

What is a CSV?

CSV stands for “Comma Separated Values.” It is the simplest form of storing data in tabular form as plain text. The first line of a CSV file is the header. It contains the names of the fields/features

Methods to Read a CSV File in Python

1. Using `csv.reader` function by importing `csv` module : The `next()` method returns the current row and moves to the next row.

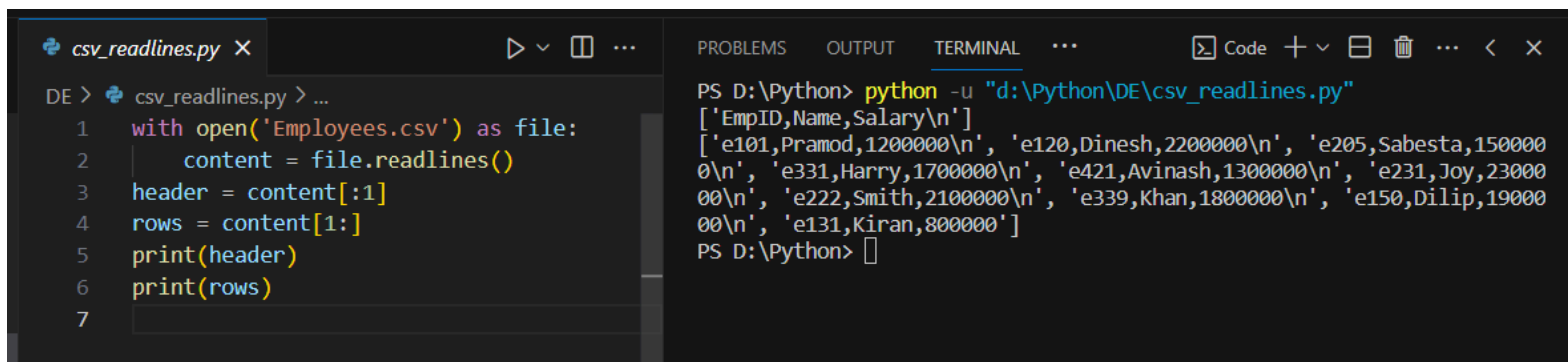


```
csv_reader.py X
DE > csv_reader.py > ...
1 import csv
2 rows = []
3 with open("Employees.csv", 'r') as file:
4     csvreader = csv.reader(file)
5     header = next(csvreader)
6     for row in csvreader:
7         rows.append(row)
8     print(header)
9     print(rows)
10
11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Python> python -u "d:\Python\DE\csv_reader.py"
['EmpID', 'Name', 'Salary']
[['e101', 'Pramod', '1200000'], ['e120', 'Dinesh', '2200000'], ['e205', 'Sabesta', '1500000'], ['e331', 'Harry', '1700000'], ['e421', 'Avinash', '1300000'], ['e231', 'Joy', '2300000'], ['e222', 'Smith', '2100000'], ['e339', 'Khan', '1800000'], ['e150', 'Dilip', '1900000'], ['e131', 'Kiran', '800000']]
PS D:\Python>
```

2. Reading CSV file using `.readlines()` function

Using `readlines()` which returns a list of data in the file then slicing the list to display header and rows.

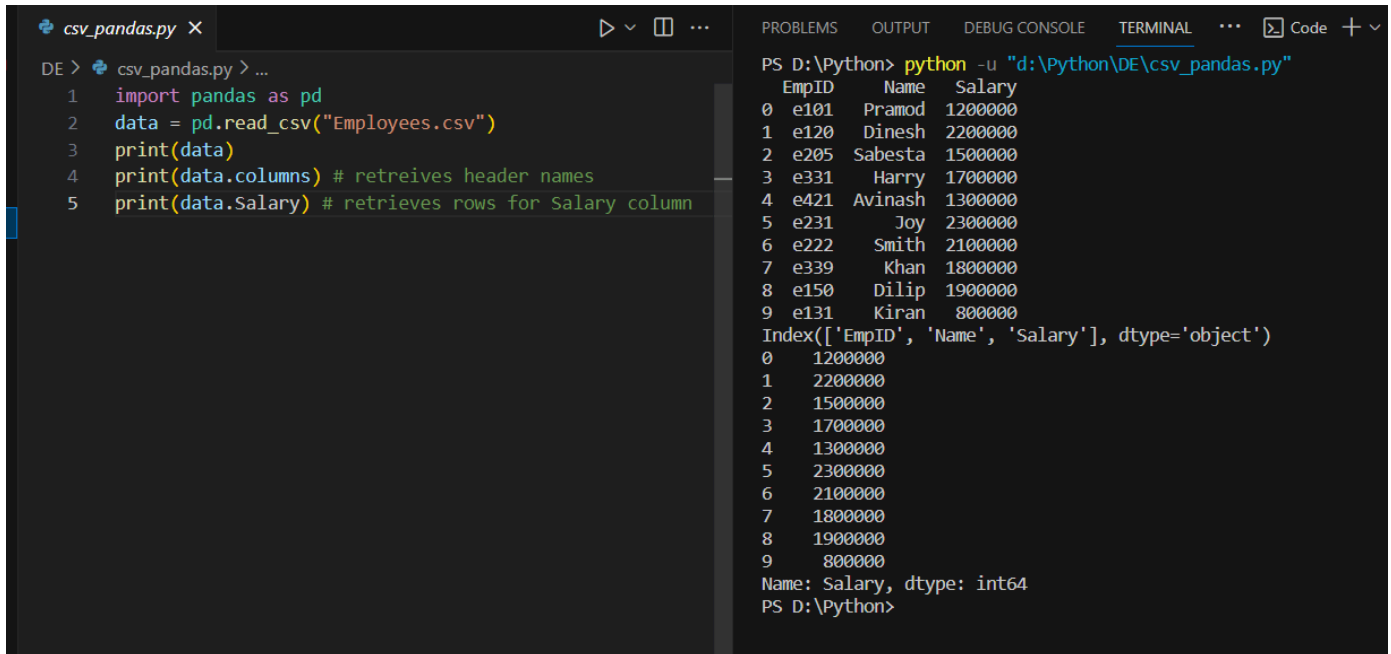


```
csv_readlines.py X
DE > csv_readlines.py > ...
1 with open('Employees.csv') as file:
2     content = file.readlines()
3 header = content[:1]
4 rows = content[1:]
5 print(header)
6 print(rows)
7

PROBLEMS OUTPUT TERMINAL ...
PS D:\Python> python -u "d:\Python\DE\csv_readlines.py"
['EmpID,Name,Salary\n']
['e101,Pramod,1200000\n', 'e120,Dinesh,2200000\n', 'e205,Sabesta,1500000\n', 'e331,Harry,1700000\n', 'e421,Avinash,1300000\n', 'e231,Joy,2300000\n', 'e222,Smith,2100000\n', 'e339,Khan,1800000\n', 'e150,Dilip,1900000\n', 'e131,Kiran,800000\n']
PS D:\Python>
```

