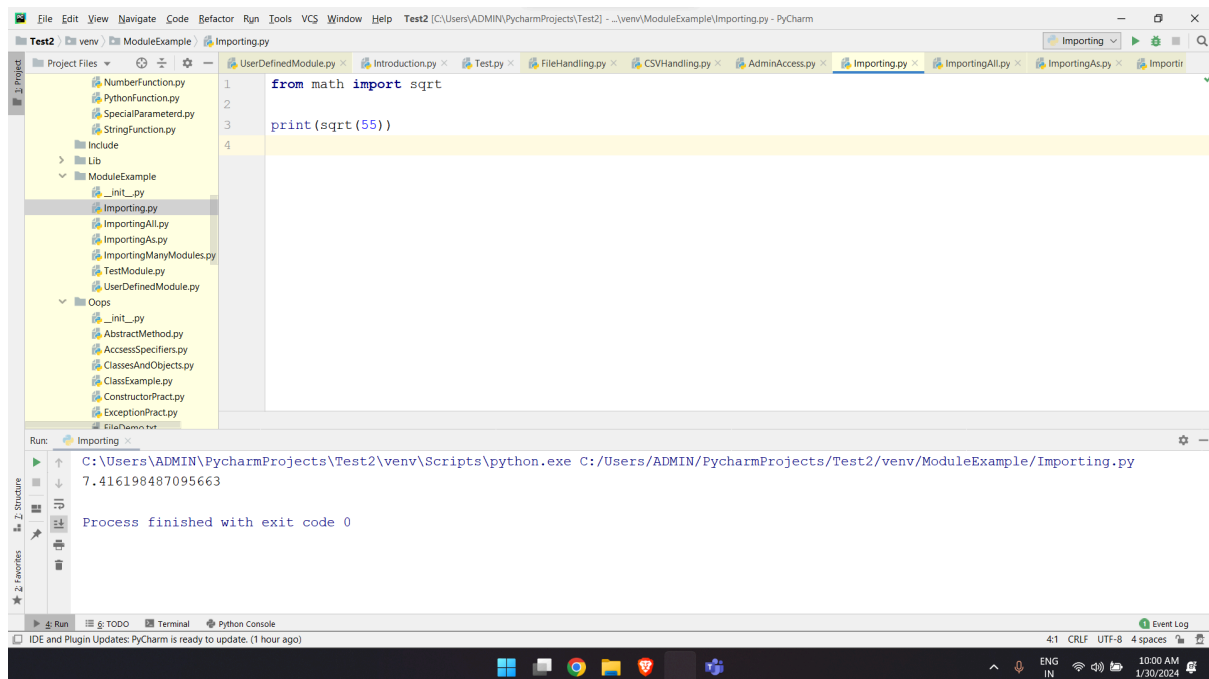


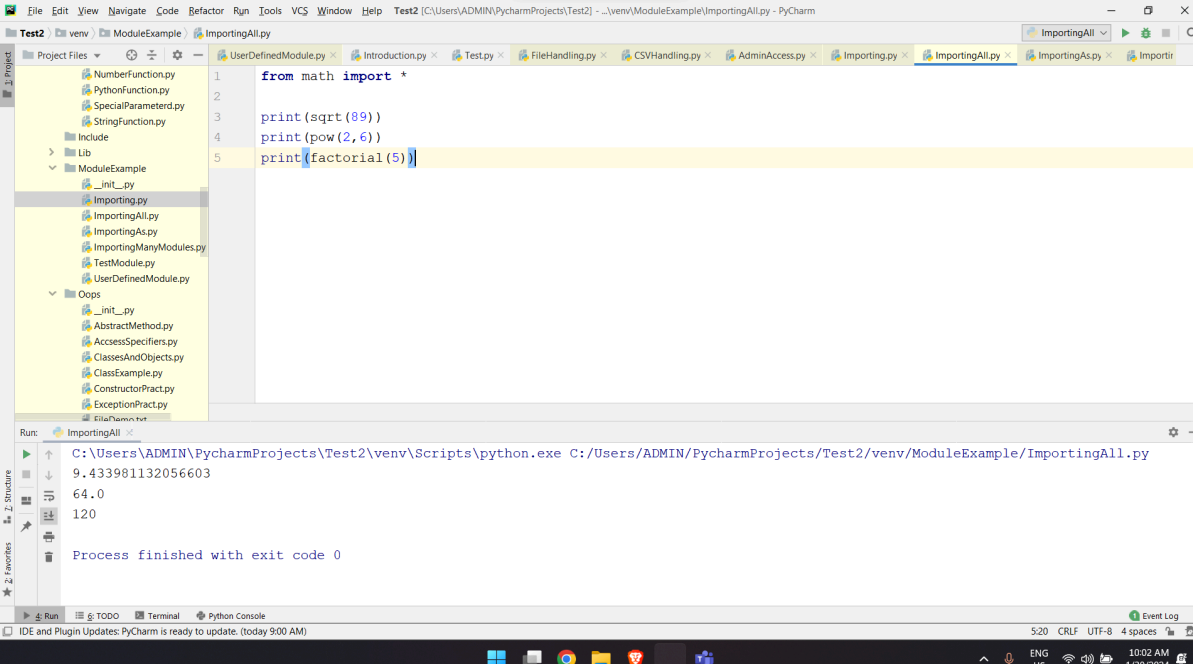
Batch : Data Engineering Batch-1

In Python, the `import` keyword is used to include external modules or libraries into a script, allowing access to their functions, classes, and variables.



Importing using * (astrix) :

Importing with * imports all functions and variables from a module into the current namespace, potentially causing name clashes and making it harder to track the origin of each object.



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameterd.py', 'StringFunction.py', 'Include', 'Lib', 'ModuleExample', 'UserDefinedModule.py', and 'Oops'. The main editor displays a Python script 'ImportingAll.py' with the following code:

```
1 from math import *
2
3 print(sqrt(89))
4 print(pow(2,6))
5 print(factorial(5))
```

