

File Handling

Creating and Manipulating Files



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.org>

sli.do

#python-advanced

1. Python File Object
2. Opening a File
 - File Modes
3. Reading a File
4. Writing and Creating a File
 - Closing a File
 - **with** Statement
5. Deleting a File





Python File Object

Built-In Functions to Create and Manipulate Files

Python File Object

- Built-in functions to create and manipulate files
- **io** module is the default module for accessing files
 - Don't need to import it
- **open()** returns a **file object** whose type depends on:
 - The mode
 - File operations such as reading and writing
 - Files in a text mode ('**w**', '**r**', '**wt**', '**rt**', etc.) return a **TextIOWrapper**





Opening a File

Opens a File and Returns a File Object

open() Function

- **open()** function in Python opens a file and returns a file object `file = open('text.txt', 'r')`
- The arguments are **file name** and the **mode** (reading, etc.)
- All arguments except **file** are optional and have default values
- If the file is not in the current directory, the full path to the file can be provided

```
file = open('D:\\text.txt', 'r')
```

open() Function with Invalid File

- If the file does not exist, **FileNotFoundError** is thrown

```
file = open('invalid.txt', 'r') # FileNotFoundError
```

- It can be caught as a try-finally block

```
try:  
    file = open('invalid.txt', 'r') # FileNotFoundError  
except FileNotFoundError:  
    print("File not found or path is incorrect")  
finally:  
    print("exit")
```


- The **mode** argument is optional and specifies the mode for manipulating the file
- Its default value is '**r**' - open for reading in text mode
- File modes in Python
 - **w** - open for writing, truncating the file first
 - **x** - create a new file and open it for writing
 - **a** - open for writing, appending to the end of the file if it exists
 - **t** - text mode (default)
 - **b** - binary mode
 - **+** - open a disk file for updating (reading and writing)

Problem: File Opener

- Create a program that opens the file called '**text.txt**'
- If the file is found, print '**File found**'
- If the file is not found, print '**File not found**'

```
try:  
    text_file = open('text.txt', 'r')  
    print("File found")  
except FileNotFoundError:  
    print("File not found")
```



Reading a File

Build-In Methods for Reading from File

Reading Functions - read()

- Returns the first **n** bytes of the file
- Returns the **entire file** if a number of bytes is not passed as an argument

```
file = open("asd.txt") # 'Hello, SoftUni!'
print(file.read(2))    # 'He'
print(file.read(2))    # 'LL'
print(file.read(2))    # 'o,'
print(file.read())     # ' SoftUni!'
```

Reading Functions - readline()

- Returns at most **n** bytes of a single line of a file
- It does not read more than one line
- If no argument is passed, the entire line (or rest of the line) is read

```
file = open("text.txt") # 'Hello, SoftUni!'
print(file.readline(5)) # 'Hello'
print(file.readline(5)) # ',Sof'
print(file.readline(5)) # 'tUni!'
print(file.readline())  # '' Goes to the new line
print(file.readline(5)) # 'Secon' Print second line
```

Reading Functions - readlines()

- Read the remaining lines from the file object and return them as a list


```
file = open("text.txt")  
print(file.readlines())  
# ['Every\n', 'Word\n', 'is\n', 'Line']
```

- Keep in mind every line in the file treats the **new line symbol** as a string

```
file = open("text.txt")  
lines = file.readlines()  
[print(line, end="") for line in lines]
```

Looping Over a File Object

- To return all lines from a file you can loop over it
- More memory efficient and fast manner
 - Simple and easy to read



```
file = open("python.txt", 'r')  
  
for line in file:  
    print(line, end="")  
    # print every line in a new line
```

Problem: File Reader

- Create a program that reads the numbers from the file called '**numbers.txt**'
- Print on the console the sum of those numbers

```
numbers_file = open('numbers.txt', 'r')
numbers_sum = 0
for number in numbers_file:
    numbers_sum += int(number)
print(numbers_sum)
```




Writing and Creating a File

Write, Append, Close Methods and **with** Statement

- Using '**w**' (write) mode - **creates** a file with the given name
 - If the file exists, this mode **overwrites** it

```
file = open('python.txt', 'w')
```

```
# Creates or opens the file
```

```
file.write("This is the write command.\n")
```

```
file.write("It allows us to write in a particular file")
```

```
file.close()
```

