File I/O Handling

Python File Handling

We often use files to store data (and code) permanently on a storage device

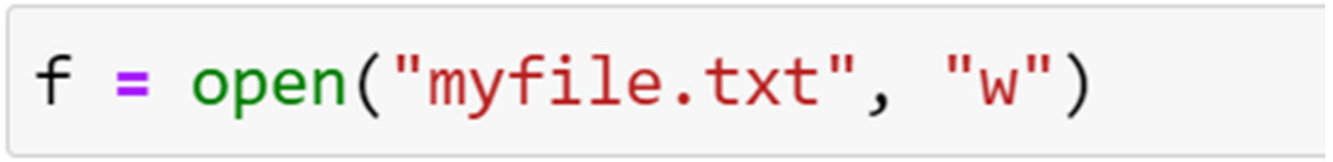
- which can also be conveniently shared and transferred.

Python has several built-in functions/methods to create/read/update/delete files

- first function for working with files is the open() function, which creates a file object

The open() function takes two parameters: *filename* and *mode*

First, type the command to create a new file for writing



```
f = open("myfile.txt", "w")
```

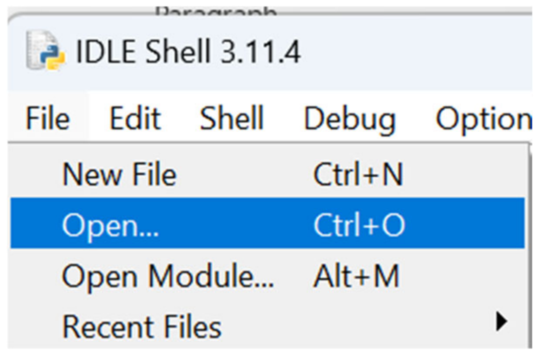
Go to Python’s working directory

- you should see the file just created with a file size of 0

Python File Handling

Back to IDLE Shell, select File->Open to open the file

- type in some texts
- save the file and close it.



```
Create a new text file!ile.txt
This file is for testing purposes.
Good Luck!
```

Back to the IDLE Shell

- open the file for reading using the `read()` method
- print the content

```python
f = open("myfile.txt", "r")
```

```python
print(f.read())
```

```
Create a new text file!ile.txt
This file is for testing purposes.
Good Luck!
```

Python File Handling - Reading

Instead of reading the whole file

- you can read one line of the file using the readline() method

```
f = open("myfile.txt", "r")
print(f.readline())
```

```
Create a new text file!ile.txt
```

- or part of the line in the file using read(index) method

```
f = open("myfile.txt", "r")
```

```
print(f.read(10))
```

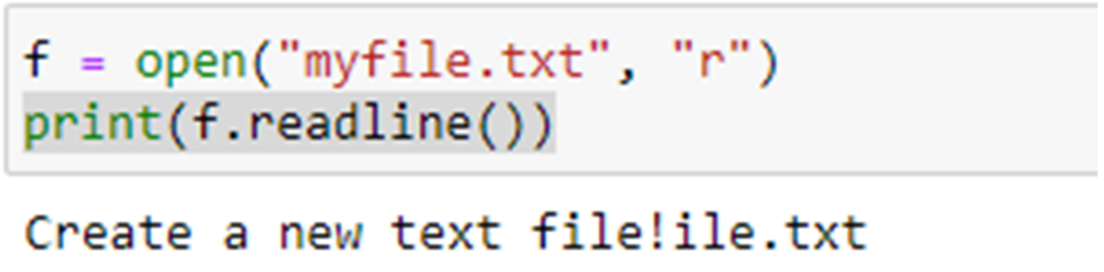
```
Create a n
```

```
Create a new text file!ile.txt
This file is for testing purposes.
Good Luck!
```