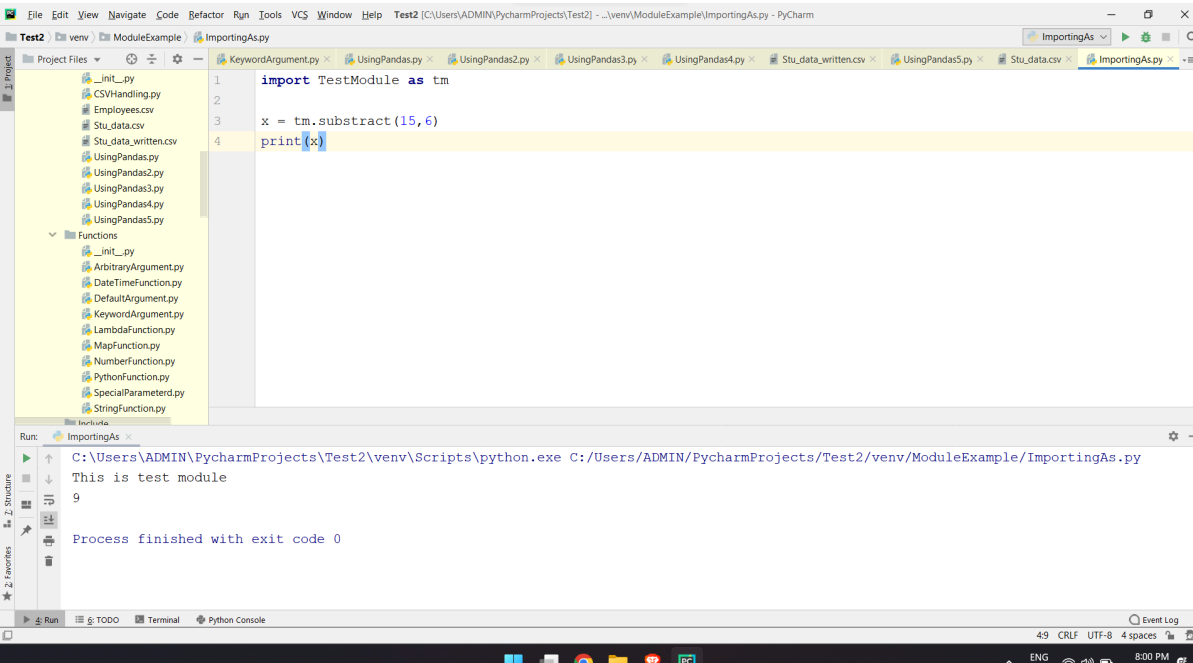
The Run window at the bottom shows the execution of 'ImportingAll.py' using the command 'C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\Test2\venv\ModuleExample\ImportingAll.py'. The output is:

```
9.433981132056603
64.0
120
```

The process finished with exit code 0.

Importing using as :

The as keyword allows renaming a module or its components during import, providing a way to create more readable code or to avoid naming conflicts with existing identifiers in the script.



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like '._init_.py', 'CSVHandling.py', 'Employees.csv', 'Stu_data.csv', 'Stu_data_written.csv', 'UsingPandas.py', 'UsingPandas2.py', 'UsingPandas3.py', 'UsingPandas4.py', 'UsingPandas5.py', 'Functions', 'ArbitraryArgument.py', 'DateTimeFunction.py', 'DefaultArgument.py', 'KeywordArgument.py', 'LambdaFunction.py', 'MapFunction.py', 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameterd.py', and 'StringFunction.py'. The main editor displays a Python script 'ImportingAs.py' with the following code:

```
1 import TestModule as tm
2
3 x = tm.subtract(15,6)
4 print(x)
```

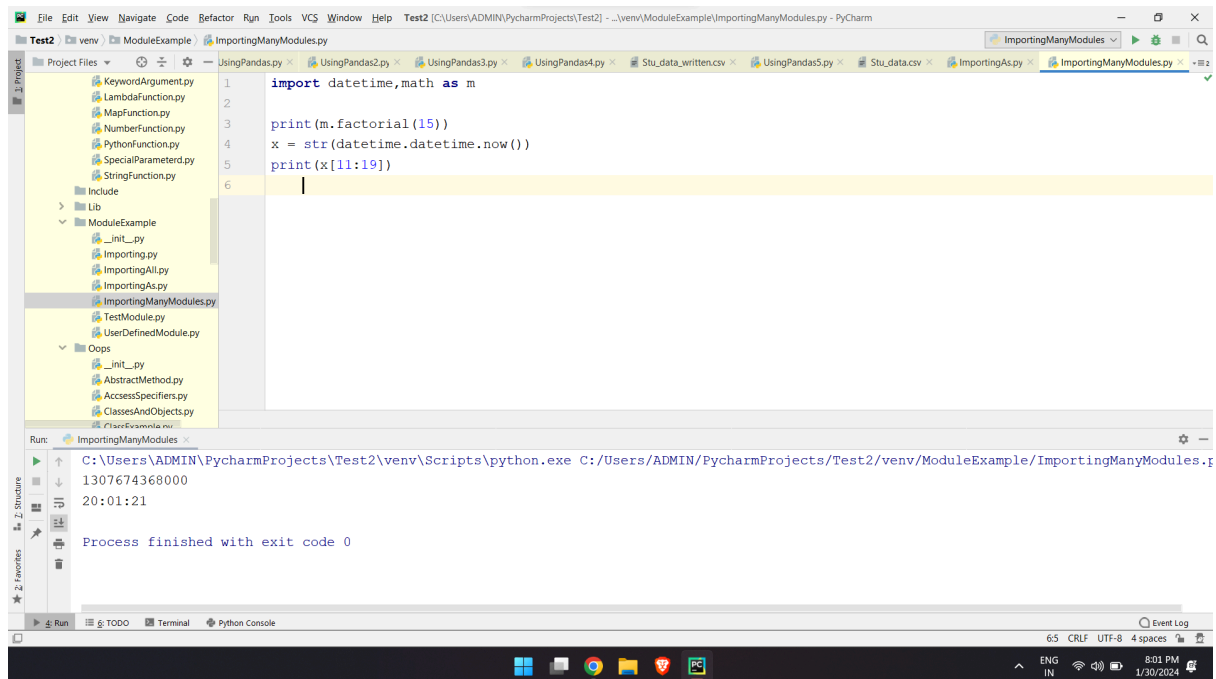
The Run window at the bottom shows the execution of 'ImportingAs.py' using the command 'C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\Test2\venv\ModuleExample\ImportingAs.py'. The output is:

```
This is test module
9
```

The process finished with exit code 0.

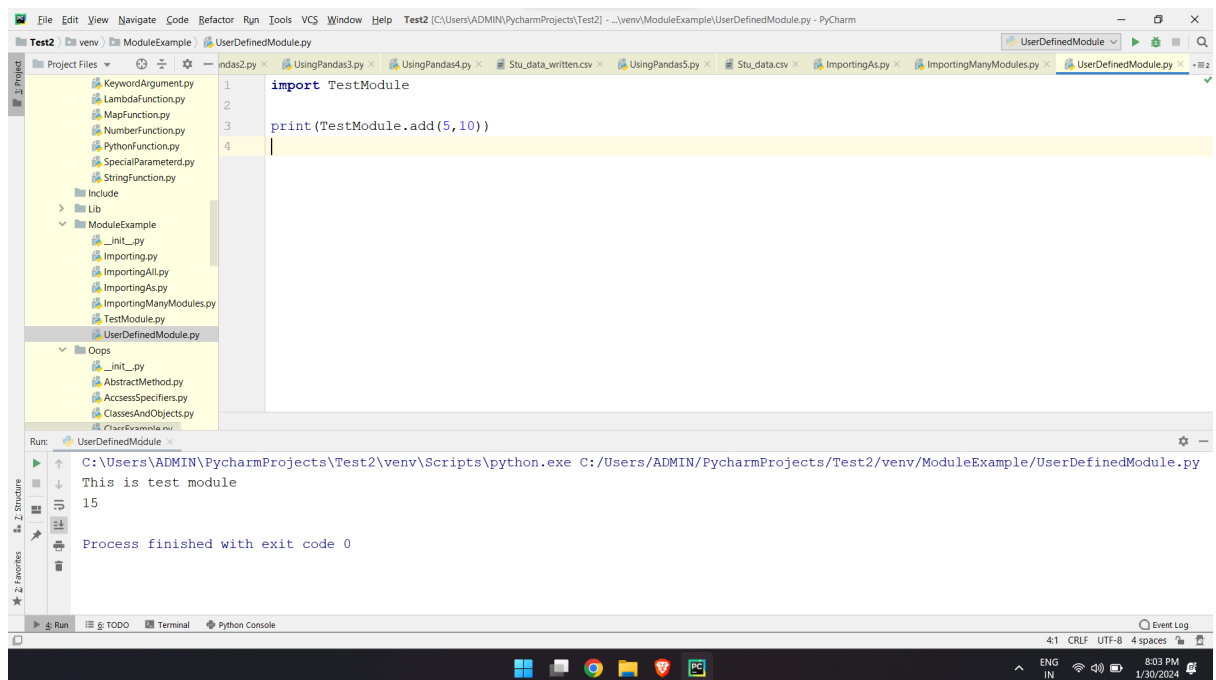
Importing many modules :

