CS6.201: Introduction to Software Systems
Python Session - 1
`conda`, `pip`, File Handling in Python

Prakhar Jain, Neel Amrutia, Amey Karan, Kevin Thakkar, Rahul Singal

February 19, 2025

Laptops Off!

conda

conda Commands

conda is a package, environment, and dependency manager for Anaconda and Miniconda. Below are essential commands categorized for convenience.

Environment Management with Conda

- **Create a new environment:** `conda create --name <env_name>`
  Example: `conda create --name myenv`
- **Activate an environment:** `conda activate <env_name>`
  Example: `conda activate myenv`
- **Deactivate an environment:** `conda deactivate`
- **List all environments:** `conda env list`
- **Remove an environment:** `conda remove --name <env_name> --all`

Package Management with Conda

- **Install a package:** `conda install <package_name>`
  Example: `conda install numpy`
- **Install a specific version of a package:** `conda install <package_name>=<version>`
  Example: `conda install pandas=1.3.3`
- **Uninstall a package:** `conda remove <package_name>`
  Example: `conda remove scipy`
- **Update a package:** `conda update <package_name>`
  Example: `conda update matplotlib`
- **Update all packages:** `conda update --all`
- **List installed packages:** `conda list`

Clearing Space with Conda

- **Remove an unused environment:** `conda remove --name <env_name> --all`
- **Uninstall specific packages:** `conda remove <package_name>`
- **Clean unused files and caches:** `conda clean --all`
- **Clean specific components:**
  - **Package cache only:** `conda clean --packages`
  - **Tarballs only:** `conda clean --tarballs`
  - **Log files:** `conda clean --logfiles`

Other Useful Conda Commands

- **Get help:** : `conda --help` or `conda -h`
- **Check conda version:** `conda --version`
- **Search for a package:** `conda search <package_name>`
  Example: `conda search requests`

Questions?

pip

pip Commands

pip is a package manager for Python packages. Below are the essential commands categorized for convenience.

Basic pip Commands

- **Install a package:** `pip install <package_name>`
  Example: `pip install numpy`
- **Upgrade a package:** `pip install --upgrade <package_name>`
  Example: `pip install --upgrade pandas`
- **Uninstall a package:** `pip uninstall <package_name>`
  Example: `pip uninstall matplotlib`

Information Commands in pip

- **Show installed packages:** `pip list`
- **Check for outdated packages:** `pip list --outdated`
- **Show details of a package:** `pip show <package_name>`
  Example: `pip show flask`

Working with Requirements Files

- **Install from a requirements file:** `pip install -r requirements.txt`
- **Generate a requirements file:** `pip freeze > requirements.txt`

Clearing Space with pip

- **Clear pip cache:** `pip cache purge` (usually works)
- **Uninstall unused packages:** `pip uninstall <package_name>` (not that helpful)
- **Remove all packages:** `pip freeze | xargs pip uninstall -y` (use as a last resort)

Other Useful pip Commands

- **Get help:** : `pip --help` or `pip -h`
- **Check pip version:** `pip --version`
- **Upgrade pip:** `pip install --upgrade pip`

Introduction

- Python provides built-in functions for handling files.
- Modes: Read (‘r‘), Write (‘w‘), Append (‘a‘), and Binary (‘b‘).

File Formats

- Python supports multiple file formats:
  - **Text files (.txt)**
  - JSON files (.json)
  - XML files (.xml)
  - CSV files (.csv)
  - Excel files (.xlsx)

Opening a File

Syntax:

```
file = open("filename.txt", "mode")
```

Modes:

- `"r"` – Read mode (default), raises an error if the file does not exist.
- `"w"` – Write mode, creates a new file if it does not exist, overwrites if it does.
- `"a"` – Append mode, creates a new file if it does not exist, appends to an existing file.
- `"x"` – Exclusive creation mode, raises an error if the file already exists.
- `"t"` – Text mode (default), used for reading and writing text files.
- `"b"` – Binary mode, used for handling binary files (e.g., images, videos).