3. Reading CSV file using pandas library:

Using `read_csv(file_name)` and storing in the variable and then retrieving columns and rows using `.columns` and `.column_name`.

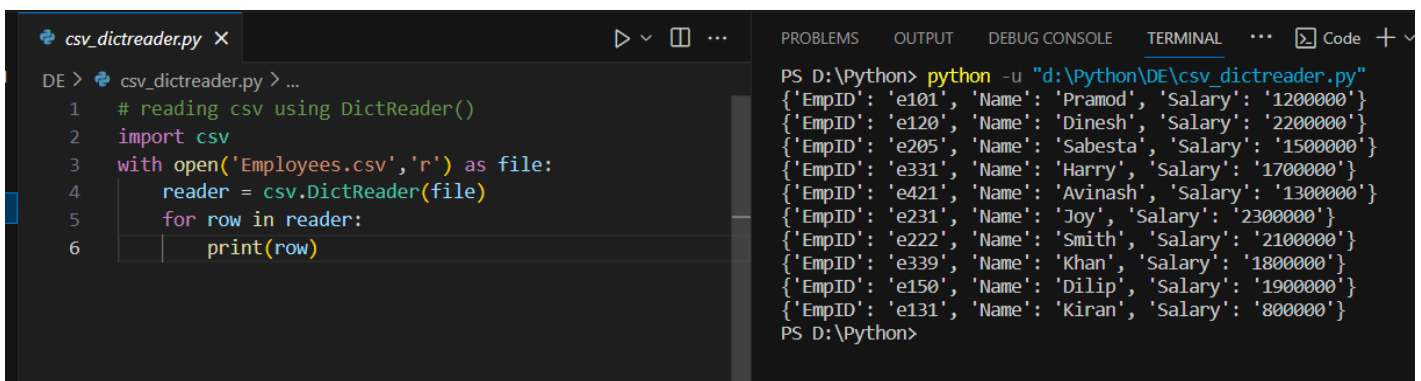


The screenshot shows a code editor with a file named `csv_pandas.py`. The code reads a CSV file named `Employees.csv` using `pd.read_csv()`. It prints the entire data frame, the column names, and the salary values. The terminal output shows the execution of the script, displaying the data frame as a table and the column names as a list.

```
DE > csv_pandas.py > ...
1 import pandas as pd
2 data = pd.read_csv("Employees.csv")
3 print(data)
4 print(data.columns) # retrieves header names
5 print(data.Salary) # retrieves rows for Salary column
```

```
PS D:\Python> python -u "d:\Python\DE\csv_pandas.py"
EmpID      Name      Salary
0  e101    Pramod    1200000
1  e120    Dinesh    2200000
2  e205    Sabesta    1500000
3  e331    Harry     1700000
4  e421    Avinash    1300000
5  e231      Joy     2300000
6  e222    Smith     2100000
7  e339     Khan     1800000
8  e150    Dilip     1900000
9  e131     Kiran      800000
Index(['EmpID', 'Name', 'Salary'], dtype='object')
Name: Salary, dtype: int64
PS D:\Python>
```

4. Using `.dictReader()` to read csv file : Dict is a hash table of keys and values structured in Python. The `dict()` method is used to create a dictionary object from either a specified set or iterables of keys and values. The `csv` module `.DictReader` is used to read CSV files.



