# Lecture notes on Python Programming

**By Jason Pandian**
*Assistant Professor, Department of Information Technology*

# FILES AND OOPS CONCEPT IN PYTHON

Files, Text files, reading and writing files-format operator; Files and exception handling -Introduction to Object Oriented Programming – Basic principles of Object Oriented Programming in Python – Class definition-Object Creation - Inheritance, Composition, Operator Overloading.

- In Python, file operations are straightforward and intuitive.
- You can open, read, write, and manipulate files easily using built-in functions and methods.

## Opening and Reading Files:

You can open a file using the open() function and then read its contents using various methods.

## Example 1: Reading a Text File

In [28]:
```
!touch example.txt
```

In [29]:
```
# Open a text file in read mode
with open('example.txt', 'r') as file:
    # Read the entire file content
    content = file.read()
    print(content)
```

```
Hello Students, This is python class.
This is unit four.
```

## Example 2: Reading Lines from a Text File

In [30]:
```
# Open a text file in read mode
with open('example.txt', 'r') as file:
    # Read lines from the file
```

```
    lines = file.readlines()
    for line in lines:
        print(line.upper())  # Remove newline character
```

```
HELLO STUDENTS, THIS IS PYTHON CLASS.

THIS IS UNIT FOUR.
```

### Writing to Files:

You can write data to a file using the write() method or other methods like writelines().

## Example 3: Writing to a Text File

In [32]:
```
!touch output.txt
```

In [38]:
```
# Open a text file in write mode
with open('output.txt', 'w') as file:
    # Write data to the file
    file.write(input("Write the context"))
```

### File Formats and Operations:

Python can handle various file formats and operations, including text files, CSV files, JSON files, etc.

## Example 4: Reading and Writing CSV Files

In [4]:
```
import csv

# Writing data to a CSV file
with open('data.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Name', 'Age'])
    writer.writerow(['John', 25])
    writer.writerow(['Alice', 30])

# Reading data from a CSV file
with open('data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        print(row)
```

```
['Name', 'Age']
['John', '25']
['Alice', '30']
```

# Example 5: Reading and Writing JSON Files

In [5]:
```python
import json

# Writing data to a JSON file
data = {'name': 'John', 'age': 30}
with open('data.json', 'w') as jsonfile:
    json.dump(data, jsonfile)

# Reading data from a JSON file
with open('data.json', 'r') as jsonfile:
    data = json.load(jsonfile)
    print(data)
```

```
{'name': 'John', 'age': 30}
```

- These examples cover basic file operations and demonstrate how to read and write text, CSV, and JSON files in Python.
- Remember to handle exceptions and close files properly after operations to avoid resource leaks.

## 1. Files in Python

In Python, files are objects that allow you to interact with external files on your computer's file system. You can perform various operations on files, such as reading from them, writing to them, and appending data to them.

## 2. Text Files

Text files are files that contain human-readable text. They are commonly used for storing data that can be easily interpreted by both humans and computers. Text files can be created, read, written to, and appended in Python.

## Opening a File

You can open a file using the open() function. It takes two arguments: the filename and the mode in which to open the file.

```
file = open("example.txt", "r")  # Opens "example.txt" in
read mode
```

## Closing a File

After you've finished working with a file, it's essential to close it using the close() method. This ensures that any resources associated with the file are released.

```
file.close()
```

Using `with` is recommended as it ensures that files are always closed properly, even if exceptions occur during file operations.

In [ ]:
```
# Open the file
file = open("example.txt", "r")

# Read data from the file
data = file.read()
print(data)

# Close the file
file.close()
```

## 3. Reading and Writing Files

Reading and writing files in Python involves several modes, each with its specific behavior:

- `r` (Read Mode)
- Opens a file for reading.
- Raises an error if the file does not exist.
- The file pointer is placed at the beginning of the file.

In [ ]:
```
with open('example.txt', 'r') as file:
    data = file.read()
    print(data)
```

- `r+` (Read/Write Mode)
- Opens a file for both reading and writing.
- Raises an error if the file does not exist.
- The file pointer is placed at the beginning of the file.

In [ ]:
```
with open('example.txt', 'r+') as file:
    data = file.read()
    print(data)
    file.write("New data")
```

- `w` (Write Mode)
- Opens a file for writing.
- Creates a new file if it does not exist.
- Truncates the file if it exists.
- The file pointer is placed at the beginning of the file.

In [ ]:
```
with open('example.txt', 'w') as file:
    file.write("Hello, world!\n")
    file.write("This is a sample text.")
```

- `w+` (Write/Read Mode)
- Opens a file for both writing and reading.
- Creates a new file if it does not exist.
- Truncates the file if it exists.
- The file pointer is placed at the beginning of the file.