Importing multiple modules at once allows access to various functionalities in a single script by using the import keyword followed by module names, separated by commas.



User Defined Module :

A user-defined module is a Python script or file containing reusable code or functions that can be imported and utilized in other Python scripts for code organization and reusability.



File Handling :

File Handling:

File handling in Python involves operations related to reading from and writing to files. The `open()` function is used to open a file, and various methods like `read()`, `readlines()`, `write()`, and `close()` are used for different file operations.

CSV (Comma-Separated Values):

CSV is a file format used for storing tabular data, where each line represents a row, and the values in each row are separated by commas. Python's `csv` module provides functionality to read from and write to CSV files.

`open()` Function:

The `open()` function is used to open a file in Python. It takes the file name and an optional mode as parameters. The mode can be 'r' for reading, 'w' for writing, and 'a' for appending, among others.

`csv.reader` Object:

The `csv.reader` object is created by passing a file object to it. It reads the contents of the file in CSV format and provides an iterator to traverse through rows.

`readlines()` Method:

The `readlines()` method is used to read all lines from a file and returns a list of strings where each string represents a line in the file.

Header in CSV File:

The header in a CSV file is the first line that typically contains column names. It provides labels for the data in each column.

List of Lists (rows):

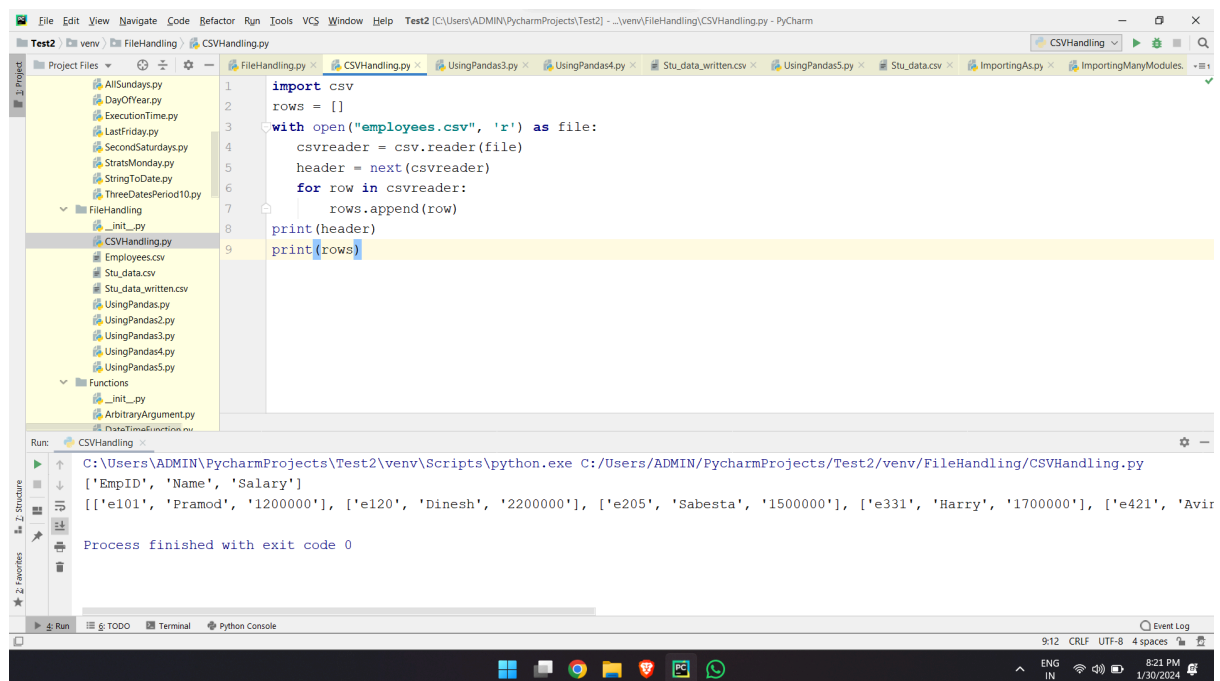
The program creates a list of lists (rows) to store the data read from the CSV file. Each inner list represents a row in the CSV file, and the outer list contains all the rows.

Now, looking at your program:

The first block of code uses the `csv.reader` object to read the contents of the CSV file, storing the header in the `header` variable and the rows in the `rows` list.

The second block of code uses the `readlines()` method to read the entire content of the file. It then extracts the header and rows from the content and prints them.

The given program reads the same CSV file using two different approaches, showcasing both the use of the `csv` module and basic file reading methods. It prints the header and rows obtained from each approach.



```
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)
```

