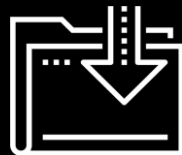




Introduction to Pandas & Jupyter

Data Boot Camp
Lesson 4.1



Class Objectives

By the end of today's class you will be able to:



Serve Jupyter Notebook files from local directories and connect their development environment.



Create Pandas DataFrames from scratch.



Understand how to run functions on Pandas DataFrame.



Read/write DataFrames from/to CSV files using Pandas.

"We set sail on this new sea because there is new knowledge to be gained..."

We choose to go to the Moon!

We choose to go to the Moon...

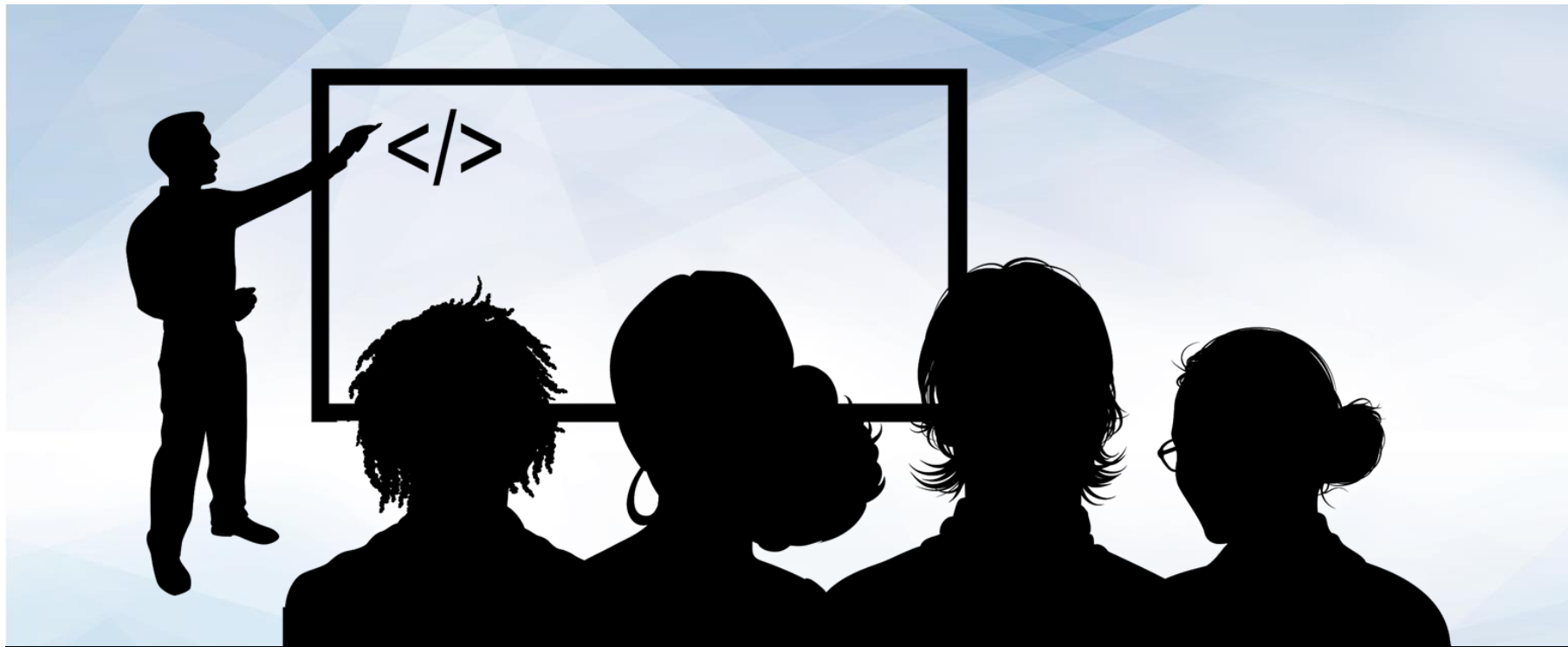
...not because they are easy, but because they are
HARD...

Parts of President John F. Kennedy
Address at the Rice University on the
Nation's Space Effort delivered on September 12, 1962.



A little something
to assist you
throughout your
journey..





Instructor Demonstration

Introduction to Jupyter Notebook

Introduction to Jupyter Notebook

Before diving into Pandas, let's take few notes on Jupyter Notebook



- Jupyter Notebook is an open-source application that allows its users to create documents that contain live code, equations, visualizations, and explanatory text.
- In other words, Jupyter Notebook combines a text editor, the console, and a markdown file into one application.



Introduction to Jupyter Notebook

Before diving into Pandas, let's take few notes on Jupyter Notebook

- Create a Python file with Jupyter Notebook. **Set the kernel as 'PythonData'**
 - Setting the kernel for Jupyter projects is important because these kernels let the program know which libraries will be available for use. Only those libraries loaded into the development environment selected can be used in a Jupyter Notebook project.
 - If the user's development environment does not show up within Jupyter Notebook, simply run the command `conda install -c anaconda nb_conda_kernels` within the terminal so that anaconda environments can be used as kernels.



Introduction to Jupyter Notebook

Before diving into Pandas, let's take few notes on Jupyter Notebook

- Check out the structure of the file in Jupyter Notebook and navigating thru it.
 - Each cell contains Python code which can be run independently by placing the cursor inside a cell and pressing `Shift + Enter`.
 - Jupyter notebooks allow users to both experiment with code in real time, and save it for later.
 - The running order of the cells won't dictate the stored value of the code, but rather which cell ran last.





Activity: Netflix Remix

In this activity, you will create a Jupyter Notebook that performs the same functions as the Netflix activity from last week.

