File IO

You will learn about Python file operations. More specifically, opening a file, reading from it, writing into it, closing it and various file methods.

Write a Cell to File in Jupytor Notebook

Jupyter Notebook provides a magic function `%%file` to export the content in a cell to a file.

- Note: Any magic function staring with `%%` must be at first line of the cell, and it is applied to all lines below it.

Try Code:

Create a file `quote.txt` by running following code.

```
%%file 'quote.txt'
Give a man a program, frustrate him for a day.
Teach a man to program, frustrate him for a lifetime.
```

In [5]:
```
%%file 'quote.txt'
Give a man a program, frustrate him for a day.
Teach a man to program, frustrate him for a lifetime.
```

```
Overwriting quote.txt
```

1. Introduction

What is File?

File is a named location on disk to store related information.

- It uses non-volatile memory, e.g. hard disk, to store data permanently.

A file operation takes place in following order:

- Open a file
- Read or Write (perform operations)
- Close the file

A file can be **text** or **binary** format.

Open File

Python has a built-in function `open(file_path)` to open a file.

- The `open()` function returns a file object, also called a file handler, as it is used to read or modify the file accordingly.

To read data from a file object, use its `read()` method.

To close a file object, uses its `close()` method.

Exercise:

Read and print out content in file `quote.txt`, which is created in previous step.

In [22]:
```
f = open('quote.txt')    # open file in current directory
s = f.read()
print(s)
f.close()
```

Question:

What is the object type of a file handler, e.g. object `f`?

In [18]:
```
type(f)
```

Out[18]:
```
_io.TextIOWrapper
```

Question:

What will be the returned value of `read()` function when the function is called second time?

In [11]:
```
f = open('quote.txt')    # open file in current directory
s = f.read()
# f.seek(0)  # Move pointer to start of file
s = f.read()
print(s)
f.close()
```

Filepath ¶

The `filepath` can be a relative or absolute path.

- If only file name is specified, Python assume the file is in the same folder as current Python kernel
- When specifying full path, use `/` instead of `\`, which is used as escape character in string

In [10]:
```
import os
path = os.path.join(os.path.abspath(''), 'quote.txt')
print(path)

f = open(path)  # specifying full path
f.close()
```

```
D:\GoogleDrive\Learn-Python\Python-MOE-Teacher-Training-2020\09 File IO\quote.t
xt
```

Question:

What if I foget to close the file?

```
For Read operation, unclosed file objects will be garbage collected.

But for Write operation, close() function will flush content in buffer to the
file before closing it. The content may not be written to file if the file is
not closed.
```

2. Context Manager (Optional)

What is Context Manger?

Programming commonly involves usage of resources, e.g. file operations, network and database connections etc. It's important to release these resources after usage because they may be limited in supply.

- **Context manager** is a mechanism in Python for the automatic setup and teardown of resources.
- When an object (function or class) is implemented as a Context Manager, it can be used by `with` operator.

The file `open()` method is implemented as a Context Manager.

- With context manager, `f.close()` will be automatically called when `with` code block exits.
- You can use `f.closed` attribute to check if a file object is closed.

Try Code:

Use `with` code block to read and print out the content of file `quote.txt` .

```
with open('quote.txt') as f:
    print(f.read())

# check if the file is closed
print(f.closed)
```

In [2]:
```
with open('quote.txt') as f:
    print(f.read())

# check if the file is closed
print(f.closed)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-f62975a284a7> in <module>
----> 1 with open('quote.txt') as f:
      2     print(f.read())
      3
      4 # check if the file is closed
      5 print(f.closed)

FileNotFoundError: [Errno 2] No such file or directory: 'quote.txt'
```

Custom Context Manager

You can make your function to work with context manager using decorator `@contextmanager`, which is from built-in module `contextlib`.

- Add `@contextmanager` decorator to the function
- Add `yield` statement to a position in the function
- Statements before `yield` will be run before the `with` code block
- Statements after `yield` will be run after the `with` code block

Exercise:

Imagine you would like to have a meal in harker center. You need to reserve a seat first before your meal. You also need to return the seat after your meal.

