Reading and Writing to Files

Why is this important?

CSVs are really common in data science and data analysis. They allow us to store data in a structured format that can be easily read and written by many different programs. Being able to read and write CSV files is a fundamental skill that every programmer should have.

What are we going to do?

We're going to read csv files to analyze a bit of data.

For this example we're going to use a dataset of NHL teams and their statistics.

Full credit to MoneyPuck for making this dataset available. PS. there's a lot of other really cool datasets on that site, and other datasets for other sports.

Steps

1. Let's open our "NHL Teams" CSV file and read the contents.

CSV files are a common format for storing tabular data. They are easy to read and write in Python using the `csv` module.

I'm sure you folks have used CSV files in excel before or google sheets, so you understand the format.

Let's open this with python so that we can read the data.

```
import csv

def read_nhl_teams(file_path):
    """Read the NHL teams data from a CSV file."""
    all_teams = []
    try:
        with open(file_path, 'r') as file:
            reader = csv.DictReader(file)

            for team in reader:
                all_teams.append(team)
    except FileNotFoundError:
        print(f"Error: The file {file_path} does not exist.")
    return all_teams
```

You can see the that file reading is very similar to the previous example, but we use the `csv.DictReader` to read the CSV file. This allows us to read each row as a dictionary, where the keys are the column headers.

2. Let's create the `__main__` function to read the NHL teams and print them.

```
import csv

from pprint import pprint # this allows us to print dictionaries in a more
readable format



# ... read_nhl_teams function ...

def main():
    file_path = 'data/nhl_teams_data_2024_2025.csv'
    teams = read_nhl_teams(file_path)
    print(f"NHL Teams from {file_path}:\n")
    pprint(teams)

if __name__ == "__main__":
    main()
```

This code reads the NHL teams data from the CSV file and prints it in a more readable format using `pprint`.

Let's run this code and see the output, you can observe that it prints a list of dictionaries, where each dictionary represents a team and its statistics.

3. Let's write a function that will sort the data by a specific column and print out the top 5 teams of that column with their statistics.

Here we're going to learn about a the `sorted()` function and how to use it to sort a list of dictionaries by a specific key.

```
def sort_teams_by_stat(teams, column, reverse=True, top_n=5):
    """Sort the teams by a specific statistic and return the top N
teams."""

    # a special function to get the value of the column we want to sort by
    def get_score(entry):
        return entry[column]

    sorted_teams = sorted(teams, key=get_score, reverse=reverse)
    return sorted_teams[:top_n]
```

Let's break down the code:

- We define a function `sort_teams_by_stat` that takes a list of teams, a column to sort by, a boolean `reverse` to indicate if we want to sort in descending order, and an integer `top_n` to specify how many teams we want to return.

- We define a helper function `get_score` that takes an entry (a dictionary) and returns the value of the column we want to sort by.
- We use the `sorted()` function to sort the teams by the specified column using the `key` parameter to specify the sorting function. The `reverse` parameter is used to sort in descending order.
- Finally, we return the top N teams by slicing the sorted list.

4. Let's modify the `__main__` function to sort the teams by a specific statistic and print the top 5 teams.

```
def main():
    file_path = 'data/nhl_teams_data_2024_2025.csv'
    teams = read_nhl_teams(file_path)
    COLUMN = 'xGoalsFor'

    # Sort the teams by points and print the top 5 teams
    top_teams = sort_teams_by_stat(teams, COLUMN)
    print(f"Top 5 NHL Teams by {COLUMN}:\n")
    for index, team in enumerate(top_teams):
        rank = index + 1
        print(f"{rank}. {team['team']} - {team[COLUMN]}")
```

Let's break this down.

- We define a constant `COLUMN` to specify the statistic we want to sort by (in this case, 'xGoalsFor').
- We call the `sort_teams_by_stat` function to get the top 5 teams by the specified statistic.
- We print the top 5 teams with their statistics.

5. Let's make this a bit more interactive and allow the user to choose which statistic they want to sort by.

Let's modify the `__main__` function to allow the user to choose which statistic they want to sort by.

```
# ... imports and other functions ...

def main():
    file_path = 'data/nhl_teams_data_2024_2025.csv'
    teams = read_nhl_teams(file_path)

    # Get the column names from the first team
    columns = list(teams[0].keys())
    print("Available statistics to sort by:")
    for index, column in enumerate(columns):
        print(f"{index + 1}. {column}")

    choice = int(input("Enter the number of the statistic you want to sort
by: ")) - 1
    COLUMN = columns[choice]

    # Sort the teams by the chosen statistic and print the top 5 teams
```

```
    top_teams = sort_teams_by_stat(teams, COLUMN)
    print(f"\nTop 5 NHL Teams by {COLUMN}:\n")
    for index, team in enumerate(top_teams):
        rank = index + 1
        print(f"{rank}. {team['team']} - {team[COLUMN]}")
```

Now when you run the program you can see the available statistics to sort by, and you can choose which one you want to sort by.

6. Let's talk about `lambda` functions and how they can be used to simplify our sorting function.

We can simplify our sorting function by using a `lambda` function instead of defining a separate `get_score` function. A `lambda` function is an anonymous function that can take any number of arguments but can only have one expression.

Let's modify the `sort_teams_by_stat` function to use a `lambda` function.

```
def sort_teams_by_stat(teams, column, reverse=True, top_n=5):
    """Sort the teams by a specific statistic and return the top N
teams."""

    # changed the get_score function to a lambda function
    sorted_teams = sorted(teams, key=lambda entry: entry[column],
reverse=reverse)

    data = []
    for team in sorted_teams:
        data.append({
            'Team': team['team'],
            column: team[column],
        })

    return sorted_teams[:top_n]
```

So in the above code we replaced:

```
    def get_score(entry):
        return entry[column]
```

with a `lambda` function that does the same thing:

```
    # inside of the sorted function
    lambda entry: entry[column]
```

Lambda functions are useful for short, throwaway functions that you don't need to reuse elsewhere. These are used alot in Python, especially in functional programming contexts like sorting, filtering, and mapping type of operations.

Challenge

1. Go to the MoneyPuck NHL Data and download the NHL player data for the season.
2. Open the CSV file and read the data using the `csv` module.
3. Write a function that sorts the players by a specific statistic (e.g., goals, assists, points) and returns the top 10 players. Filter only the data for 5on5 play.

Conclusion

Reading and writing CSV files is a fundamental skill in data analysis and programming. It allows us to work with structured data in a simple and efficient way. In this lesson, we learned how to read CSV files, sort the data by specific statistics, and make our code more interactive by allowing user input. We also explored the use of `lambda` functions to simplify our code.