

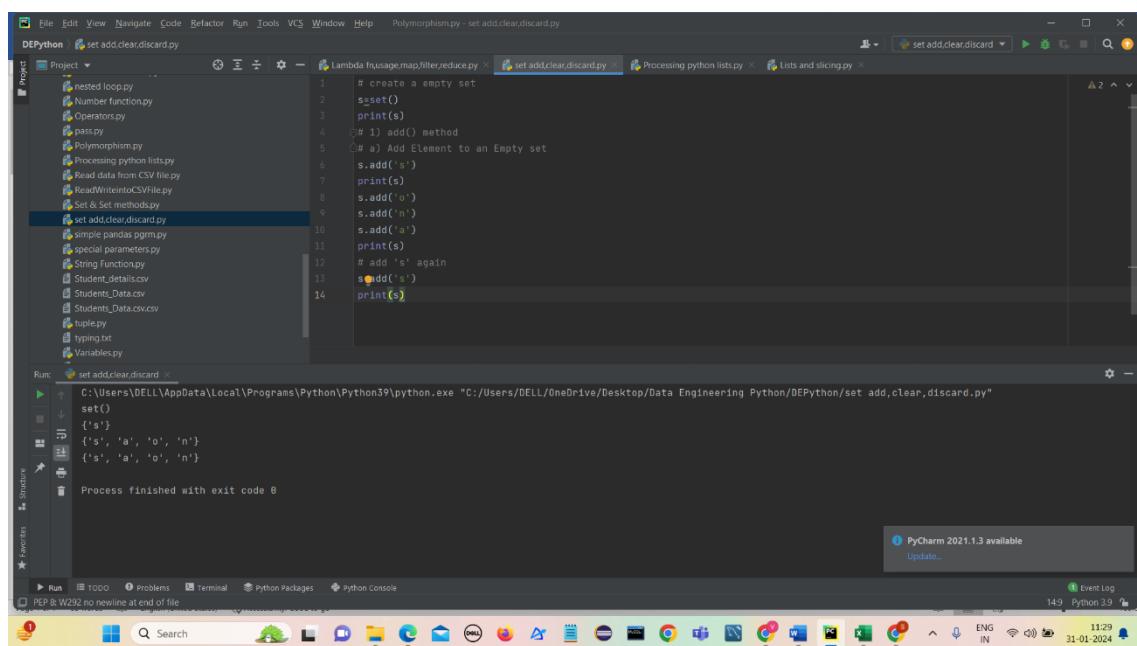
# DAY-9

## 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



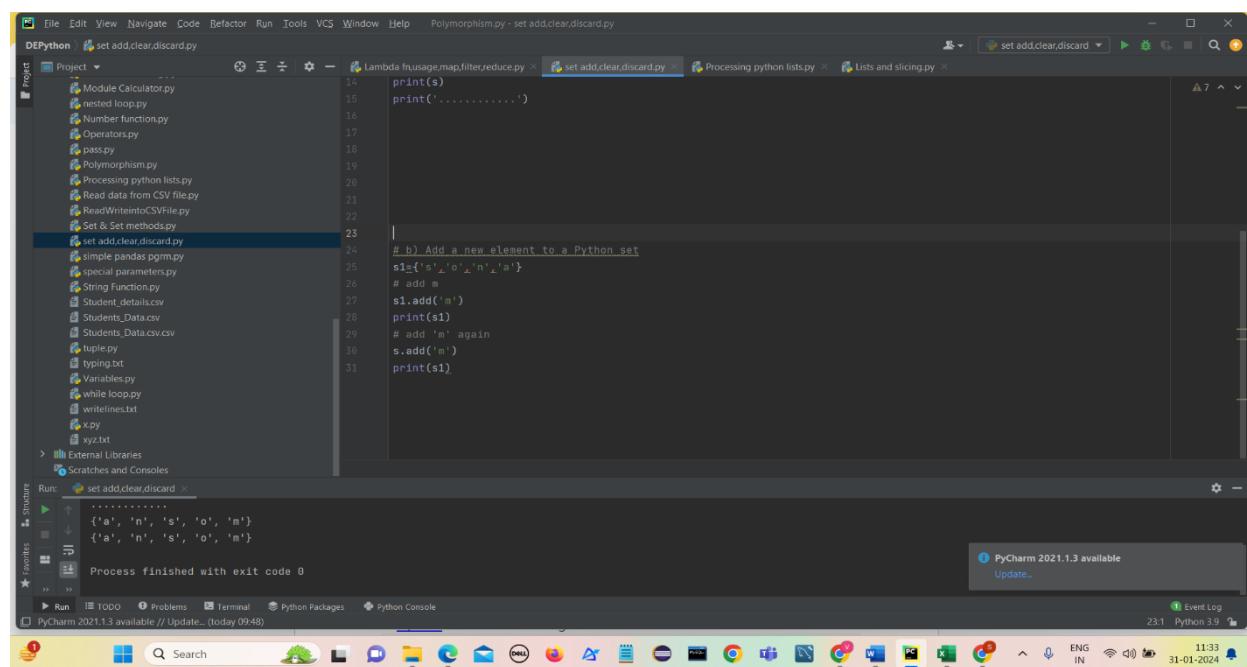
```
# create an empty set
s=set()
print(s)
# 1) add() method
# a) Add Element to an Empty set
s.add('s')
print(s)
s.add('o')
s.add('n')
print(s)
# add 's' again
s.add('s')
print(s)
```

Process finished with exit code 0

PyCharm 2021.1.3 available  
Update...

### 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](#).



```
# b) Add a new element to a Python set
s1={'e','o','n','a'}
# add m
s1.add('m')
print(s1)
# add 'm' again
s1.add('m')
print(s1)
```