In [ ]:
```
with open('example.txt', 'w+') as file:
    file.write("Hello, world!\n")
    file.seek(0)
    data = file.read()
    print(data)
```

- `a` (Append Mode)
- Opens a file for appending.
- Creates a new file if it does not exist.
- The file pointer is placed at the end of the file.

In [ ]:
```
with open('example.txt', 'a') as file:
    file.write("\nThis is appended data.")
```

In [ ]:
```
- `a+` (Append/Read Mode)
- Opens a file for both appending and reading.
- Creates a new file if it does not exist.
- The file pointer is placed at the end of the file.
```

In [ ]:
```
with open('example.txt', 'a+') as file:
    file.write("\nThis is appended data.")
    file.seek(0)
    data = file.read()
    print(data)
```

File format operators in Python are used to format strings before writing them to a file. These operators allow you to insert values into strings in a specific format, making it easier to generate structured text output for files. The format operators include %s, %d, %f, %x, %o, %e, %c, and %%, each serving a specific purpose:

- %s: String
- - Used to insert a string into the formatted string.
- - Example: "Hello, %s!" % "world"
- %d, %i: Integer
- - Inserts an integer into the formatted string.

- Example: "The answer is %d." % 42
- %f, %F: Floating point number
- Inserts a floating-point number into the formatted string.
- Example: "Pi is approximately %f." % 3.14159
- %x, %X: Hexadecimal
- Inserts an integer in hexadecimal format into the formatted string.
- Example: "The hexadecimal representation of 10 is %x." % 10
- %o, %O: Octal
- Inserts an integer in octal format into the formatted string.
- Example: "The octal representation of 10 is %o." % 10
- %e, %E: Exponential
- Inserts a floating point number in exponential notation into the formatted string.
- Example: "Avogadro's number is %e." % 6.022e23
- %c: Character
- Inserts a single character into the formatted string.
- Example: "The first letter of the alphabet is %c." % 'A'
- %%: Percentage
- Inserts a literal % character into the formatted string.
- Example: "I got 90%% on my test."

## Example

Let's say you want to generate a formatted string with some data and write it to a file using these format operators:

In [1]:
```
# Data
name = "Alice"
age = 30
height = 5.8

# Format string using format operators
formatted_string = "Name: %s, Age: %d, Height: %.2f" % (name, age, height)

# Writing formatted string to a file
with open('formatted_output.txt', 'w') as file:
    file.write(formatted_string)
```

## Exception Handling

In [3]:
```
try:
    # Code that may raise exceptions
    result = 10 / 0  # This will raise a ZeroDivisionError
except Exception as e:
    # Handling any exception that occurs
    print("An error occurred:", e)
```

```
finally:
    print("Check the code")
```

```
An error occurred: division by zero
Check the code
```

## Opening a File

To open a file, you can use the open() function. This function can raise various exceptions, such as FileNotFoundError if the file does not exist or PermissionError if you don't have the required permissions to open the file.

In [ ]:
```
try:
    with open("example.txt", "r") as file:
        data = file.read()
        print(data)
except FileNotFoundError:
    print("File not found.")
except PermissionError:
    print("Permission denied to open the file.")
except Exception as e:
    print("An error occurred:", e)
```

## Reading from a File

Reading from a file can also raise exceptions, such as IOError if there's an issue reading from the file or UnicodeDecodeError if the file contains non-text data

In [ ]:
```
try:
    with open("example.txt", "r") as file:
        data = file.read()
        print(data)
except IOError:
    print("Error reading from the file.")
except UnicodeDecodeError:
    print("Error decoding file contents.")
except Exception as e:
    print("An error occurred:", e)
```

## Writing to a File

When writing to a file, exceptions like IOError or PermissionError might occur if there are issues with file permissions or disk space.

In [ ]:
```
try:
    with open("example.txt", "w") as file:
        file.write("Hello, world!")
except IOError:
    print("Error writing to the file.")
except PermissionError:
    print("Permission denied to write to the file.")
```

```
except Exception as e:
    print("An error occurred:", e)
```

## Closing a File

Although using the with statement handles file closure automatically, you can still handle exceptions related to file closure explicitly.

In [ ]:
```
try:
    file = open("example.txt", "r")
    try:
        data = file.read()
        print(data)
    finally:
        file.close()
except FileNotFoundError:
    print("File not found.")
except Exception as e:
    print("An error occurred:", e)
```