Reading a File

Modes:

- `"r"` - Opens the file for reading.
- `"r+"` - Opens the file for both reading and writing.

Methods to read a file:

- `read()` - Reads the entire file.
- `readline()` - Reads one line at a time.
- `readlines()` - Reads all lines into a list.

Example:

```
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()  # Ensure the file is closed properly
```

Writing to a File

Modes:

- `"w"` - Opens the file for writing; **overwrites** the file if it exists, or creates a new one if it doesn't.
- `"a"` - Opens the file for writing; **appends** data to the file if it exists, or creates a new one if it doesn't.

Writing Functions:

- `write()` - Writes a single string to the file. It does not automatically add a newline.
- `writelines()` - Writes a list of strings to the file. Newline characters must be explicitly included.

Writing to a File

Examples:

```
file = open("example.txt", "w")
file.write("Hello, Python!")
file.write("\nThis is a new line.")
file.close()  # Ensure the file is closed properly

lines = ["First line\n", "Second line\n", "Third line\n"]
file = open("example.txt", "w")  # Open the file again to
↪  overwrite
file.writelines(lines)
file.close()  # Close the file after writing
```

Closing a File

- Always close a file after use.
- Prevents data loss, resource leaks, file corruption and also ensures data consistency (Other processes may need to access the same file).

Example:

```
file = open("example.txt", "r")
content = file.read()
file.close()
```

Using with Statement

- Ensures file closure automatically.
- Reduces risk of memory leaks.

Example:

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Working with Binary Files

- Use mode `"rb"` (read binary) or `"wb"` (write binary).
- Used for handling non-text files such as images, audio, video, compressed files, PDFs, and serialized objects (`pickle`).
- Although specialized libraries are preferred for working with media files, `rb`/`wb` modes are still useful for low-level operations.

Example:

```
with open("image.jpg", "rb") as file:
    data = file.read()
```

Exception Handling in File Handling

- Errors like missing files or permission issues can cause crashes.
- Use `try-except` blocks to handle exceptions gracefully.

Example:

```
try:
    with open("nonexistent.txt", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("File not found!")
except Exception as e:
    print(f"An error occurred: {e}")
```

- Catch specific exceptions like `FileNotFoundError`, `PermissionError`, `IOError`.
- `finally` block can be used to ensure resources are released (if you are not using `with` statement).

Using the `re` Module in Python

- The `re` module provides support for regular expressions.
- Useful for pattern matching, searching, and text processing.
- Common functions: `re.match()`, `re.search()`, `re.findall()`, `re.sub()`, `re.compile()`.

Example:

```
import re

text = "Order 123, item 456, price $78"
pattern = re.compile(r"\d+")  # Precompile the regex pattern
numbers = pattern.findall(text)
```

Why Use `re.compile()`?

- Improves performance when using the same pattern multiple times.
- Makes the code cleaner and easier to read.

File Handling Modes in Python (Reading)

Mode	Description
`rb`	Opens a file for reading only in binary format. Pointer at the beginning.
`r+`	Opens a file for both reading and writing. Pointer at the beginning.
`rb+`	Opens a file for both reading and writing in binary format. Pointer at the beginning.

File Handling Modes in Python (Writing)

Mode	Description
`wb`	Opens a file for writing only in binary format. Overwrites if it exists, creates new if not.
`w+`	Opens a file for both writing and reading. Overwrites if it exists, creates new if not.
`wb+`	Opens a file for both writing and reading in binary format. Overwrites if it exists, creates new if not.

File Handling Modes in Python (Appending)

Mode	Description
`ab`	Opens a file for appending in binary format. Pointer at the end. Creates new if not exists.
`a+`	Opens a file for both appending and reading. Pointer at the end. Creates new if not exists.
`ab+`	Opens a file for both appending and reading in binary format. Pointer at the end. Creates new if not exists.