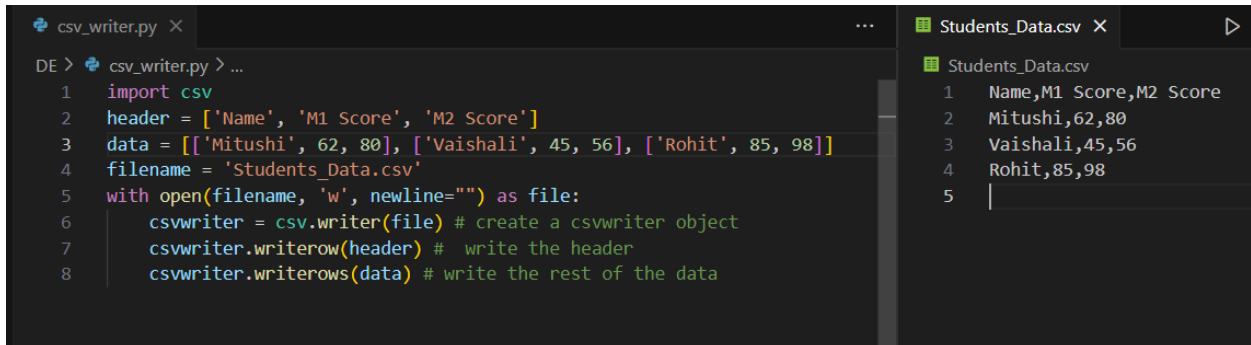
The screenshot shows a code editor with a file named `csv_dictreader.py`. The code uses `csv.DictReader()` to read the `Employees.csv` file. It prints each row as a dictionary. The terminal output shows the execution of the script, displaying each row as a dictionary with keys for `EmpID`, `Name`, and `Salary`.

```
DE > csv_dictreader.py > ...
1 # reading csv using DictReader()
2 import csv
3 with open('Employees.csv','r') as file:
4     reader = csv.DictReader(file)
5     for row in reader:
6         print(row)
```

```
PS D:\Python> python -u "d:\Python\DE\csv_dictreader.py"
{'EmpID': 'e101', 'Name': 'Pramod', 'Salary': '1200000'}
{'EmpID': 'e120', 'Name': 'Dinesh', 'Salary': '2200000'}
{'EmpID': 'e205', 'Name': 'Sabesta', 'Salary': '1500000'}
{'EmpID': 'e331', 'Name': 'Harry', 'Salary': '1700000'}
{'EmpID': 'e421', 'Name': 'Avinash', 'Salary': '1300000'}
{'EmpID': 'e231', 'Name': 'Joy', 'Salary': '2300000'}
{'EmpID': 'e222', 'Name': 'Smith', 'Salary': '2100000'}
{'EmpID': 'e339', 'Name': 'Khan', 'Salary': '1800000'}
{'EmpID': 'e150', 'Name': 'Dilip', 'Salary': '1900000'}
{'EmpID': 'e131', 'Name': 'Kiran', 'Salary': '800000'}
PS D:\Python>
```

Methods to Write a CSV file in python

1. **Write CSV file using csv.writer** : Creating a new csv file and opening it in write mode. Then creating a csv.writer object and adding the header and data rows using writerow() and writerows() from csv module.



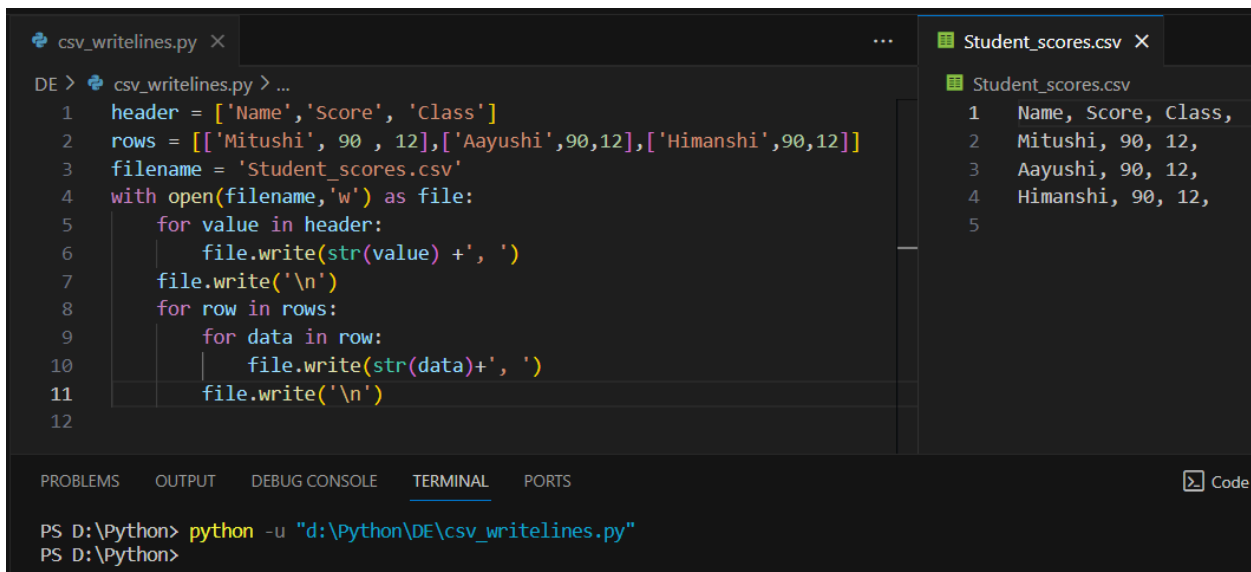
The screenshot shows a code editor with two files. The left file, `csv_writer.py`, contains the following Python code:

```
1 import csv
2 header = ['Name', 'M1 Score', 'M2 Score']
3 data = [['Mitushi', 62, 80], ['Vaishali', 45, 56], ['Rohit', 85, 98]]
4 filename = 'Students_Data.csv'
5 with open(filename, 'w', newline='') as file:
6     csvwriter = csv.writer(file) # create a csvwriter object
7     csvwriter.writerow(header) # write the header
8     csvwriter.writerows(data) # write the rest of the data
```

The right file, `Students_Data.csv`, displays the resulting CSV content:

1	Name,M1 Score,M2 Score
2	Mitushi,62,80
3	Vaishali,45,56
4	Rohit,85,98
5	

2. **Write CSV File Using writelines()** : writelines() iterates through each list, converts the list elements to a string, and then writes it to the csv file.