Suggested Time:
Enough



Activity: Netflix Remix

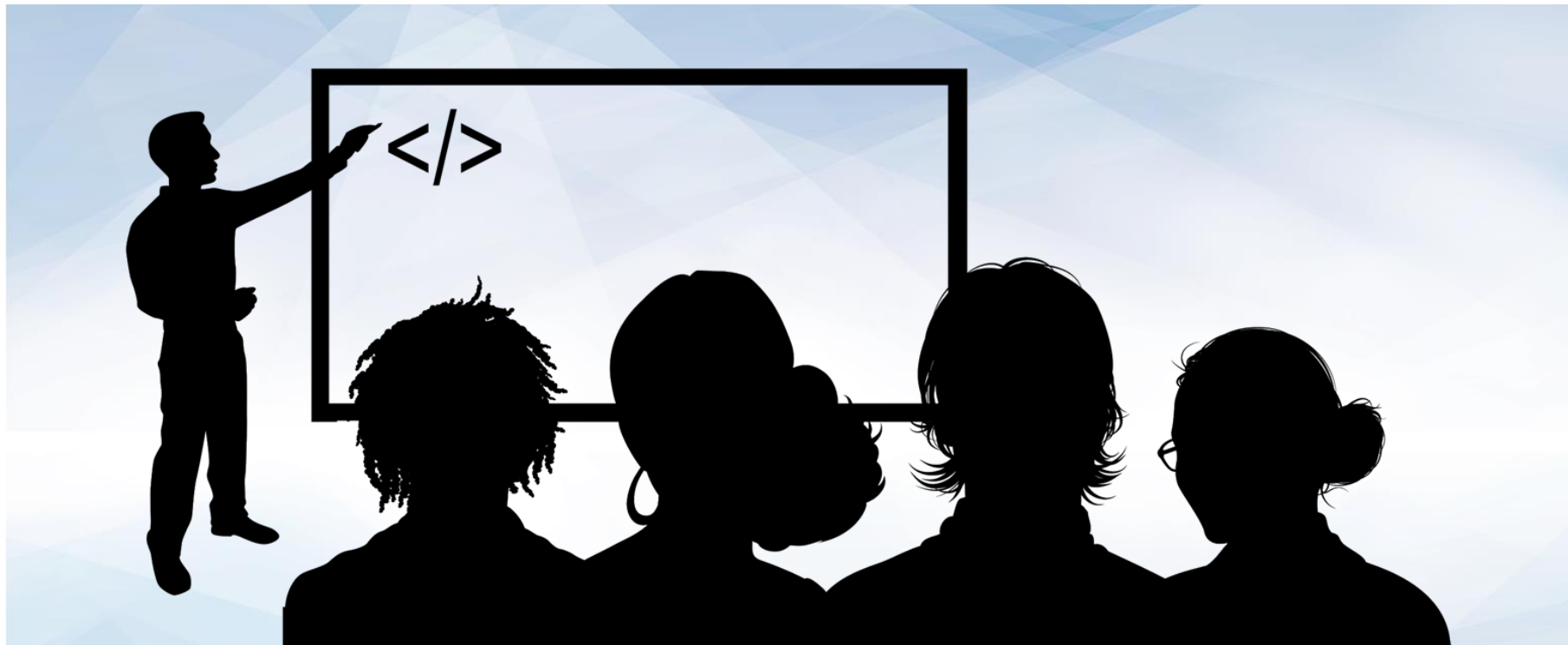
Instructions:

- Using `Netflix.py` as a jumping off point, convert the application so that it runs properly within a Jupyter Notebook.
- Make sure to have the application print out the user's input, the path to `Netflix_Ratings.csv`, and the final rating/review for the film in different cells.
- **Bonus:**
 - Go through any of the activities from last week and attempt to convert them to run within a Jupyter Notebook. While doing this, try to split up the code into cells and print out the outputs.
- **Hints:**
 - If your development environment does not appear as a potential kernel, close Jupyter Notebook and run `conda install -c anaconda nb_conda_kernels` within the terminal. Upon reloading Jupyter Notebook, all possible kernels should now appear.





Time's Up! Let's Review.

