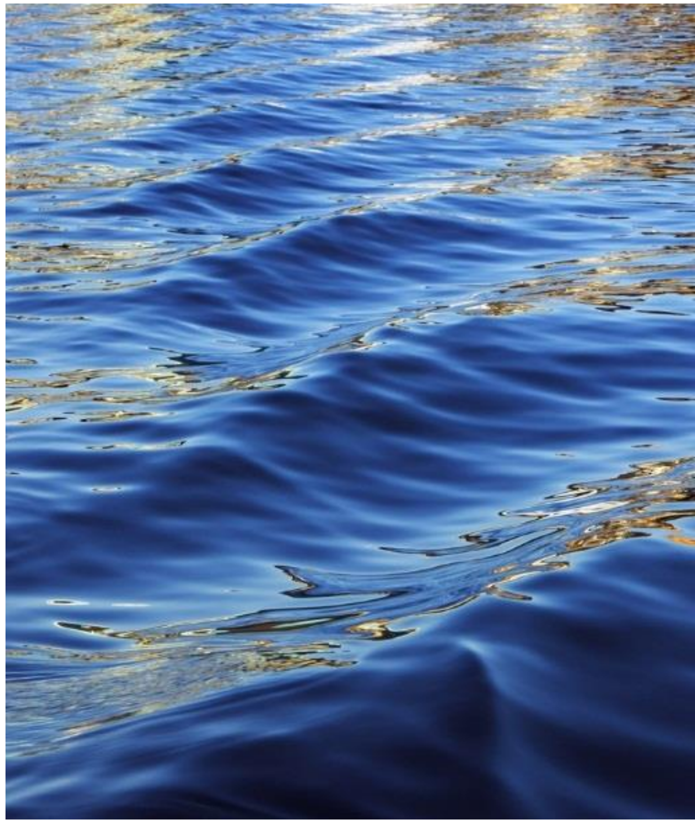
Working with Files in Python

Lecture 9

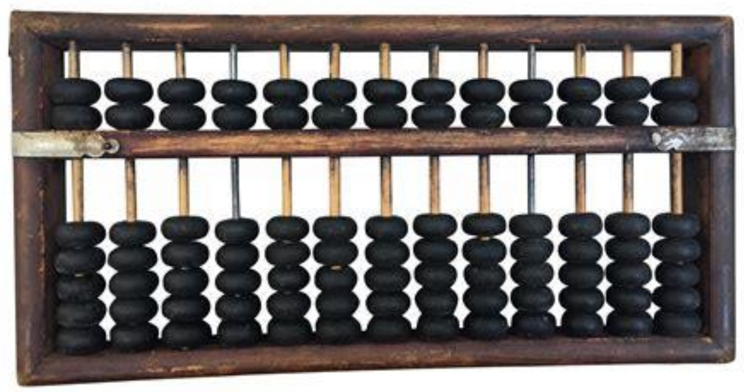


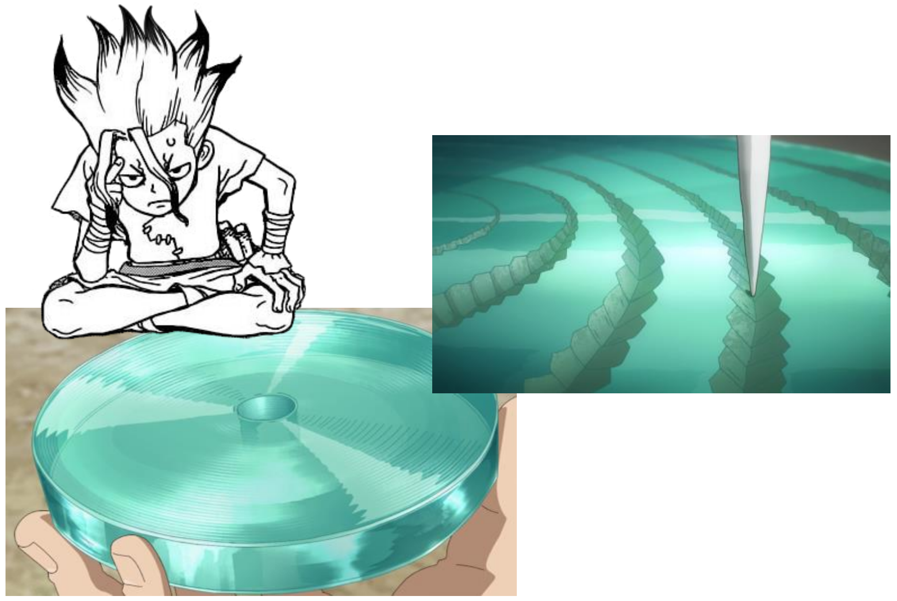
Files

Files are named locations on disk to store related information. They are used to permanently store data in a **non-volatile memory** (e.g., hard disk).

Since Random Access Memory (RAM) is **volatile** (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.





[1] Opening Files in Python

Python has a built-in **open()** function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt")    # open file in current directory
>>> f = open("C:/Python38/README.txt")  # specifying full path
```

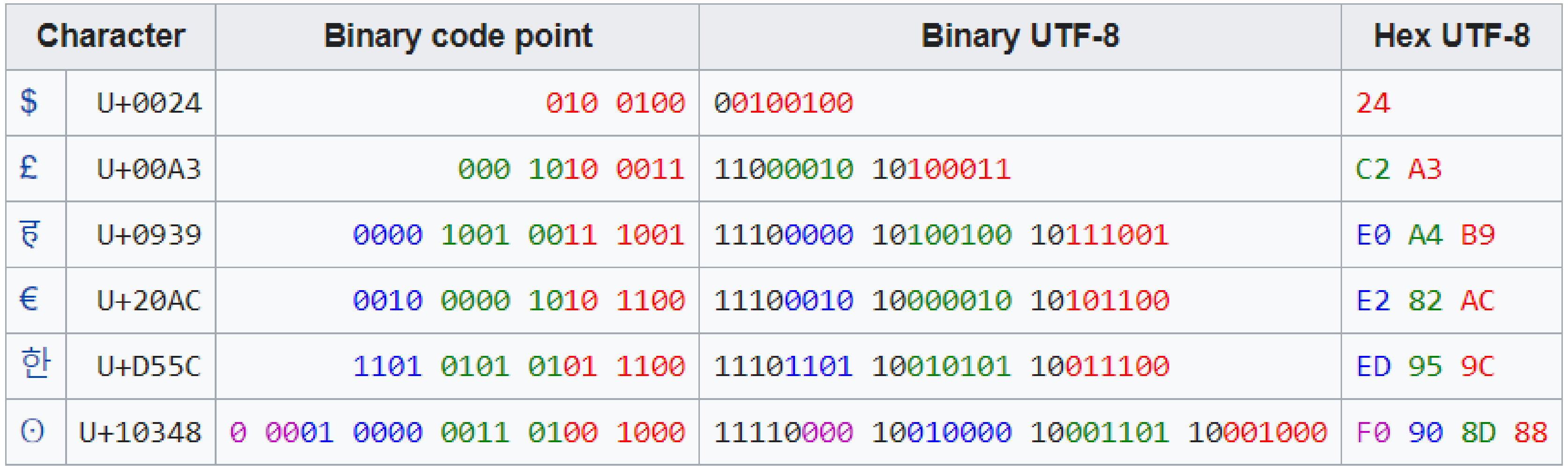
We can specify the mode while opening a file.

`"r"`	- Read - Default value. Opens a file for reading, error if the file does not exist
`"a"`	- Append - Opens a file for appending, creates the file if it does not exist
`"w"`	- Write - Opens a file for writing, creates the file if it does not exist
`"x"`	- Create - Creates the specified file, returns an error if the file exists

`"t"`	- Text - Default value. Text mode
`"b"`	- Binary - Binary mode (e.g. images)

[1] Opening Files in Python

Character encoding, the default encoding is platform dependent.
In the past... windows use **CP-1252 (windows-1252)** and Linux use **UTF-8**.

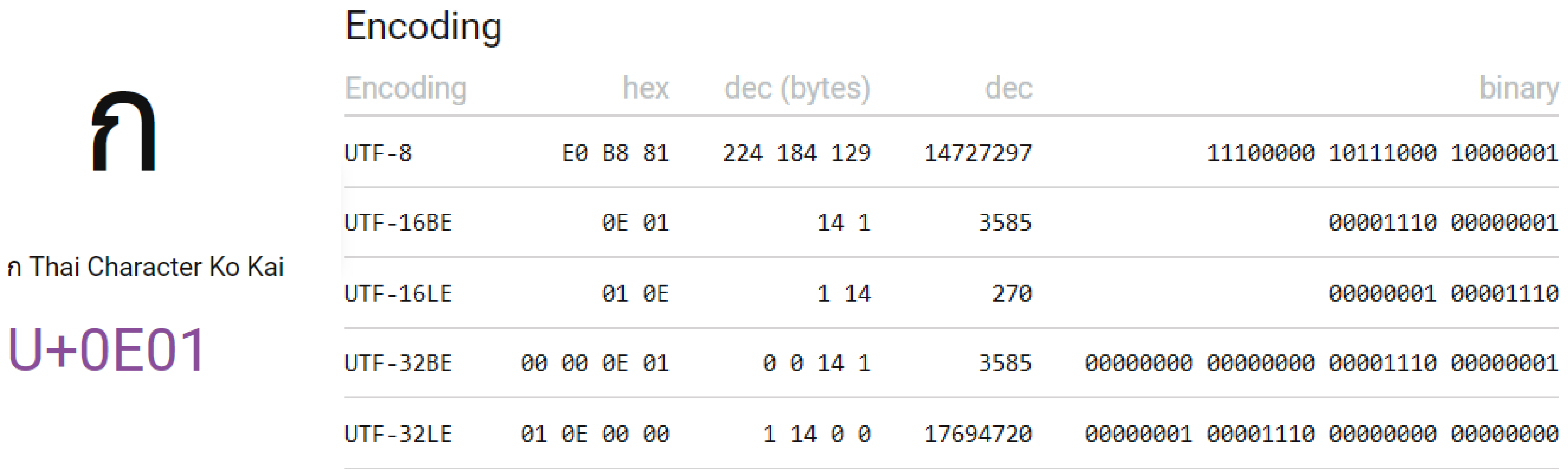


Character		Binary code point	Binary UTF-8	Hex UTF-8
$	U+0024	010 0100	00100100	24
£	U+00A3	000 1010 0011	11000010 10100011	C2 A3
ह	U+0939	0000 1001 0011 1001	11100000 10100100 10111001	E0 A4 B9
€	U+20AC	0010 0000 1010 1100	11100010 10000010 10101100	E2 82 AC
한	U+D55C	1101 0101 0101 1100	11101101 10010101 10011100	ED 95 9C
𐍈	U+10348	0 0001 0000 0011 0100 1000	11110000 10010000 10001101 10001000	F0 90 8D 88

UTF-8 : *Unicode* (or *Universal Coded Character Set*) *Transformation Format – 8-bit*

Now, most of them transition to **Unicode**

[1] Opening Files in Python



ก

ก Thai Character Ko Kai

U+0E01

Encoding

Encoding	hex	dec (bytes)	dec	binary
UTF-8	E0 B8 81	224 184 129	14727297	11100000 10111000 10000001
UTF-16BE	0E 01	14 1	3585	00001110 00000001
UTF-16LE	01 0E	1 14	270	00000001 00001110
UTF-32BE	00 00 0E 01	0 0 14 1	3585	00000000 00000000 00001110 00000001
UTF-32LE	01 0E 00 00	1 14 0 0	17694720	00000001 00001110 00000000 00000000

https://unicode-table.com/en/blocks/thai/

```
f = open("test.txt", mode='r', encoding='utf-8')
```

[2] Closing Files in Python

When we are done with performing operations on the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file.
It is done using the **close()** method available in Python.