File Handling Methods in Python

Method	**Description**
`close()`	Closes an opened file. It has no effect if the file is already closed.
`detach()`	Separates the underlying binary buffer from `TextIOBase` and returns it.
`fileno()`	Returns an integer number (file descriptor) of the file.
`flush()`	Flushes the write buffer of the file stream.
`isatty()`	Returns `True` if the file stream is interactive.

File Handling Methods in Python

Method	**Description**
`read(n)`	Reads at most `n` characters from the file. Reads till end of file if it is negative or `None`.
`readable()`	Returns `True` if the file stream can be read from.
`readline(n=-1)`	Reads and returns one line from the file. Reads in at most `n` bytes if specified.
`readlines(n=-1)`	Reads and returns a list of lines from the file. Reads in at most `n` bytes/characters if specified.

File Handling Methods in Python

Method	**Description**
`seek(offset, from=SEEK_SET)`	Changes the file position to `offset` bytes, in reference to `from` (start, current, end).
`seekable()`	Returns `True` if the file stream supports random access.
`tell()`	Returns the current file location.
`truncate(size=None)`	Resizes the file stream to `size` bytes. If `size` is not specified, resizes to current location.

File Handling Methods in Python

Method	**Description**
`writable()`	Returns `True` if the file stream can be written to.
`write(s)`	Writes the string `s` to the file and returns the number of characters written.
`writelines(lines)`	Writes a list of `lines` to the file.

Handling JSON Files in Python

- Use the `json` package to work with JSON data.
- **Reading JSON:**

```
import json

with open("data.json", "r") as file:
    data = json.load(file)  # Load JSON data
```

- **Writing JSON:**

```
with open("data.json", "w") as file:
    json.dump(data, file)  # Save data as JSON
```

Handling CSV Files in Python

- Use the `csv` package to work with CSV data.
- **Reading CSV:**

```
import csv

with open("data.csv", newline="") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

- **Writing CSV:**

```
with open("data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Name", "Age"])
    writer.writerow(["Alice", 25])
```

What is the use of `newline`?

Handling Excel (XLSX) Files in Python

- Use the `openpyxl` package to read **'.xlsx'** files.
- **Reading an Excel File:**

```python
import openpyxl

# Open the Excel file (.xlsx format)
workbook = openpyxl.load_workbook("data.xlsx")

# Select the active sheet
sheet = workbook.active

# Read and print all rows
for row in sheet.iter_rows(values_only=True):
    print(row)
```

Handling Excel (XLSX) Files in Python

- Use the `openpyxl` package to read and write **'.xlsx'** files.
- **Writing an Excel File:**

```python
import openpyxl

# Create a new workbook and select the active sheet
workbook = openpyxl.Workbook()
sheet = workbook.active

# Add data to the sheet
sheet["A1"] = "Name"
sheet["B1"] = "Age"
sheet.append(["Alice", 25])
sheet.append(["Bob", 30])

# Save the workbook
workbook.save("data.xlsx")
```

Handling XML Files in Python

- Use `xml.etree.ElementTree` to work with XML.
- **Parsing XML:**

```
import xml.etree.ElementTree as ET

tree = ET.parse("data.xml")
root = tree.getroot()

for child in root:
    print(child.tag, child.text)
```

Conclusion

- Python provides simple and efficient file handling.
- Use `open()` and `with` for file operations.
- Handle exceptions properly to avoid crashes.

Questions?

Important Warning for Lab Activity

Encoding Issues!!

- **All files must be saved using `UTF-8` encoding!**
- **Windows Defaults to `CP1252`:**
  - Windows uses `cp1252` (or `latin1`), causing issues with special characters.
  - If you don't specify `encoding="utf-8"`, your code may fail or produce incorrect results.
- **Best Practice**: **Regardless** of your **Operating System** use `encoding="utf-8"` in Python to ensure *proper handling of text files*.