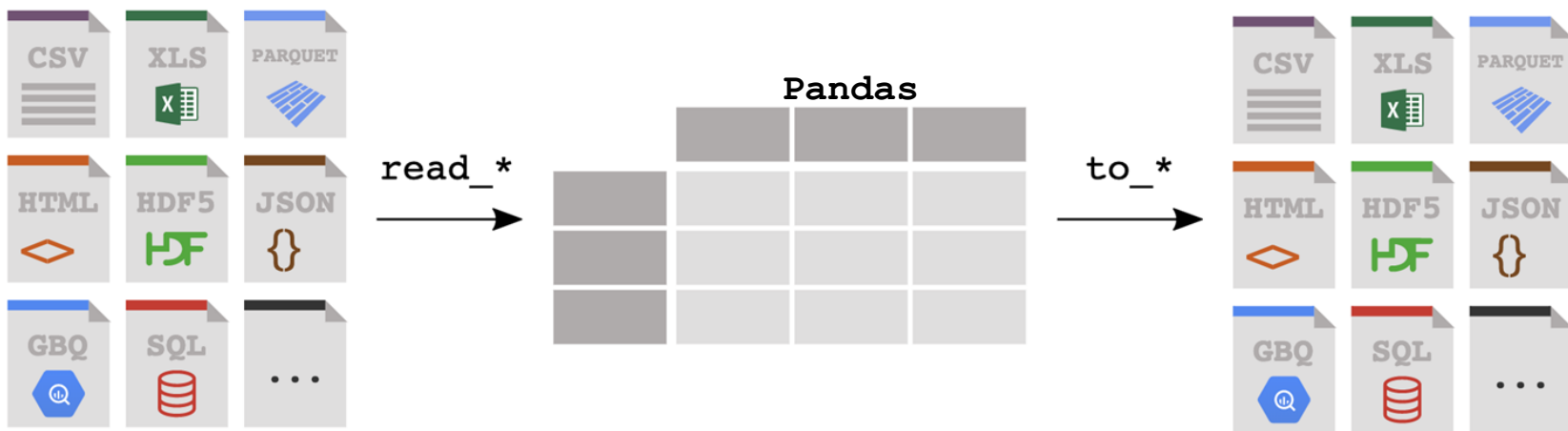


Instructor Demonstration

Introduction to Pandas

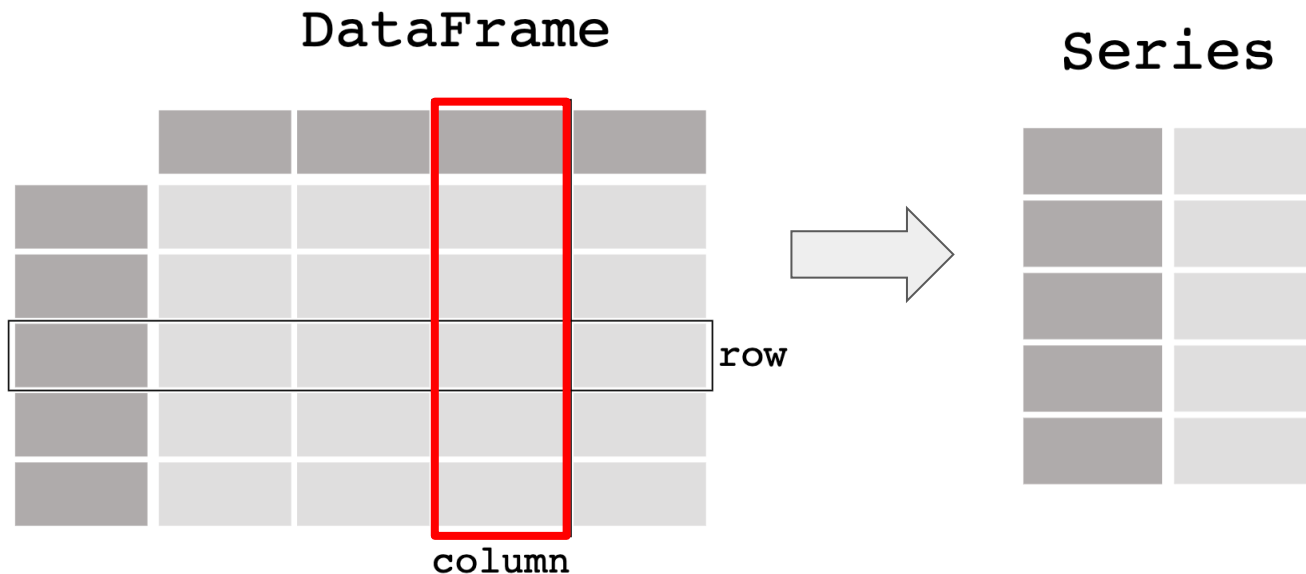
Introduction to Pandas

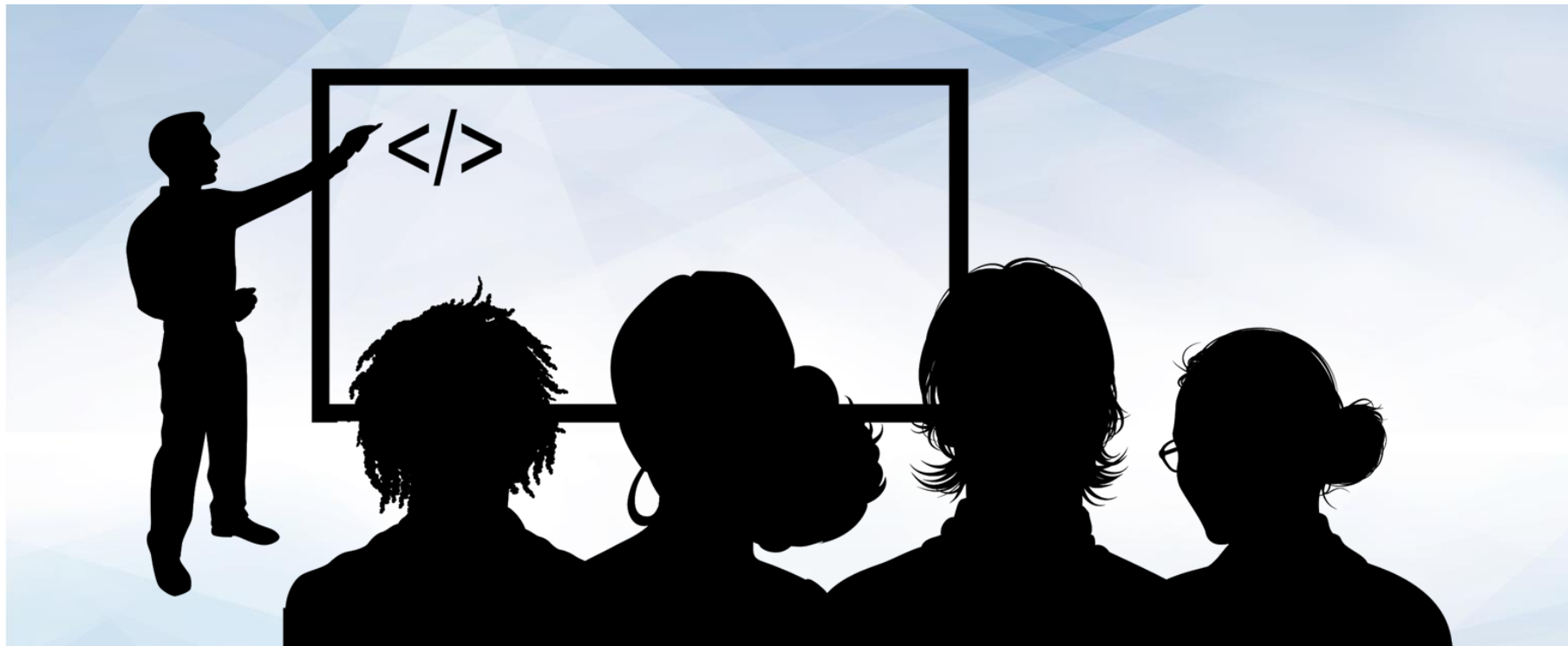
- Thankfully there is a library that, in addition to having a cute and cuddly name, is extraordinarily powerful when it comes to visualizing, analyzing, and altering large datasets. This library is **Pandas**.



