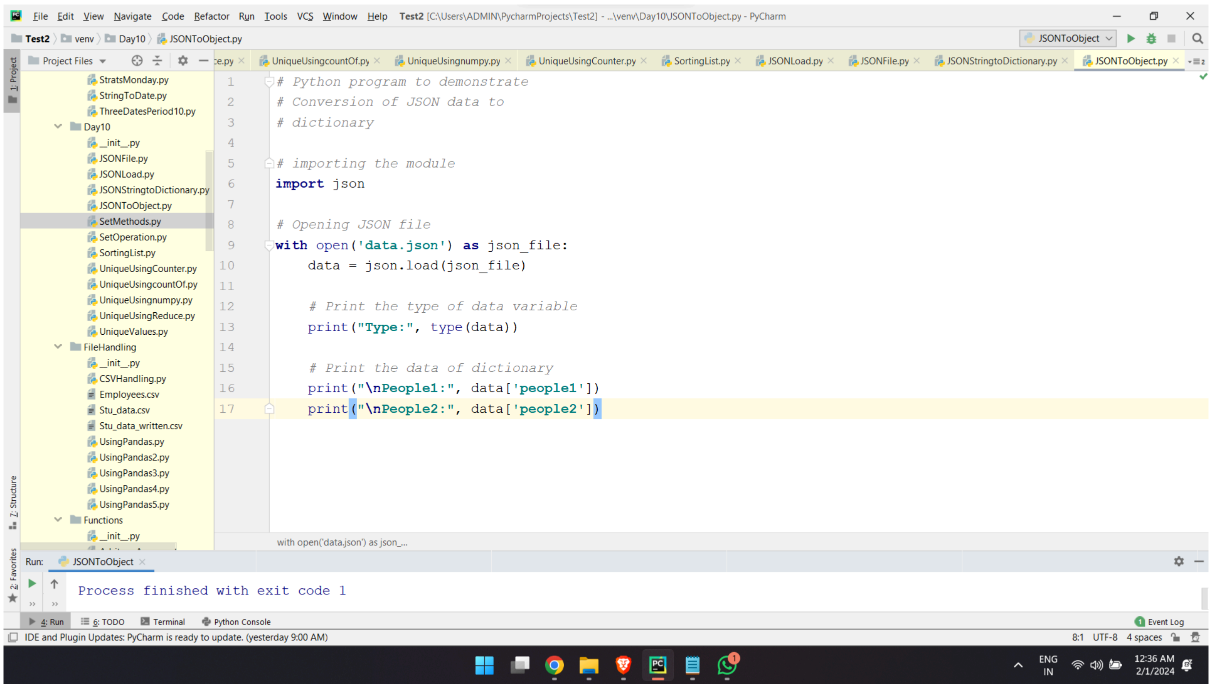
## Convert JSON String to Dictionary Python :



```python
# Import JSON module
import json

# Define JSON string
jsonString = '{ "id": 121, "name": "Naveen", "course": "MERN Stack"}'

# Convert JSON String to Python
student_details = json.loads(jsonString)

# Print Dictionary
print(student_details)

# Print values using keys
print(student_details['name'])
print(student_details['course'])
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/JSONFile.py
{'id': 121, 'name': 'Naveen', 'course': 'MERN Stack'}
Naveen
MERN Stack

Process finished with exit code 0
```

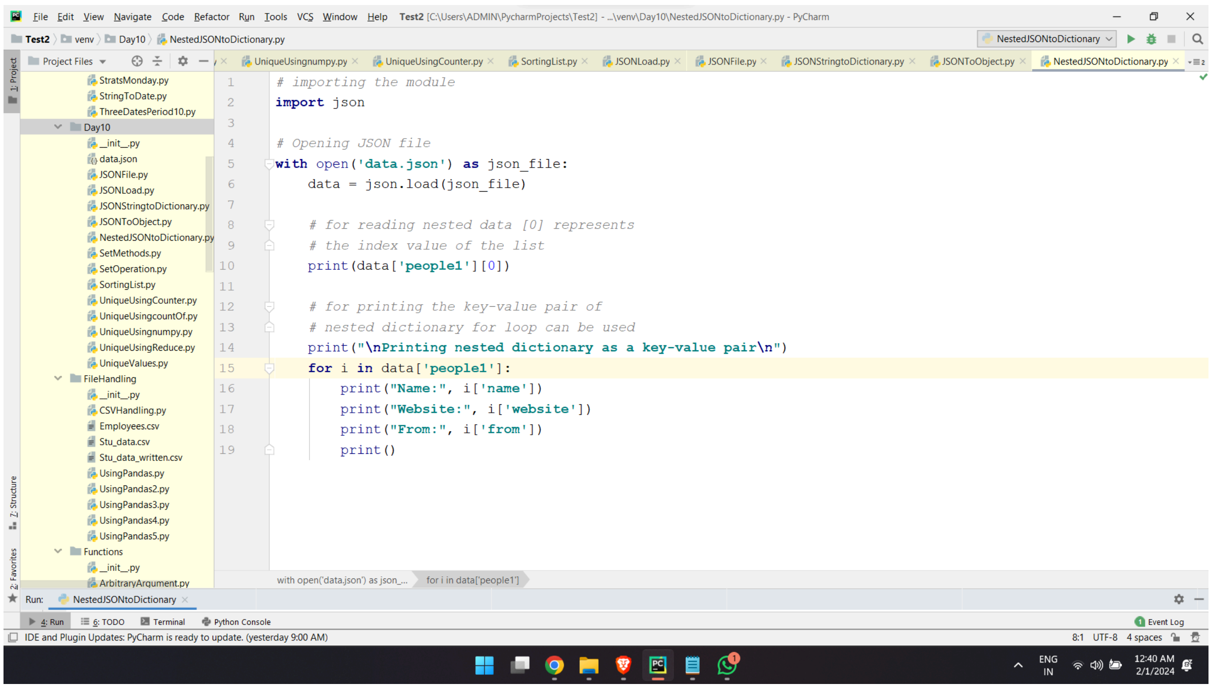
## Convert JSON File to Python Object :



```python
# Python program to demonstrate
# Conversion of JSON data to
# dictionary

# importing the module
import json

# Opening JSON file
with open('data.json') as json_file:
    data = json.load(json_file)

    # Print the type of data variable
    print("Type:", type(data))

    # Print the data of dictionary
    print("\nPeople1:", data['people1'])
    print("\nPeople2:", data['people2'])
```