```
f = open("test.txt", encoding = 'utf-8')
# perform file operations
f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a **try...finally** block.

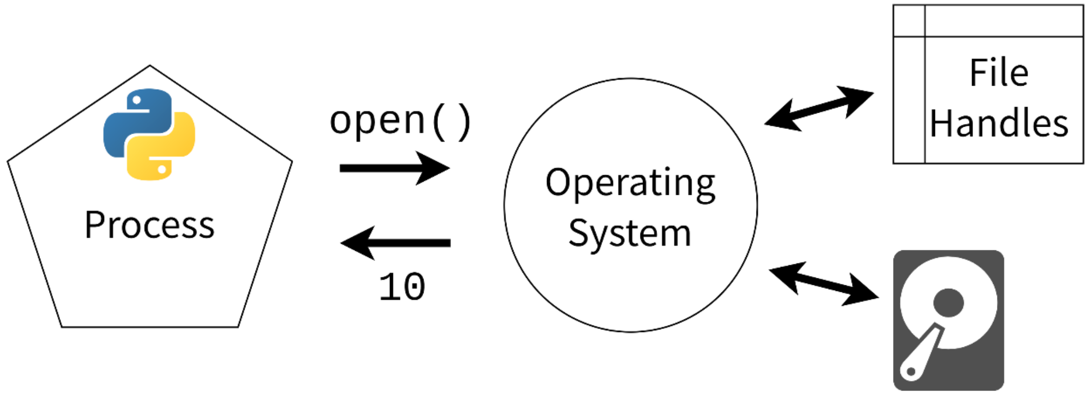
[2] Closing Files in Python

A safer way is to use a **try...finally** block.

```
try:
    f = open("test.txt", encoding = 'utf-8')
    # perform file operations
finally:
    f.close()
```

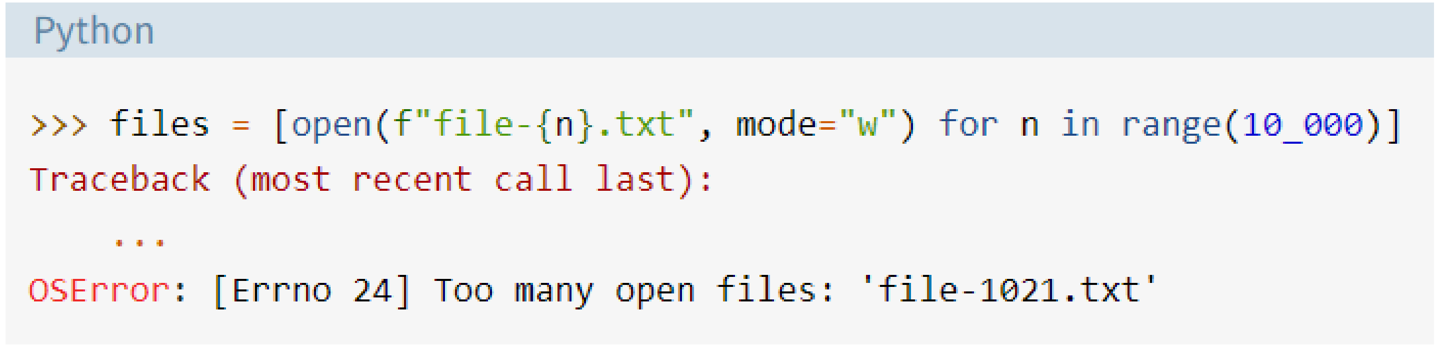
This way, we are guaranteeing that the file is properly closed even if an exception is raised that causes program flow to stop.

Operating systems limit the number of open files any single process can have.



[2] Closing Files in Python

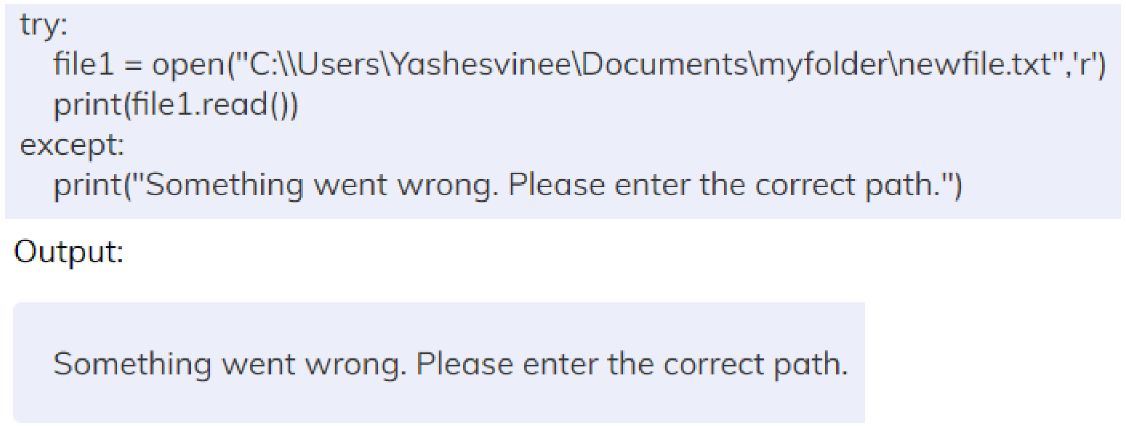
(Good to Know) What Happens When You Open Too Many Files?



Python

```
>>> files = [open(f"file-{n}.txt", mode="w") for n in range(10_000)]
Traceback (most recent call last):
    ...
OSError: [Errno 24] Too many open files: 'file-1021.txt'
```

(Good to Know) When You Open File that Not Exist?

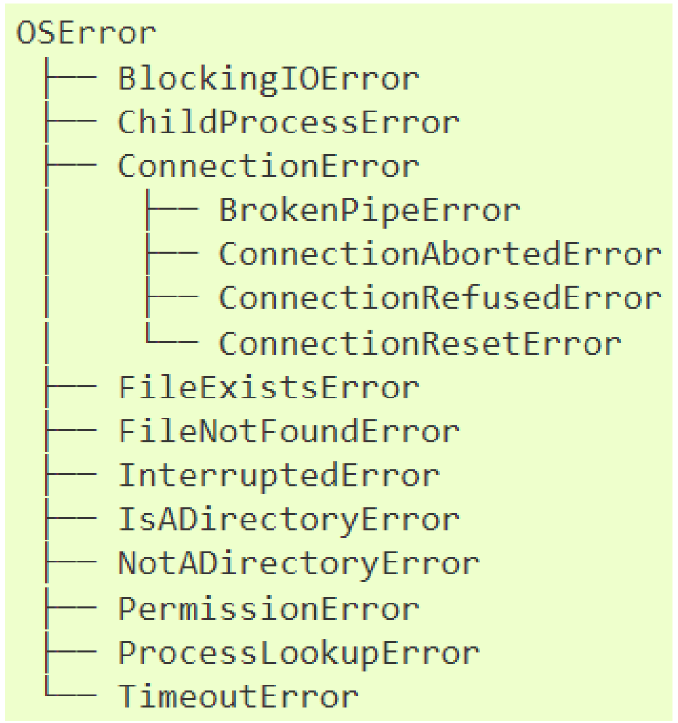


```
try:
   file1 = open("C:\\Users\Yashesvinee\Documents\myfolder\newfile.txt",'r')
   print(file1.read())
except:
   print("Something went wrong. Please enter the correct path.")
```

Output:

```
Something went wrong. Please enter the correct path.
```

try...except



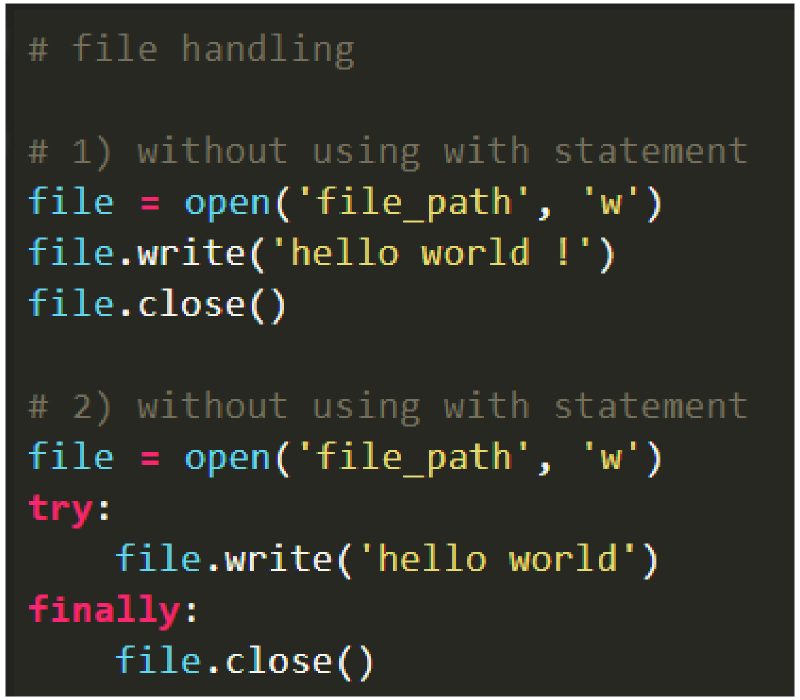
```
OSError
 ├── BlockingIOError
 ├── ChildProcessError
 ├── ConnectionError
 │    ├── BrokenPipeError
 │    ├── ConnectionAbortedError
 │    ├── ConnectionRefusedError
 │    └── ConnectionResetError
 ├── FileExistsError
 ├── FileNotFoundError
 ├── InterruptedError
 ├── IsADirectoryError
 ├── NotADirectoryError
 ├── PermissionError
 ├── ProcessLookupError
 └── TimeoutError
```

