Writing on a file with Open()

Estimated time needed: 10 minutes

Objective

1. Create and write data to a file in Python
2. Write multiple lines of text to a file using lists and loops
3. Add new information to an already existing file without erasing its content
4. Compare and contrast the different file modes in Python, what they mean, and when to use them

Writing to a file

You can create a new text file and write data to it using Python's `open()` function. The `open()` function takes two main arguments: the file path (including the file name) and the mode parameter, which specifies the operation you want to perform on the file. For writing, you should use the mode 'w' Here's an example:

```
# Create a new file Example2.txt for writing
with open('Example2.txt', 'w') as file1:
    file1.write("This is line A\n")
    file1.write("This is line B\n")
    # file1 is automatically closed when the 'with' block exits
```

Line 2 explanation: `with open('Example2.txt', 'w') as file1:`

- We start by using the open function to open or create a file named `Example2.txt` for writing ('w' mode).
- The `'w'` mode specifies that we intend to write data to the file.
- We use the `with` statement to ensure that the file is automatically closed when the code block exits. This helps manage resources efficiently.

Line 3 explanation: `file1.write("This is line A\n")`

- Here, we use the `write()` method on the file object, `file1`, to add the text `This is line A` to the file.
- The `\n` at the end represents a newline character, which starts a new line in the file.

Line 4 explanation `file1.write("This is line" B\n")`

- Similarly, we use the `write()` method again to add the text `This is line B` to the file on a new line.

Writing multiple lines to a file using a list and loop

In Python, you can use a list to store multiple lines of text and then write these lines to a file using a loop. Here's an example code snippet that demonstrates this:

```
# List of lines to write to the file
Lines = ["This is line 1", "This is line 2", "This is line 3"]
# Create a new file Example3.txt for writing
with open('Example3.txt', 'w') as file2:
```

```
    for line in Lines:
        file2.write(line + "\n")
    # file2 is automatically closed when the 'with' block exits
```

Here's an explanation of the code:

- Line 2: We start by defining a list called `Lines`, which contains multiple lines of text that we want to write to the file. Each line is a string.
- Line 5: Next, we use the `open()` function to create a new text file named `Example3.txt` for writing, `'w'` mode. The `'w'` mode indicates that we intend to write data to the file.
- Line 6: We then enter a for loop to iterate through each element (line) in the `Lines` list.
- Line 7: Inside the loop, we use the `write()` method on the file object `file2` to write the current line of text (line) to the file. We add `\n` at the end of each line to ensure that each line is followed by a newline character, which separates them in the file.
- Line 8: Finally, we add a comment indicating that the file `file2` will be automatically closed when the code block within the with statement exits. Properly closing the file is essential for good resource management.

Appending data to an existing file

In Python, you can use the `'a'` mode when opening a file to append new data to an existing file without overwriting its contents. Here's an example code snippet that demonstrates this:

```
# Data to append to the existing file
new_data = "This is line C"
# Open an existing file Example2.txt for appending
with open('Example2.txt', 'a') as file1:
    file1.write(new_data + "\n")
    # file1 is automatically closed when the 'with' block exits
```

Here's an explanation of the code:

- Line 2: We start by defining a variable `new_data` that contains the text we want to append to the existing file. In this case, it's the string `This is line C.``
- Line 5: Next, we use the `open()` function to open an existing file named `Example2.txt` for appending, `'a'` mode. The `'a'` mode indicates that we intend to append data to the file, and if the file doesn't exist, it will be created.
- Line 6: Within the with block, we use the `write()` method on the file object `file1` to append the `new_data` to the file. We add `"\n"` at the end to ensure that the appended data starts on a new line, maintaining the file's readability.
- Finally, we add a comment indicating that the file `file1` will automatically close when the code block within the `with` statement exits. Properly closing the file is essential for good resource management.

Copying contents from one file to another

In Python, you can copy the contents of one file to another by reading from the source file and writing to the destination file. Here's an example code snippet that demonstrates this:

```
# Open the source file for reading
with open('source.txt', 'r') as source_file:
    # Open the destination file for writing
    with open('destination.txt', 'w') as destination_file:
        # Read lines from the source file and copy them to the destination file
```

```
            for line in source_file:
                destination_file.write(line)
        # Destination file is automatically closed when the 'with' block exits
    # Source file is automatically closed when the 'with' block exits
```

Here's an explanation of the code:

- Line 2: We start by opening the source file, `source.txt` for reading, `r` mode, using the `with` statement and the `open()` function. This allows us to read data from the source file.
- Line 4: Inside the first with block, we open the destination file, `destination.txt` for writing, `w` mode, using another `with` statement and the `open()` function. This prepares the destination file for writing.
- Line 6: We use a `for` loop to iterate through each line in the source file `source_file`. This loop reads each line from the source file one by one.
- Line 7: Within the loop, we use the `write()` method to write each line from the source file to the destination file `destination_file`. This effectively copies the content of the source file to the destination file.
- Lines 8 and 9: After copying all the lines, both the source and destination files are automatically closed when their respective with blocks exit. Proper file closure is crucial for managing resources efficiently.

File modes in Python (syntax and use cases)

The following table provides an overview of different file modes, their syntax, and common use cases. Understanding these modes is essential when working with files in Python for various data manipulation tasks.

Mode	Syntax	Description
‘r’	`'r'`	Read mode. Opens an existing file for reading. Raises an error if the file doesn't exist.
‘w’	`'w'`	Write mode. Creates a new file for writing. Overwrites the file if it already exists.
‘a’	`'a'`	Append mode. Opens a file for appending data. Creates the file if it doesn't exist.
‘x’	`'x'`	Exclusive creation mode. Creates a new file for writing but raises an error if the file already exists.
‘rb’	`'rb'`	Read binary mode. Opens an existing binary file for reading.
‘wb’	`'wb'`	Write binary mode. Creates a new binary file for writing.
‘ab’	`'ab'`	Append binary mode. Opens a binary file for appending data.
‘xb’	`'xb'`	Exclusive binary creation mode. Creates a new binary file for writing but raises an error if it already exists.
‘rt’	`'rt'`	Read text mode. Opens an existing text file for reading. (Default for text files)
‘wt’	`'wt'`	Write text mode. Creates a new text file for writing. (Default for text files)
‘at’	`'at'`	Append text mode. Opens a text file for appending data. (Default for text files)
‘xt’	`'xt'`	Exclusive text creation mode. Creates a new text file for writing but raises an error if it already exists.

Mode	Syntax	Description
‘r+’	`'r+'`	Read and write mode. Opens an existing file for both reading and writing.
‘w+’	`'w+'`	Write and read mode. Creates a new file for reading and writing. Overwrites the file if it already exists.
‘a+’	`'a+'`	Append and read mode. Opens a file for both appending and reading. Creates the file if it doesn't exist.
‘x+’	`'x+'`	Exclusive creation and read/write mode. Creates a new file for reading and writing but raises an error if it already exists.

Conclusion

Working with files is a fundamental aspect of programming, and Python provides powerful tools to perform various file operations. In this summary, we covered key concepts and code examples related to file handling in Python, including writing, appending, and copying files.

Author(s)

Akansha Yadav