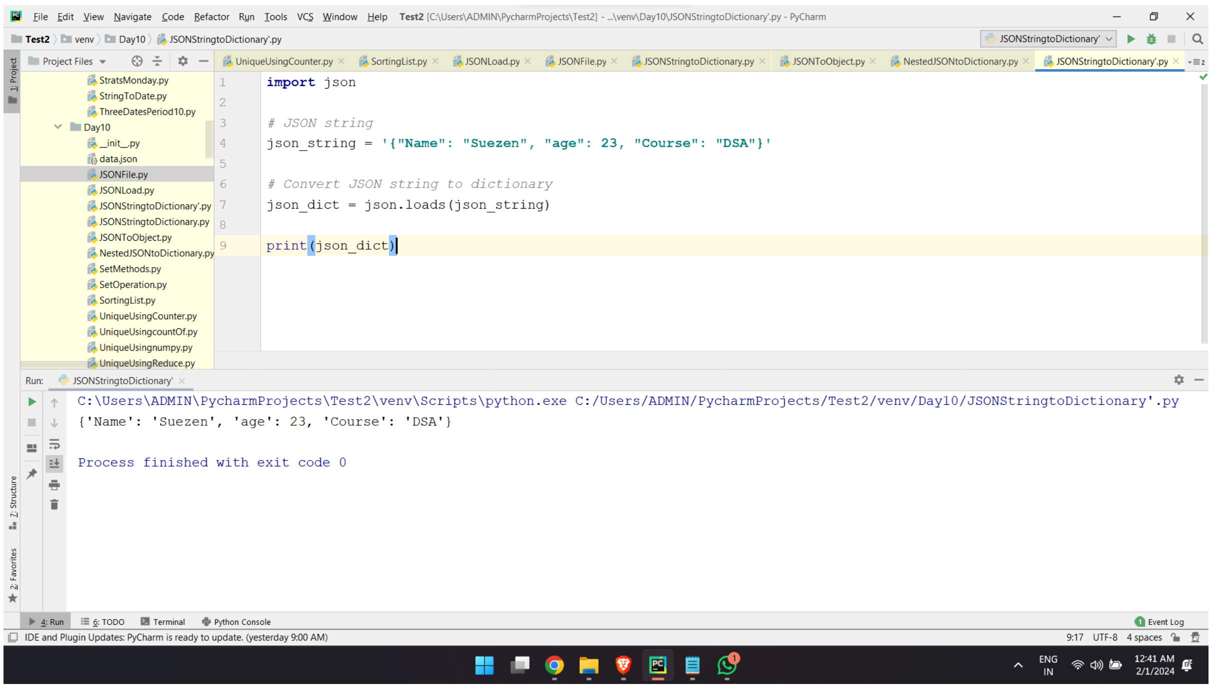
```
Process finished with exit code 1
```

## Convert Nested JSON Object to Dictionary :



```python
# importing the module
import json

# Opening JSON file
with open('data.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()
```

## Convert JSON String to Dictionary in Python :



```python
import json

# JSON string
json_string = '{"Name": "Suezen", "age": 23, "Course": "DSA"}'

# Convert JSON string to dictionary
json_dict = json.loads(json_string)

print(json_dict)
```

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/JSONStringtoDictionary'.py
{'Name': 'Suezen', 'age': 23, 'Course': 'DSA'}

Process finished with exit code 0
```