The screenshot shows a code editor with two files. The left file, `csv_writelines.py`, contains the following Python code:

```
1 header = ['Name','Score', 'Class']
2 rows = [['Mitushi', 90 , 12],['Aayushi',90,12],['Himanshi',90,12]]
3 filename = 'Student_scores.csv'
4 with open(filename,'w') as file:
5     for value in header:
6         file.write(str(value) + ', ')
7     file.write('\n')
8     for row in rows:
9         for data in row:
10            file.write(str(data)+' , ')
11        file.write('\n')
12
```

The right file, `Student_scores.csv`, displays the resulting CSV content:

1	Name, Score, Class,
2	Mitushi, 90, 12,
3	Aayushi, 90, 12,
4	Himanshi, 90, 12,
5	

At the bottom, the terminal shows the command to run the script:

```
PS D:\Python> python -u "d:\Python\DE\csv_writelines.py"
PS D:\Python>
```

3. **Write in CSV using pandas**

Writing into csv using `dataframe()` and `columns` attribute takes the list of header names. The `to_csv()` function from pandas is used to write into csv files.

The screenshot shows an IDE with a Python script named `csv_write_pandas.py` and a generated CSV file named `Stud_score.csv`.

```
DE > csv_write_pandas.py > ...
1 import pandas as pd
2 header = ['Name','Score','Class']
3 rows = [['Mitushi', 90 , 12],['Aayushi',90,12],['Himanshi',90,12]]
4 data = pd.DataFrame(rows, columns=header)
5 data.to_csv('Stud_score.csv', index=False)
```

The `Stud_score.csv` file contains the following data:

	Name	Score	Class
1	Mitushi	90	12
2	Aayushi	90	12
3	Himanshi	90	12

The terminal at the bottom shows the command to run the script:

```
PS D:\Python> python -u "d:\Python\DE\csv_writelines.py"
PS D:\Python> python -u "d:\Python\DE\csv_write_pandas.py"
```

4. Write CSV File Using `csv.DictWriter` : Writing into csv file by creating dictionary and using `DictWriter()` with `fieldnames` parameter

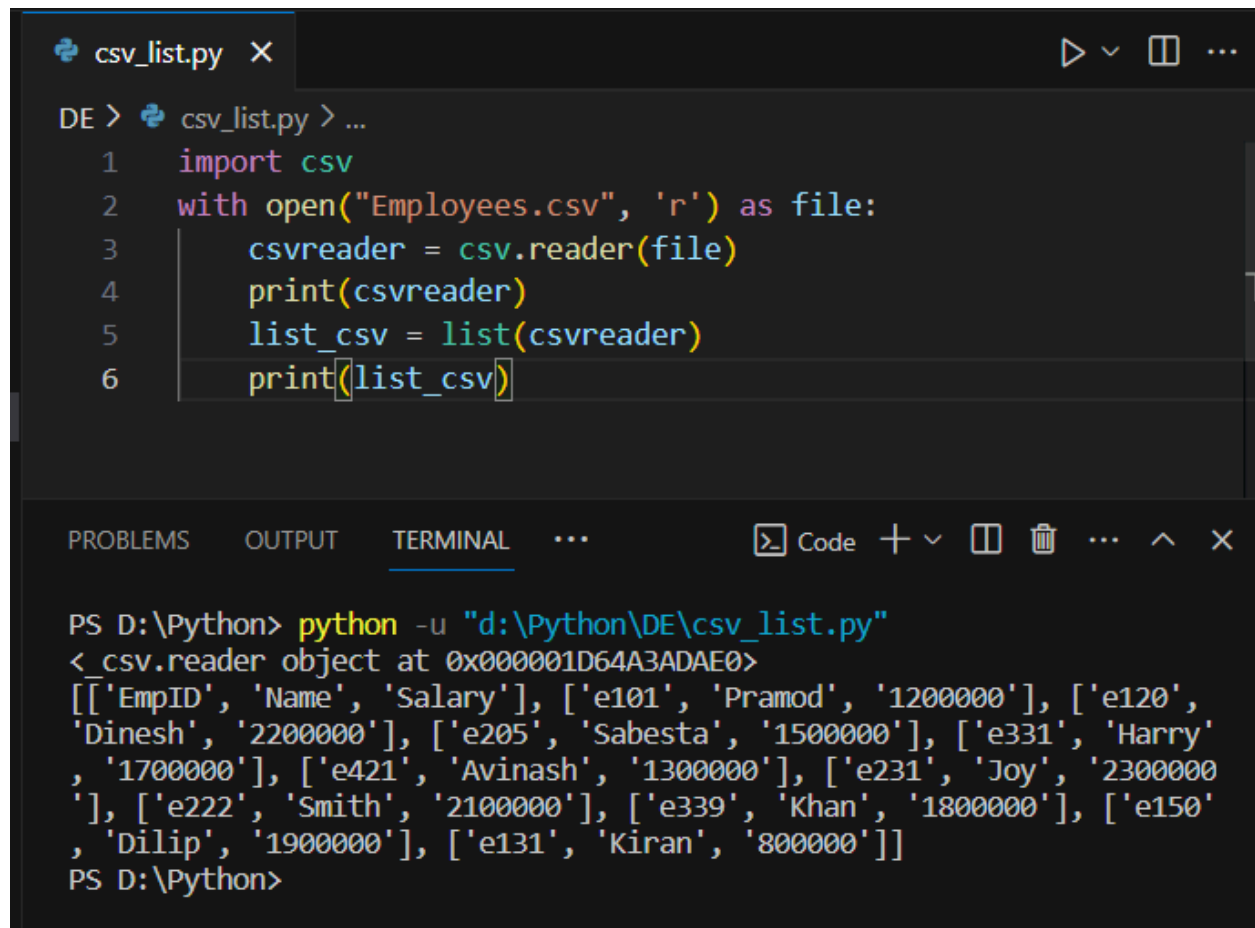
The screenshot shows an IDE with a Python script named `csv_dictwriter.py` and a generated CSV file named `Emp_Data.csv`.

```
DE > csv_dictwriter.py > ...
1 import csv
2 data=[{'Name': 'Rudra', 'Age': 25, 'Salary': 300000},
3       {'Name': 'Krushna', 'Age': 24, 'Salary': 400000},
4       {'Name': 'Ram', 'Age': 21, 'Salary': 20000}]
5 with open('Emp_Data.csv', 'w', newline='') as file:
6     field_names=['Name', 'Age', 'Salary']
7     writer=csv.DictWriter(file, fieldnames=field_names)
8     writer.writeheader()
9     writer.writerows(data)
```