Define a function `chope_seat()` which implements context manager. It will

- Reserve a seat by printing `Chope a seat...` before `with` code block exection
- Release the seat by printing `Give up the seat...` after `with` code block execution

Sample Output:

```
Chope a seat...
Enjoy my meal
Give up the seat...
```

In [55]:
```
from contextlib import contextmanager

@contextmanager
def chope_seat():
    print('Chope a seat...')
    yield
    print('Give up the seat...')

with chope_seat():
    print('Enjoy my meal')
```

```
Chope a seat...
Enjoy my meal
Give up the seat...
```

Custom Context Manager with Return Object

The `yield` function can return an object to the `with` statement

Try Code:

Following function `my_open()` is mimicking the `open()` function. It is implemented using `@contextmanager` decorator.

```
from contextlib import contextmanager

@contextmanager
def my_open(filename, mode):
    print('openning')
    f = open(filename, mode)
    yield f
    print('closing')
    f.close()

with my_open('test.txt', 'w') as f:
    print('using file')
    f.write('Test context manager')
```

In [32]:
```
from contextlib import contextmanager

@contextmanager
def my_open(filename, mode):
    print('openning')
    f = open(filename, mode)
    yield f
    print('closing')
    f.close()

with my_open('test.txt', 'w') as f:
    print('using file')
    f.write('Test context manager')
```

```
openning
using file
closing
```

3. Operation Mode

You can specify the mode used to open a file by applying a second argument to the open function.

- `r` / `w` / `a` : Are you reading, writing or appending a file?
- `t` / `b` : Is it a text or binary file?

```
f = open(filepath, mode)
```

Read / Write / Append

The `mode` specifies how you want to work with the file.

- `'r'` : read mode, which is the default.
- `'w'` : write mode, for overwriting the contents of a file. Existing file content will be lost.
- `'a'` : append mode, for appending new content to the end of the file. Existing content in the file will not be lost.

Exercise:

Complete following operations using `with` code block:

- Write `"Alexa, "` to a file `test.txt` . This operator will overwrite any content in the file.
- Append `"Good morning!\n"` to the file `test.txt` .
- Append `"What time is it?"` to the file `test.txt` .
- Read and print out content from the file `test.txt` .

In [44]:
```
# write/create a file
with open("test.txt", "w") as f:
    f.write("Alexa, ")

# append/create a file
with open("test.txt", "a") as f:
    f.write("Good morning!\n")
    f.write("What time is it?")

# read a file
with open("test.txt", "r") as f:
    s = f.read()
#     print(repr(s))
    print(s)
```

```
Alexa, Good morning!
What time is it?
```

Read by Characters

The `read()` method also accepts an argument which specifies the number of character to read.

- Without this argument, it will read till EOF.

Check out the documentation of of `read()`.

```
f.read?
```

In [13]:
```
f.read?
```

Try Code:

```
with open('test.txt', 'r') as f:
    print(repr(f.read(10)))
    print(repr(f.read(10)))
    print(repr(f.read()))
```

The `repr()` function will represent special characters as symbols in a string. It helps to print string unambigously.

In [60]:
```
with open('test.txt', 'r') as f:
    print(repr(f.read(10)))
    print(repr(f.read(10)))
    print(repr(f.read()))
```

```
'Hello\nfrom'
'\nSingapore'
''
```

Write a String

The `write()` method returns number of characters written to the file.

- Note: It counts special characters too.

Try Code:

```
s = "Alexa\tGood morning!\nWhat time is it?"
print(len(s))
with open("test.txt", "w") as f:
    x = f.write(s)
    print(x)
```

In [7]:
```
s = "Alexa\tGood morning!\nWhat time is it?"
print(len(s))
with open("test.txt", "w") as f:
    x = f.write(s)
    print(x)
```

```
36
36
```

Open the file `test.txt` and examine text inside it. All special characters are handled properly.

Try Code:

```
!notepad test.txt
```

In [38]:
```
!notepad test.txt
```

Question:

- What happens if you open a non-existing file in `read` mode?
- What happens if you open a non-existing file in `write` mode?
- What happens if you open a non-existing file in `append` mode?
- What happens if you open an existing file with mode `write` and close immediately?

```

```

Read by Lines

Compared to `read()` function, which return all content in a single string, the `readlines()` function returns a list, where each item contains a line.

NOTE: No character is removed, e.g. new line character `\n` at the end of a line.