https://docs.python.org/3/library/exceptions.html

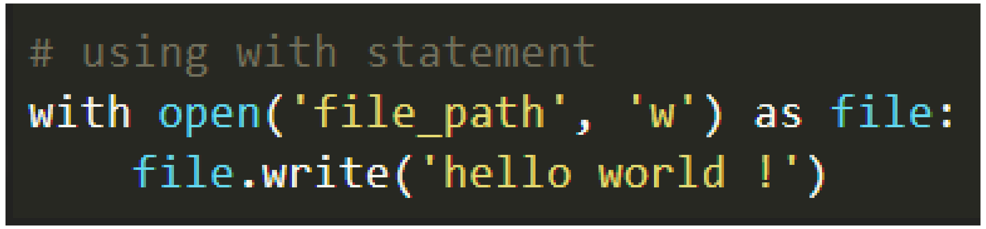
[2] Closing Files in Python

(Good to Know) [with] statement in Python

In Python, **with** statement is used in **exception handling** to make the code cleaner and much more readable. It simplifies the management of common resources like file streams.



```python
# file handling

# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```



```python
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

[3] Writing to Files in Python

In order to write into a file in Python,

we need to open it in write **w**, append **a**, exclusive creation **x** **mode**

[Warning] Write mode (W) will overwrite into the file if it already exists

```python
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

`"x"` - Create - will create a file, returns an error if the file exist

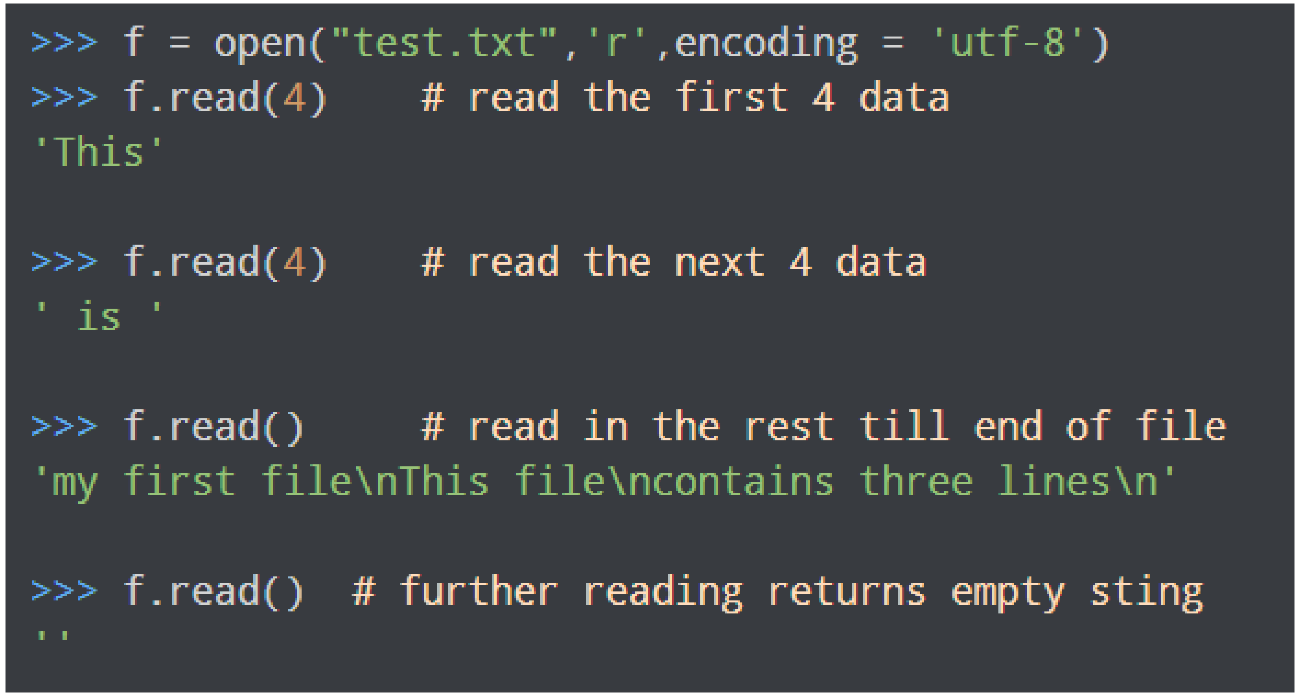
`"a"` - Append - will create a file if the specified file does not exist

`"w"` - Write - will create a file if the specified file does not exist

[4] Reading Files in Python

To read a file in Python, we must open the file in reading **r mode.**

There are various methods available for this purpose. We can use the **read(size)** method to read in the size amount of data. If the size parameter is not specified, it reads and returns up to the end of the file.

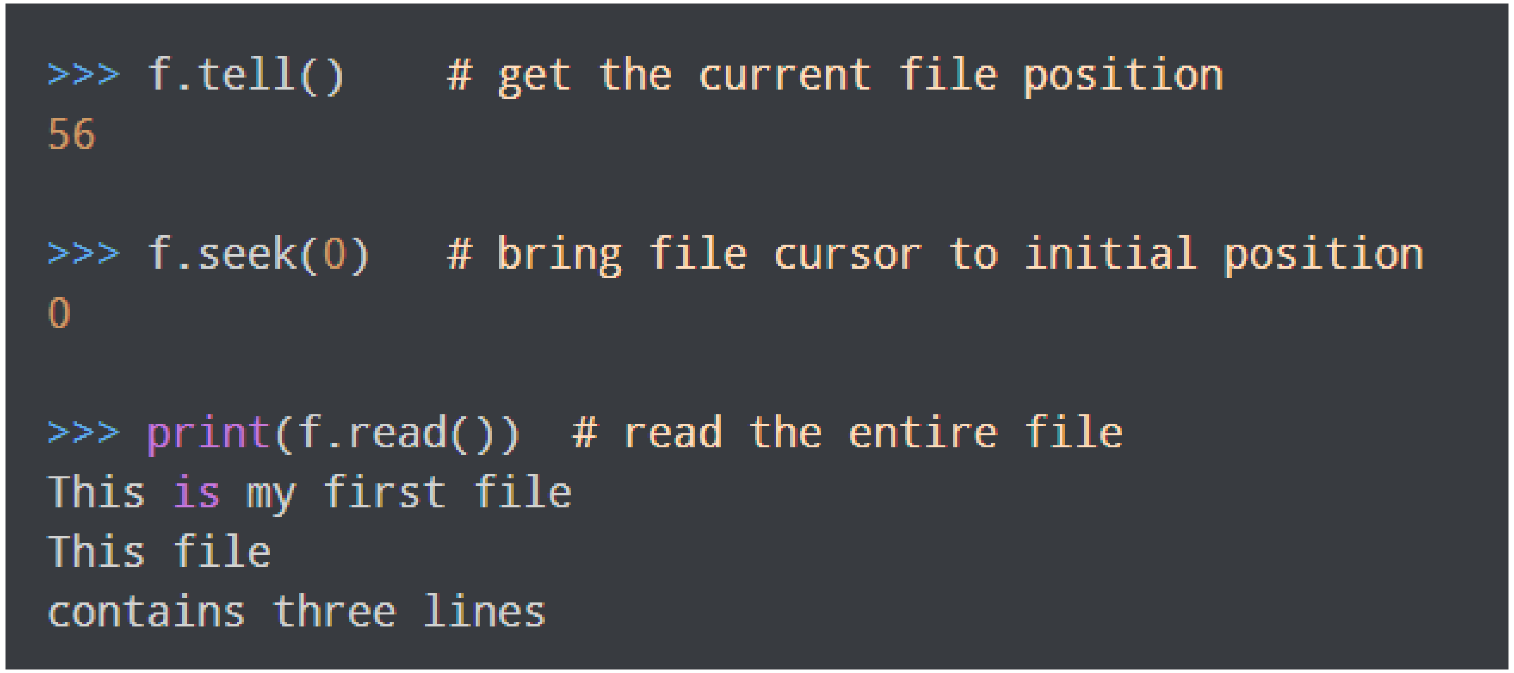


```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4)    # read the first 4 data
'This'

>>> f.read(4)    # read the next 4 data
' is '

>>> f.read()     # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'

>>> f.read()  # further reading returns empty sting
''
```

[4] Reading Files in Python

We can change our current file cursor (position) using the **seek()** method.

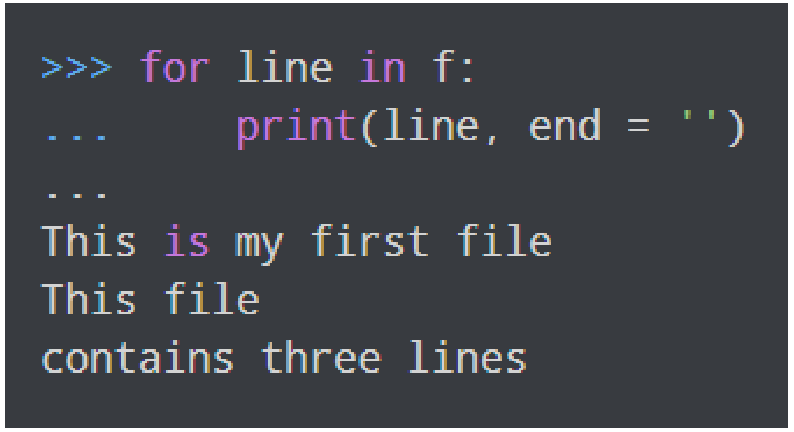
Similarly, the **tell()** method returns our current position **(in number of bytes)**.