The `Emp_Data.csv` file contains the following data:

	Name	Age	Salary
1	Rudra	25	300000
2	Krushna	24	400000
3	Ram	21	20000

Read CSV file data into list :



The screenshot shows a Python IDE with a file named `csv_list.py` open. The code in the editor reads a CSV file named `Employees.csv` and prints its contents as a list. The terminal output shows the execution of the script, displaying the CSV data as a list of lists.

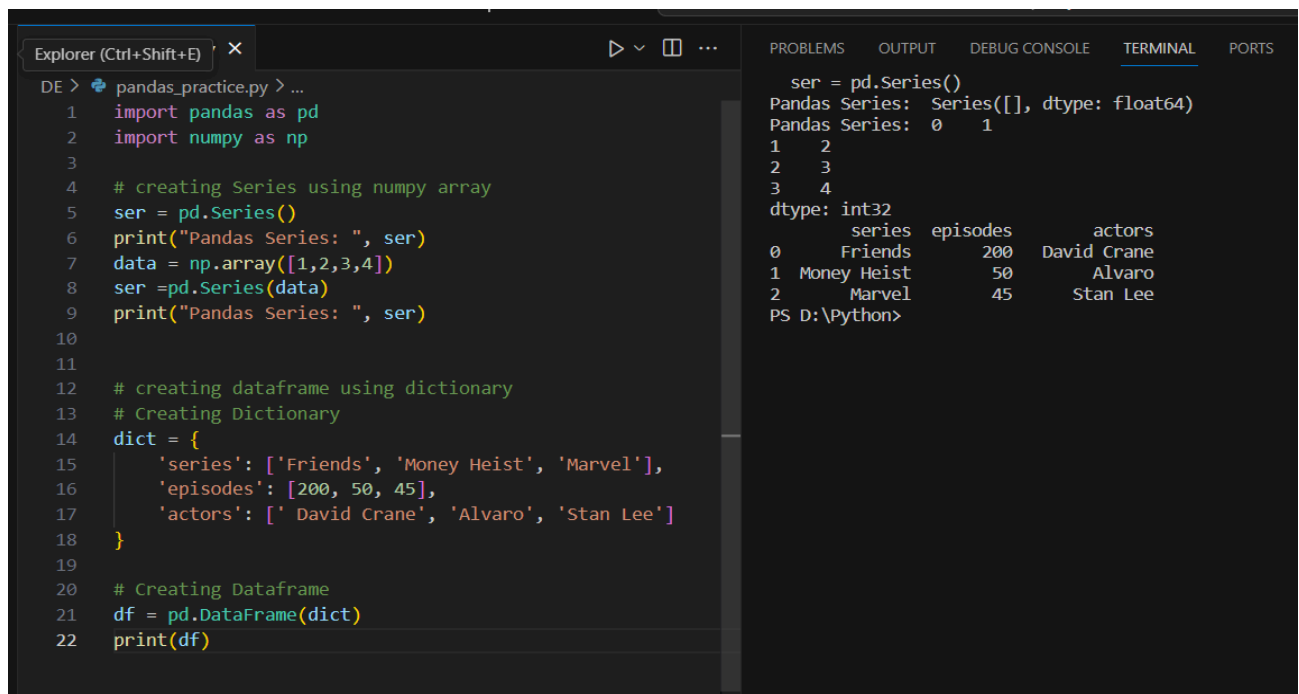
```
DE > csv_list.py > ...
1 import csv
2 with open("Employees.csv", 'r') as file:
3     csvreader = csv.reader(file)
4     print(csvreader)
5     list_csv = list(csvreader)
6     print(list_csv)
```

PROBLEMS OUTPUT TERMINAL ... Code + - [] [X] ... ^ X

```
PS D:\Python> python -u "d:\Python\DE\csv_list.py"
<_csv.reader object at 0x000001D64A3ADAE0>
[['EmpID', 'Name', 'Salary'], ['e101', 'Pramod', '1200000'], ['e120', 'Dinesh', '2200000'], ['e205', 'Sabesta', '1500000'], ['e331', 'Harry', '1700000'], ['e421', 'Avinash', '1300000'], ['e231', 'Joy', '2300000'], ['e222', 'Smith', '2100000'], ['e339', 'Khan', '1800000'], ['e150', 'Dilip', '1900000'], ['e131', 'Kiran', '800000']]
PS D:\Python>
```

Pandas : Pandas is a library used for data manipulation in python. There are two types of data structures in pandas.