Run: CSVHandling.py

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\Test2\venv\FileHandling\CSVHandling.py
['EmpID', 'Name', 'Salary']
[['e101', 'Pramod', '1200000'], ['e120', 'Dinesh', '2200000'], ['e205', 'Sabesta', '1500000'], ['e331', 'Harry', '1700000'], ['e421', 'Avir', '1800000']]
```

Process finished with exit code 0

Pandas :

Pandas is a popular open-source data manipulation and analysis library for Python. It provides data structures and functions needed to efficiently manipulate large datasets and perform various data analysis tasks. Here are some key aspects of the Pandas library:

Data Structures:

Series: A one-dimensional labeled array capable of holding any data type. It is similar to a column in a spreadsheet or a single column in a DataFrame.

DataFrame: A two-dimensional labeled data structure with columns that can be of different types. It is similar to a spreadsheet or a SQL table.

Key Features:

Data Cleaning: Pandas provides various methods for handling missing data, filtering, and cleaning datasets.

Data Transformation: It supports operations like merging, reshaping, and pivoting for transforming datasets.

Data Analysis: Pandas allows for descriptive statistics, aggregations, and other statistical operations on datasets.

Data Visualization: It integrates well with other libraries like Matplotlib and Seaborn for data visualization.

Reading and Writing Data:

Pandas supports reading data from and writing data to various file formats, including CSV, Excel, SQL databases, and more.

Common functions include `pd.read_csv()`, `pd.to_csv()`, `pd.read_excel()`, `pd.to_excel()`, `pd.read_sql()`, and `pd.to_sql()`.

Indexing and Selecting Data:

Pandas provides powerful indexing capabilities, allowing users to select, filter, and manipulate data using labels or numerical indices.

Operations like slicing, boolean indexing, and query expressions make it easy to access and modify data.

Grouping and Aggregation:

Pandas supports grouping data based on one or more criteria, facilitating aggregation and summarization of data.

The `groupby()` function is commonly used, followed by aggregation functions like `sum()`, `mean()`, `count()`, etc.

Merging and Joining:

Pandas allows combining datasets through merging and joining operations, similar to SQL joins.

The `merge()` function is used to combine DataFrames based on a specified key or column.

Time Series Data:

Pandas provides specialized data structures and functions for handling time series data, making it suitable for financial and time-based analyses.

Flexibility:

Pandas is highly flexible and can handle diverse types of data, including numerical, categorical, and textual data.

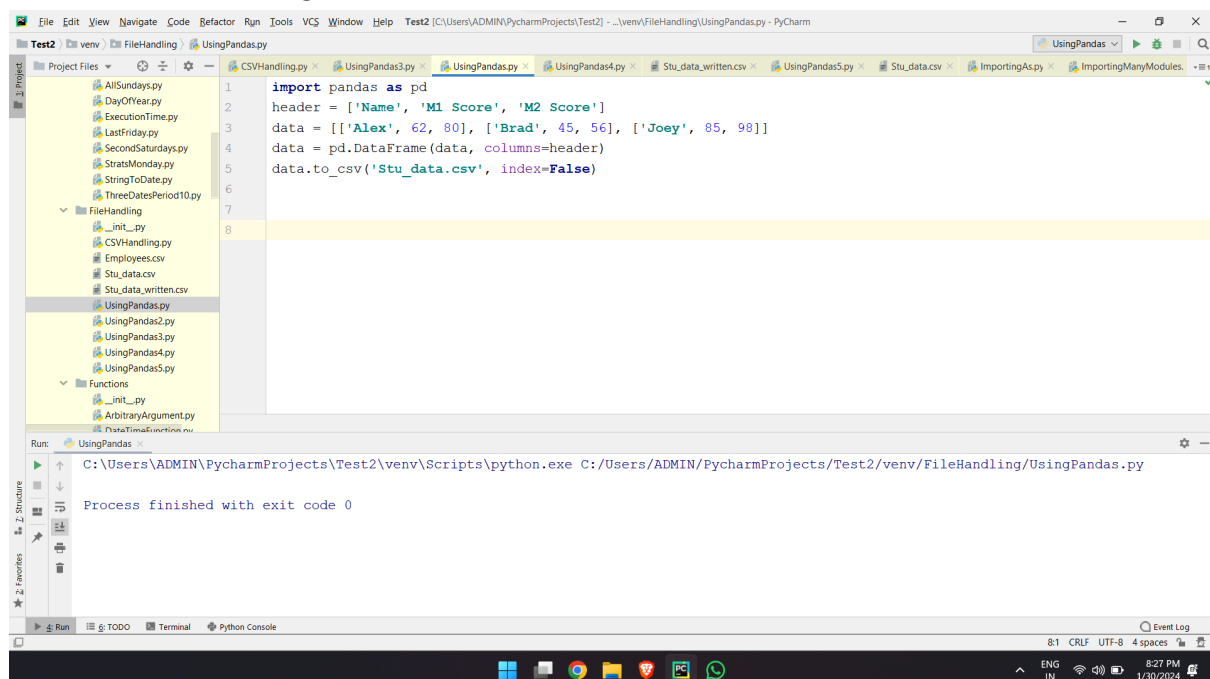
It is widely used in various domains, including data science, finance, biology, and more.

Community and Ecosystem:

Pandas has a large and active community, which contributes to its continuous development and improvement.

It integrates well with other popular Python libraries like NumPy, Matplotlib, and Scikit-Learn.

In summary, Pandas is a powerful and versatile library that simplifies data manipulation and analysis in Python, making it a go-to choice for working with structured data. It is an essential tool for data scientists, analysts, and researchers working with tabular data.

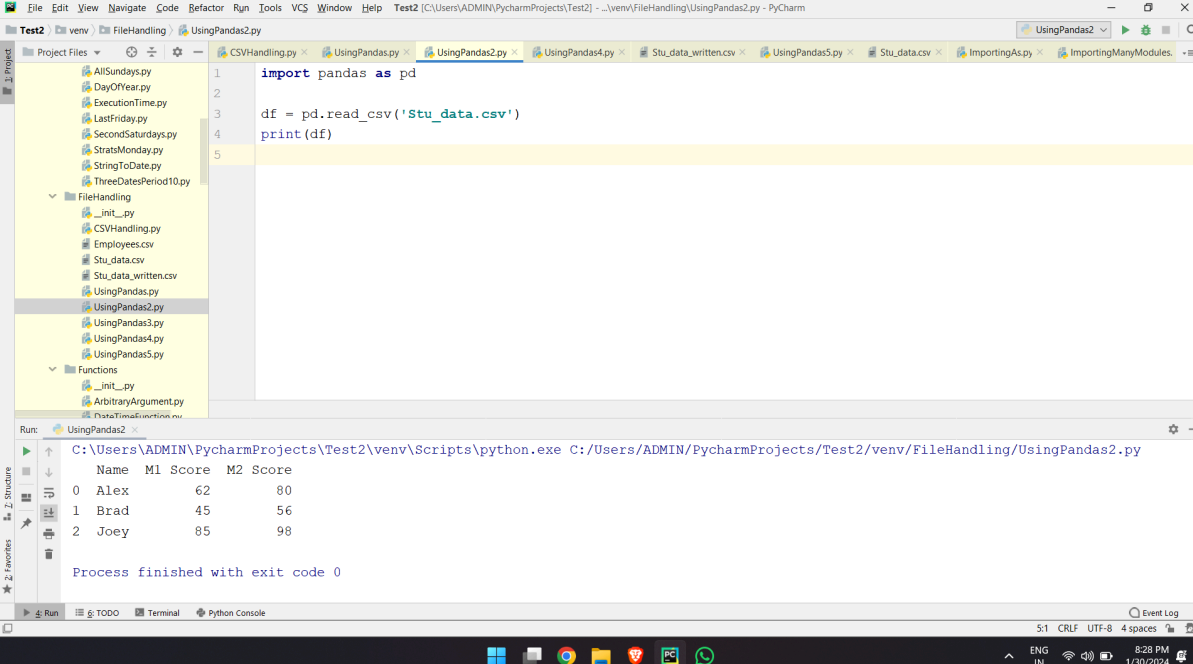


The screenshot displays the PyCharm IDE interface. The main editor window shows a Python script named 'UsingPandas.py' with the following code:

```
1 import pandas as pd
2 header = ['Name', 'M1 Score', 'M2 Score']
3 data = [['Alex', 62, 80], ['Brad', 45, 56], ['Joey', 85, 98]]
4 data = pd.DataFrame(data, columns=header)
5 data.to_csv('Stu_data.csv', index=False)
6
7
8
```

The left sidebar shows the project structure, including a 'FileHandling' folder with various Python files and CSV files. The bottom panel shows the 'Run' output, indicating that the process finished with exit code 0.

Data frames :



The screenshot shows a PyCharm IDE window titled 'Test2'. The left sidebar displays a project structure with a folder named 'FileHandling' containing several Python files, including 'UsingPandas2.py'. The main editor window shows the code for 'UsingPandas2.py':

```
1 import pandas as pd
2
3 df = pd.read_csv('Stu_data.csv')
4 print(df)
5
```

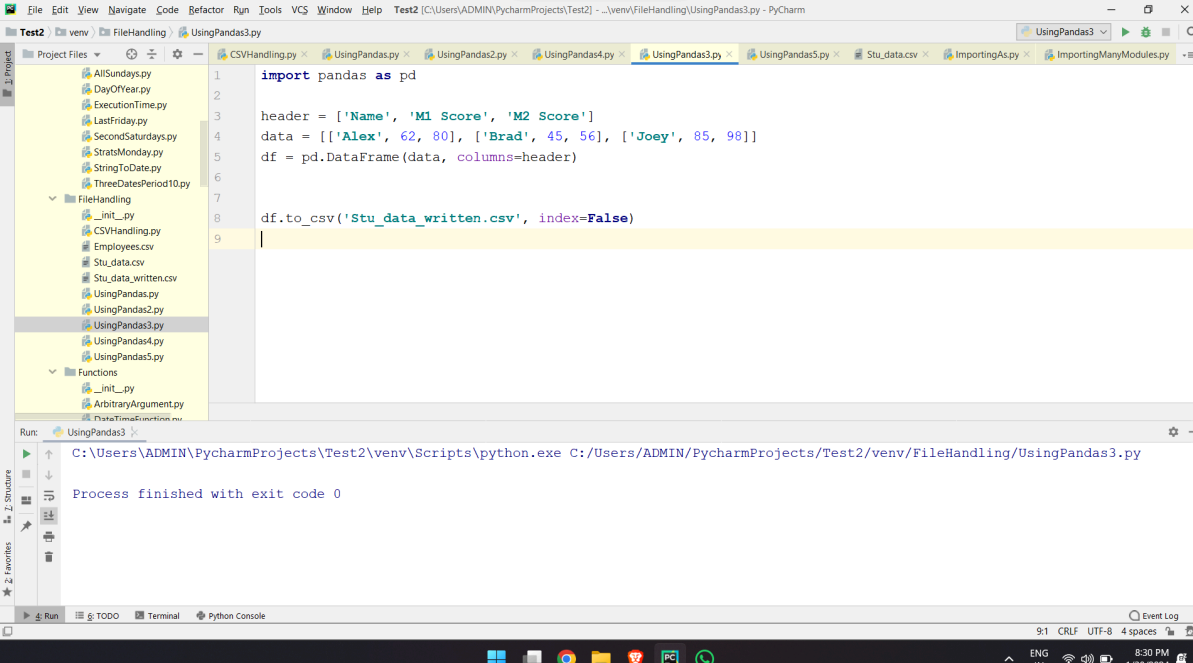
The bottom 'Run' console shows the output of the script, which is a pandas DataFrame with 3 rows and 4 columns:

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/FileHandling/UsingPandas2.py
   Name  M1 Score  M2 Score
0  Alex         62         80
1  Brad         45         56
2  Joey         85         98

Process finished with exit code 0
```

The status bar at the bottom indicates the file encoding is UTF-8 with 4 spaces.

Writing data into file :



The screenshot shows a PyCharm IDE window titled 'Test2'. The left sidebar displays a project structure with a folder named 'FileHandling' containing several Python files, including 'UsingPandas3.py'. The main editor window shows the code for 'UsingPandas3.py':

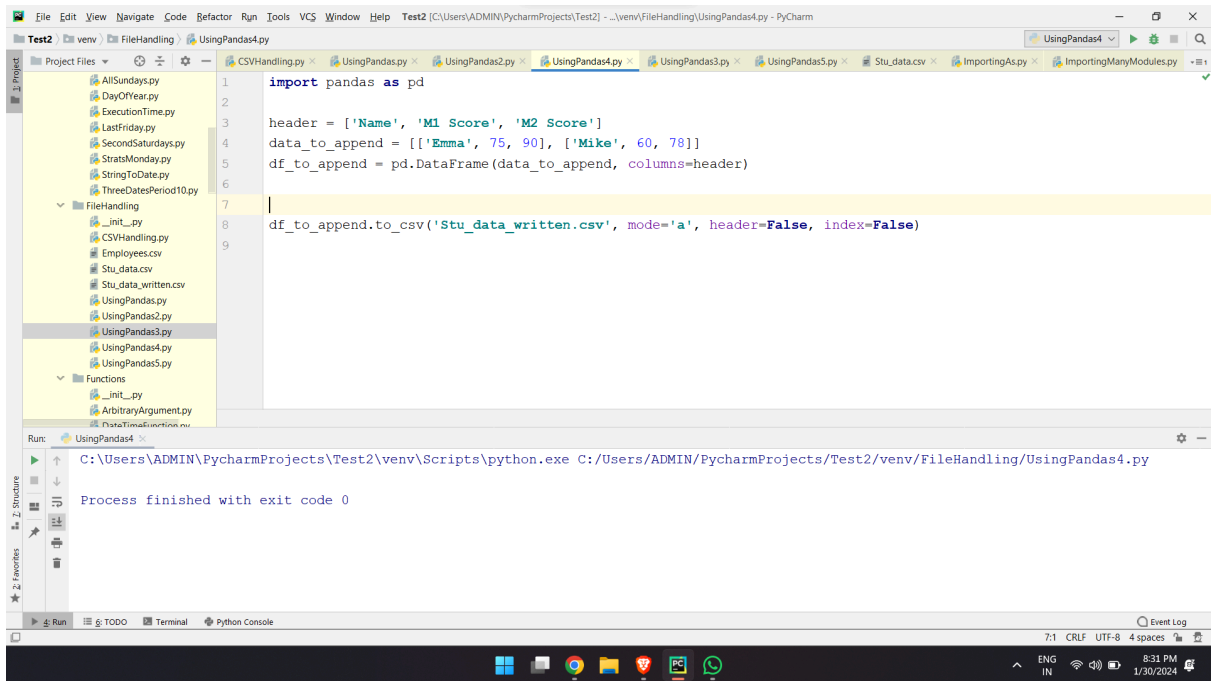
```
1 import pandas as pd
2
3 header = ['Name', 'M1 Score', 'M2 Score']
4 data = [['Alex', 62, 80], ['Brad', 45, 56], ['Joey', 85, 98]]
5 df = pd.DataFrame(data, columns=header)
6
7
8 df.to_csv('Stu_data_written.csv', index=False)
9
```

The bottom 'Run' console shows the output of the script, which is a pandas DataFrame with 3 rows and 4 columns:

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/FileHandling/UsingPandas3.py
Process finished with exit code 0
```

The status bar at the bottom indicates the file encoding is UTF-8 with 4 spaces.

Appending data :



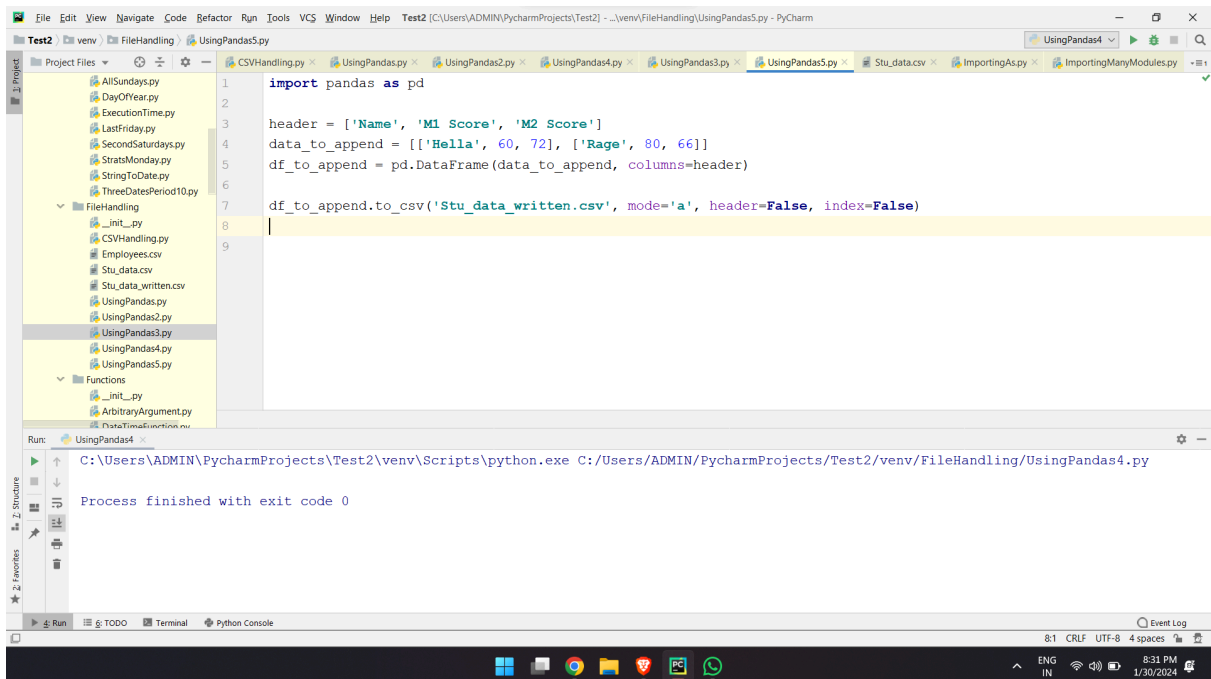
This screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with various Python files and CSV files. The main editor window displays the code for 'UsingPandas4.py'. The code imports pandas as 'pd', defines a header with columns 'Name', 'M1 Score', and 'M2 Score', and appends two rows of data: ['Emma', 75, 90] and ['Mike', 60, 78]. The data is then saved to 'Stu_data_written.csv' in append mode ('a'). The Run console at the bottom shows the command executed and the message 'Process finished with exit code 0'.

```
1 import pandas as pd
2
3 header = ['Name', 'M1 Score', 'M2 Score']
4 data_to_append = [['Emma', 75, 90], ['Mike', 60, 78]]
5 df_to_append = pd.DataFrame(data_to_append, columns=header)
6
7
8 df_to_append.to_csv('Stu_data_written.csv', mode='a', header=False, index=False)
9
```

Run: UsingPandas4

C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/FileHandling/UsingPandas4.py

Process finished with exit code 0



This screenshot shows the PyCharm IDE with the same project 'Test2'. The file explorer on the left shows the same directory structure. The main editor window displays the code for 'UsingPandas5.py'. The code imports pandas as 'pd', defines a header with columns 'Name', 'M1 Score', and 'M2 Score', and appends two rows of data: ['Hella', 60, 72] and ['Rage', 80, 66]. The data is then saved to 'Stu_data_written.csv' in append mode ('a'). The Run console at the bottom shows the command executed and the message 'Process finished with exit code 0'.