```
>>> f.tell()    # get the current file position
56

>>> f.seek(0)   # bring file cursor to initial position
0

>>> print(f.read())  # read the entire file
This is my first file
This file
contains three lines
```

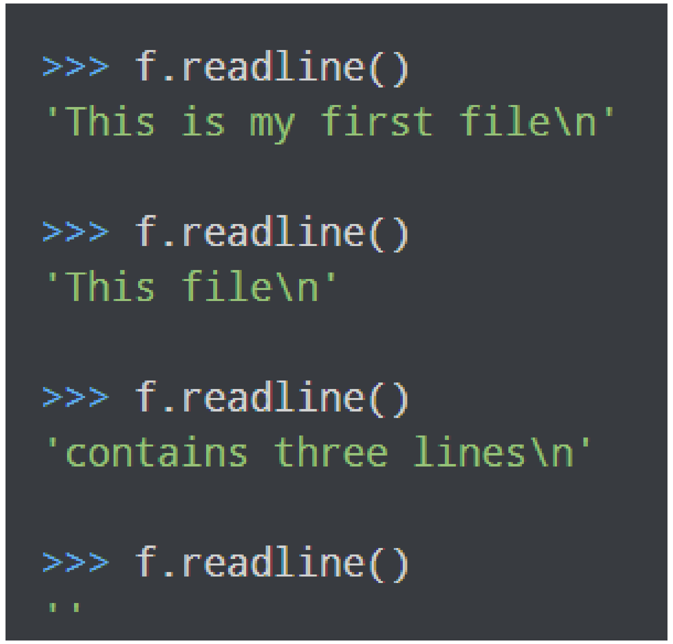
[4] Reading Files in Python

We can read a file line-by-line using a for loop. This is both efficient and fast.



```
>>> for line in f:
...     print(line, end = '')
...
This is my first file
This file
contains three lines
```

We can use the **readline()** method to read individual lines of a file.



```
>>> f.readline()
'This is my first file\n'

>>> f.readline()
'This file\n'

>>> f.readline()
'contains three lines\n'

>>> f.readline()
''
```

[5] Renaming files

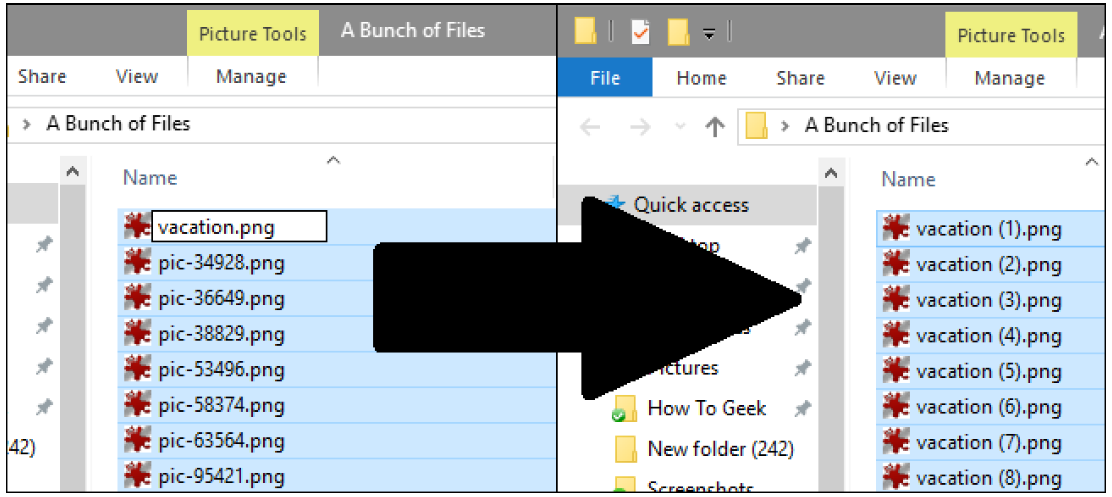
Python **OS** module provides methods that help you perform file-processing operations

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

```
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```



Batch Renaming

[6] Remove files (Delete)

You can use the ***remove()*** method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

```
import os

# Delete file test2.txt
os.remove("text2.txt")
```

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

[7] Directories in Python

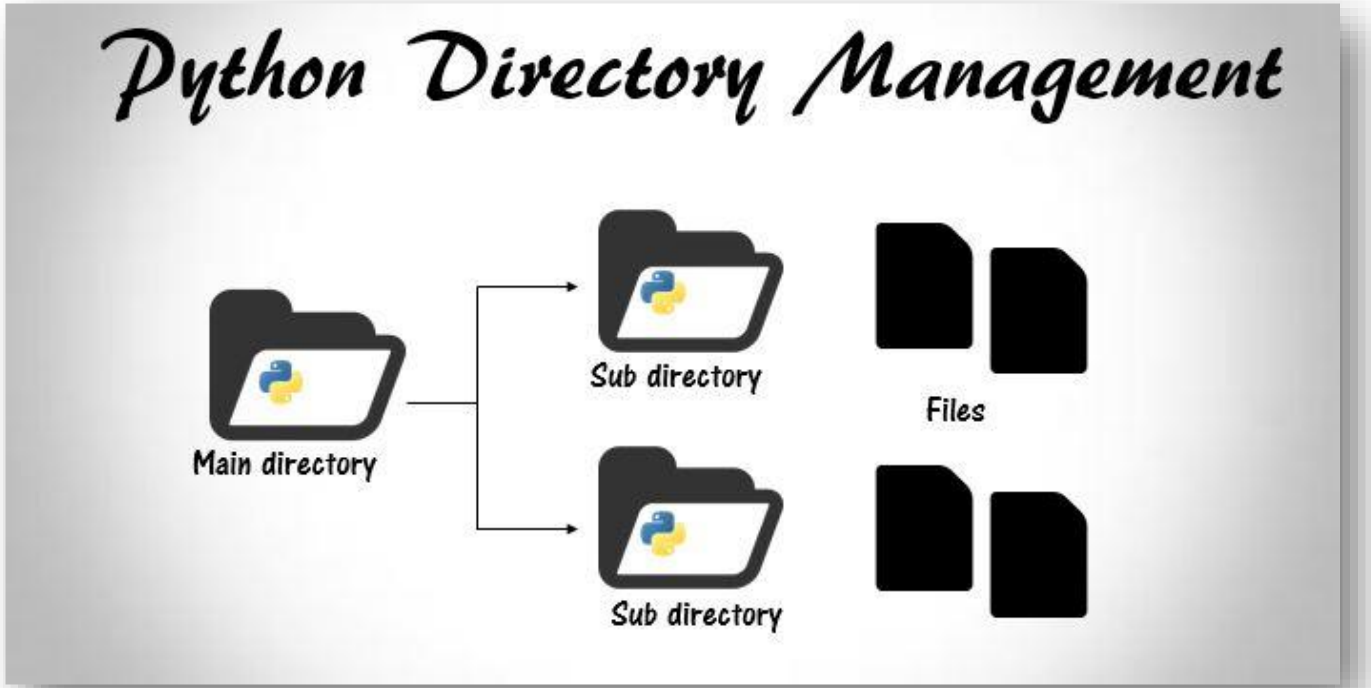
Create Folder. You can use the ***mkdir()*** method of the **os** module to create directories in the current directory.

Syntax

```
os.mkdir("newdir")
```

Example

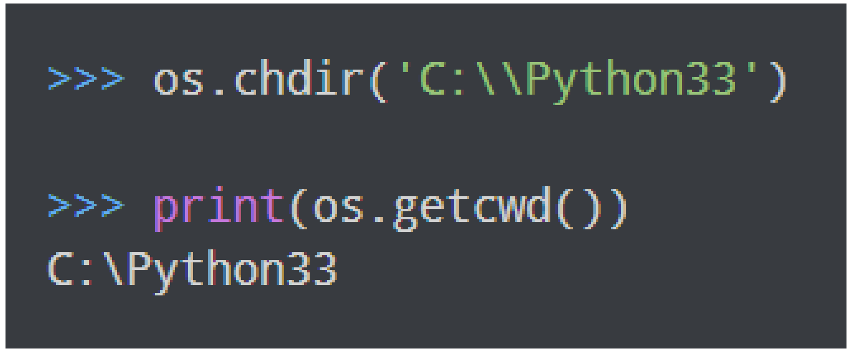
```
import os

# Create a directory "test"
os.mkdir("test")
```



[7] Directories in Python

Get Current Directory and Change Current Directory

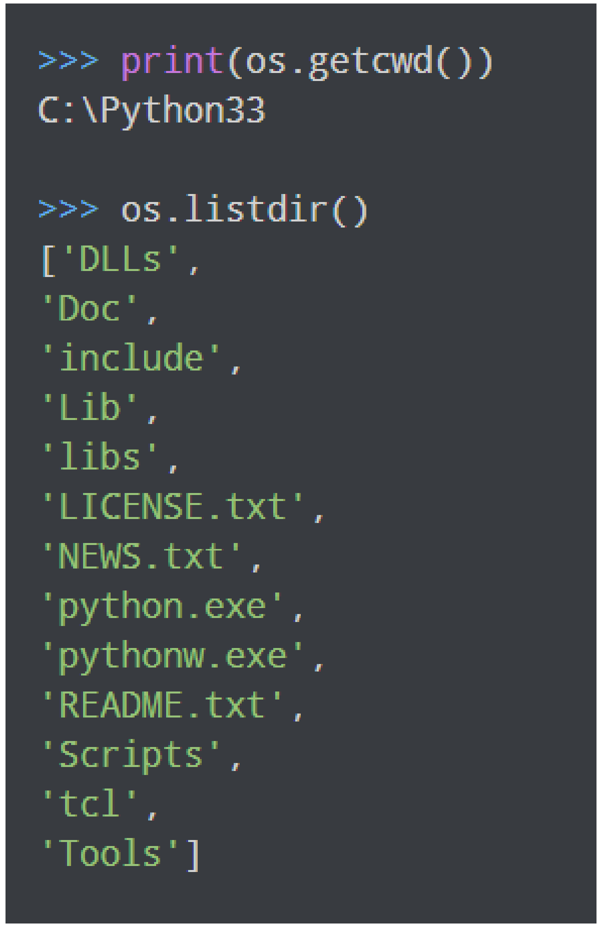
- We can get the present working directory using the **getcwd()** method.
- We can change the current working directory by using the **chdir()** method.