Series and DataFrame : Series being one-dimensional same as array capable of storing different types of values.



```
Explorer (Ctrl+Shift+E) x
DE > pandas_practice.py > ...
1 import pandas as pd
2 import numpy as np
3
4 # creating Series using numpy array
5 ser = pd.Series()
6 print("Pandas Series: ", ser)
7 data = np.array([1,2,3,4])
8 ser =pd.Series(data)
9 print("Pandas Series: ", ser)
10
11
12 # creating dataframe using dictionary
13 # Creating Dictionary
14 dict = {
15     'series': ['Friends', 'Money Heist', 'Marvel'],
16     'episodes': [200, 50, 45],
17     'actors': ['David Crane', 'Alvaro', 'Stan Lee']
18 }
19
20 # Creating Dataframe
21 df = pd.DataFrame(dict)
22 print(df)
```

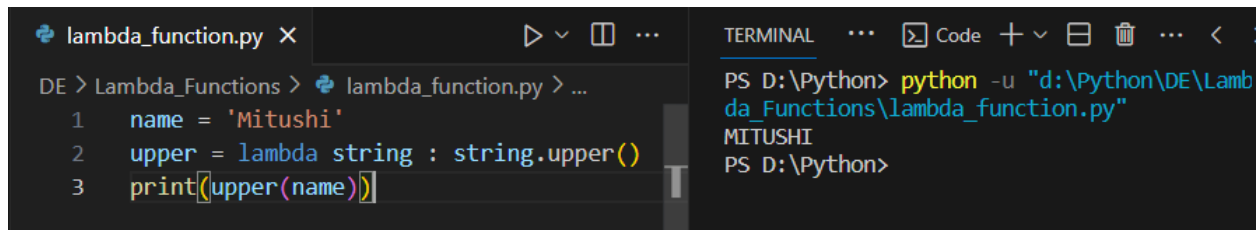
ser = pd.Series()
Pandas Series: Series([], dtype: float64)
Pandas Series: 0 1
1 2
2 3
3 4
dtype: int32

	series	episodes	actors
0	Friends	200	David Crane
1	Money Heist	50	Alvaro
2	Marvel	45	Stan Lee

PS D:\Python>

Lambda Functions

Lambda functions are anonymous or nameless function written in a single expression. The syntax is like lambda arguments: expression. Lambda works for only one expression.

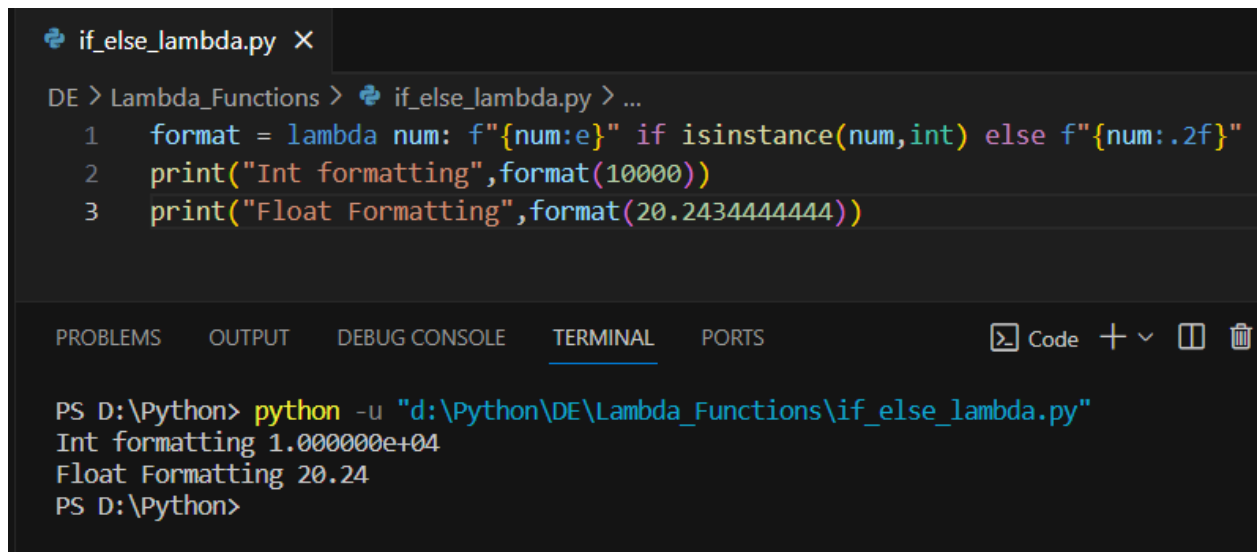


The screenshot shows a code editor with a file named `lambda_function.py`. The code defines a variable `name` with the value `'Mitushi'`, a lambda function `upper` that takes a string and returns its uppercase version using `string.upper()`, and then prints the result of `upper(name)`. To the right, a terminal window shows the command `python -u "d:\Python\DE\Lambda_Functions\lambda_function.py"` being executed, with the output `MITUSHI`.

```
DE > Lambda_Functions > lambda_function.py > ...
1  name = 'Mitushi'
2  upper = lambda string : string.upper()
3  print(upper(name))

TERMINAL
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\lambda_function.py"
MITUSHI
PS D:\Python>
```