- Using 'a' mode - **opens** a file and writes at the end of the file
 - If the file does not exist, it will be **created**

```
file = open('python.txt', 'a')
file.write("This is the write command.\n")
file.write("It allows us to write in a particular file")
file.close()
```

```
file = open('python.txt', 'a')
lines = ["Write ", "in ", "file"]
file.writelines(lines) # Write multiple strings
file.close()
```

Closing Files

- We should always make sure that an open file is properly **closed**
- In **most cases**, upon the **termination** of an application or script, a file will be **closed eventually**
 - There is **no guarantee** when exactly that will happen
- To avoid **unwanted behavior**, **always close** the files
 - This is a **good practice!**



with Statement

- Files opened with **with** statement will be **closed automatically** once the **with** block is left
- Provides much cleaner syntax and exception handling

```
with open("file.txt", "w") as f:  
    f.write("Hello World!!!")
```



Problem: File Writer

- Create a program that creates a file called **'my_first_file.txt'**
- In that file write a single line with the content: **'I just created my first file!'**

```
created_file = open('my_first_file.txt', 'w')  
created_file.write('I just created my first file!')
```



Deleting a File

OS module, Check if File Exists and Delete File

Deleting a File

- To delete a file, you must import the **os** module

```
import os
```

```
os.remove("python.txt")
```

```
os.remove("D:\\text.txt")  # Can use full path
```



Deleting a File

- Keep in mind if the file does not exist, an error will be raised
- To avoid getting an error
 - Check whether the file exists
 - Delete file



```
import os

file_path = "text.txt"
if os.path.exists(file_path):
    os.remove(file_path)
```

Problem: File Delete

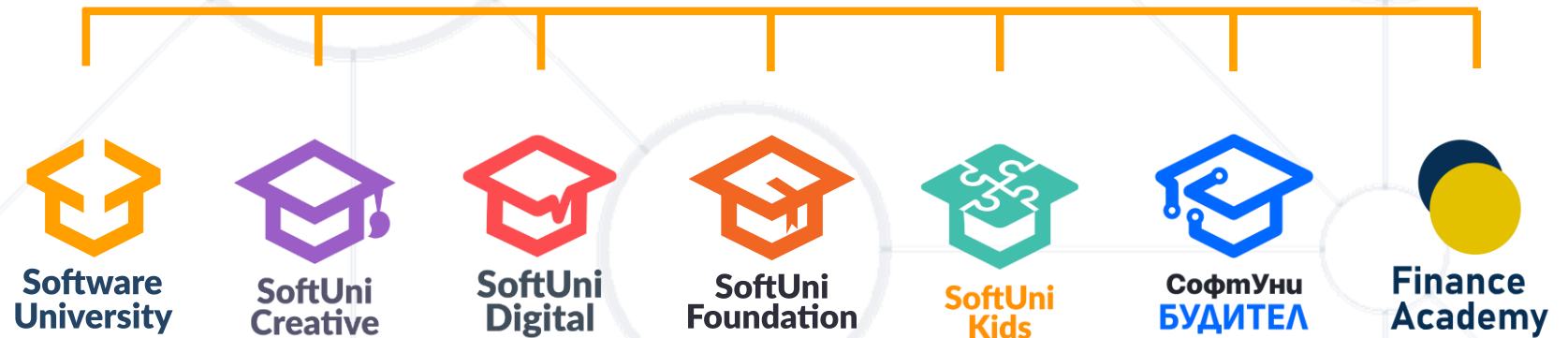
- Create a program that deletes the file you created in the previous task
- If you try to delete the file multiple times, print the message: **'File already deleted!'**

```
import os
try:
    os.remove('my_first_file.txt')
except FileNotFoundError:
    print("File already deleted!")
```

- **Open** or **create** a file
- Choose the appropriate **file mode**
- **Manipulate** the file
- **Close** the file
- **Delete** file
 - Import OS module
 - Check whether the file exists
 - Delete file



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, softuni.org
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

