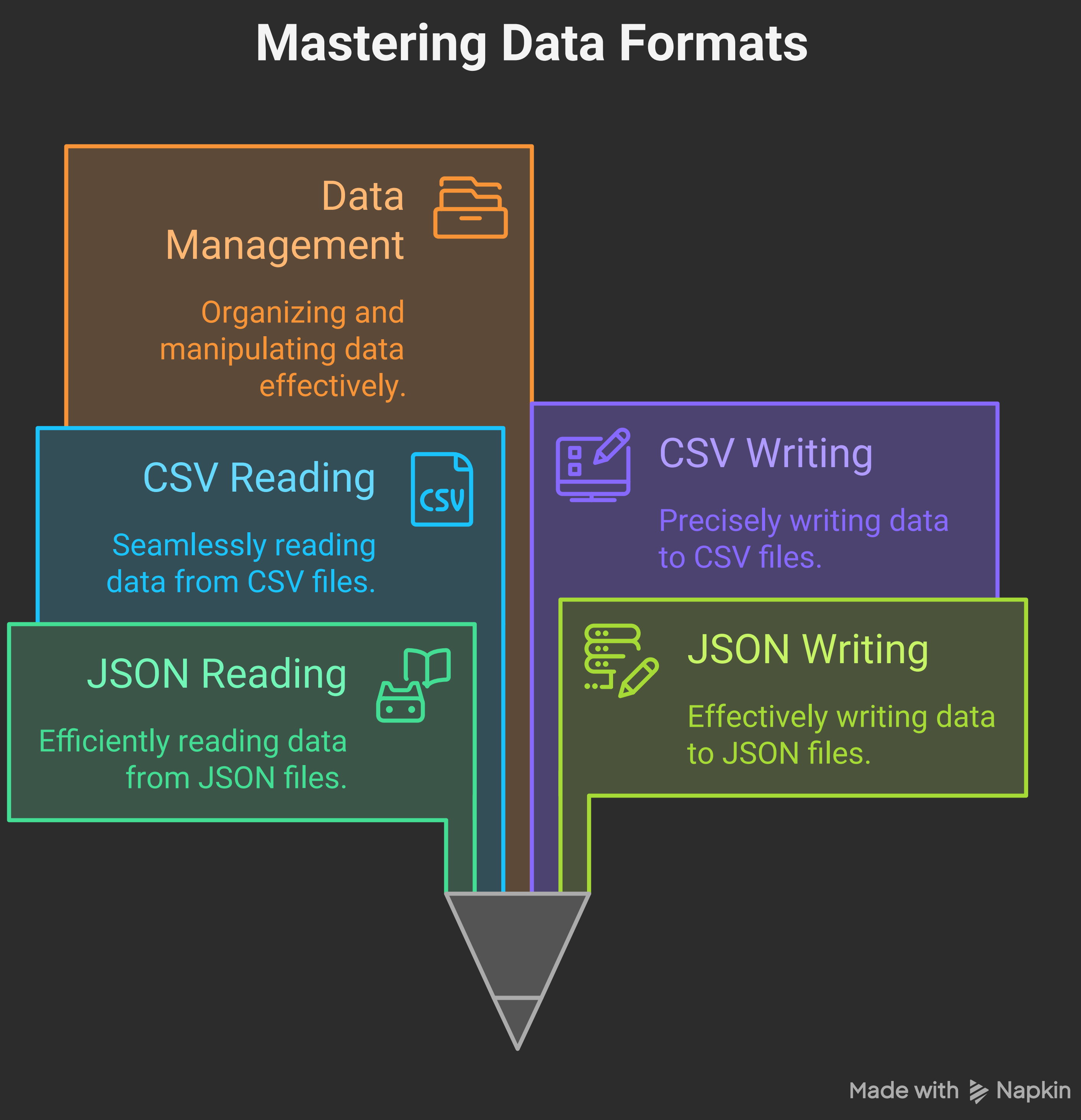