```
1 import pandas as pd
2
3 header = ['Name', 'M1 Score', 'M2 Score']
4 data_to_append = [['Hella', 60, 72], ['Rage', 80, 66]]
5 df_to_append = pd.DataFrame(data_to_append, columns=header)
6
7 df_to_append.to_csv('Stu_data_written.csv', mode='a', header=False, index=False)
8
9
```

Run: UsingPandas4

C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/FileHandling/UsingPandas4.py

Process finished with exit code 0

Lambda Function :

Lambda functions in Python, also known as anonymous functions or lambda expressions, are a concise way to create small, inline functions. Here are key aspects of the theory behind lambda functions:

Definition:

A lambda function is defined using the lambda keyword, followed by a list of parameters, a colon (:), and an expression. The syntax is: lambda parameters: expression.

Anonymous Functions:

Lambda functions are often referred to as anonymous because they don't have a name like regular functions defined using the def keyword.

Single Expression:

Lambda functions are designed for simple operations and are limited to a single expression. They are suitable for situations where a full function definition with a def statement might be too verbose.

Usage:

Lambda functions are commonly used in situations where a small, throwaway function is needed for a short duration and doesn't warrant a full function definition.

Functional Programming:

Lambda functions align with principles of functional programming. They can be passed as arguments to higher-order functions like map(), filter(), and sorted().

Syntax and Examples:

The basic syntax is: lambda parameters: expression.

Example 1: add = lambda x, y: x + y

Example 2: square = lambda x: x ** 2

Example 3: is_even = lambda x: x % 2 == 0

Scope:

Lambda functions have their own local scope and cannot access variables from the surrounding scope unless explicitly passed as arguments.

Use Cases:

Lambda functions are often used for short-lived operations where creating a named function would be overkill.

Common use cases include sorting, filtering, and mapping elements in lists.

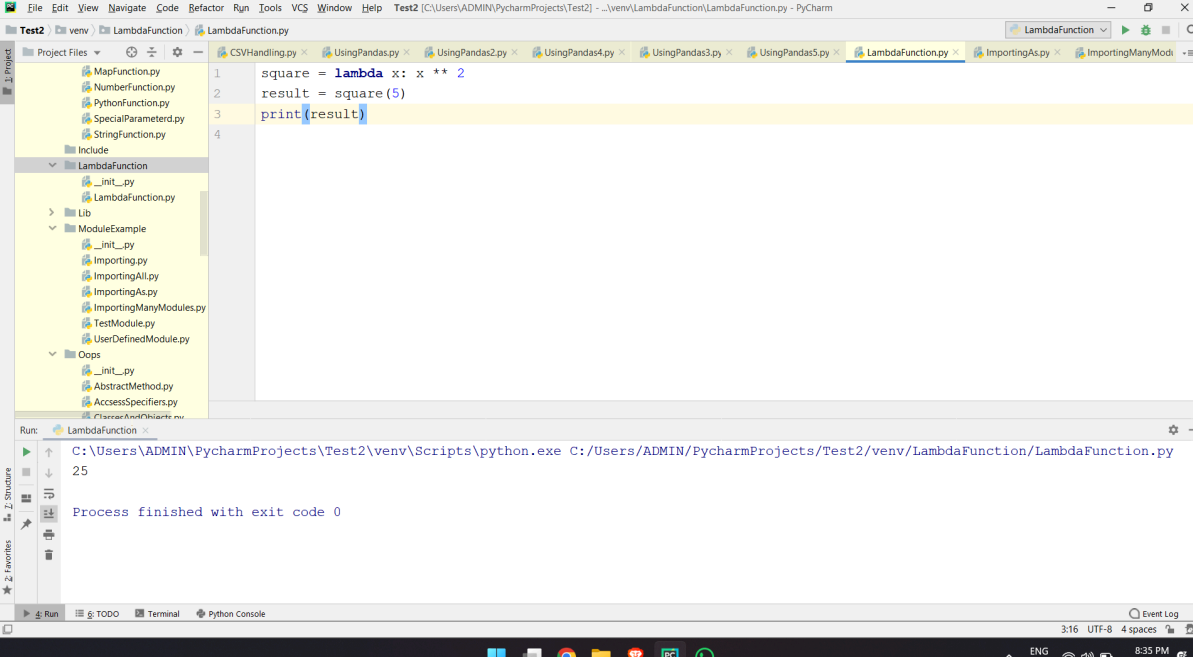
Integration with Higher-Order Functions:

Lambda functions are frequently used with higher-order functions like `map()`, `filter()`, and `sorted()` to perform operations on iterable objects.

Avoiding Named Functions:

In situations where a function is used only once and its behavior is simple, using a lambda function can be more concise and avoids cluttering the code with unnecessary named functions.

Example,



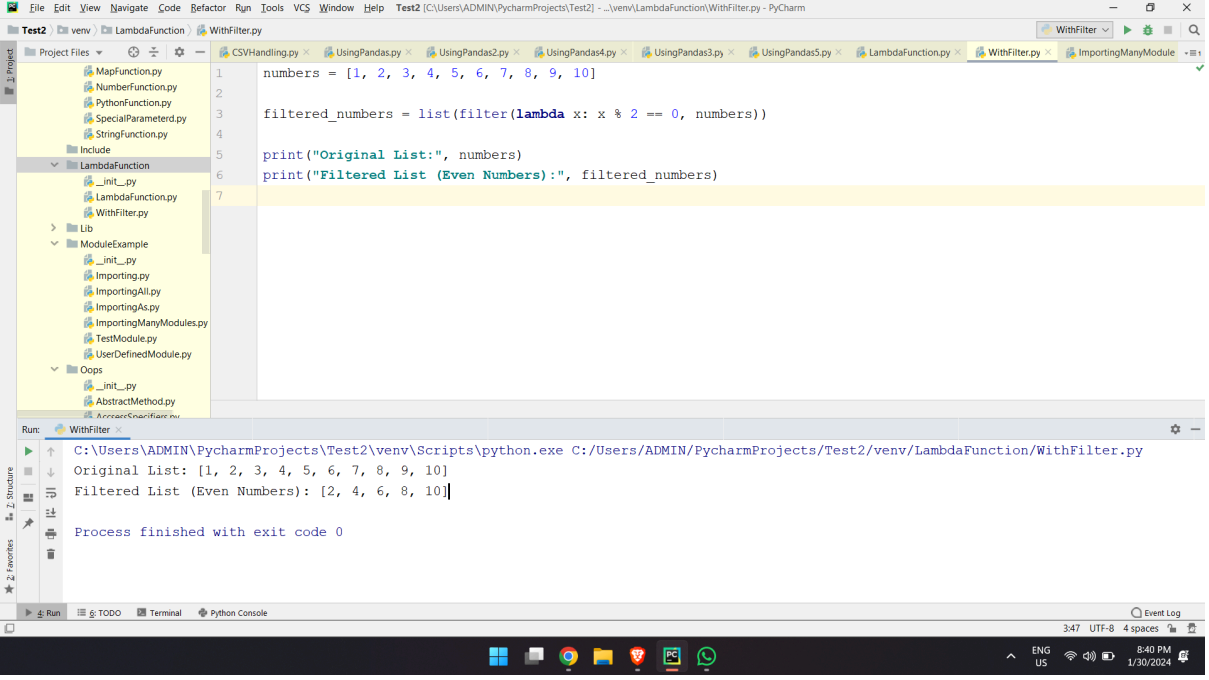
The screenshot shows a PyCharm IDE window with a project named 'Test2'. The 'Project Files' sidebar on the left shows a directory structure with files like 'MapFunction.py', 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameterd.py', 'StringFunction.py', and a subdirectory 'LambdaFunction' containing 'LambdaFunction.py'. The main editor window displays the code in 'LambdaFunction.py':

```
1 square = lambda x: x ** 2
2 result = square(5)
3 print(result)
4
```

The 'Run' console at the bottom shows the execution output: '25' and 'Process finished with exit code 0'. The status bar at the bottom right indicates '3:16 UTF-8 4 spaces' and the date '1/30/2024'.

Lambda with filter() :

In Python, you can use the filter() function along with a lambda function to filter data from a list based on a specified condition. Here's a simple example to illustrate how to filter data in a Python list using filter() and lambda:



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'MapFunction.py', 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameterd.py', 'StringFunction.py', 'Include', 'LambdaFunction', 'Lib', 'ModuleExample', and 'Oops'. The main editor window displays a Python script in 'WithFilter.py' with the following code:

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 filtered_numbers = list(filter(lambda x: x % 2 == 0, numbers))
4
5 print("Original List:", numbers)
6 print("Filtered List (Even Numbers):", filtered_numbers)
7
```