Python File Handling - Reading

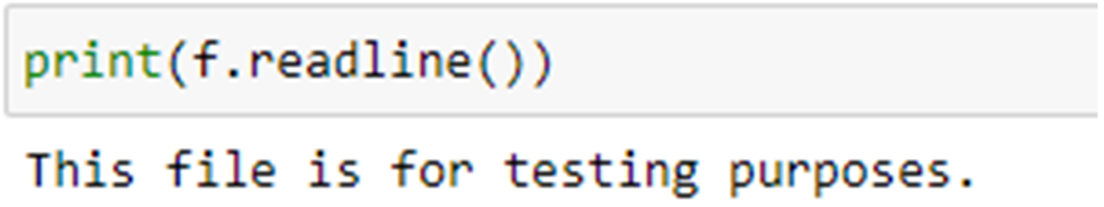
Once you read one line of the file

- the next read will be continued from the next line



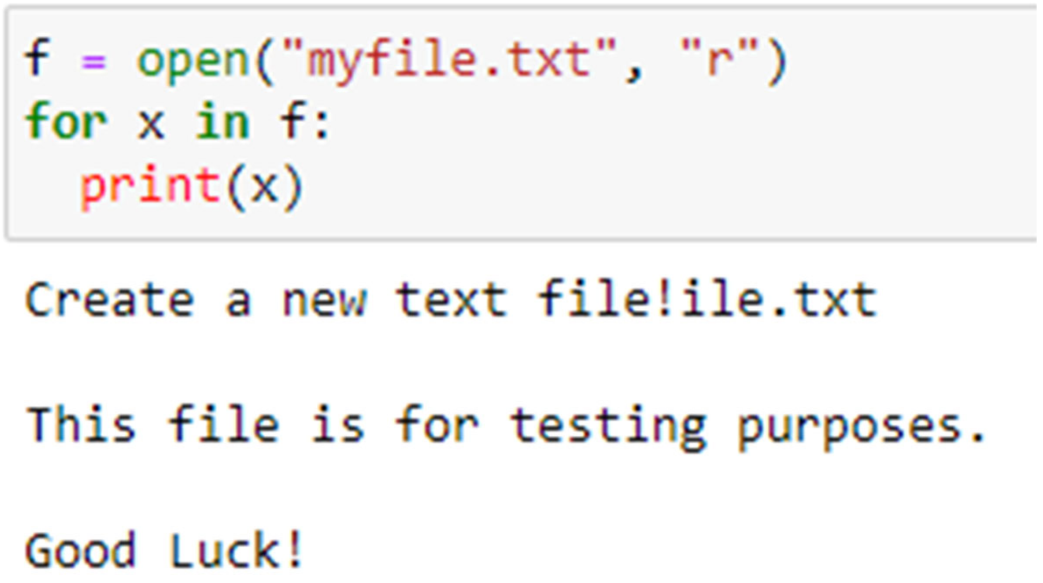
```
f = open("myfile.txt", "r")
print(f.readline())
```

```
Create a new text file!ile.txt
```



```
print(f.readline())
```

```
This file is for testing purposes.
```



```
f = open("myfile.txt", "r")
for x in f:
  print(x)
```

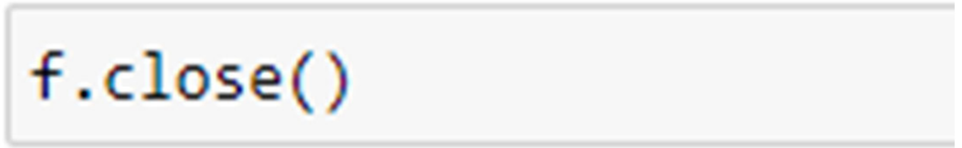
```
Create a new text file!ile.txt

This file is for testing purposes.

Good Luck!
```

You can loop through the file line by line

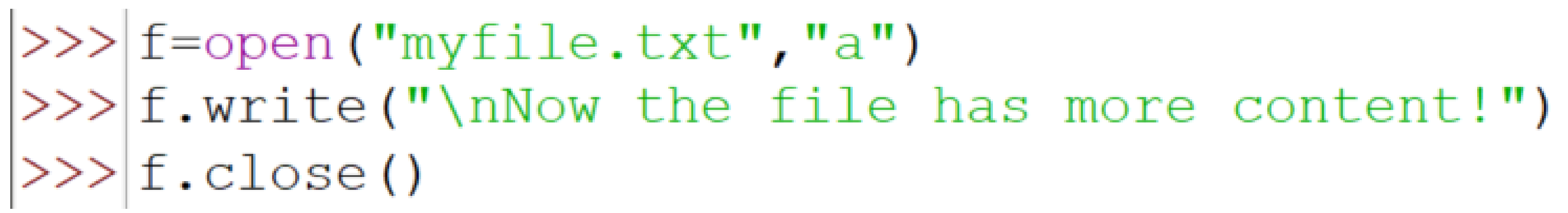
You should close the file when you finish with it