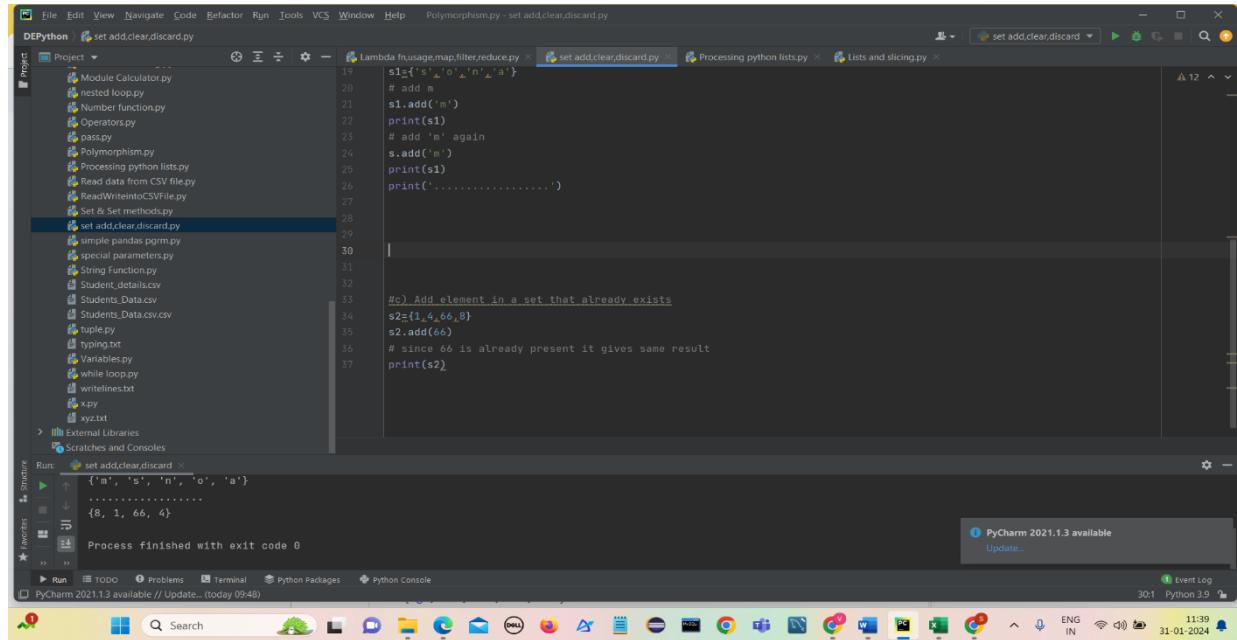
.....  
{'a', 'n', 's', 'o', 'm'}  
{'a', 'n', 's', 'o', 'm'}  
Process finished with exit code 0

PyCharm 2021.1.3 available // Update... (today 09:48)  
Event Log 23:1 Python 3.9

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.



```
#c) Add element in a set that already exists
s2={1,4,66,8}
s2.add(66)
# since 66 is already present it gives same result
print(s2)
```

Output:

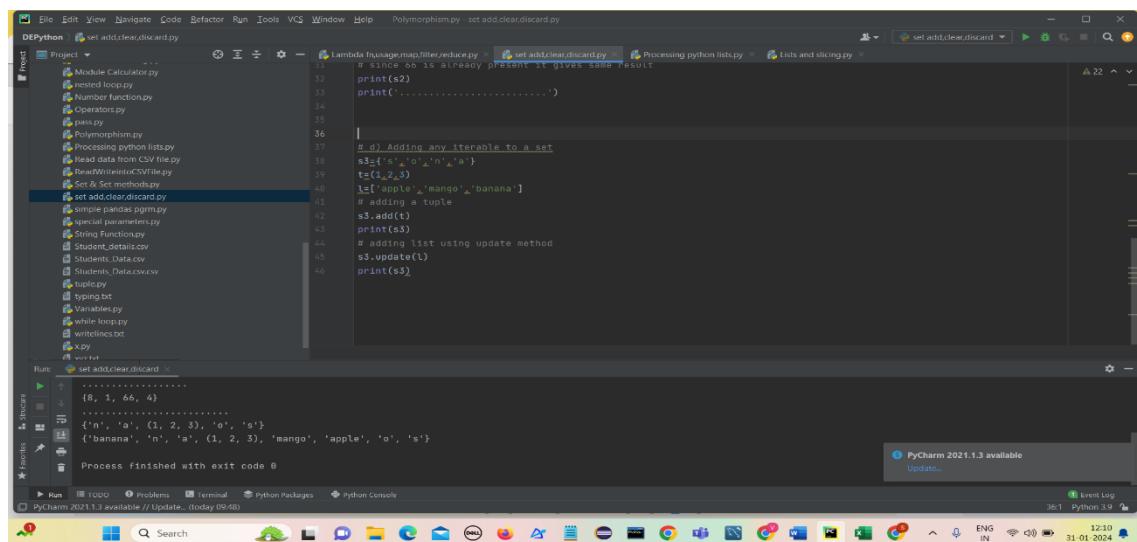
```
{8, 1, 66, 4}
```

Process finished with exit code 0

### 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



```
#_d) Adding any iterable to a set
s3={'s','o','n','a'}
t=(1,2,3)
l=['apple','mango','banana']
# adding a tuple
s3.add(t)
print(s3)
# adding list using update method
s3.update(l)
print(s3)
```

Output:

```
{'n', 'a', (1, 2, 3), 'o', 's'}
{'banana', 'n', 'a', (1, 2, 3), 'mango', 'apple', 'o', 's'}
```

Process finished with exit code 0

## 2) Set clear() method:

Set clear() method removes all elements from the set and it does not take any parameters.

**Syntax:** `set_name.clear()`

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists various Python files. In the center, the code editor displays a script named `set.add.clear.discard.py`. The code contains two sections: one for adding elements to a set using the `update()` method, and another for clearing the set using the `clear()` method. The run tab at the bottom shows the output of the script, which includes the original set and an empty set after clearing. A status bar at the bottom right indicates the Python version is 3.9.

```
# adding list using update method
s3.update(l)
print(s3)
print('.....')

#_2) clear
s={ 'sona' , 1 , 67 , 'Hello' , 78.90}
print(s)
s.clear()
print(s)
```