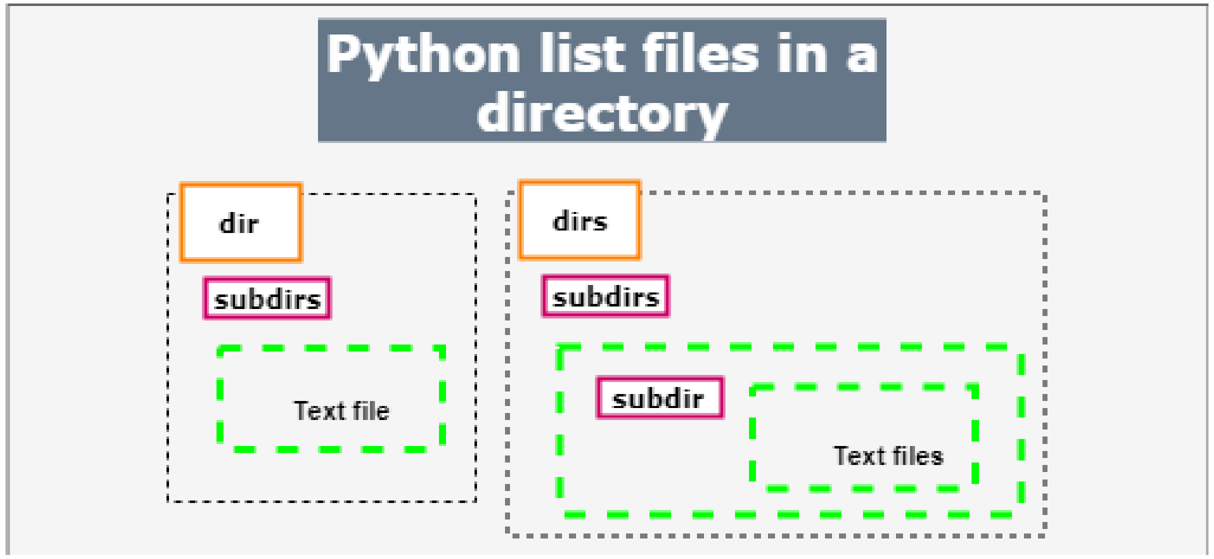
```
>>> os.chdir('C:\\Python33')

>>> print(os.getcwd())
C:\Python33
```

[7] Directories in Python

List directories and files



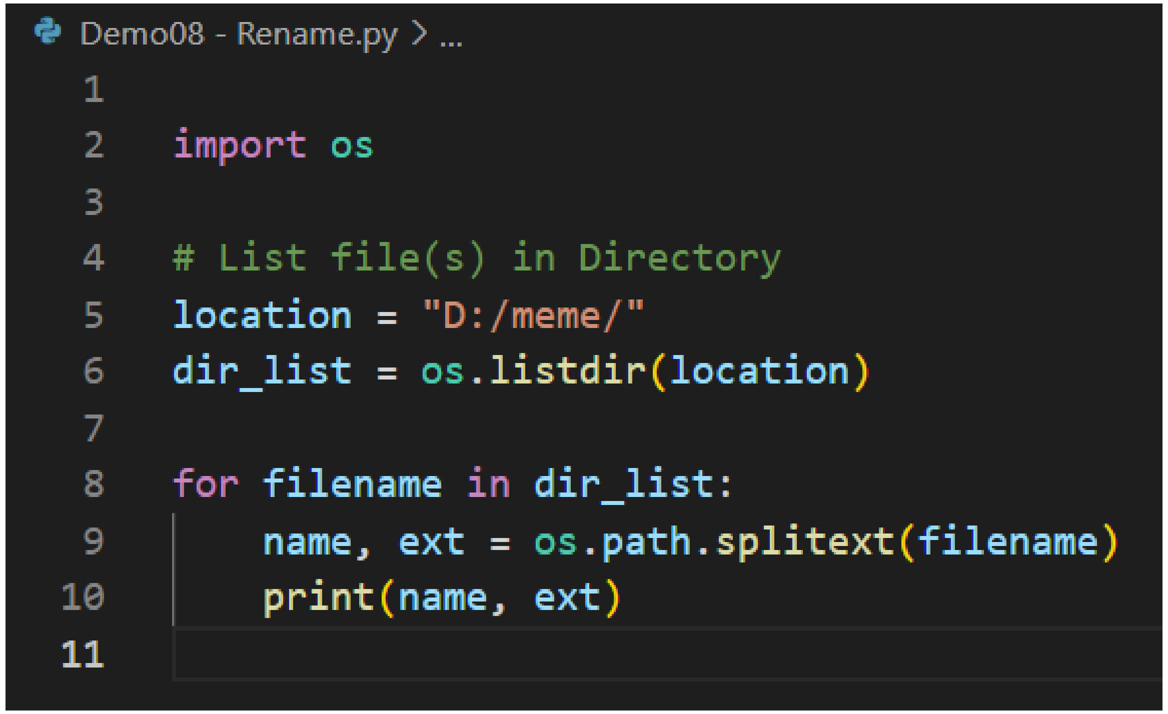
```
>>> print(os.getcwd())
C:\Python33

>>> os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
'tcl',
'Tools']
```



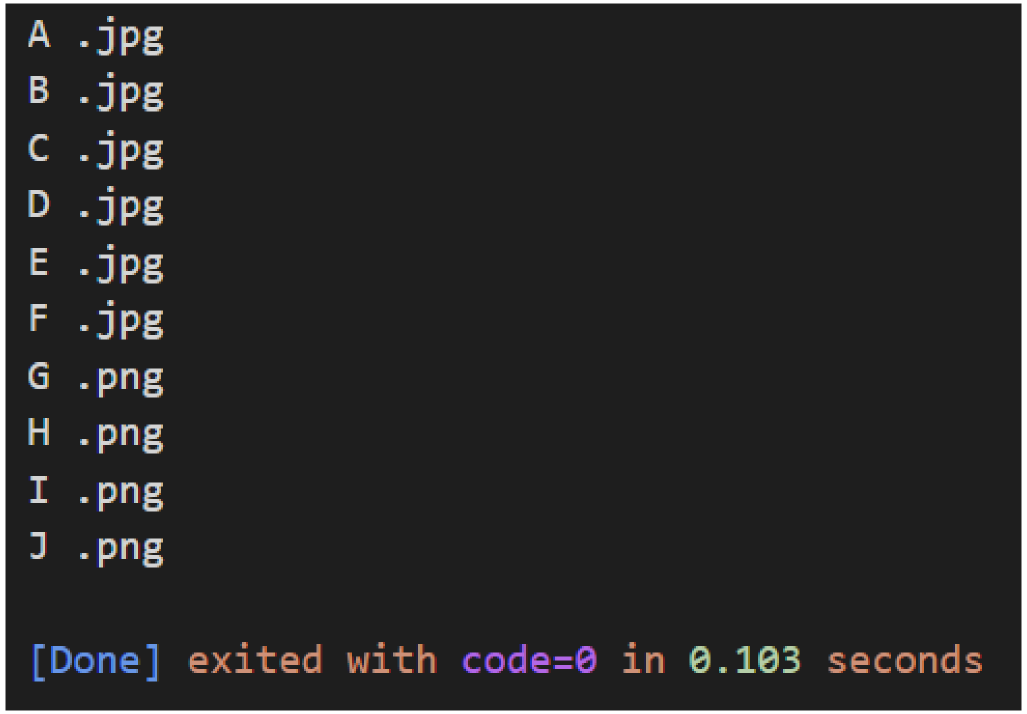
https://techbeamers.com/python-list-all-files-directory/

[8] Get extension of file

How to get the extension of file (file type)



```
Demo08 - Rename.py > ...

import os

# List file(s) in Directory
location = "D:/meme/"
dir_list = os.listdir(location)

for filename in dir_list:
    name, ext = os.path.splitext(filename)
    print(name, ext)

```



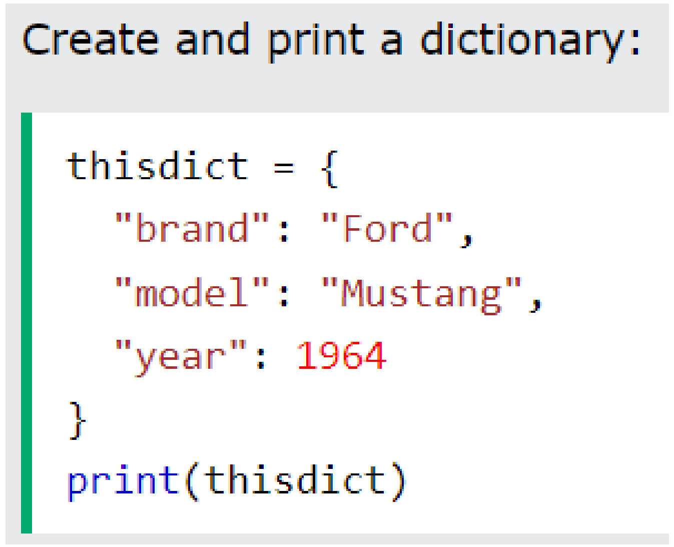
```
A .jpg
B .jpg
C .jpg
D .jpg
E .jpg
F .jpg
G .png
H .png
I .png
J .png

[Done] exited with code=0 in 0.103 seconds
```

Dictionary (Python data structure)

Dictionaries are used to store data values in **key:value pairs**.

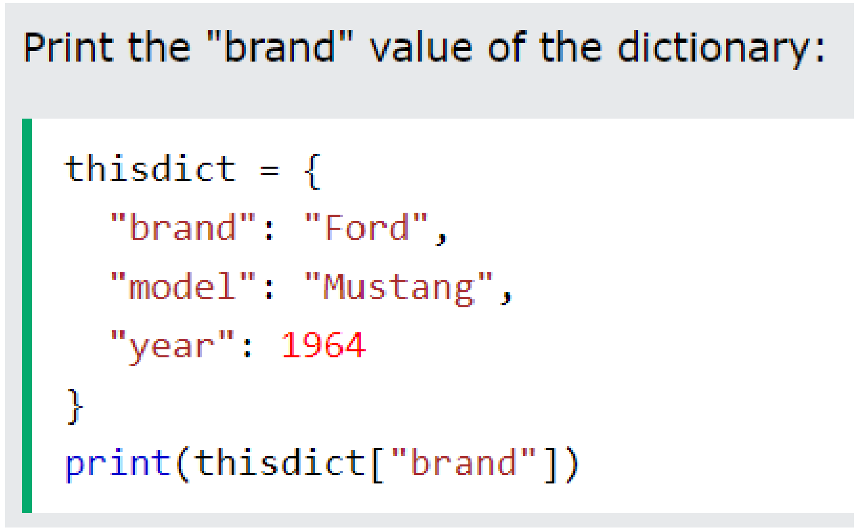
A dictionary is a collection which is **ordered**, changeable and **do not allow duplicates**.

