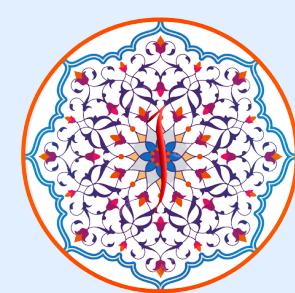


Working with Files in Python

Working with Files in Python

Python makes it easy to work with files, enabling you to read from and write to them.

Files can be in different formats, such as text files (.txt) and binary files (e.g., .bin, images). Python provides built-in functions to handle files efficiently.



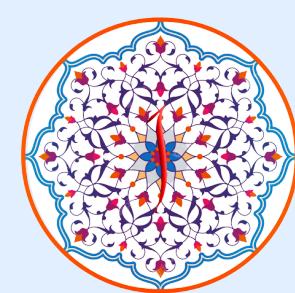
Working with Files in Python

Opening and Closing Files

To work with a file, you first need to open it.

The `open()` function is used for this, and it returns a file object, which allows you to interact with the file (read/write).

After working with the file, it's essential to close it using `close()` to free up system resources.



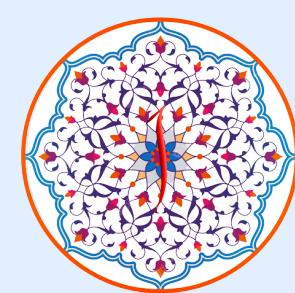
Working with Files in Python

Opening and Closing Files

Syntax of open():

```
python  
  
file = open("filename", "mode")
```

- **filename:** The name of the file to open.
- **mode:** The mode in which to open the file:
- "r": Read (default mode)
- "w": Write (creates the file if it doesn't exist, overwrites if it does)
- "a": Append (creates the file if it doesn't exist, adds to the end of the file)
- "b": Binary mode (used with other modes like "rb", "wb")
- "x": Create (fails if the file already exists)



Working with Files in Python

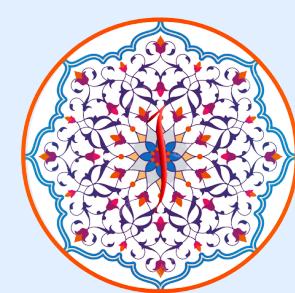
Opening and Closing Files

Example:

```
python

file = open("example.txt", "r") # Opens the file in read mode
# Do some operations
file.close() # Always close the file after use
```

Alternatively, you can use the `with` statement to automatically close the file after the block of code is executed



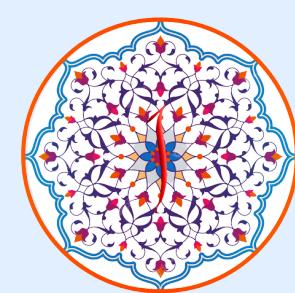
Working with Files in Python

Opening and Closing Files

Example with `with`:

```
python

with open("example.txt", "r") as file:
    content = file.read() # Read the entire content of the file
    print(content)
# No need to explicitly close the file
```



Working with Files in Python

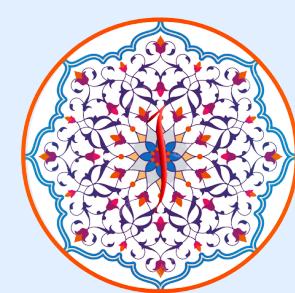
Reading Files

There are several ways to read data from a file:

read() Method:

Reads the entire file as a single string.

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



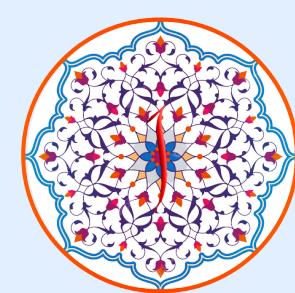
Working with Files in Python

Reading Files

readline() Method:

Reads one line at a time.

Files can be in different formats, such as text files (.txt) and binary files (e.g., .bin, images). Python provides built-in functions to handle files efficiently.