## 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.

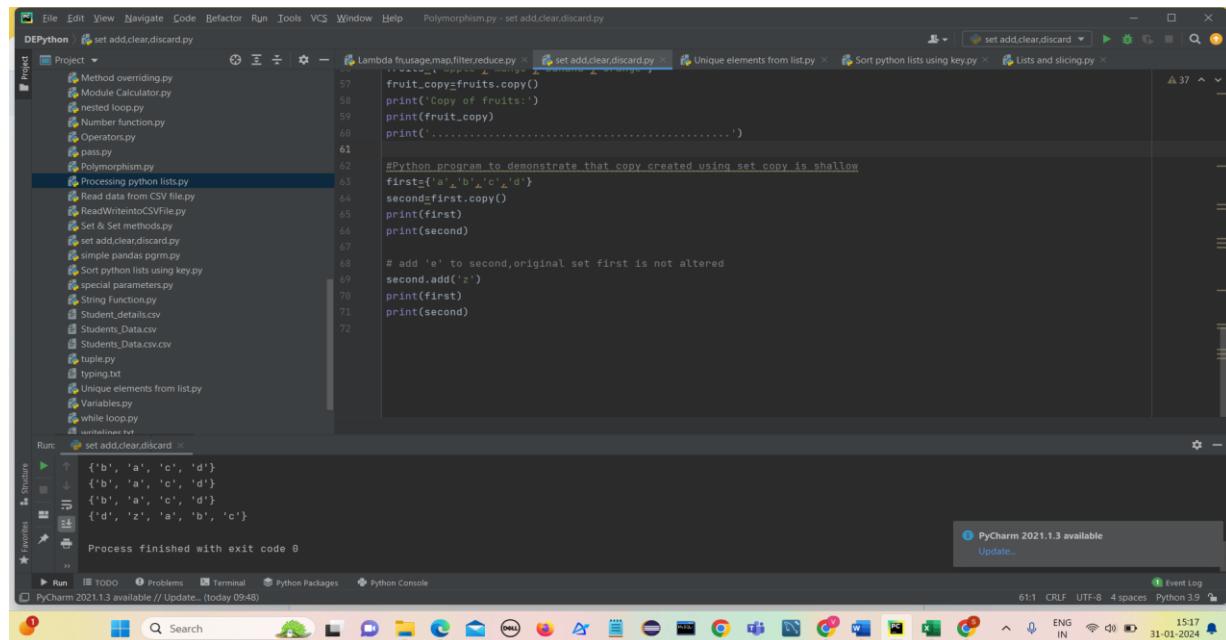
The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists various Python files. In the center, the code editor displays a script named `set.add.clear.discard.py`. The code demonstrates creating a copy of a set and modifying the copied set to show that changes do not affect the original. The run tab at the bottom shows the output of the script, which includes the original set and the modified copied set. A status bar at the bottom right indicates the Python version is 3.9.

```
se={'sona' , 1 , 67 , 'Hello' , 78.90}
print(s)
s.clear()
print(s)
print('.....')

#_3) copy()
#_it creates a shallow copy of set
fruits={'apple','mango','banana','orange'}
fruit=fruits.copy()
print(fruit)
print(fruits)
print(fruit.copy())
```

## Shallow Copy Example :

Python program to demonstrate copy created is shallow



```
fruits=fruits.copy()
print('Copy of fruits:')
print(fruit_copy)
print(.....)

#Python program to demonstrate that copy created using set copy is shallow
first=['a','b','c','d']
second=first.copy()
print(first)
print(second)

# add 'e' to second,original set first is not altered
second.add('z')
print(first)
print(second)
```

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.

The screenshot shows the PyCharm IDE interface. The main window displays a Python script named 'set.add.clear.discard.py'. The code demonstrates the use of the set module:

```

print('.....')

#_discard
first={'a', 'b', 'c', 'd'}
first.discard('c')
print(first)

# Discarding an element that is not present in set
first.discard('m')
print(first)

```

The 'Run' tab shows the output of the script:

```

{'d', 'c', 'b', 'z', 'a'}
.....
{'a', 'd', 'b'}
{'a', 'd', 'b'}

```

The status bar at the bottom right indicates 'PyCharm 2021.1.3 available // Update... (today 09:48)'.

## 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.

The screenshot shows the PyCharm IDE interface. The main window displays a Python script named 'Unique elements from list.py'. The code uses the set module to find unique elements in two lists:

```

# Unique elements from list
# 1) Using set method
def unique(list1):
    list_set=set(list1)
    unique_list=list(list_set)
    for i in unique.list:
        print(i,end=' ')

```

```

# list1
list1=[1,3,2,1,2,67,22,44,33,33,22]
print('The unique elements in list1')
unique(list1)

# list2
list2=['a','b','c','d','b','c','e']
print('\nThe unique elements in list2')
unique(list2)

```

The 'Run' tab shows the output of the script:

```

The unique elements in list1
1 2 67 3 33 44 22
The unique elements in list2
c b a d e

```

The status bar at the bottom right indicates 'PyCharm 2021.1.3 available // Update... (today 09:48)'.

## b) Using reduce() function

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists various Python files. The main code editor window contains the following Python code:

```
# 2) Using reduce() function
from functools import reduce
def unique(list1):
    res=reduce(lambda re,x:re+[x] if x not in re else re,list1[])
    print(res)

# list1
list1=[12,54,8,12,9,3,8,9,3,8,10,1,3,2,2]
print("Unique elements in list1:")
unique(list1)
# list2
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique values from 2nd list is")
unique(list2)
```

The run tab below shows the output of the code execution:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Unique elements from list.py"
Unique elements in list1:
[12, 54, 8, 9, 3, 10, 1, 2]