Dictionaries are written with **curly brackets**, and have keys and values:



Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```



Print the "brand" value of the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

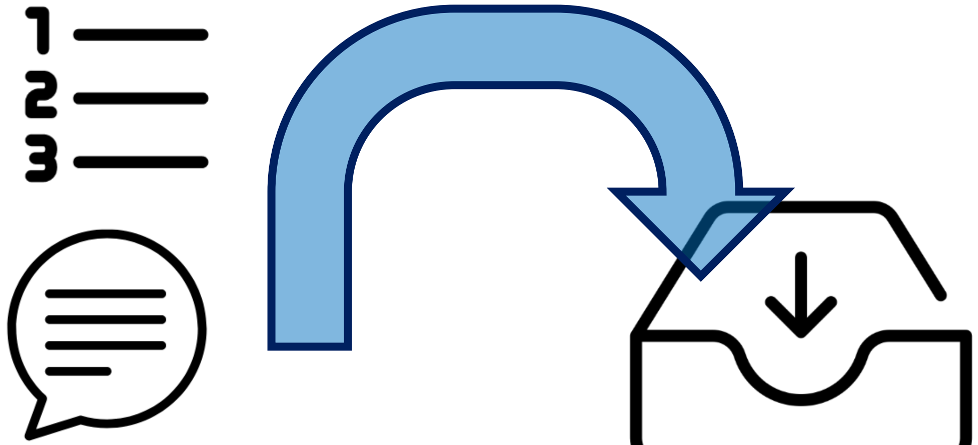
Dictionary Items

JSON with Python

JSON is text, written with **J**ava**S**cript **O**bject **N**otation

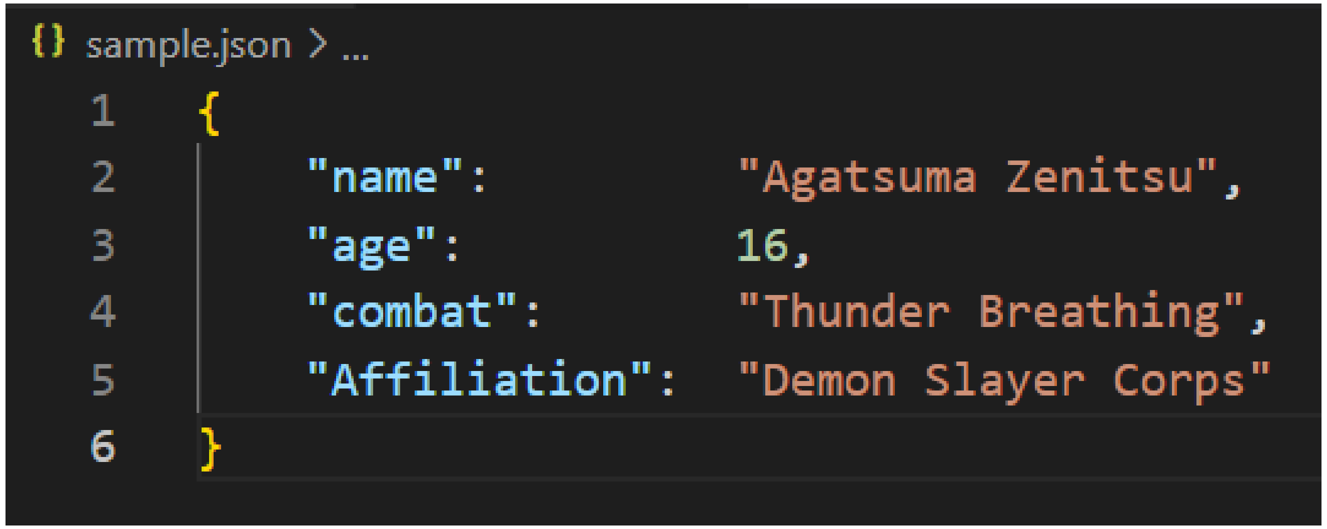
- ❑ JSON is a lightweight format for storing and transporting data
- ❑ JSON is **"self-describing"** and easy to understand
- ❑ JSON is often used when data is sent from a server to a web page

JSON



JSON Structure

- ❑ JSON format uses curly brackets to mark objects.
- ❑ JSON files **do not contain header section**.
- ❑ Each JSON entry consists of a **name**-**value** pair.

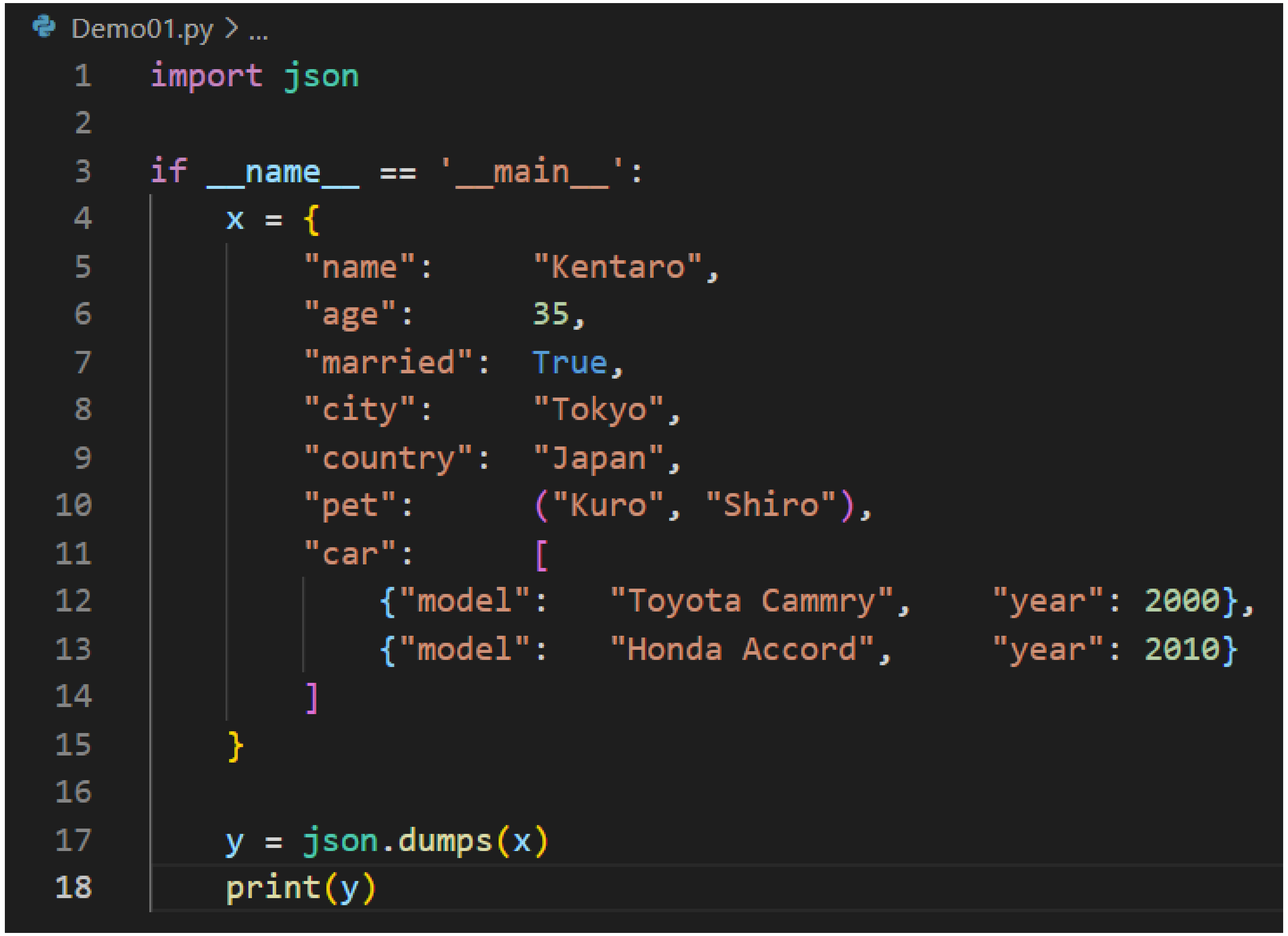


sample.json > ...

```
{
    "name":         "Agatsuma Zenitsu",
    "age":          16,
    "combat":       "Thunder Breathing",
    "Affiliation":  "Demon Slayer Corps"
}
```

Variable **names** **must be** quoted.

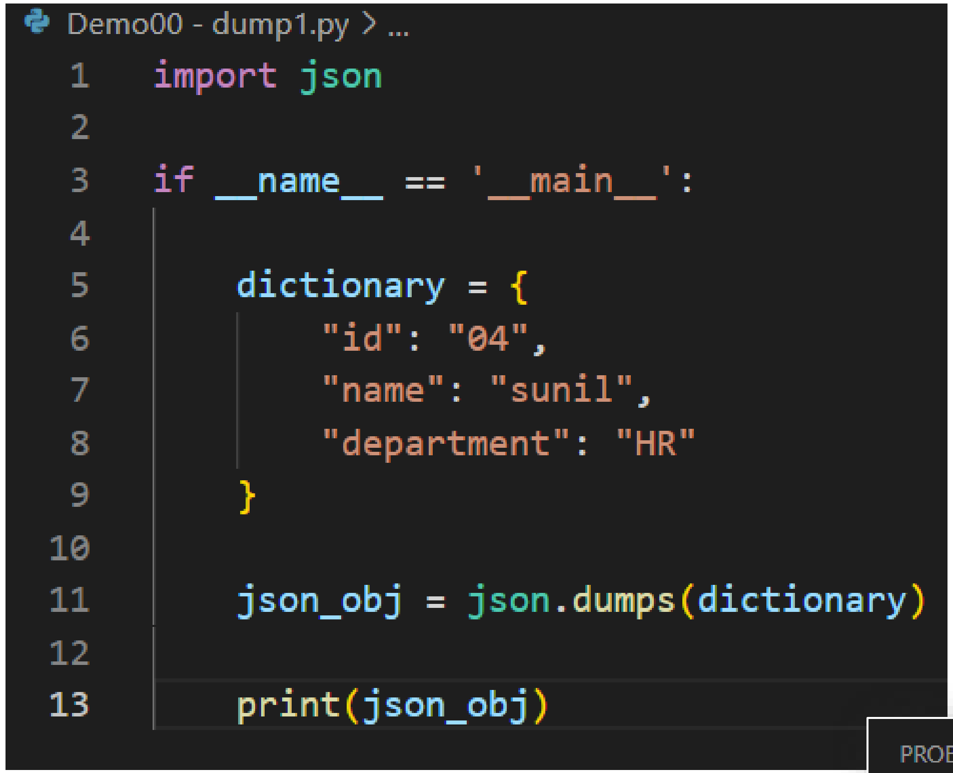
JSON Convert Python ⇔ JSON



Demo01.py > ...

```python
import json

if __name__ == '__main__':
    x = {
        "name":     "Kentaro",
        "age":      35,
        "married":  True,
        "city":     "Tokyo",
        "country":  "Japan",
        "pet":      ("Kuro", "Shiro"),
        "car":      [
            {"model":   "Toyota Cammry",    "year": 2000},
            {"model":   "Honda Accord",     "year": 2010}
        ]
    }

    y = json.dumps(x)
    print(y)
```