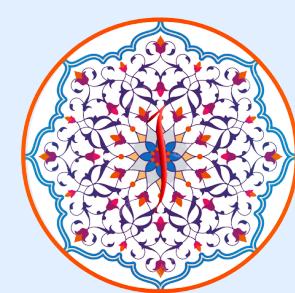
Working with Files in Python

Reading Files

readline() Method:

```
python

with open("example.txt", "r") as file:
    line = file.readline()
    while line:
        print(line, end=' ')
        line = file.readline()
```



Working with Files in Python

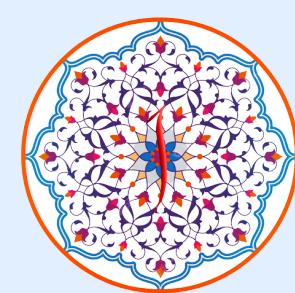
Reading Files

readlines() Method:

Reads all lines at once and returns a list of lines.

```
python

with open("example.txt", "r") as file:
    lines = file.readlines()
    print(lines) # Each line is an element in the list
```

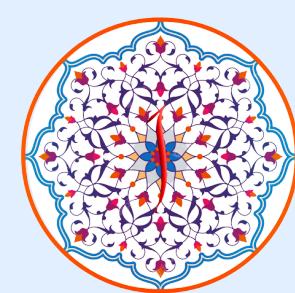


Working with Files in Python

Reading Files

Example of Reading Line by Line:

```
python  
  
with open("example.txt", "r") as file:  
    for line in file:  
        print(line, end='')
```

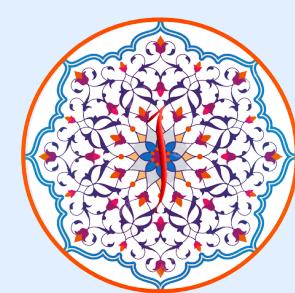


Working with Files in Python

Writing to Files

Python makes it easy to work with files, enabling you to read from and write to them.

Files can be in different formats, such as text files (.txt) and binary files (e.g., .bin, images). Python provides built-in functions to handle files efficiently.

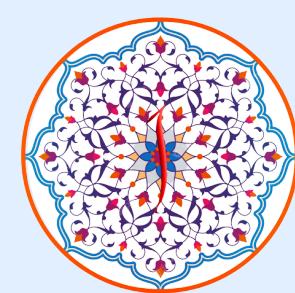


Working with Files in Python

Writing to Files

Python makes it easy to work with files, enabling you to read from and write to them.

Files can be in different formats, such as text files (.txt) and binary files (e.g., .bin, images). Python provides built-in functions to handle files efficiently.

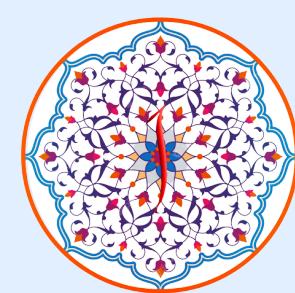


Working with Files in Python

Writing to Files

Python makes it easy to work with files, enabling you to read from and write to them.

Files can be in different formats, such as text files (.txt) and binary files (e.g., .bin, images). Python provides built-in functions to handle files efficiently.



Working with Files in Python

Writing to Files

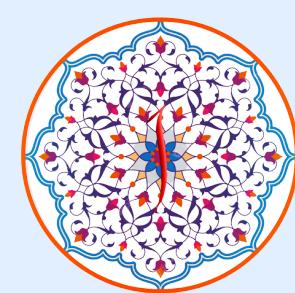
You can write to a file using the `write()` method or the `writelines()` method.

Be careful when using "w" mode, as it will overwrite the file if it exists.

`write()` Method:

Writes a string to the file.

```
python
with open("output.txt", "w") as file:
    file.write("Hello, world!")
```



Working with Files in Python

Writing to Files

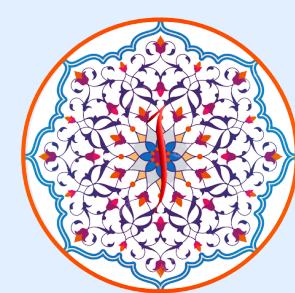
You can write to a file using the `write()` method or the `writelines()` method.

Be careful when using "w" mode, as it will overwrite the file if it exists.