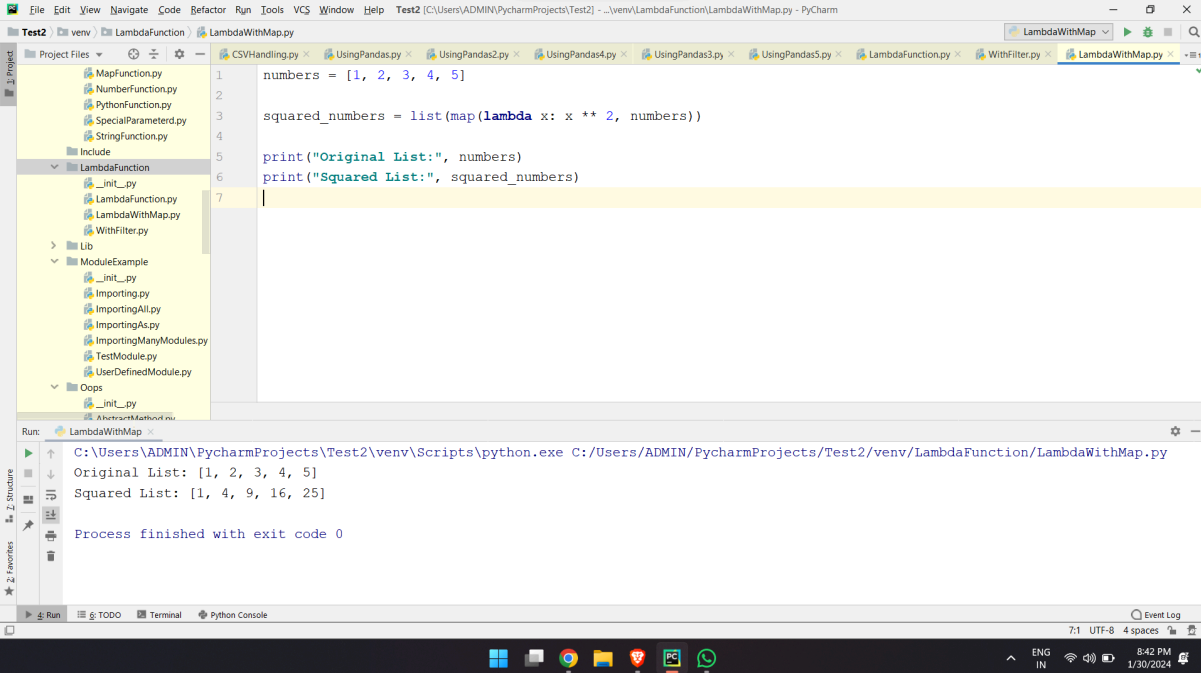
The Run window at the bottom shows the output of the script:

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\Test2\venv\LambdaFunction\WithFilter.py
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered List (Even Numbers): [2, 4, 6, 8, 10]
Process finished with exit code 0
```

The status bar at the bottom indicates the file is 'WithFilter.py', the encoding is 'UTF-8', and the line length is '4 spaces'. The system tray shows the time as 8:40 PM on 1/30/2024.

Lambda with map() :

The map() function in Python is often used in conjunction with lambda functions to apply a specific operation to each item in an iterable. Here's a simple example illustrating the use of map() with a lambda function:



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'MapFunction.py', 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameter.py', 'StringFunction.py', and a 'Lib' directory. The main editor window displays a Python script named 'LambdaWithMap.py' with the following code:

```
1 numbers = [1, 2, 3, 4, 5]
2
3 squared_numbers = list(map(lambda x: x ** 2, numbers))
4
5 print("Original List:", numbers)
6 print("Squared List:", squared_numbers)
7
```

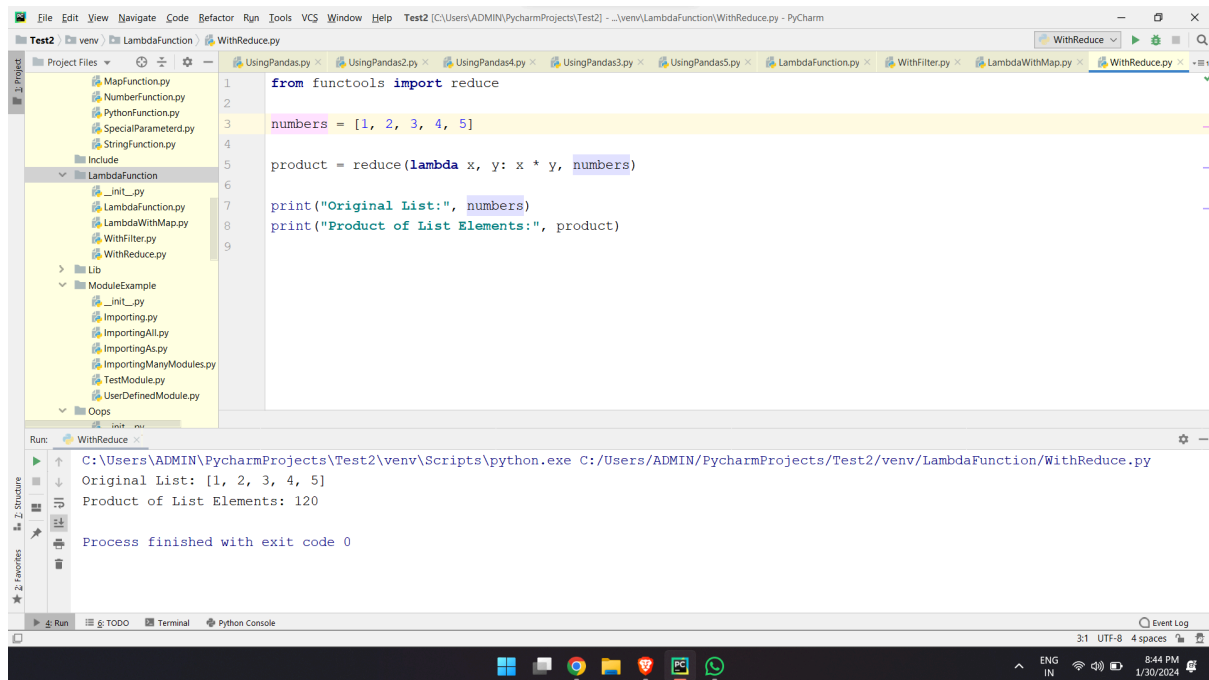
The Run tool window at the bottom shows the execution output:

```
Run: LambdaWithMap
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/LambdaFunction/LambdaWithMap.py
Original List: [1, 2, 3, 4, 5]
Squared List: [1, 4, 9, 16, 25]
Process finished with exit code 0
```

The status bar at the bottom indicates the file is encoded in UTF-8 with 4 spaces, and the system clock shows 8:42 PM on 1/30/2024.

Lambda with reduce() :

The reduce() function is part of the functools module in Python and is often used with a lambda function to successively apply a binary function to the items of an iterable, reducing it to a single accumulated result. Here's an example using reduce() with a lambda function:



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'MapFunction.py', 'NumberFunction.py', 'PythonFunction.py', 'SpecialParameterd.py', 'StringFunction.py', 'Include', 'LambdaFunction', 'LambdaFunction.py', 'LambdaWithMap.py', 'WithFilter.py', 'WithReduce.py', 'Lib', 'ModuleExample', 'ModuleExample.py', 'Importing.py', 'ImportingAll.py', 'ImportingAs.py', 'ImportingManyModules.py', 'TestModule.py', 'UserDefinedModule.py', and 'Oops'. The main editor window displays the following Python code in 'WithReduce.py':

```
1 from functools import reduce
2
3 numbers = [1, 2, 3, 4, 5]
4
5 product = reduce(lambda x, y: x * y, numbers)
6
7 print("Original List:", numbers)
8 print("Product of List Elements:", product)
9
```

The Run window at the bottom shows the execution output:

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\Test2\venv\LambdaFunction\WithReduce.py
Original List: [1, 2, 3, 4, 5]
Product of List Elements: 120
Process finished with exit code 0
```

The status bar at the bottom indicates the file encoding is UTF-8, the line length is 4 spaces, and the time is 8:44 PM on 1/30/2024.