JSON Convert Python ⇔ JSON



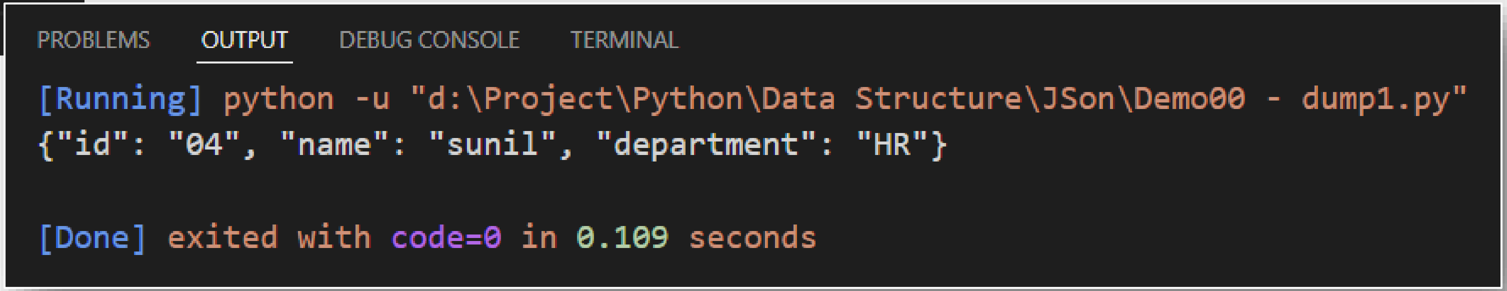
Demo00 - dump1.py > ...

```python
import json

if __name__ == '__main__':

    dictionary = {
        "id": "04",
        "name": "sunil",
        "department": "HR"
    }

    json_obj = json.dumps(dictionary)

    print(json_obj)
```

Dump: การถ่ายโอนข้อมูล, เท, โละทิ้ง

Python	JSON
None	null
str	String
int	Number
float	Number

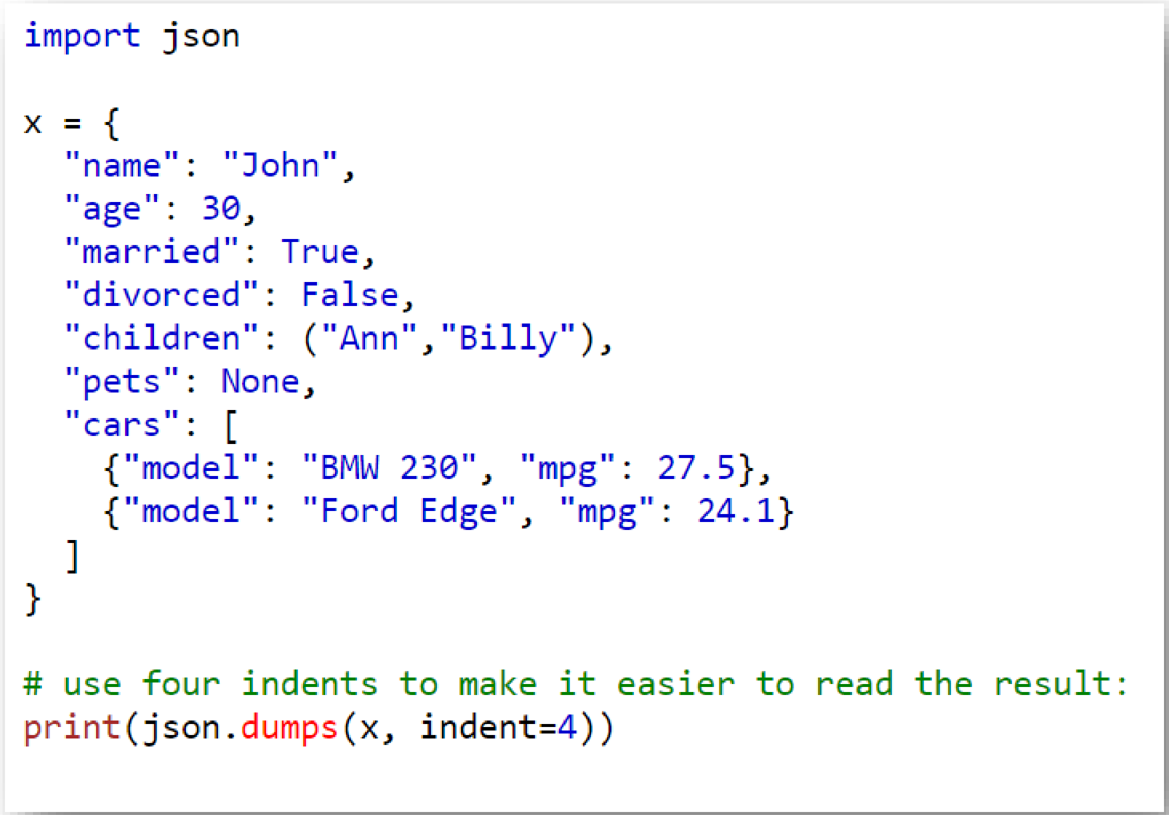
Python	JSON
dict	Object
list	Array
tuple	Array
Boolean (True / False)	true / false



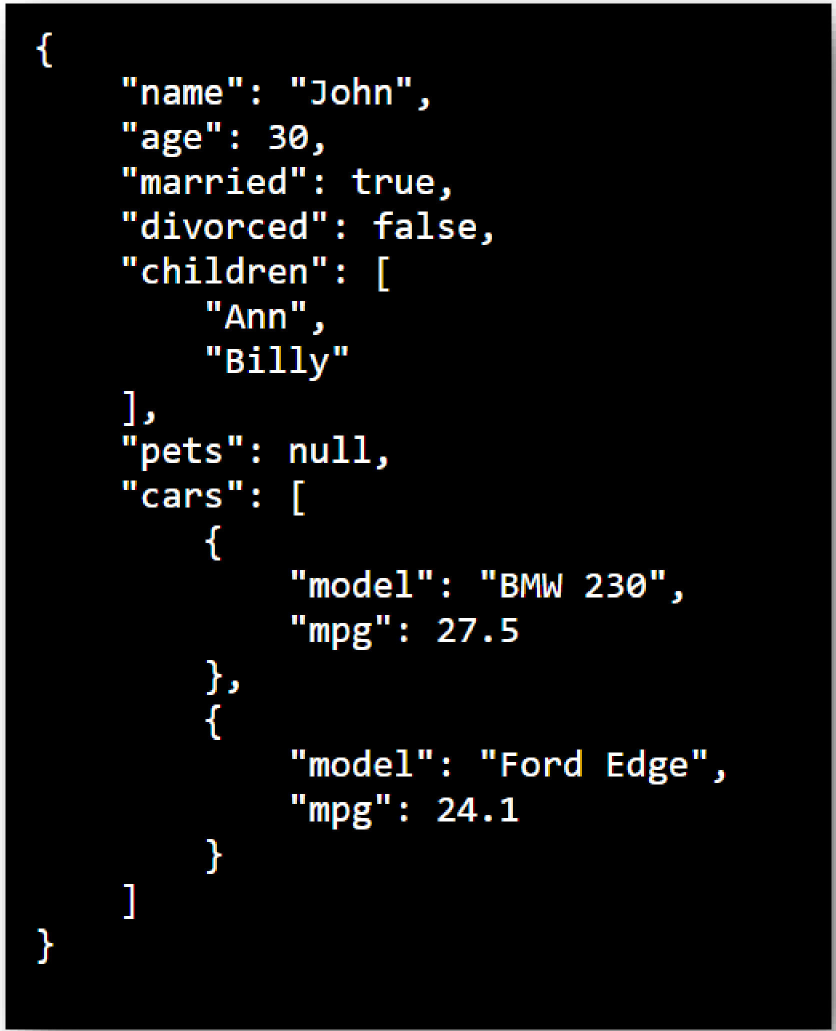
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[Running] python -u "d:\Project\Python\Data Structure\JSon\Demo00 - dump1.py"
{"id": "04", "name": "sunil", "department": "HR"}