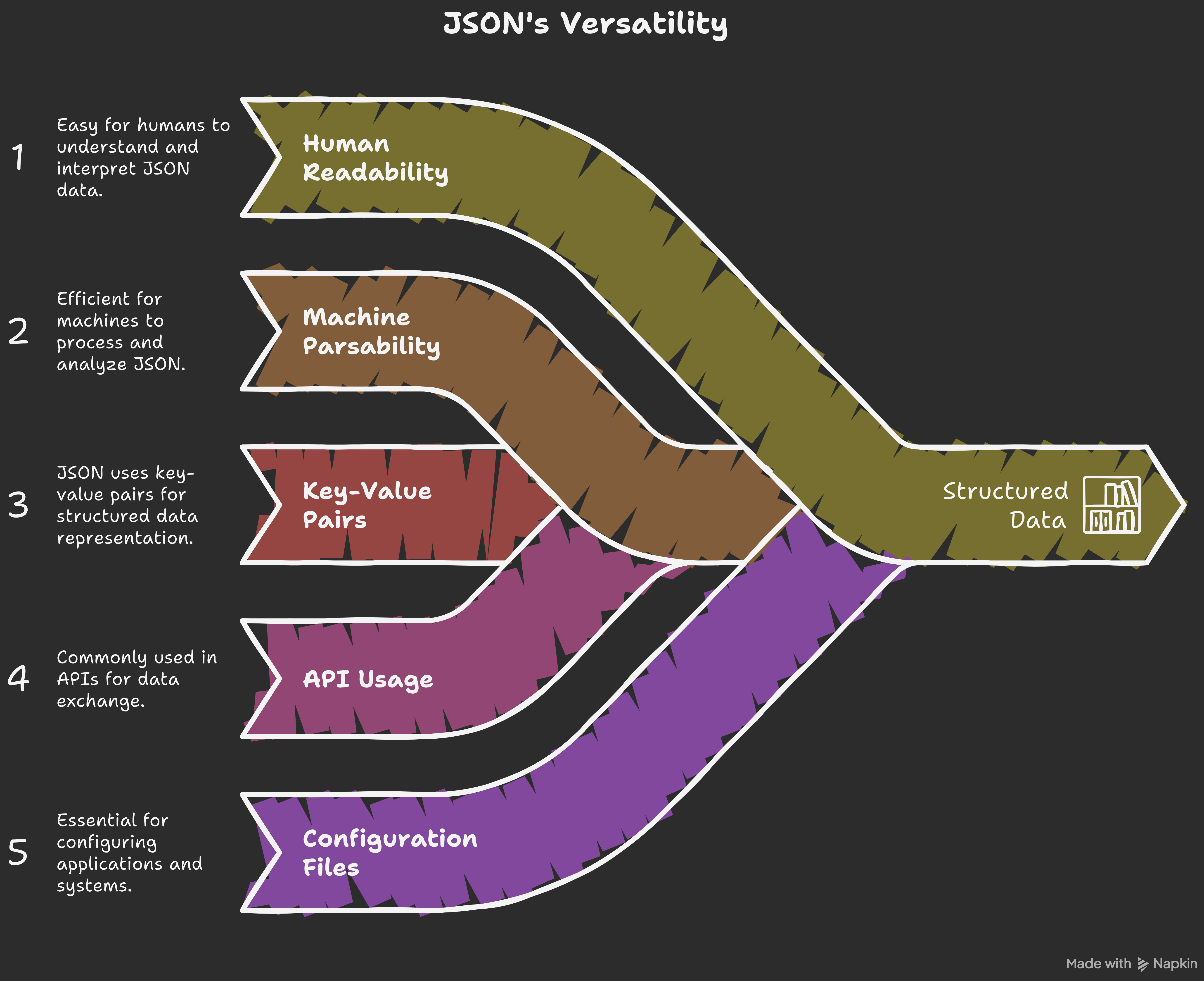
DAY-02: JSON & CSV in Python