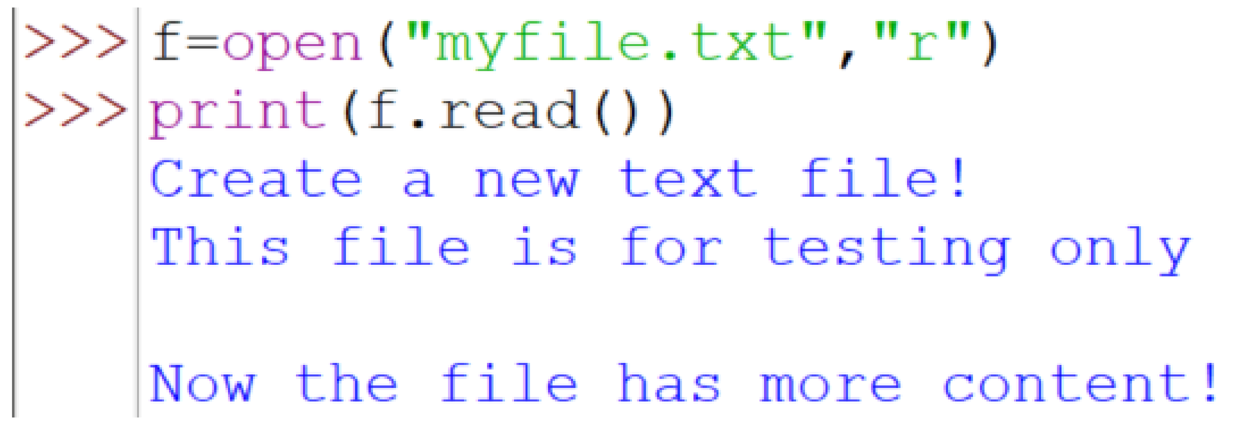
```
f.close()
```

Python File Handling - Writing

You can write to an existing file by opening with “a” or “w”

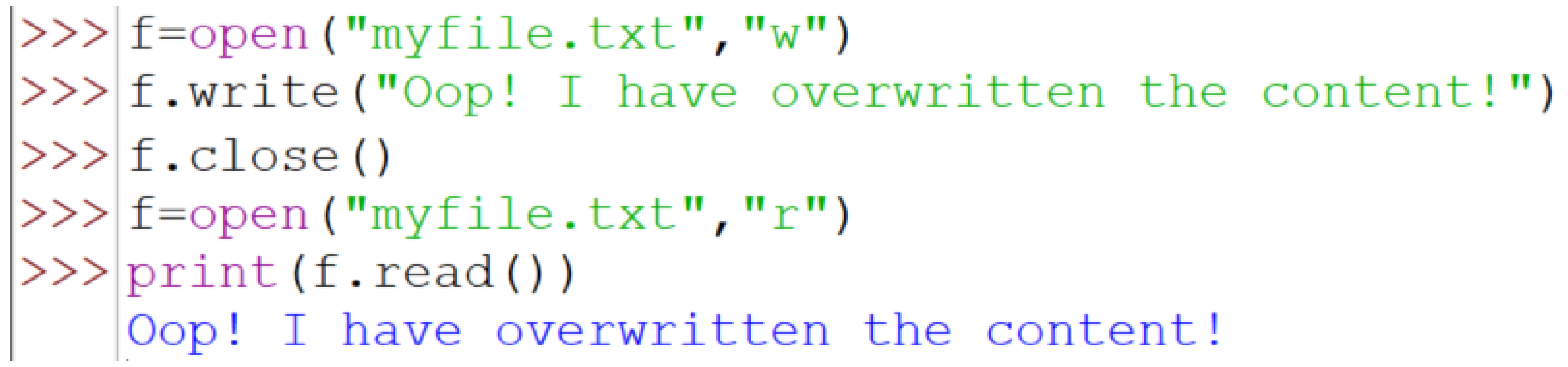


```
>>> f=open("myfile.txt","a")
>>> f.write("\nNow the file has more content!")
>>> f.close()
```



```
>>> f=open("myfile.txt","r")
>>> print(f.read())
Create a new text file!
This file is for testing only

Now the file has more content!
```