`write()` Method:

Writes a string to the file.

```
python
with open("output.txt", "w") as file:
    file.write("Hello, world!")
```



Working with Files in Python

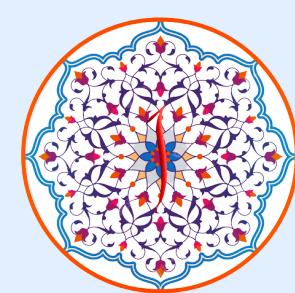
Writing to Files

writelines() Method:

Writes a list of strings to the file (each string should represent a line).

```
python

lines = ["First line\n", "Second line\n", "Third line\n"]
with open("output.txt", "w") as file:
    file.writelines(lines)
```



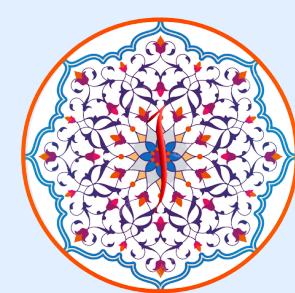
Working with Files in Python

Writing to Files

Appending to a File (a Mode):

Use the "a" mode to add content to the end of a file without overwriting it.

```
python  
  
with open("output.txt", "a") as file:  
    file.write("This will be appended to the file.\n")
```



Working with Files in Python

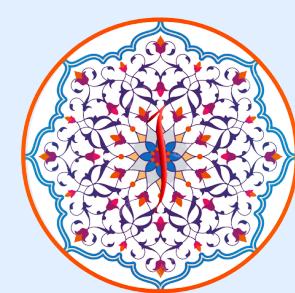
Writing to Files

Working with Binary Files

When working with non-text files like images or videos, you need to open the file in binary mode by adding "b" to the mode ("rb", "wb", etc.).

Reading Binary Files:

```
python
with open("image.png", "rb") as binary_file:
    data = binary_file.read()
    print(data) # Displays binary content
```



Working with Files in Python

Writing to Files

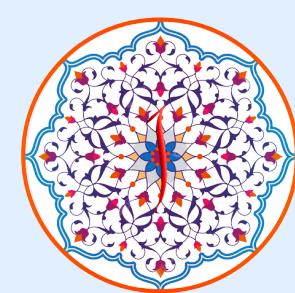
You can write to a file using the `write()` method or the `writelines()` method.

Be careful when using "w" mode, as it will overwrite the file if it exists.

`write()` Method:

Writes a string to the file.

```
python
with open("output.txt", "w") as file:
    file.write("Hello, world!")
```

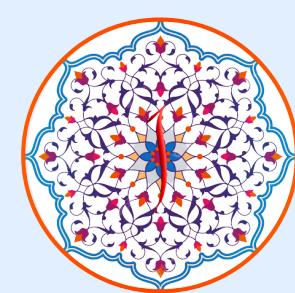


Working with Files in Python

Writing to Files

Writing to Binary Files:

```
python  
  
with open("output.bin", "wb") as binary_file:  
    binary_file.write(b'Some binary data')
```



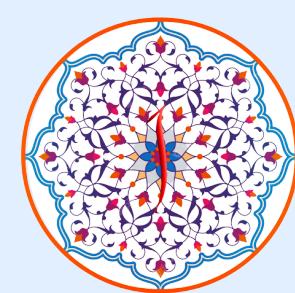
Working with Files in Python

File Positioning with `seek()` and `tell()`

- **`seek(offset, whence)`: Moves the file pointer to a specific location.**
- **`offset`: The number of bytes to move.**
- **`whence`: Optional, defaults to 0 (start of the file). Other values: 1 (current position), 2 (end of the file).**
- **`tell()`: Returns the current position of the file pointer.**

```
python
```

```
with open("example.txt", "r") as file:  
    file.seek(10) # Move to the 10th byte in the file  
    print(file.tell()) # Output the current position (should be 10)  
    content = file.read() # Read from the 10th byte onward  
    print(content)
```



Working with Files in Python

File Methods and Attributes

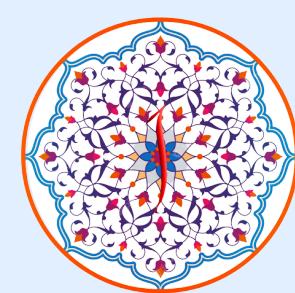
Here are some useful file-related methods and attributes:

- **file.name:** Returns the name of the file.
- **file.mode:** Returns the mode in which the file was opened.
- **file.closed:** Returns True if the file is closed.
- **file.flush():** Flushes the internal buffer (writes buffered data to disk).