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

Process finished with exit code 0
```

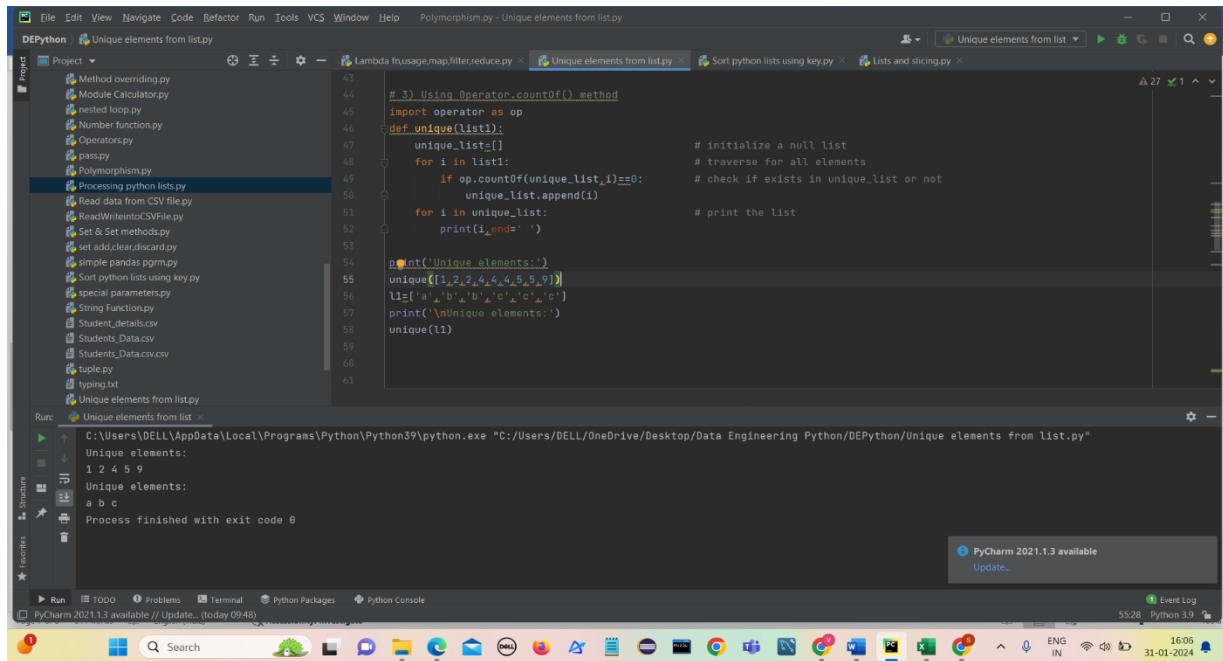
A status bar at the bottom right indicates the system is connected to the internet and shows the date and time.

- `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.



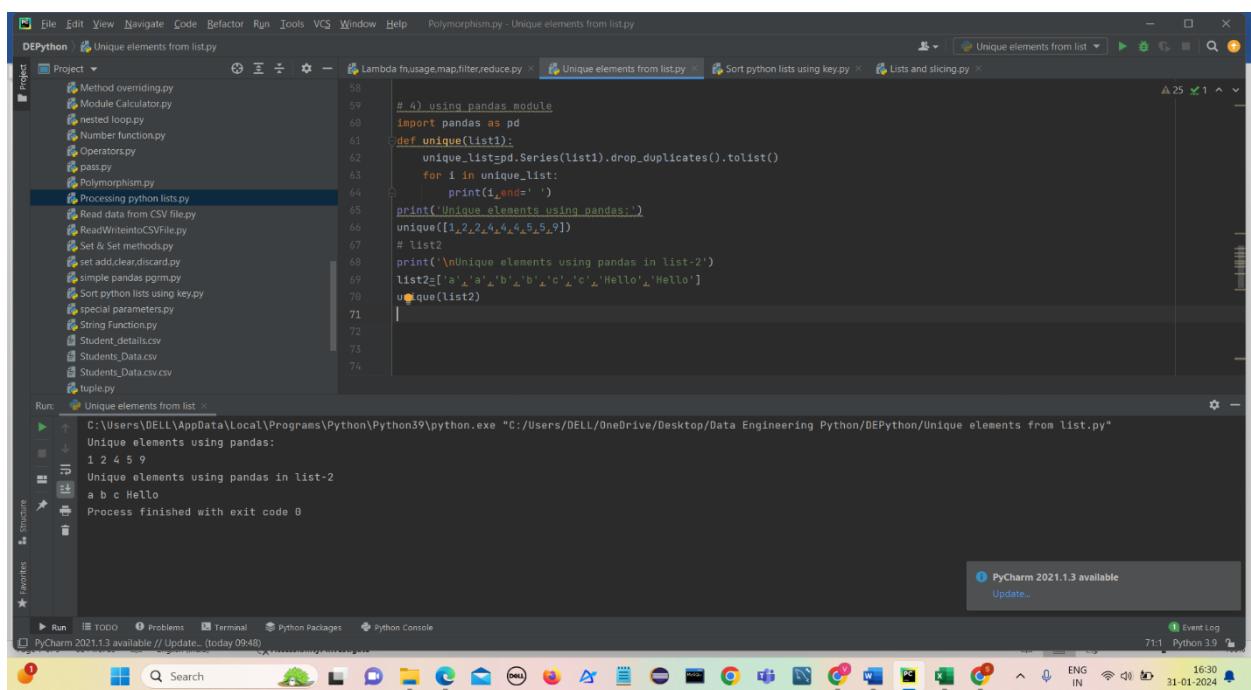
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help Polymorphism.py - Unique elements from list.py
DEPython Unique elements from list.py
Project Lambda fusage,map/filter/reduce.py Unique elements from list.py Sort python lists using key.py Lists and slicing.py
43
44 # 3) Using Operator.countOf() method
45 import operator as op
46 def unique(list1):
47     unique_list=[]
48     for i in list1:
49         if op.countOf(unique_list,i)==0:
50             unique_list.append(i)
51     for i in unique_list:
52         print(i,end=' ')
53
54 print('Unique elements:')
55 unique([1,2,2,4,4,4,5,5,9])
56 li=[ 'a' , 'b' , 'b' , 'c' , 'c' , 'c' ]
57 print( '\nUnique elements:' )
58 unique(li)
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
Run: Unique elements from list x
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Unique elements from list.py"
Unique elements:
1 2 4 5 9
Unique elements:
a b c
Process finished with exit code 0
PyCharm 2021.1.3 available
Update...
Event Log
55:28 Python 3.9
16:06 31-01-2024

```

## 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.



```

File Edit View Navigate Code Refactor Run Tools VCS Window Help Polymorphism.py - Unique elements from list.py
DEPython Unique elements from list.py Sort python lists using key.py Lists and slicing.py
Project Lambda fusage,map/filter/reduce.py Unique elements from list.py Sort python lists using key.py Lists and slicing.py
58
59 # 4) using pandas module
60 import pandas as pd
61 def unique(list1):
62     unique_list=pd.Series(list1).drop_duplicates().tolist()
63     for i in unique_list:
64         print(i,end=' ')
65 print('Unique elements using pandas:')
66 unique([1,2,2,4,4,4,5,5,9])
67 # list2
68 print('\nUnique elements using pandas in list-2')
69 list2=[ 'a' , 'a' , 'b' , 'b' , 'c' , 'c' , 'Hello' , 'Hello' ]
70 unique(list2)
71
72
73
74
Run: Unique elements from list x
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Unique elements from list.py"
Unique elements using pandas:
1 2 4 5 9
Unique elements using pandas in list-2
a b c Hello
Process finished with exit code 0
PyCharm 2021.1.3 available
Update...
Event Log
71:1 Python 3.9
16:30 31-01-2024

```

## e) Using collections.Counter()

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

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several Python files. The main editor window displays a Python script named 'Unique elements from list.py'. The code uses the `Counter` class from the `collections` module to count occurrences of each element in a list. It then prints the unique elements of both the original list and a modified list. The run tab at the bottom shows the output of the script, which lists the unique elements of the first list as 12, 54, 8, 9, 3, 10, 1, 2, and 2. The status bar at the bottom right indicates the current date and time as 31-01-2024 16:34.

```
# 5) Using collections.Counter()
from collections import Counter
def unique(list1):
    print(*Counter(list1))

list1=[12, 54, 8, 12, 9, 3, 8, 9, 3, 8, 10, 1, 3, 2, 2]
print('Unique elements in list1:')
unique(list1)
# list2
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique elements in list2:")
unique(list2)
```

## 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.

The screenshot shows the PyCharm IDE interface. The project name is 'DEPython'. In the 'Run' tab, the command 'C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Unique elements from list.py"' is run, resulting in the output '[10, 20, 30, 40, 70]'. The status bar at the bottom right shows '90:1 Python 3.9'.

```
# 6) Using dict.fromkeys()
list1=[10,10,20,30,40,70,30,40]
unique_list1=list(dict.fromkeys(list1))
print(unique_list1)
```