Mastering Data Formats
Data Management
Organizing and manipulating data effectively.
CSV Reading
Seamlessly reading data from CSV files.
JSON Reading
Efficiently reading data from JSON files.
CSV Writing
Precisely writing data to CSV files.
JSON Writing
Effectively writing data to JSON files.
Made with Napkin

Introduction to JSON and CSV

What is JSON?

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is commonly used for APIs and configuration files. JSON data is represented in key-value pairs, making it a versatile format for structured data.


JSON's Versatility
1
Easy for humans to understand and interpret JSON data.
Human Readability
2
Efficient for machines to process and analyze JSON.
Machine Parsability
3
JSON uses key-value pairs for structured data representation.
Key-Value Pairs
4
Commonly used in APIs for data exchange.
API Usage
5
Essential for configuring applications and systems.
Configuration Files
Structured Data
Made with Napkin

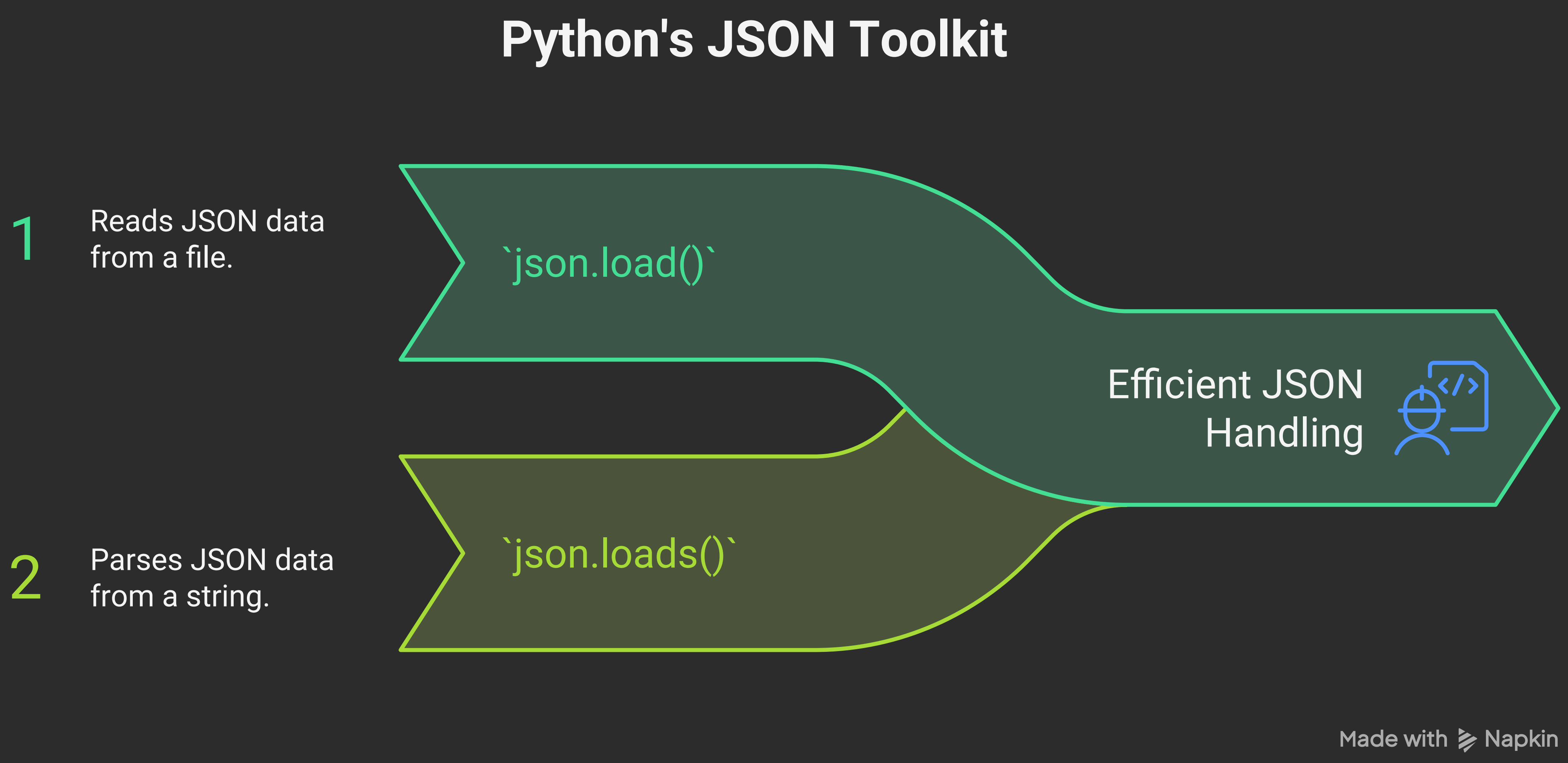
What is CSV?