Python Lambda Function with if-else

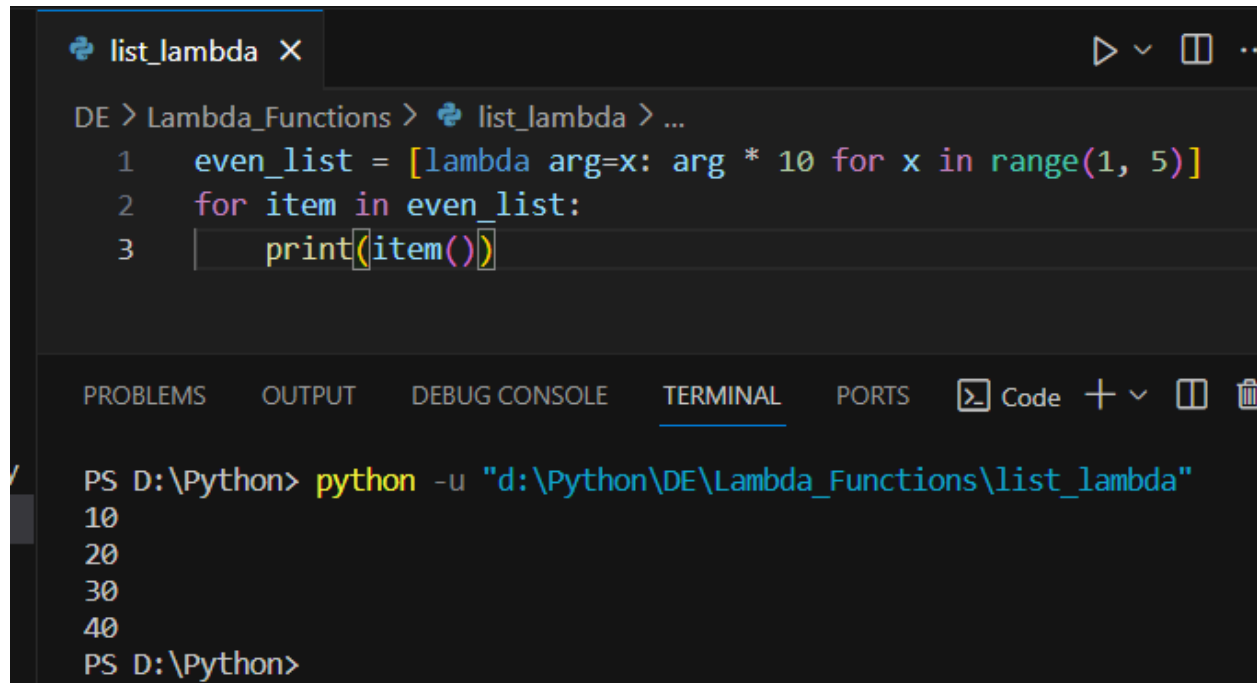


The screenshot shows a code editor with a file named `if_else_lambda.py`. The code defines a lambda function `format` that takes a number and returns a string formatted as either scientific notation (using `{num:e}`) if the number is an integer, or as a float (using `{num:.2f}`) otherwise. It then prints the results of `format(10000)` and `format(20.2434444444)`. Below the code, a terminal window shows the command `python -u "d:\Python\DE\Lambda_Functions\if_else_lambda.py"` being executed, with the output `Int formatting 1.000000e+04` and `Float Formatting 20.24`.

```
DE > Lambda_Functions > if_else_lambda.py > ...
1  format = lambda num: f"{num:e}" if isinstance(num,int) else f"{num:.2f}"
2  print("Int formatting",format(10000))
3  print("Float Formatting",format(20.2434444444))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\if_else_lambda.py"
Int formatting 1.000000e+04
Float Formatting 20.24
PS D:\Python>
```

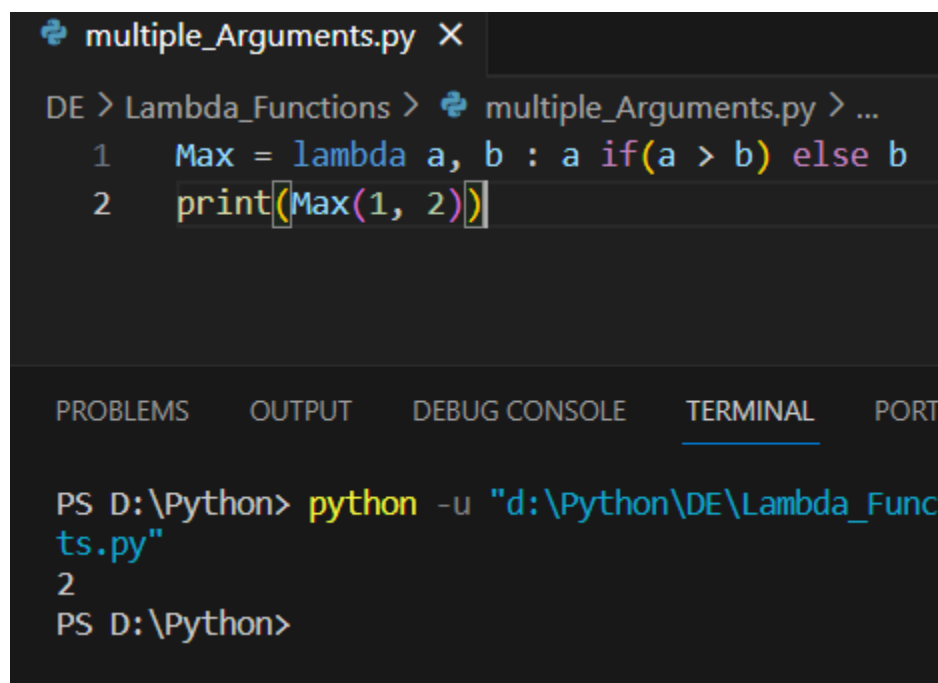
Lambda inside list comprehension : Here a list of lambda functions is created using list comprehension and each list item is called outside the for loop.



```
list_lambda X
DE > Lambda_Functions > list_lambda > ...
1  even_list = [lambda arg=x: arg * 10 for x in range(1, 5)]
2  for item in even_list:
3      print(item())

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code + - 
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\list_lambda"
10
20
30
40
PS D:\Python>
```