## g) Using map and set() method

The screenshot shows the PyCharm IDE interface. The project name is 'DEPython'. In the 'Run' tab, the command 'C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Unique elements from list.py"' is run, resulting in the output '[1, 2, 3, 4, 5]'. The status bar at the bottom right shows '97:1 Python 3.9'.

```
#7) Using map and set() method
my_list = [1, 2, 2, 3, 3, 4, 4, 5]
unique_elements = set(map(lambda x: x, my_list))
print(list(unique_elements))
```

# 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

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and the current file name: Polymorphism.py - Sort python lists using key.py. The Project tool window on the left lists various Python files and CSV files. The main code editor contains three examples of sorting lists:

```
# Sort python lists
# a) using sorted() function
l1=[89,23,11,1,-2,-1000]
print(l1)
# using sorted function
print('The sorted list is :')
print(sorted(l1))
print('.....')

# b) sorted() function with reverse=True
print('The sorted list in descending order :')
print(sorted(l1,reverse=True))
print('.....')

# c) sorted() function on strings
l2=['aa', 'BB', 'zz', 'CC']
print('Sorting strings :')
print(sorted(l2))
```

The Run tool window at the bottom shows the output of running the script:

```
C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Sort python lists using key.py"
[89, 23, 11, 1, -2, -1000]
The sorted list is:
[-1000, -2, 1, 11, 23, 89]
.....
The sorted list in descending order:
[89, 23, 11, 1, -2, -1000]
.....
Sorting strings:
['BB', 'CC', 'aa', 'zz']
```

The status bar at the bottom right indicates PyCharm 2021.1.3 available // Update... and the system clock shows 12:31 Python 3.9.

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.

```

# Custom sorting with key
l3=['aaaa','bbbbbbb','ccccccc','ddd','ee','f']
print("Sorting with len key")
print(sorted(l3,key=len))

# with reverse=True
print('Sorting with len key in descending order')
print(sorted(l3,key=len,reverse=True))

```

Process finished with exit code 0

## b) key=str.lower()

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

```

# 2)key=str.lower()
l4=['y','A','x','B','a','b','Y','X']
print('Sorting with str.lower() key')
print(sorted(l4,key=str.lower()))

# 3) Creating our own key

```

Process finished with exit code 0

## c) Creating our own key

### Sorting based on last letter of string

The screenshot shows the PyCharm IDE interface with a project named "DEPython". The code editor displays a Python script titled "Sort python lists using key.py". The code defines a function `myFunc` that sorts a list of strings based on their last character. The run output shows the sorted list: ['bd', 'Welcome', 'owl', 'Hello', 'ax']. A notification at the bottom right indicates "PyCharm 2021.1.3 available // Update...".

```
# 3) Creating our own key
# sorting based on last letter of string
def myFunc(s):
    return s[-1]

str=['ax','bd','Hello','Welcome','owl']
print('Creating own key')
print(sorted(str,key=myFunc))

# 4) itemgetter()
```

## itemgetter()

(firstname , lastname, score) tuples

The screenshot shows the PyCharm IDE interface with a project named "DEPython". The code editor displays a Python script titled "Sort python lists using key.py". The code uses the `itemgetter` function from the `operator` module to sort a list of tuples based on different criteria. The run output shows two sorted lists: one using `itemgetter(1, 0)` and another using `itemgetter(0, -1)`. A notification at the bottom right indicates "PyCharm 2021.1.3 available // Update...".

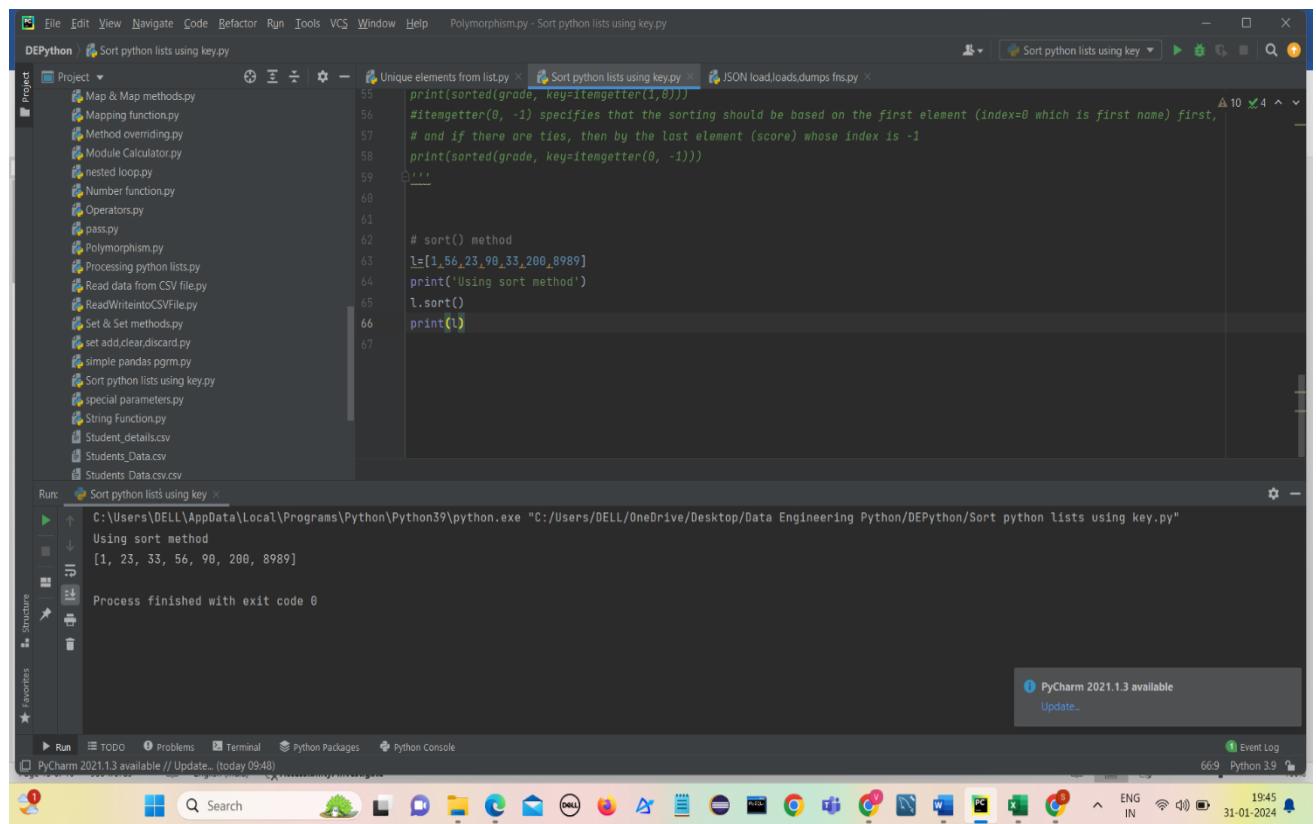
```
# (first name, last name, score) tuples
#itemgetter(1, 0) specifies that the sorting should be based on the second element(index=1 which is last name) first,
#and if there are ties, then by the first element(index=0 which is first name).
from operator import itemgetter
grade = [('Freddy', 'Frank', 3), ('Anil', 'Frank', 100), ('Anil', 'Wang', 24)]
print('Sorted using item getter')
print(sorted(grade, key=itemgetter(1,0)))
print(sorted(grade, key=itemgetter(0, -1)))
#itemgetter(0, -1) specifies that the sorting should be based on the first element (index=0 which is first name) first,
#and if there are ties, then by the last element (score) whose index is -1
print(sorted(grade, key=itemgetter(0, -1)))
```

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 the first 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.