Introduction to Pandas

- While Python alone is stuck using lists, tuples, and dictionaries, Pandas lets Python programmers work with "Series" and "DataFrames"
- Each Column in a DataFrame is a Series.





Instructor Demonstration

DataFrame Creation

DataFrame Creation

Some things to note when creating a DataFrame

- First, import the Pandas library by running `import pandas as pd`. This method of import allows Pandas functions/methods to be called using the variable `pd`.
- To create a Series, simply run `pd.Series()` with a list as an argument (within the parentheses). Note that the index for the values within the Series will be the numeric index of the initial list.

DataFrame Creation

Few thing to note when creating a DataFrame

- One of many different ways to create DataFrames from scratch is to use `pd.DataFrame()` and provide it with a list of dictionaries. Each dictionary will represent a new row where the keys become column headers and the values will be placed inside the table.
- Another way to use `pd.DataFrame()` is to provide a dictionary of lists. The keys of the dictionary will be the column headers and the listed values will be placed into their respective rows.



Activity: Data-Frame Shop

In this activity, you will create DataFrames from scratch using the two methods discussed earlier.

Suggested Time:
Several Minutes



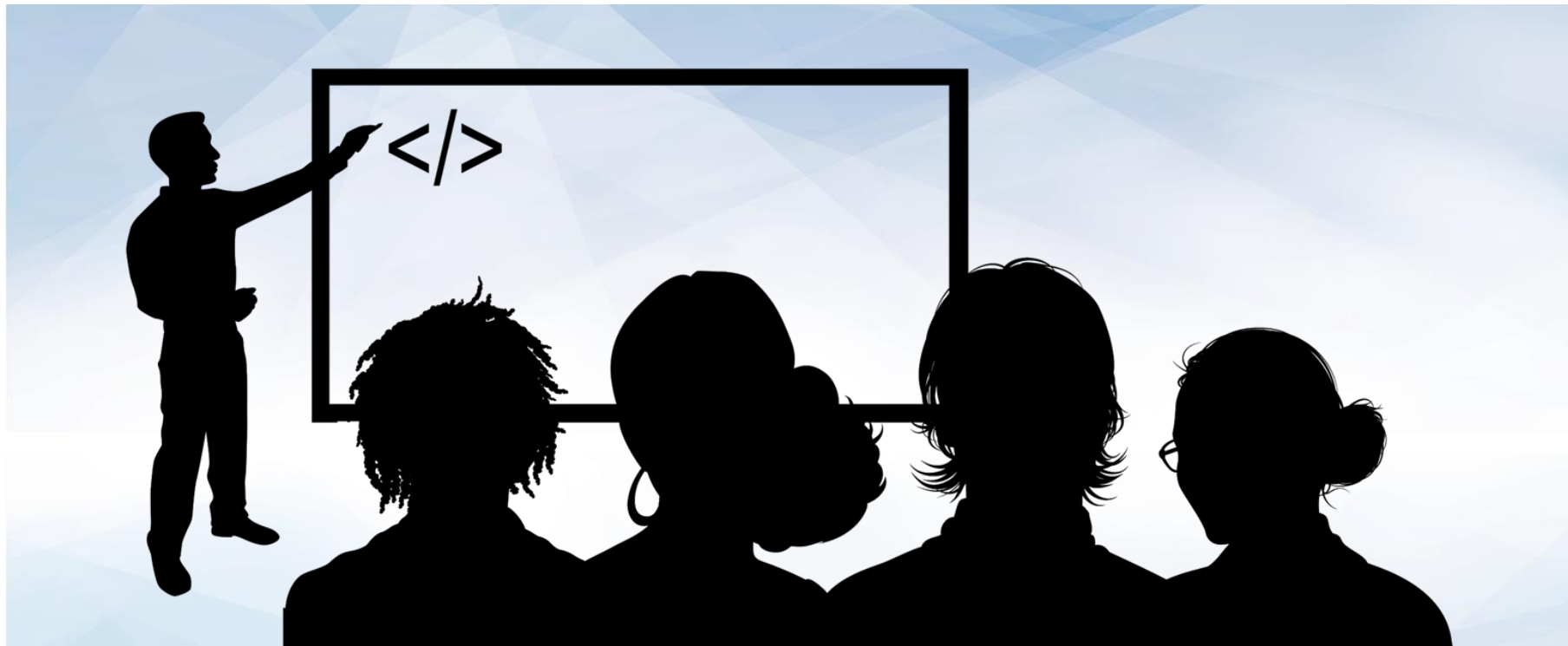
Activity: DataFrame Shop

Instructions:

- Create a DataFrame for a frame shop that contains three columns - "Frame", "Price", and "Sales" - and has five rows of data stored within it.
- Using an alternate method from that used before, create a DataFrame for an art gallery that contains three columns - "Painting", "Price", and "Popularity" - and has four rows of data stored within it.
- **Bonus:**
 - Once both of the DataFrames have been created, discuss with those around you which method you prefer to use and why.