Example:

```
python

with open("example.txt", "r") as file:
    print(file.name) # Output: example.txt
    print(file.mode) # Output: r
    print(file.closed) # Output: False
    print(file.closed) # Output: True (after the file is closed)
```



Working with Files in Python

Checking If a File Exists

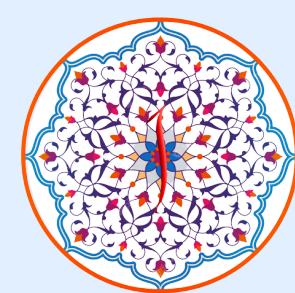
You can check whether a file or directory exists using the **os** module.

Example:

```
python

import os

if os.path.exists("example.txt"):
    print("File exists")
else:
    print("File does not exist")
```



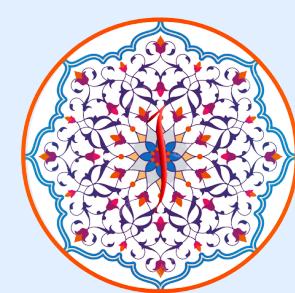
Working with Files in Python

Deleting Files and Directories

To delete files and directories, Python provides functions from the `os` and `shutil` modules.

Deleting Files:

```
python  
  
import os  
  
os.remove("output.txt") # Deletes the file named output.txt
```

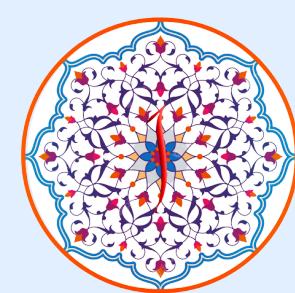


Working with Files in Python

Writing to Files

Deleting an Empty Directory:

```
python  
  
os.rmdir("empty_folder") # Deletes an empty directory
```

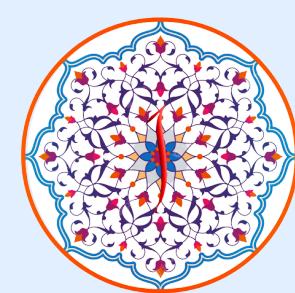


Working with Files in Python

Writing to Files

Deleting a Directory with Contents:

```
python  
  
import shutil  
  
shutil.rmtree("folder_with_files") # Deletes a directory and all its co
```



Working with Files in Python

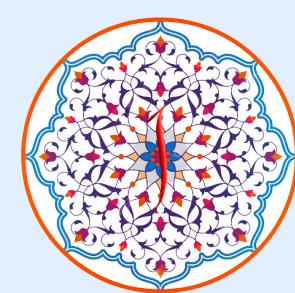
Working with CSV Files

The **csv** module provides functionality to work with **CSV (Comma-Separated Values)** files.

Reading a CSV File:

```
python
import csv

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



Working with Files in Python

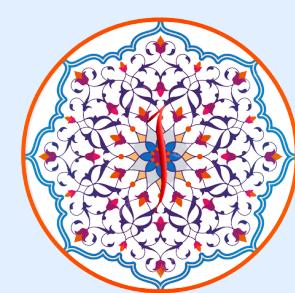
Working with CSV Files

Writing to a CSV File:

```
python

import csv

with open("output.csv", "w", newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Name", "Age", "City"])
    writer.writerow(["Alice", 25, "New York"])
    writer.writerow(["Bob", 30, "Los Angeles"])
```



Working with Files in Python

Working with files in Python is a crucial skill for handling data input and output in various formats, including text, binary, and CSV.

Python provides convenient built-in functions and modules to manage files efficiently.

By properly handling file operations (like reading, writing, and closing files), you can ensure your programs handle data safely and effectively.