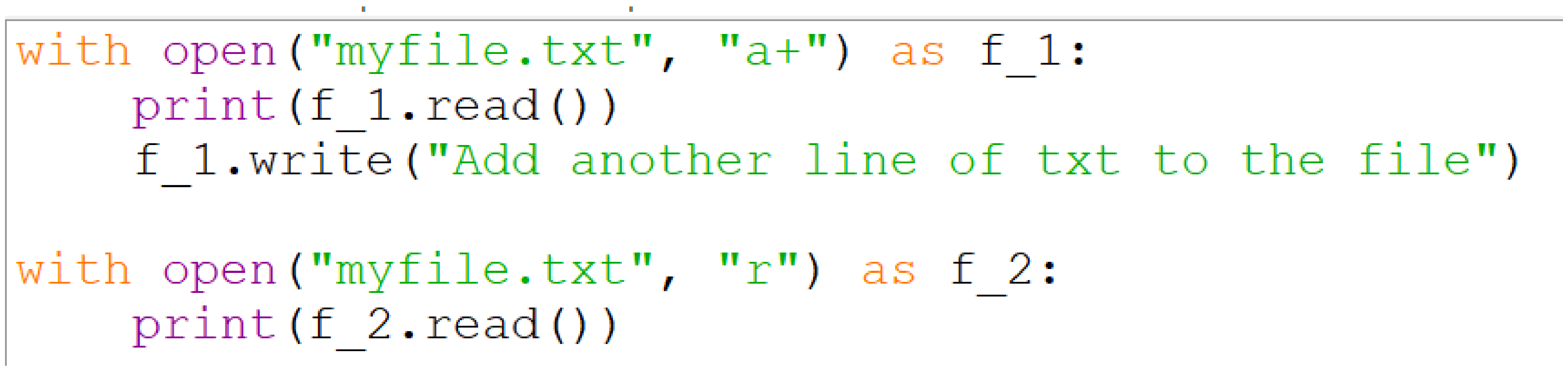
```
>>> f=open("myfile.txt","w")
>>> f.write("Oop! I have overwritten the content!")
>>> f.close()
>>> f=open("myfile.txt","r")
>>> print(f.read())
Oop! I have overwritten the content!
```

`With` statement

It is important to close the file once we have completed with it

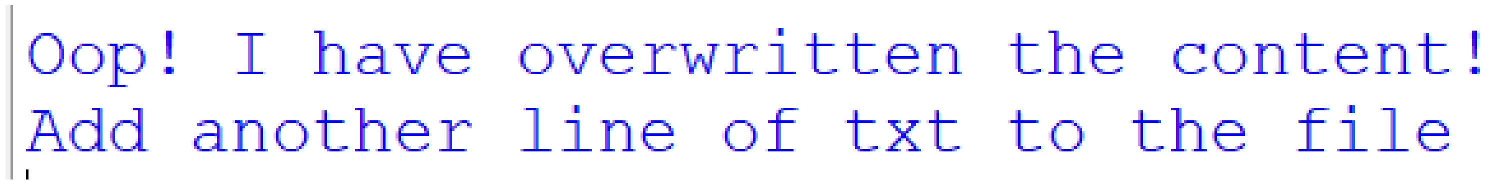
- free up the memory used by the file buffer
- ‘flushes’ the contents in the buffer to the file

Closing of the file can be automatically done by using the **with** keyword when opening the file



```
with open("myfile.txt", "a+") as f_1:
    print(f_1.read())
    f_1.write("Add another line of txt to the file")

with open("myfile.txt", "r") as f_2:
    print(f_2.read())
```



```
Oop! I have overwritten the content!
Add another line of txt to the file
```

Summary of Python File Handling

`“r”`	Open text file for reading. The stream is positioned at the beginning of the file.
`“r+”`	Open for reading and writing. The stream is positioned at the beginning of the file.
`“w”`	Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
`“w+”`	Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
`“a”`	Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file for subsequent writes.
`“a+”`	Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file for subsequent writes to.