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

# sort() method
l=[1,56,23,90,33,200,8989]
print('Using sort method')
l.sort()
print(l)
```

C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Sort python lists using key.py"  
Using sort method  
[1, 23, 33, 56, 90, 200, 8989]  
Process finished with exit code 0

PyCharm 2021.1.3 available  
Update...

# JSON

- **JSON** stands for **JavaScript Object Notation**. 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 structure the data compared 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.

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - JSON load.loads.dumps fns.py. The main editor window contains the following Python code:

```
# Convert JSON String to Dictionary Python
#json.loads(); to convert JSON string to a dictionary

# import json module
import json

# Define json string
jsonString='{"Name":"Sriganga", "age":22, "Course":"Data Engineering"}'

# Convert JSON string into Python
student_details=json.loads(jsonString)

#Print Dictionary
print(student_details)

# To find type
print(type(student_details))

# Print values using keys
print(student_details['Name'])
print(student_details['age'])
print(student_details['Course'])
```

The Run tab at the bottom shows the output of the script:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/JSON load.loads.dumps fns.py"
{'Name': 'Sriganga', 'age': 22, 'Course': 'Data Engineering'}
<class 'dict'>
Sriganga
22
Data Engineering

Process finished with exit code 0
```

The status bar at the bottom right indicates PyCharm 2021.1.3 available // Update... (today 09:48), 11:23 CRLF UTF-8 4 spaces Python 3.9, and 17:51 31-01-2024.

## Example-2

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - JSON load.loads.dumps fns.py. The main editor window contains the following Python code:

```
# Example-2
data='''{
    "Name": "Sriganga",
    "Contact": [6767676767, 9090990000],
    "Email": "sri@gmail.com",
    "Hobbies": ["Reading", "Listening Music"]
}'''

res=json.loads(data)
print(res)
print(res['Contact'])
```

The Run tab at the bottom shows the output of the script:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/JSON load.loads.dumps fns.py"
{'Name': 'Sriganga', 'Contact': [6767676767, 9090990000], 'Email': 'sri@gmail.com', 'Hobbies': ['Reading', 'Listening Music']}
[6767676767, 9090990000]

Process finished with exit code 0
```

The status bar at the bottom right indicates PEP 8: W292 no newline at end of file, 31-22 Python 3.9, and 17:51 31-01-2024.

# 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.

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several files, including abc.txt, Access Specifiers.py, app.txt, Arbitrary args.py, Book1.xlsx, breakpy, Calculator.py, Class and obj.py, Constructor.py, continue.py, data.json, Datatypes.py, Date and time function.py, Default arguments.py, Dictionary and dict methods.py, Emp\_data.csv, Example-1.csv, Example-2.csv, Exception.py, extra.py, Family\_Data.csv, fileHandling.py, fileO.py, First program.py, for loop.py, and functions.py. Below the navigation bar, the main code editor window contains the following Python code:

```
27     }
28     res=json.loads(data)
29     print(res)
30     print(res[ 'Contact'])
31     print('.....')
32
33
34
35
36
37
38 #Convert JSON File to Python Object
39 with open('data.json') as json_file:
40     data=json.load(json_file)
41     print('Type:',type(data))
42     print(data)
43     print('Hobbies are:',data['Hobbies'])
```

The Run tab at the bottom shows the output of the code execution:

```
Type: <class 'dict'>
{'Name': 'Sriganga', 'Contact': [6767676767, 9090990000], 'Email': 'sri@gmail.com', 'Hobbies': ['Reading', 'Listening Music']}
Hobbies are: ['Reading', 'Listening Music']

Process finished with exit code 0
```

The status bar at the bottom right indicates the system date and time as 31-01-2024 20:18.

## Example-2

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several files, including abc.txt, Access Specifiers.py, app.txt, Arbitrary args.py, Book1.xlsx, breakpy, Calculator.py, Class and obj.py, Constructor.py, continue.py, data.json, Datatypes.py, Date and time function.py, Default arguments.py, Dictionary and dict methods.py, Emp\_data.csv, and Example-2.csv. Below the navigation bar, the main code editor window contains the following Python code:

```
39
40
41
42
43
44
45
46
47
48 # Example-2
49 with open('data1.json') as json_file:
50     data=json.load(json_file)
51     print('Type:',type(data))
52     print(data)
53     print('Employees are:',data['employees'])
```

The Run tab at the bottom shows the output of the code execution:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering/DEPython/JSON load.loads.dumps fns.py"
Type: <class 'dict'>
{'employees': [{'name': 'Shyam', 'email': 'shyamjaiswal@gmail.com'}, {'name': 'Bob', 'email': 'bob32@gmail.com'}, {'name': 'Jai', 'email': 'jai87@gmail.com'}]}
Employees are: [{'name': 'Shyam', 'email': 'shyamjaiswal@gmail.com'}, {'name': 'Bob', 'email': 'bob32@gmail.com'}, {'name': 'Jai', 'email': 'jai87@gmail.com'}]

Process finished with exit code 0
```

The status bar at the bottom right indicates the system date and time as 31-01-2024 20:21.