CSV (Comma-Separated Values) is a simple file format used to store tabular data, such as spreadsheets or databases. Each line in a CSV file corresponds to a row in the table, and each value is separated by a comma. CSV files are widely used for data exchange due to their simplicity and compatibility with various applications.

Working with JSON in Python

Reading JSON Files

To read JSON files in Python, we can use the built-in json library. Here's how to do it:



```
import json
```

with open('data.json', 'r') as file:

```
data = json.load(file)
```

print(data)

```
In this example, we open a JSON file named `data.json`, read its contents, and
parse it into a Python dictionary.
```

Writing JSON Files

To write data to a JSON file, we can also use the json library. Here's an example:

```
import json
```

data = {

```
"name": "John Doe",
```

```
"age": 30,
```

```
"city": "New York"
```

with open('output.json', 'w') as file:

```
json.dump(data, file, indent=4)
```

print("Data written to output.json")

```
In this code snippet, we create a dictionary and write it to a file named
`output.json` with indentation for better readability.
```

Handling JSON Errors

When working with JSON, it's important to handle potential errors, such as file not found or JSON decoding errors. Here's how you can do that:

```
import json

try:
    with open('data.json', 'r') as file:
        data = json.load(file)
except FileNotFoundError:
    print("The file was not found.")
except json.JSONDecodeError:
    print("Error decoding JSON.")
```

This code snippet demonstrates how to catch common exceptions when reading a JSON file.

Working with CSV in Python

Reading CSV Files

Python provides the csv module to handle CSV files. Here's how to read a CSV file:

```
import csv
```

with open('data.csv', 'r') as file:

```
reader = csv.reader(file)
```

```
for row in reader:
```

```
    print(row)
```

```
In this example, we open a CSV file named `data.csv` and read its contents line by line.
```

Writing CSV Files

To write data to a CSV file, we can use the csv module as well. Here's an example:

```
import csv
```

data = [

```
["name", "age", "city"],
```

```
["John Doe", 30, "New York"],
```

```
["Jane Smith", 25, "Los Angeles"]
```

]

with open('output.csv', 'w', newline='') as file:

```
writer = csv.writer(file)
```

```
writer.writerows(data)
```

print("Data written to output.csv")

```
In this code snippet, we create a list of lists and write it to a file named `output.csv`.
```

Handling CSV Errors

Similar to JSON, it's important to handle errors when working with CSV files. Here's an example:

```
import csv

try:
    with open('data.csv', 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            print(row)
except FileNotFoundError:
    print("The file was not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

This code snippet demonstrates how to catch common exceptions when reading a CSV file.

Conclusion

In this document, we have covered the basics of handling JSON and CSV files in Python. We explored how to read and write data using the built-in json and csv libraries, along with error handling techniques. Mastering these skills will enable you to effectively manage real-world data files and integrate them into your Python applications. As you continue to work with data, you will find these formats to be invaluable tools in your programming toolkit.