Time's Up! Let's Review.



Instructor Demonstration

DataFrame Functions

DataFrame Functions

Built in Function: `head()`

- The `head()` method allows the programmer to look at a truncated version of a much larger table, thus allowing them to make informed changes without having to search through the entire dataset.

```
In [3]: # Use Pandas to read data
data_file_df = pd.read_csv(data_file)
data_file_df.head()
```

```
Out[3]:
```

| | id | First Name | Last Name | Gender | Amount |
|---|----|------------|-----------|--------|---------|
| 0 | 1 | Todd | Lopez | M | 8067.7 |
| 1 | 2 | Joshua | White | M | 7330.1 |
| 2 | 3 | Mary | Lewis | F | 16335.0 |
| 3 | 4 | Emily | Burns | F | 12460.8 |
| 4 | 5 | Christina | Romero | F | 15271.9 |

DataFrame Functions

Built in Function: `describe()`

- The `describe()` method will print out a DataFrame containing some analytic information on the table and its columns. It is also helpful in showing what other data functions can be performed on a DataFrame or Series.

```
In [4]: # Display a statistical overview of the DataFrame
data_file_df.describe()
```

Out[4]:

| | id | Amount |
|-------|-------------|--------------|
| count | 1000.000000 | 1000.000000 |
| mean | 500.500000 | 10051.323600 |
| std | 288.819436 | 5831.230806 |
| min | 1.000000 | 3.400000 |
| 25% | 250.750000 | 4854.875000 |
| 50% | 500.500000 | 10318.050000 |
| 75% | 750.250000 | 15117.425000 |
| max | 1000.000000 | 19987.400000 |

DataFrame Functions

Working with a single column

- Most data functions can also be performed on a Series by referencing a single column within the whole DataFrame.
- This is done in a similar way to referencing a key from a dictionary using bracket notation.

```
In [5]: # Reference a single column within a DataFrame  
data_file_df["Amount"].head()
```

```
Out[5]: 0      8067.7  
        1      7330.1  
        2     16335.0  
        3     12460.8  
        4     15271.9  
        Name: Amount, dtype: float64
```


DataFrame Functions

Working with multiple columns

- Multiple columns can be referenced as well by placing all of the column headers desired within a pair of *double brackets*.

```
In [6]: # Reference multiple columns within a DataFrame  
data_file_df[["Amount", "Gender"]].head()
```

Out[6]:

| | Amount | Gender |
|---|---------|--------|
| 0 | 8067.7 | M |
| 1 | 7330.1 | M |
| 2 | 16335.0 | F |
| 3 | 12460.8 | F |
| 4 | 15271.9 | F |

DataFrame Functions

Aggregating Functions

- `.mean()` computes the arithmetic mean.
- `.sum()` adds the values.

```
In [7]: # The mean method averages the series  
average = data_file_df["Amount"].mean()  
average
```

```
Out[7]: 10051.323600000002
```

```
In [8]: # The sum method adds every entry in the series  
total = data_file_df["Amount"].sum()  
total
```

```
Out[8]: 10051323.600000001
```