## Example-3

The screenshot shows the PyCharm IDE interface. The code editor displays a script named 'JSON load.loads.dumps fns.py' with the following content:

```
# Example-3
import json
with open('x.json') as json_file:
    data1 = json.load(json_file)
    print("Type:", type(data1))
    print("\nPeople1:", data1['people1'])
    print("\nPeople2:", data1['people2'])
    print('.....')

with open('x.json') as json_file
```

The Run tab shows the output of the script:

```
Type: <class 'dict'>
People1: [{'name': 'Nikhil', 'website': 'gfg.com', 'from': 'Delhi'}, {'name': 'Abhinav', 'website': 'google.com', 'from': 'Mumbai'}]
People2: [{"name": "Anshu", "website": "apple.com", "from": "Chennai"}]
.....
```

Process finished with exit code 0

## Convert Nested JSON Object to Dictionary

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

The screenshot shows the PyCharm IDE interface. The code editor displays a script named 'JSON load.loads.dumps fns.py' with the following content:

```
#Convert Nested JSON Object to Dictionary
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'])
    print("Author Details:")
    print("lastname:", data1['author']['lastname'])
    print("firstname:", data1['author']['firstname'])
    print("Editor Details:")
    print("lastname:", data1['editor']['lastname'])
    print("firstname:", data1['editor']['firstname'])
    print("title:", data1['title'])
    print("category:", data1['category'])

with open('nested_json.json') as json_file
```

The Run tab shows the output of the script:

```
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
```

The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `Polymorphism.py` which contains code to convert a nested JSON object into a dictionary. The output of the script is shown in the terminal window below, detailing the structure of a book entry with author and editor details.

```
# Convert Nested JSON Object to Dictionary
# Example-1
with open('nested_json.json') as json_file:
    data = json.load(json_file)
print(data)

# for reading nested data- [0] represents the index value of the list
print(data['isbn'][0])
print(data['isbn'][1:8])
print(data['editor']['lastname'])
print("ISBN:", data['isbn'])

Process finished with exit code 0
```

Output:

```
{'isbn': '123-456-222', 'author': {''lastname': 'Doe', 'firstname': 'Jane'}, 'editor': {''lastname': 'Smith', 'firstname': 'Jane'}, 'title': 'The Ultimate Database Study Guide', 'category': ['Non-fiction', 'Technology']}
```

## Example-2

The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `Polymorphism.py` which prints a nested dictionary as key-value pairs. The output of the script is shown in the terminal window below, displaying two entries: one for Nikhil and one for Abhinav.

```
# 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['people'][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['people']:
    print("Name:", i['name'])
    print("Website:", i['website'])
    print("From:", i['from'])
    print()

with open('x.json') as json_file
```

Output:

```
{'name': 'Nikhil', 'website': 'gfg.com', 'from': 'Delhi'}

Printing nested dictionary as a key-value pair

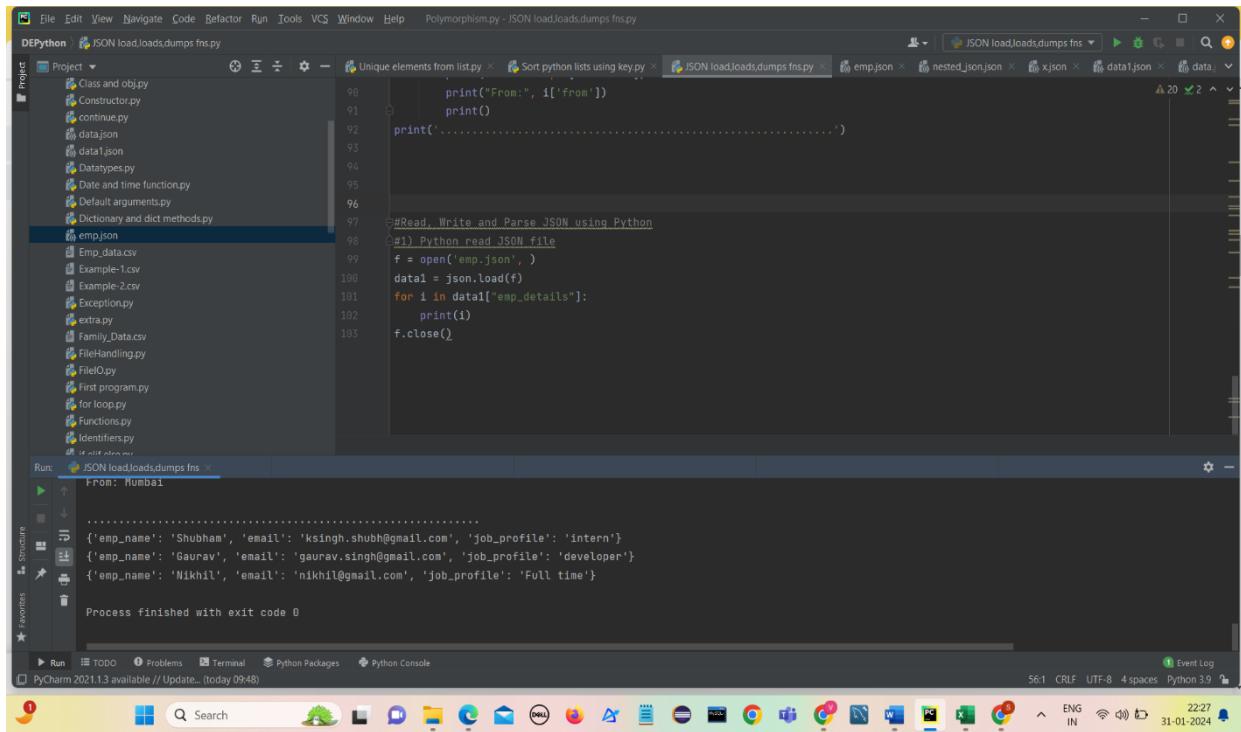
Name: Nikhil
Website: gfg.com
From: Delhi

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.



The screenshot shows the PyCharm IDE interface. The top bar has tabs for 'File', 'Edit', 'View', 'Navigate', 'Code', 'Refactor', 'Run', 'Tools', 'VCS', 'Window', and 'Help'. The current tab is 'Polymorphism.py - JSON load.loads.dumps fn.py'. The left sidebar shows a 'Project' tree with files like 'Class and obj.py', 'Constructor.py', 'continue.py', 'datajson', 'data1.json', 'Datatypes.py', 'Date and time function.py', 'Default arguments.py', 'Dictionary and dict methods.py', 'emp.json', 'Emp\_data.csv', 'Example-1.csv', 'Example-2.csv', 'Exceptions.py', 'extra.py', 'Family\_Data.csv', 'FileHandling.py', 'FileO.py', 'First program.py', 'for loop.py', 'Functions.py', 'Identifiers.py', and 'if and else.py'. The main code editor window contains the following Python code:

```
# Read, Write and Parse JSON using Python
#1) Python read JSON file
f = open('emp.json', )
data1 = json.load(f)
for i in data1['emp_details']:
    print(i)
f.close()

print("From: ", i['from'])
print()
print('.....')
```

The 'Run' tab at the bottom shows the output: 'From: Mumbai'. Below the code editor, the terminal shows the command 'python Polymorphism.py' and the output: 'Process finished with exit code 0'. The status bar at the bottom right shows 'PyCharm 2021.1.3 available // Update... (today 09:48)' and the date '31-01-2024'.

## 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.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - JSON load.loads.dumps fn.s. The main window displays a code editor with a script named JSON load.loads.dumps fn.s. The code prints a dotted separator and then converts a Python dictionary to a JSON string with an indent of 4 spaces. A run configuration at the bottom shows the output: "Dictionary is:" followed by a JSON object. The bottom status bar indicates PyCharm 2021.1.3 available // Update... (today 09:48), 106:1 CRLF, UTF-8, 4 spaces, Python 3.9, and the date/time 31-01-2024.

