AKTU B.Tech Second Year

PYTHON PROGRAMMING

UNIT 4

Python Program Flow Control Conditional blocks

LEC-1

Today's Target

AKTU PYQs

By
Pragya Rajvanshi

Gateway Classes

FILES

- A file is a container in computer storage devices used for storing data.
- When a program is terminated, the entire data is lost.
- Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
- However, if you have a file containing all the data, you can easily access the contents of the file using a few commands .

Advantages of File Handling in Python

1.Versatility: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.

2. Flexibility: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files etc.), and to perform different operations on files (e.g. read, write, append, etc.).

CSV files are plain text files that store tabular data in a simple format where each line represents a row, and values within the row are separated by commas

Gateway classes

3. User–friendly: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files

Disadvantages of File Handling in Python

1.Error-prone: File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).

2.Security risks: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.

Complexity : File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.

Performance : File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

Gateway classes

OPEN ()

The Python open() function is used to open internally stored files.

It returns the contents of the file as Python objects.

Python open() Function Syntax

Syntax: open(file_name, mode)

Parameters:.

file name: the name of the file that we want to open.

mode: This parameter is a string that is used to specify the mode in which the file is to be opened.

File opening modes

1.r: open an existing file for a read operation.

2 w: open an existing file for a write operation. If the file already contains some data, then it will be overridden but if the file is not present then it creates the file as well.

3.a: open an existing file for append operation. It won't override existing data.

4. r+: To read and write data into the file. This mode does not override the existing data, but you can modify the data starting from the beginning of the file.

Gateway classes

5 w+: To write and read data. It overwrites the previous file if one exists, it will truncate the file to zero length or create a file if it does not exist.

6.a+: To append and read data from the file. It won't override existing data.

7. rb : Open the file for reading in binary format. Raises an I/O error if the file does not exist.

8.rb+: Open the file for reading and writing in binary format. Raises an I/O error if the file does not exist

9. wb: Open the file for writing in binary format. Truncates the file if it already exists. Creates a new file if it does not exist

10. wb+ Open the file for reading and writing in binary format. Truncates the file if it already exists. Creates a new file if it does not exist.

11.ab Open the file for appending in binary format. Inserts data at the end of the file. Creates a new file if it does not exist.

12. ab+ Open the file for reading and appending in binary format. Inserts data at the end of the file. Creates a new file if it does not exist.

Gateway classes

Open a file the to read the content

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

main.py | sales.txt

```
1  hello
2  students
3  unit 4
```

```
hello
students
unit 4
```

Open a file to Write the content

```
file = open("myfile.txt", "w")
file.write("how are you?")
file.close()
```

main.py | myfile.txt

```
1  how are you?
```

Open a file to append the data

```
file = open("myfile.txt", "a")
file.write("hiiii")
file.close()
```

```
how are you?hiiii
```

Gateway classes

W+ mode example

```
file_name = "example.txt"
with open(file_name, "w+") as file:
    file.write("Hello, this is a test using w+ mode.\n")
    file.write("This mode allows both writing and reading.\n")
    file.seek(0)
    content = file.read()
    print("File content after writing:")
    print(content)
# Verify the file contents by reopening in read mode
with open(file_name, "r") as file:
    print("Verified content from the file:")
    print(file.read())
```

Gateway classes

main.py | example.txt

```
1  Hello, this is a test using w+ mode.
2  This mode allows both writing and reading.
3
```

```
File content after writing:
Hello, this is a test using w+ mode.
This mode allows both writing and reading.

Verified content from the file:
Hello, this is a test using w+ mode.
This mode allows both writing and reading.
```

Gateway Classes

Without seek ()

```
file_name = "example.txt"
with open(file_name, "w+") as file:
    file.write("Hello, this is a test using w+ mode.\n")
    file.write("This mode allows both writing and reading.\n")
    content = file.read()
    print("File content after writing:")
    print(content)
with open(file_name, "r") as file:
    print("Verified content from the file:")
    print(file.read())
```

```
File content after writing:

Verified content from the file:
Hello, this is a test using w+ mode.
This mode allows both writing and reading.
```

classes

r+ mode

main.py | example.txt

```
1  file_name = "example.txt"
2  with open(file_name, "w") as file:
3      file.write("This is the original content.\n")
4  with open(file_name, "r+") as file:
5      print("Original file content:")
6      print(file.read())
7      file.seek(0)
8      file.write("Updated content.\n")
9      file.seek(0)
10     print("\nUpdated file content:")
11     print(file.read())
12 # Verify the file content by reopening in read mode
13 with open(file_name, "r") as file:
14     print("\nVerified content from the file:")
15     print(file.read())
```

Gateway classes

```
Original file content:
This is the original content.


Updated file content:
Updated content.
nal content.


Verified content from the file:
Updated content.
nal content.
```

```
This is the original content.
```

```
Updated content.
nal content.
```

Gateway classes

a+ mode

```
file_name = "example.txt"
with open(file_name, "a+") as file:
    file.write("This is\n")
    print("\nUpdated file content:")
    file.seek(0)
    print(file.read())
```

```
Updated file content:
hi students
This is
```

main.py | example.txt

```
1  hi students
2  This is
3
```

Gateway Cl

Different ways read can be performed

Read()

main.py | example.txt

```
1  file_name = "example.txt"
2  with open(file_name, "w") as file:
3      file.write("Line 1: Hello\n")
4      file.write("Line 2: World\n")
5      file.write("Line 3: Python\n")
6  with open(file_name, "r") as file:
7      # Using read()
8      print("Using read():")
9      content = file.read()
10     print(content)
11
```

main.py | example.txt

```
1  Line 1: Hello
2  Line 2: World
3  Line 3: Python
4
```



```
Using read():
Line 1: Hello
Line 2: World
Line 3: Python
```

Gateway classes

Different ways read can be performed

readline()

```
file_name = "example.txt"
with open(file_name, "w") as file:
    file.write("Line 1: Hello\n")
    file.write("Line 2: World\n")
    file.write("Line 3: Python\n")

with open(file_name, "r") as file:
    print("\nUsing readline():")
    line = file.readline()
    print(line, end="")
    line = file.readline()
    print(line, end="")
```

```
Using readline():
Line 1: Hello
Line 2: World
```

main.py | example.txt

```
1  Line 1: Hello
2  Line 2: World
3  Line 3: Python
4
```

Gateway class

Different ways read can be performed

readlines()

main.py | example.txt

```
1  file_name = "example.txt"
2  with open(file_name, "w") as file:
3      file.write("Line 1: Hello\n")
4      file.write("Line 2: World\n")
5      file.write("Line 3: Python\n")
6  with open(file_name, "r") as file:
7      print("\nUsing readlines():")
8      lines = file.readlines()
9      print(lines)
```

```
Using readlines():
['Line 1: Hello\n', 'Line 2: World\n', 'Line 3
: Python\n']
```

Gateway Classes



Gateway classes