CSV File

In practice, it is more likely that our data in a file will be stored based on certain predefined format.

CSV (Comma Separated Values) is a text-based format for storing data in a file

- common format use to store data in spreadsheets and databases

Data are stored as plain text (numbers and text) in tabular form

- each row represents one data record
- each consists of same number of fields, separated by comma

Example: contents of a CSV file **cars.csv**

```
id,brand,model,year,color
1,Mazada,CX5,2014,red
2,VW,Jetta,2018,silver
3,Honda,Civic,2022,white
```

First row of the csv file is the header

- consists of attributes' names corresponding to each column of the data records

CSV File Handling – reader()

Python provides a csv module with various functions

- to access the tabular entries in CSV file

We can use the csv.reader function to read and manipulate the content of the file

```
import csv
file = open("cars.csv")
cars = csv.reader(file)
header = []             #create an empty list
header = next(cars)     #read 1st record - header
print(header)

rows = []
for row in cars:
    print(row)
    rows.append(row)
print(rows)             #final entries in row
file.close
```

CSV File Handling - DictReader

For entries to be loaded into our program as dictionary:

```
import csv
cars = csv.DictReader(open("cars.csv"))
for row in cars:
    print(row)
```

Each iteration of the loop

- produces a dictionary in strings
  - keys = names of the columns
  - values = corresponding data from the row being read

```
{'id': '1', 'brand': 'Mazada', 'model': 'CX5', 'year': '2014', 'color ': 'red '}
{'id': '2', 'brand': 'VW', 'model': 'Jetta', 'year': '2018', 'color ': 'silver '}
{'id': '3', 'brand': 'Honda', 'model': 'Civic', 'year': '2022', 'color ': 'white '}
```

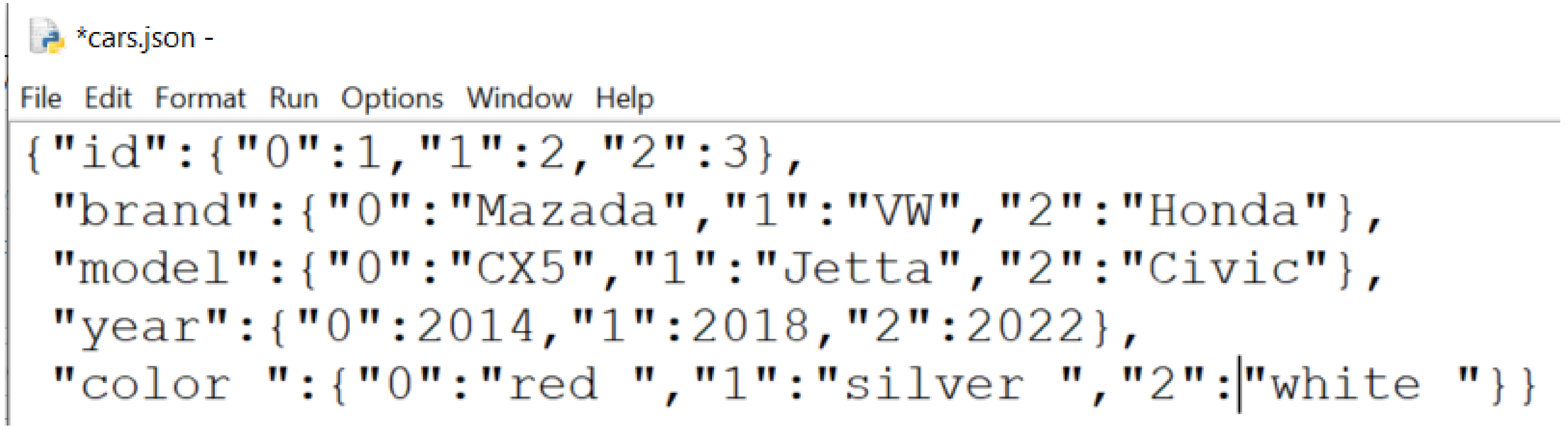
In practice, it is more common to process csv file using the Pandas module (see later)