```
print('.....')
#2) Convert Python Dict to JSON
dictionary = {
    "id": "04",
    "name": "sunil",
    "department": "HR"
}
json_obj=json.dumps(dictionary,indent=4)
print("Dictionary is:")
print(json_obj)
```

### 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.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - JSON load.loads.dumps fns.py. The left sidebar shows a project structure with files like Module Calculator.py, nested\_loop.py, nested\_json.json, Number function.py, Operators.py, pass.py, Polymorphism.py, Processing python lists.py, Read data from CSV file.py, ReadWriteintoCSVfile.py, Set & Set methods.py, set.add.clear.discard.py, simple pandas pgm.py, Sort python lists using key.py, special parameters.py, String Function.py, Student\_details.csv, Students\_Data.csv, Students\_Data.csv.csv, teacher.json, tuple.py, and typing.txt. The main code editor window contains Python code for writing JSON data to a file:

```
#3) Writing JSON to a file in Python
dictionary = {
    "t_name": "Dhana Laxmi",
    "t_id": 200,
    "t_age": 43,
    "t_salary": 45000
}
with open("teacher.json", "w") as outfile:
    json.dump(dictionary, outfile)
print("Teacher data dumped")
```

The Run tab shows the output of the script:

```
t
{
    "id": "04",
    "name": "sunil",
    "department": "HR"
}
Teacher data dumped
Process finished with exit code 0
```

The bottom status bar indicates PyCharm 2021.3 available // Update... (today 09:48), 116:1, 22:46, ENG IN, and 31-01-2024.

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

The screenshot shows the PyCharm IDE interface again. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - teacher.json. The left sidebar shows a project structure with files like Module Calculator.py, nested\_loop.py, nested\_json.json, Number function.py, Operators.py, pass.py, Polymorphism.py, Processing python lists.py, Read data from CSV file.py, ReadWriteintoCSVfile.py, Set & Set methods.py, set.add.clear.discard.py, simple pandas pgm.py, Sort python lists using key.py, special parameters.py, String Function.py, Student\_details.csv, Students\_Data.csv, Students\_Data.csv.csv, teacher.json, tuple.py, and typing.txt. The main code editor window shows the contents of the teacher.json file:

```
{"t_name": "Dhana Laxmi", "t_id": 200, "t_age": 43, "t_salary": 45000}
```

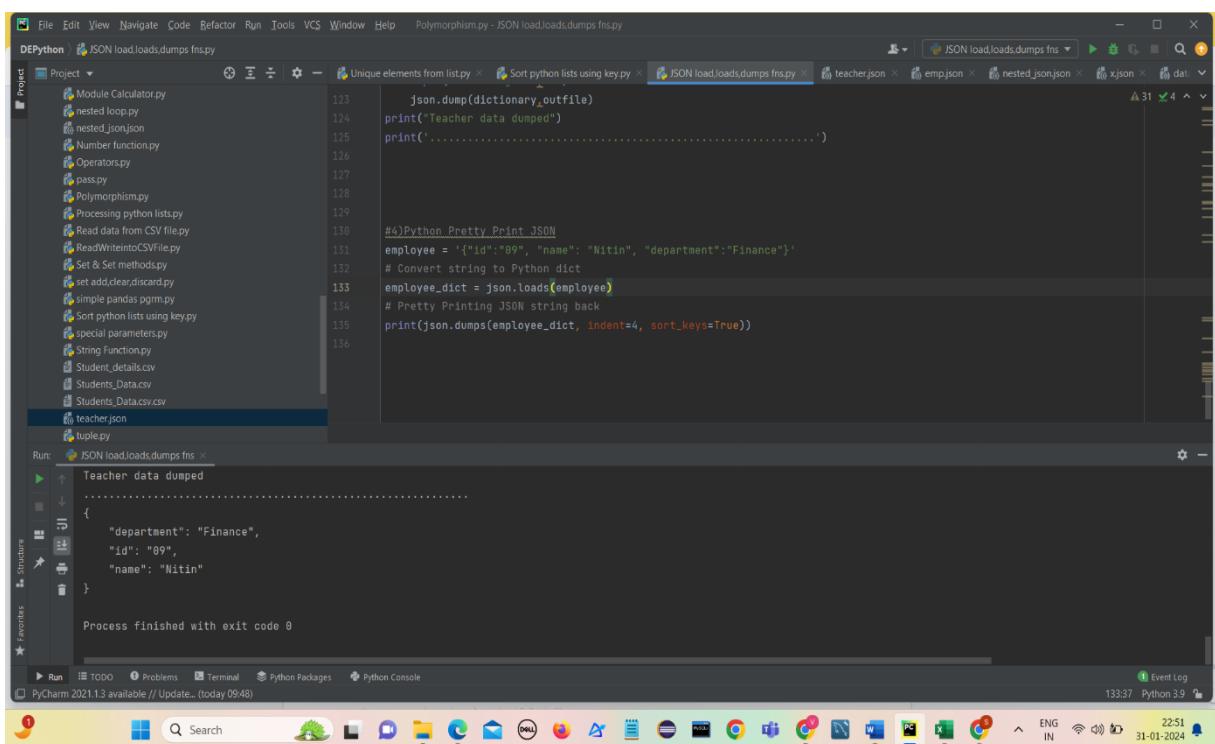
The Run tab shows the output of the script:

```
t
{
    "id": "04",
    "name": "sunil",
    "department": "HR"
}
Teacher data dumped
Process finished with exit code 0
```

The bottom status bar indicates PyCharm 2021.3 available // Update... (today 09:48), 116:1, 22:48, ENG IN, and 31-01-2024.

## 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



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - JSON load.loads.dumps fnsp.py. The title bar of the active window says "JSON load.loads.dumps fnsp.py". The code editor displays a Python script:

```
123     json.dump(dictionary_outfile)
124     print("Teacher data dumped")
125     print('.....')
126
127
128
129
130
131 #4)Python Pretty Print JSON
132 employee = '{"id":"89", "name": "Nitin", "department":"Finance"}'
133 # Convert string to Python dict
134 employee_dict = json.loads(employee)
135 # Pretty Printing JSON string back
136 print(json.dumps(employee_dict, indent=4, sort_keys=True))
```

The run tab shows the output:

```
Teacher data dumped
.....
{
    "department": "Finance",
    "id": "89",
    "name": "Nitin"
}

Process finished with exit code 0
```

The bottom status bar indicates PyCharm 2021.1.3 available // Update.. (today 09:48), 133:37, Python 3.9, 22:51, ENG IN, and 31-01-2024.