Try Code:

```
with open('test.txt') as f:
    s = f.read()
    print(repr(s))

with open('test.txt', 'r') as f:
    s = f.readlines()
    print(s)
```

In [54]:
```
with open('test.txt') as f:
    s = f.read()
    print(repr(s))

with open('test.txt', 'r') as f:
    s = f.readlines()
    print(s)
```

```
'HelloWorld\nfrom\nSingapore'
['HelloWorld\n', 'from\n', 'Singapore']
```

In fact, file object can be passed directly to `for-loop`, which will iterate through the file line by line.

Try Code:

```
f = open('test.txt')

for i in f:
    print(i)
```

In [8]:
```

```

```
Alexa   Good morning!

What time is it?
```

Question:

- Why above there is an empty line between `'Alexa   Good morning!'` and `'What time is it?'`?
- How to remove trailing `\n` from each line?

```

```

Write Multiple Lines

To write a list of stirngs to a file, method `writelines()` can be used.

NOTE: No character, e.g. `\n`, will be added or removed.

Try Code:

```
s = ['Hello', 'World', '\nfrom', '\nSingapore']
with open('test.txt', 'w') as f:
    f.writelines(s)
```

Use `!notepad test.txt` to examine the content in the file.

In [48]:
```
s = ['Hello', 'World', '\nfrom', '\nSingapore']
with open('test.txt', 'w') as f:
    f.writelines(s)
```

In [51]:
```
!notepad test.txt
```

Hybrid Mode (Optional)

When a file is opened with mode `r`, it only supports read operation. Likewise, if a file is open with `w`, it only supports write operation.

Following mode supports both read and write. But it is **not recommended** because you may need to use f.seek() to move cursor to correct position between read and write operation.

- `'r+'` : Open for reading and writing. Cause exception if file does not exists.
- `'w+'` : Open for reading and writing. The file is created if it does not exist, otherwise it is truncated.
- `'a+'` : Open for reading and appending. The file is created if it does not exist.

4. Text File / Binary File

By default, `open()` assumes the file is a **text** file. To work with **binary** files, e.g. images and sound files, adding `"b"` to the `mode`.

- Use `rb` to read a binary file
- Use `wb` to write binary content to a file

Excercise:

Write code to copy an image file `test.jpg` in `./images/` folder to current folder with name `dup.jpg`.

In [15]:
```
## Read binary content of a image and create a duplicate image
with open('./images/test.jpg', 'rb') as f:
    s = f.read()

with open('dup.jpg', 'wb') as f:
    f.write(s)
```

5. Basic CSV File Processing

Comma-Separated Values (CSV) is a common text file format to store tabular data.

- It has simple structure where each row contains one record
- A record may have multiple attributes delimited by common `,`

Sample CSV data:

```
Name,Email,Phone Number,Address
Bob Smith,bob@example.com,123-456-7890,123 Fake Street
Mike Jones,mike@example.com,098-765-4321,321 Fake Avenue
```

Read File into List

- Read the csv file using `readlines()` method, which returns data in a list.
- Use list slicing to remove header row
- Use `strip()` method to remove any surrounding white spaces (space, tab, new line characters)

Exercise:

- Read `sample-sales-data.csv` file into a list.
- Discard header row
- Strip any leading & trailing white space from each line
- Print out first 3 items of the list

In [58]:
```
with open('sample-sales-data.csv') as f:
    arr = f.readlines()

arr = arr[1:]
arr = [i.strip() for i in arr]

# print first 3 rows
print(arr[:3])
```

```
['2015-01-21,Streeplex,Hardware,11', '2015-01-09,Streeplex,Service,abc', '2015-01-06,Initech,Hardware,-17']
```

Split a Record into a Tuple

Each a row in csv file is a record. The record is delimited by `comma (,)`.

- Use `split(delimiter)` method to split the record into a list or a tuple.
- Use multi-level indexing to get a cell value

Exercise:

- Read the file into a list such that each record is represented as a tuple
- Print out first 2 records in the list
- Print out company name of 1st record

In [60]:
```
table = [tuple(i.split(',')) for i in arr]

print(table[:2])
print(table[0][1])
```

```
[('2015-01-21', 'Streeplex', 'Hardware', '11'), ('2015-01-09', 'Streeplex', 'Se
rvice', 'abc')]
Streeplex
```