Multiple arguments in lambda function with if-else



```
multiple_Arguments.py X
DE > Lambda_Functions > multiple_Arguments.py > ...
1  Max = lambda a, b : a if(a > b) else b
2  print(Max(1, 2))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\multiple_Arguments.py"
2
PS D:\Python>
```


Filter with lambda : The filter() function in Python is used to filter elements from an iterable (e.g., a list) based on a given function.

```
lambda_filter.py X
DE > Lambda_Functions > lambda_filter.py > ...
1  #-----Example 1
2  words = ["apple", "banana", "cherry", "date", "elderberry"]
3
4  # Using filter with lambda
5  filtered_words = filter(lambda x: len(x) <= 5, words)
6
7  # Converting the filter object to a list for display
8  result_list = list(filtered_words)
9
10 print("Original words:", words)
11 print("Filtered words with length <= 5:", result_list)
12
13 #-----Example 2
14 # Filter out all odd numbers
15 li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
16 print('Original List',li)
17 final_list = list(filter(lambda x: (x % 2 != 0), li))
18 print("list with odd numbers",final_list)
19
20 #-----Example 3
21 # Filter all people having age more than 18
22 ages = [13, 90, 17, 59, 21, 60, 5]
23 adults = list(filter(lambda age: age > 18, ages))
24
25 print(adults)
26

PROBLEMS OUTPUT TERMINAL ...
Original words: ['apple', 'banana', 'cherry', 'date', 'elderberry']
Filtered words with length <= 5: ['apple', 'date']
Original List [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
list with odd numbers [5, 7, 97, 77, 23, 73, 61]
[90, 59, 21, 60]
PS D:\Python>
```

Using lambda() Function with map()

The map() function in Python is used to apply a specific function to all items in an iterable (e.g., a list). The map() function in Python takes in a function and a list as an argument.

```
result = map(lambda x: expression(x), iterable)
```

lambda_map.py X

DE > Lambda_Functions > lambda_map.py > ...

```
1  # Multiply all elements of a list by 2
2
3  li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
4
5  final_list = list(map(lambda x: x*2, li))
6  print(final_list)
7
8  # Squaring numbers
9  numbers = [1, 2, 3, 4, 5]
10
11 # Using map with lambda to square each number
12 squared_numbers = map(lambda x: x ** 2, numbers)
13
14 # Converting the map object to a list for display
15 result_list = list(squared_numbers)
16
17 print("Original numbers:", numbers)
18 print("Squared numbers:", result_list)
19
20 # Transform all elements of a list to upper case
21 animals = ['dog', 'cat', 'parrot', 'rabbit']
22 uppered_animals = list(map(lambda animal: animal.upper(), animals))
23
24 print(uppered_animals)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Code + -

```
Filtered words with length <= 5: ['apple', 'date']
Original List [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
list with odd numbers [5, 7, 97, 77, 23, 73, 61]
[90, 59, 21, 60]
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\lambda_map.py"
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
Original numbers: [1, 2, 3, 4, 5]
Squared numbers: [1, 4, 9, 16, 25]
['DOG', 'CAT', 'PARROT', 'RABBIT']
PS D:\Python>
```

Using lambda() Function with reduce()

The `reduce()` function is part of the `functools` module in Python and is used to apply a specified binary function to the items of an iterable successively to reduce the iterable to a single cumulative value.

```
lambda_reduce.py X [play] [dropdown] [icon] ...

DE > Lambda_Functions > lambda_reduce.py > ...
1  # A sum of all elements in a list
2  import functools
3  li = [5, 8, 10, 20, 50, 100]
4  sum = functools.reduce((lambda x, y: x + y), li)
5  print(sum)
6
7  # Find the maximum element in a list
8  lis = [1, 3, 5, 6, 2, ]
9  print("The maximum element of the list is : ", end="")
10 print(functools.reduce(lambda a, b: a if a > b else b, lis))

PROBLEMS OUTPUT TERMINAL ... [icon] Code [plus] [dropdown] [icon] [trash] ... [up] [close]

Filtered words with length <= 5: ['apple', 'date']
Original List [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
list with odd numbers [5, 7, 97, 77, 23, 73, 61]
[90, 59, 21, 60]
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\lambda_map.py"
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
Original numbers: [1, 2, 3, 4, 5]
Squared numbers: [1, 4, 9, 16, 25]
['DOG', 'CAT', 'PARROT', 'RABBIT']
PS D:\Python> python -u "d:\Python\DE\Lambda_Functions\lambda_reduce.py"
"
193
The maximum element of the list is : 6
PS D:\Python>
```