DataFrame Functions

Built in Function: `unique()`

- There are situations in which it is helpful to list out all of the unique values stored within a column. The `unique()` function returns all of the different values within a series.

```
In [9]: # The unique method shows every element of the series that appears only once
unique = data_file_df["Last Name"].unique()
unique
```

```
Out[9]: array(['Lopez', 'White', 'Lewis', 'Burns', 'Romero', 'Andrews', 'Baker',
               'Diaz', 'Burke', 'Richards', 'Hansen', 'Tucker', 'Wheeler',
               'Turner', 'Reynolds', 'Carpenter', 'Scott', 'Ryan', 'Marshall',
               'Fernandez', 'Olson', 'Riley', 'Woods', 'Wells', 'Gutierrez',
               'Harvey', 'Ruiz', 'Lee', 'Welch', 'Cooper', 'Nichols', 'Murray',
               'Gomez', 'Green', 'Jacobs', 'Griffin', 'Perry', 'Dunn', 'Gardner',
               'Gray', 'Walker', 'Harris', 'Lawrence', 'Black', 'Simpson', 'Sims',
               'Weaver', 'Carr', 'Owens', 'Stephens', 'Butler', 'Matthews', 'Cox',
               'Brooks', 'Austin', 'Moore', 'Hunter', 'Cunningham', 'Lane',
               'Montgomery', 'Vasquez', 'Freeman', 'Hernandez', 'Alexander',
               'Pierce', 'Mcdonald', 'Kelly', 'Foster', 'Bell', 'Johnson',
               'Bowman', 'Porter', 'Wood', 'Reid', 'Willis', 'Bishop',
               'Washington', 'Gonzales', 'Davis', 'Martinez', 'Martin', 'Long',
               'Howell', 'Hawkins', 'Knight', 'Price', 'Day', 'Bailey', 'Flores',
               'Young', 'Evans', 'Cruz', 'Chavez', 'Barnes', 'Coleman', 'Burton',
               'Clark', 'Carter', 'Franklin', 'Ellis', 'Miller', 'Allen', 'Mason',
               'Patterson', 'Stevens', 'Kim', 'Kelley', 'Robinson', 'Hughes',
               'Morgan', 'Dean', 'Stewart', 'Murphy', 'Fox', 'Simmons',
               'Thompson', 'Fuller', 'Peterson', 'Hanson', 'Wright', 'Reed',
               'Graham', 'Parker', 'Boyd', 'Taylor', 'Greene', 'George', 'Mills',
               'Duncan', 'Hill', 'Jordan', 'Stanley', 'Hall', 'James', 'Stone',
               'Warren', 'Fowler', 'Williamson', 'Lynch', 'Harper', 'Little',
               'Nguyen', 'Morrison', 'Ramirez', 'Howard', 'Watkins', 'Robertson',
               'Powell', 'Sanchez', 'Sanders', 'Grant', 'Ross', 'Mitchell',
               'Henderson', 'Rose', 'Perez', 'Berry', 'Watson', 'Gordon',
               'Morales', 'Arnold', 'Morris', 'Crawford', 'Smith', 'Medina',
               'Alvarez', 'Collins', 'Rodriguez', 'Mccoy', 'Bennett',
               'Richardson', 'Chapman', 'Johnston', 'Gilbert', 'Ford', 'Russell',
               'Nelson', 'Castillo', 'Cole', 'Rice', 'Payne', 'Frazier', 'Webb',
               'Armstrong', 'Wilson', 'Garza', 'Garrett', 'Spencer', 'Peters',
               'Sullivan', 'Brown', 'Williams', 'Gonzalez', 'Palmer', 'Fields',
               'Snyder', 'Jackson', 'Edwards', 'Anderson', 'Cook', 'Ramos',
               'Harrison', 'Lawson', 'Banks', 'Wallace', 'Ortiz', 'Gibson',
               'Reyes', 'Shaw', 'Ward', 'Perkins', 'Bradley', 'Rivera', 'Jenkins',
               'Hart', 'Phillips', 'Garcia', 'Fisher', 'King', 'Larson', 'Hunt',
               'Jones', 'Hudson', 'Myers', 'Hayes', 'Dixon', 'Schmidt', 'Moreno',
               'Rogers', 'Thomas', 'Meyer', 'Daniels', 'Bryant', 'Henry',
               'Campbell', 'Ferguson', 'Oliver', 'Ray', 'Carroll', 'Wagner',
               'Kennedy', 'Holmes'], dtype=object)
```

DataFrame Functions

Built in Function: `value_counts()`

- Another method that holds similar functionality is that of `value_counts()` which not only returns a list of all unique values within a series but also counts how many times a value appears.

```
In [10]: # The value_counts method counts unique values in a column  
count = data_file_df["Gender"].value_counts()  
count
```

```
Out[10]: M      515  
        F      485  
        Name: Gender, dtype: int64
```

DataFrame Functions

Beyond Pandas Visualization Power...

- Calculations can also be performed on columns and then added back into a DataFrame as a new column by referencing the DataFrame, placing the desired column header within brackets, and then setting it equal to a Series.

```
In [11]: # Calculations can also be performed on Series and added into DataFrames as new columns
thousands_of_dollars = data_file_df["Amount"]/1000
data_file_df["Thousands of Dollars"] = thousands_of_dollars

data_file_df.head()
```

```
Out[11]:
```

| | id | First Name | Last Name | Gender | Amount | Thousands of Dollars |
|---|----|------------|-----------|--------|---------|----------------------|
| 0 | 1 | Todd | Lopez | M | 8067.7 | 8.0677 |
| 1 | 2 | Joshua | White | M | 7330.1 | 7.3301 |
| 2 | 3 | Mary | Lewis | F | 16335.0 | 16.3350 |
| 3 | 4 | Emily | Burns | F | 12460.8 | 12.4608 |
| 4 | 5 | Christina | Romero | F | 15271.9 | 15.2719 |



Activity: Training Grounds

In this activity, you will now take a DataFrame consisting of 200 rows, analyze it using some data functions, and then add a new column into it.