JSON File

JSON (**J**ava**S**cript **O**bject **N**otation)

- another popular text file format
- commonly use for transferring data in server-client web applications
- more efficient than csv file for managing big data set

The file format stores the data in array of name-values pairs separated by comma

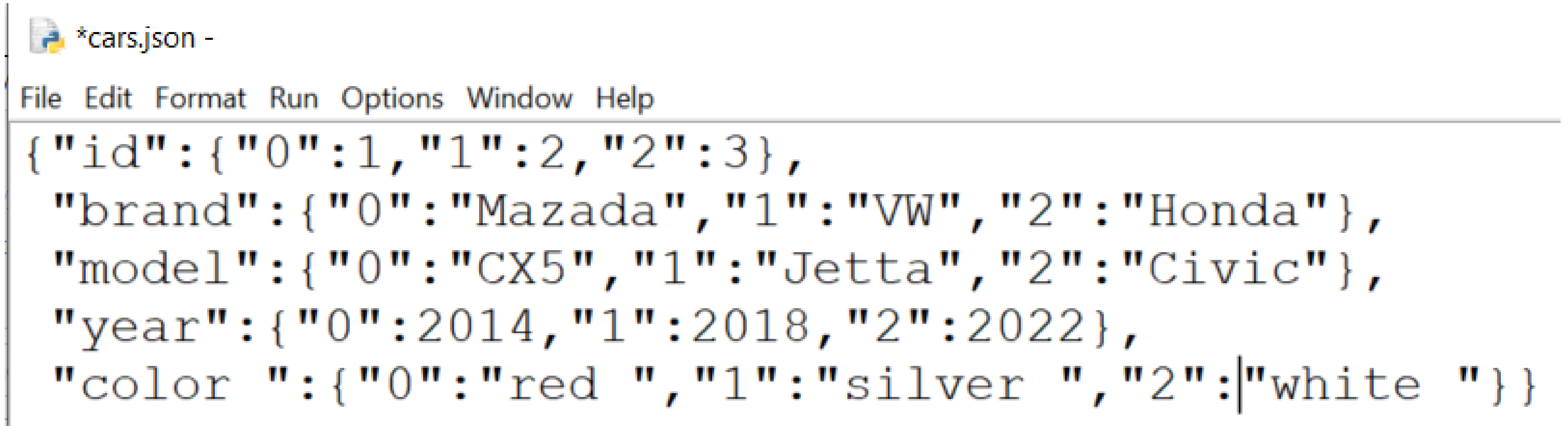


```
{"id":{"0":1,"1":2,"2":3},
 "brand":{"0":"Mazada","1":"VW","2":"Honda"},
 "model":{"0":"CX5","1":"Jetta","2":"Civic"},
 "year":{"0":2014,"1":2018,"2":2022},
 "color ":{"0":"red ","1":"silver ","2":"white "}}
```

JSON File

Data in a JSON file are grouped based on the attributes of the data

- id, brand, model, year, color



```
{"id":{"0":1,"1":2,"2":3},
 "brand":{"0":"Mazada","1":"VW","2":"Honda"},
 "model":{"0":"CX5","1":"Jetta","2":"Civic"},
 "year":{"0":2014,"1":2018,"2":2022},
 "color ":{"0":"red ","1":"silver ","2":"white "}}
```

Order of attributes is typically not significant in JSON file

- easy to extend it with additional attributes (and entries) while maintaining backward compatibility

JSON File Handling

To access data in the file (for further manipulation)

```
import json

# a JSON entry (a string)
x =  '{"brand":"Mazada", "model":"CX5",\
      "Year":2014, "color":"red"}'

y = json.loads(x)

print(f'Model = {y["model"]}')
print(f'Year = {y["Year"]}’)
```

But it is also easier to process json file through Pandas module (see later)