Suggested Time:
Certainly less than an hour



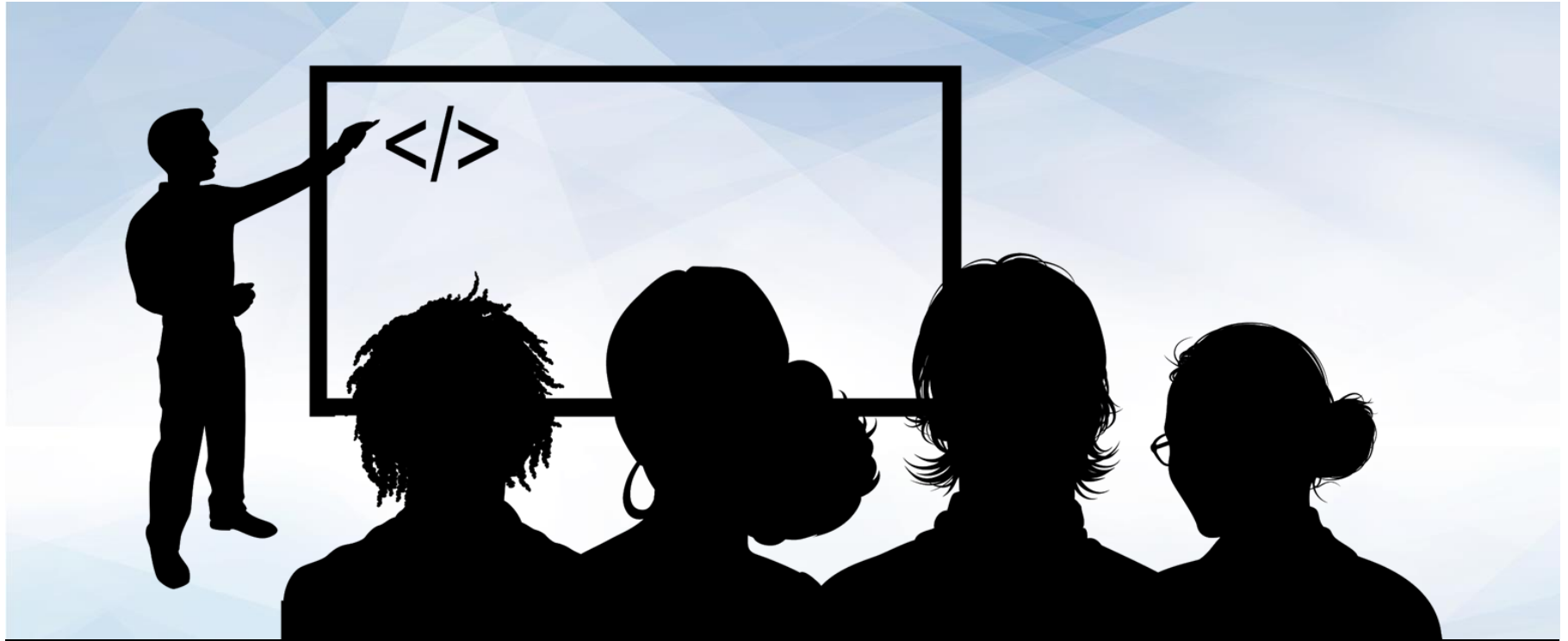
Activity: Training Grounds

Instructions:

- Using the DataFrame provided, perform all of the following actions...
- Provide a simple, analytical overview of the dataset's numeric columns.
- Collect all of the names of the trainers within the dataset.
- Figure out how many students each trainer has.
- Find the average weight of the students at the gym.
- Find the combined weight of all of the students at the gym.
- Convert the "Membership (Days)" column into weeks and then add this new series into the DataFrame.



Time's Up! Let's Review.



Instructor Demonstration

Modifying Columns

Modifying Columns

Columns Manipulation

- You can modify the names/placement of columns using `rename()`
- Use `df.columns` to print the column headers.

```
In [3]: # Collecting a list of all columns within the DataFrame  
training_df.columns
```

```
Out[3]: Index(['Membership(Days)', 'Name', 'Trainer', 'Weight'], dtype='object')
```

Modifying Columns

Columns Manipulation

- To reorder the columns, create a reference to the DataFrame followed by two brackets with the column headers placed in the order desired.
- You can also remove columns in this way by omitting them. This will, in essence, drop them from the newly made DataFrame.

```
In [4]: # Reorganizing the columns using double brackets
organized_df = training_df[["Name", "Trainer", "Weight", "Membership(Days)"]]
organized_df.head()
```

Out[4]:

| | Name | Trainer | Weight | Membership(Days) |
|---|-------------------|-----------------|--------|------------------|
| 0 | Gino Walker | Bettyann Savory | 128 | 52 |
| 1 | Hiedi Wasser | Mariah Barberio | 180 | 70 |
| 2 | Kerrie Wetzel | Gordon Perrine | 193 | 148 |
| 3 | Elizabeth Sackett | Pa Dargan | 177 | 124 |
| 4 | Jack Mitten | Blanch Victoria | 237 | 186 |

Modifying Columns

Columns Manipulation

- To rename the columns within a DataFrame, use the `df.rename()` method and place `columns={}` within the parentheses.
- Inside of the dictionary, the keys should be references to the current columns and the values should be the desired column names.

```
In [5]: # Using .rename(columns={}) in order to rename columns
renamed_df = organized_df.rename(columns={"Membership(Days)": "Membership in Days", "Weight": "Weight in Pounds"})
renamed_df.head()
```

Out[5]:

| | Name | Trainer | Weight in Pounds | Membership in Days |
|---|-------------------|-----------------|------------------|--------------------|
| 0 | Gino Walker | Bettyann Savory | 128 | 52 |
| 1 | Hiedi Wasser | Mariah Barberio | 180 | 70 |
| 2 | Kerrie Wetzel | Gordon Perrine | 193 | 148 |
| 3 | Elizabeth Sackett | Pa Dargan | 177 | 124 |
| 4 | Jack Mitten | Blanch Victoria | 237 | 186 |



Activity: Hey Arnold!

In this activity, you will be taking a pre-made DataFrame of “Hey Arnold!” characters and reorganizing it so that it is more understandable and organized.

Suggested Time:
A few minutes



Activity: Hey Arnold!

Instructions:

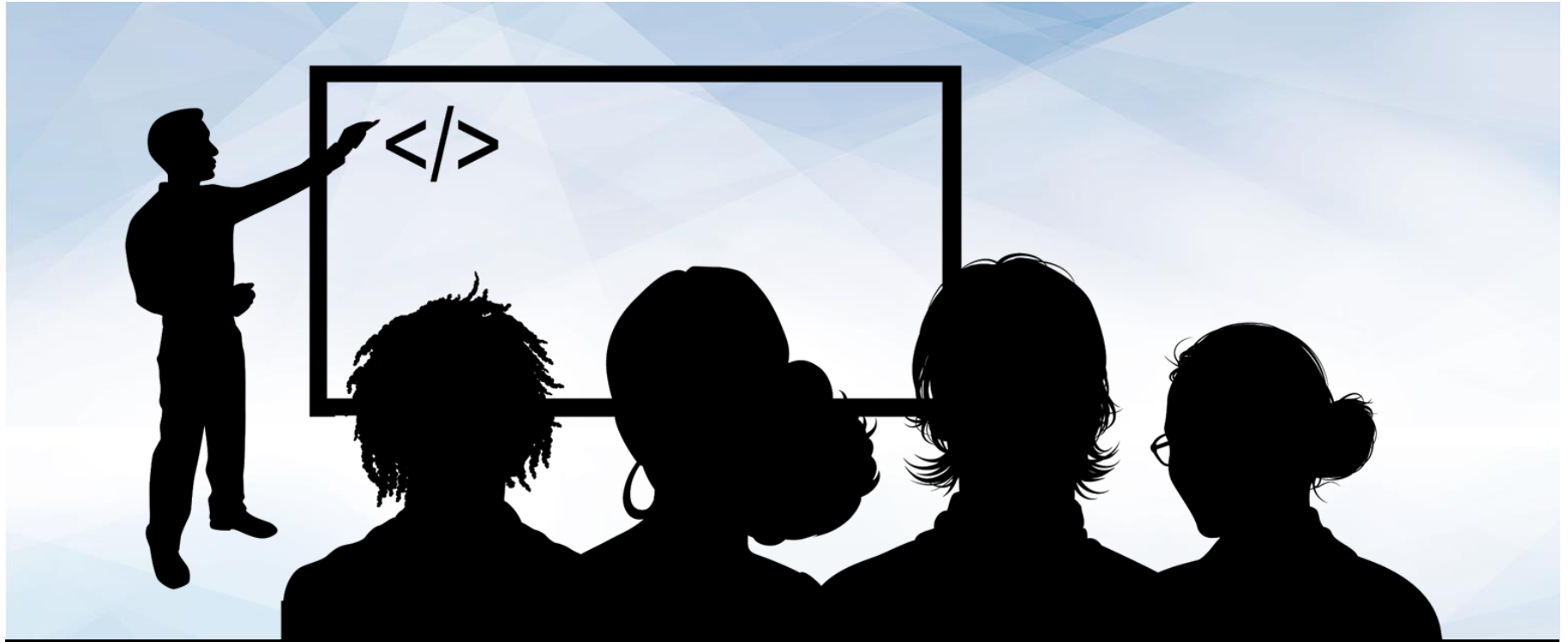
- First, use Pandas to create a DataFrame with the following columns and values:
 - `Characters_in_show` : Arnold, Gerald, Helga, Phoebe, Harold, Eugene.
 - `color_of_hair` : blonde, black, blonde, black, unknown, red.
 - `Height` : average, tallish, tallish, short, tall, short.
 - `Football_Shaped_Head` : True, False, False, False, False, False.
- You'll note that the above column names are inconsistent and difficult to work with. Rename them to the following, respectively:
 - `Character`, `Hair Color`, `Height`, `Football Head`
- Next, create a new table that contains all of the columns in the following order...
 - `Character`, `Football Head`, `Hair Color`, `Height`



Time's Up! Let's Review.



Break



Instructor Demonstration

Reading and Writing CSV Files

Reading and Writing CSV Files

A Far More Effective Way...

- A CSV file's path can be created and passed into the `pd.read_csv()` method.
- The return value is the DataFrame represented by the data in the csv.

```
In [3]: # Read our Data file with the pandas library  
# Not every CSV requires an encoding, but be aware this can come up  
file_one_df = pd.read_csv(file_one, encoding="ISO-8859-1")
```

```
In [4]: # Show just the header  
file_one_df.head()
```

Out[4]:

| | id | first_name | last_name | email | gender |
|---|----|------------|-----------|-------------------------|--------|
| 0 | 1 | David | Jordan | djordan0@home.pl | Male |
| 1 | 2 | Stephen | Riley | sriley1@hugedomains.com | Male |
| 2 | 3 | Evelyn | Grant | egrant2@livejournal.com | Female |
| 3 | 4 | Joe | Mendoza | jmendoza3@un.org | Male |
| 4 | 5 | Benjamin | Rodriguez | brodriguez4@elpais.com | Male |

Reading and Writing CSV Files

A Far More Effective Way...

- To write to a CSV file, use `df.to_csv()`, passing the path to the desired output file. By using the `index` and `header` parameters, you can indicate whether you would like the index or header for the table to be passed as well.

```
In [8]: # Export file as a CSV, without the Pandas index, but with the header  
file_one_df.to_csv("Output/fileOne.csv", index=False, header=True)
```



Activity: GoodReads Part I

In this activity, you will take a large CSV of books, read it into Jupyter Notebook using Pandas, clean up the columns, and then write their modified DataFrame to a new CSV file.

Suggested Time:
Minutes



Activity: GoodReads Part I

Instructions:

- Read in the GoodReads CSV using Pandas.
- Remove unnecessary columns from the DataFrame so that only the following columns remain: isbn, original_publication_year, original_title, authors, ratings_1, ratings_2, ratings_3, ratings_4, and ratings_5
- Rename the columns to the following: ISBN, Publication Year, Original Title, Authors, One Star Reviews, Two Star Reviews, Three Star Reviews, Four Star Reviews, and Five Star Reviews
- Write the DataFrame into a new CSV file.

- **Hints:**



- The base CSV file uses UTF-8 encoding. Trying to read in the file using some other kind of encoding could lead to strange characters appearing within the dataset.



Time's Up! Let's Review.



Activity: GoodReads - Part II

In this activity, you will take the modified version of the GoodReads DataFrame and create a new summary DataFrame based upon that dataset using some of Pandas's built-in data functions.

Suggested Time:
More Minutes



Activity: GoodReads Part II

Instructions:

- Using the modified DataFrame that was created earlier, create a summary table for the dataset that includes the following pieces of information...
- The count of unique authors within the DataFrame.
- The year of the earliest published book in the DataFrame.
- The year of the latest published book in the DataFrame.
- The total number of reviews within the DataFrame.



Time's Up! Let's Review.

*The
End*