[Done] exited with code=0 in 0.109 seconds
```

JSON Convert Python ⇔ JSON



```
import json

x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}

# use four indents to make it easier to read the result:
print(json.dumps(x, indent=4))
```

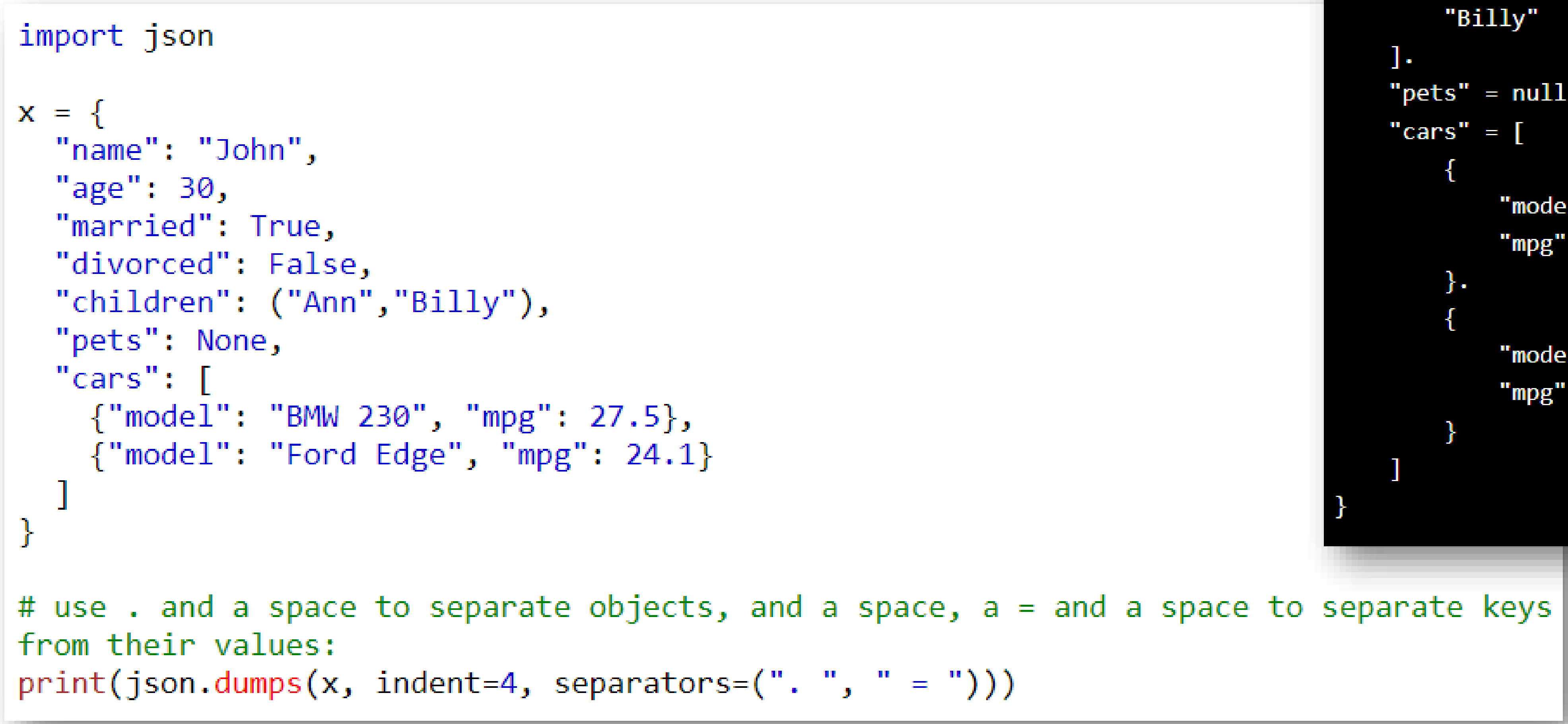


```
{
    "name": "John",
    "age": 30,
    "married": true,
    "divorced": false,
    "children": [
        "Ann",
        "Billy"
    ],
    "pets": null,
    "cars": [
        {
            "model": "BMW 230",
            "mpg": 27.5
        },
        {
            "model": "Ford Edge",
            "mpg": 24.1
        }
    ]
}
```

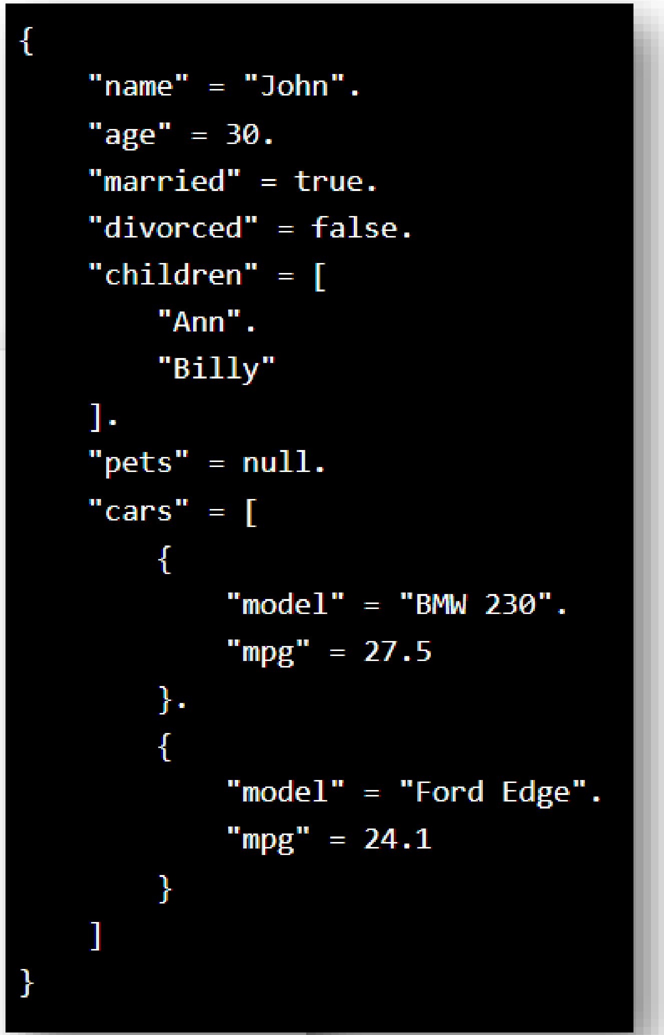
Indent: การเยื้อง

(Result Format)

JSON Convert Python ⇔ JSON

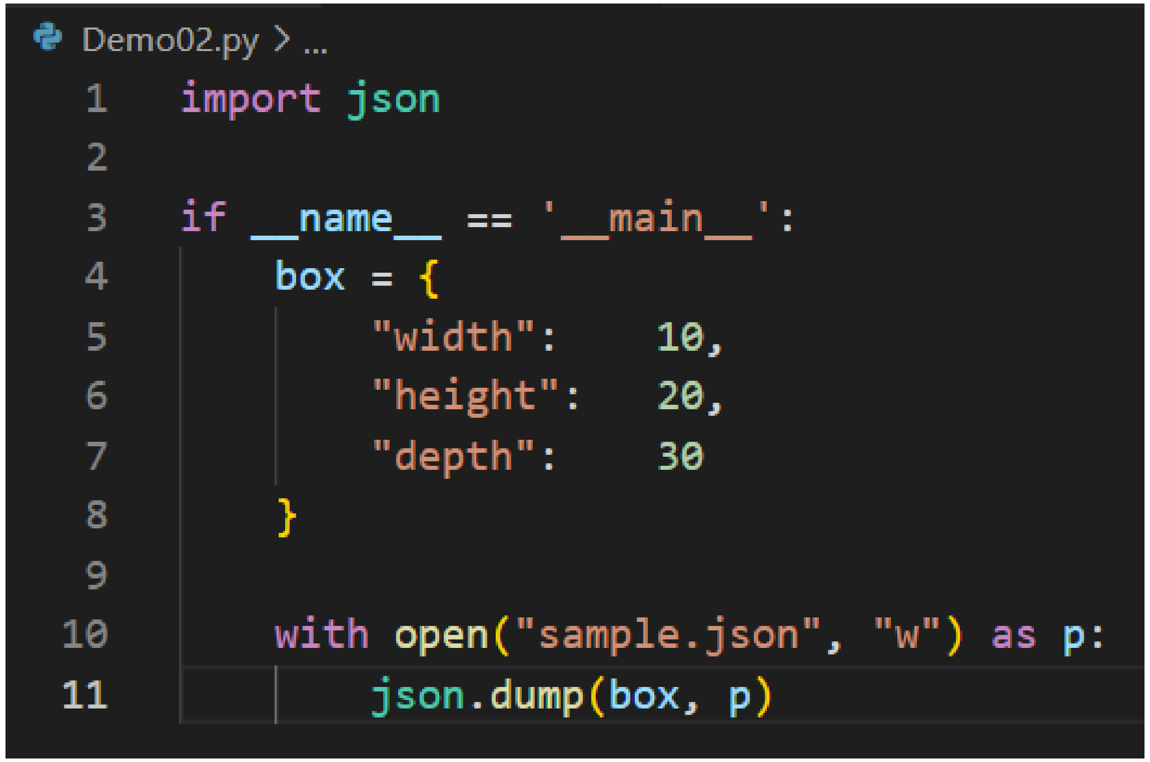


```python
import json

x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}

# use . and a space to separate objects, and a space, a = and a space to separate keys from their values:
print(json.dumps(x, indent=4, separators=(". ", " = ")))
```

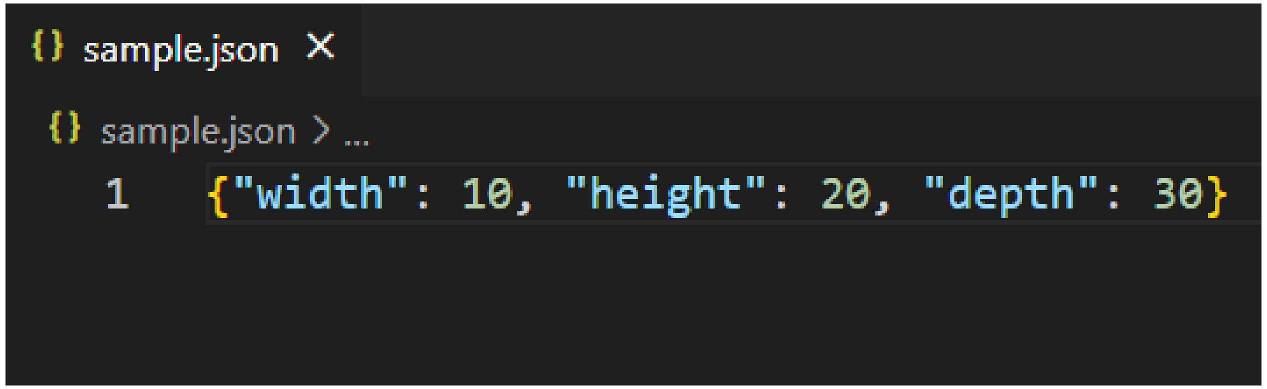


```
{
    "name" = "John".
    "age" = 30.
    "married" = true.
    "divorced" = false.
    "children" = [
        "Ann".
        "Billy"
    ].
    "pets" = null.
    "cars" = [
        {
            "model" = "BMW 230".
            "mpg" = 27.5
        }.
        {
            "model" = "Ford Edge".
            "mpg" = 24.1
        }
    ]
}
```

JSON Convert Python ⇔ JSON


Demo02.py > ...

```python
import json

if __name__ == '__main__':
    box = {
        "width":    10,
        "height":   20,
        "depth":    30
    }

    with open("sample.json", "w") as p:
        json.dump(box, p)
```


sample.json

sample.json > ...

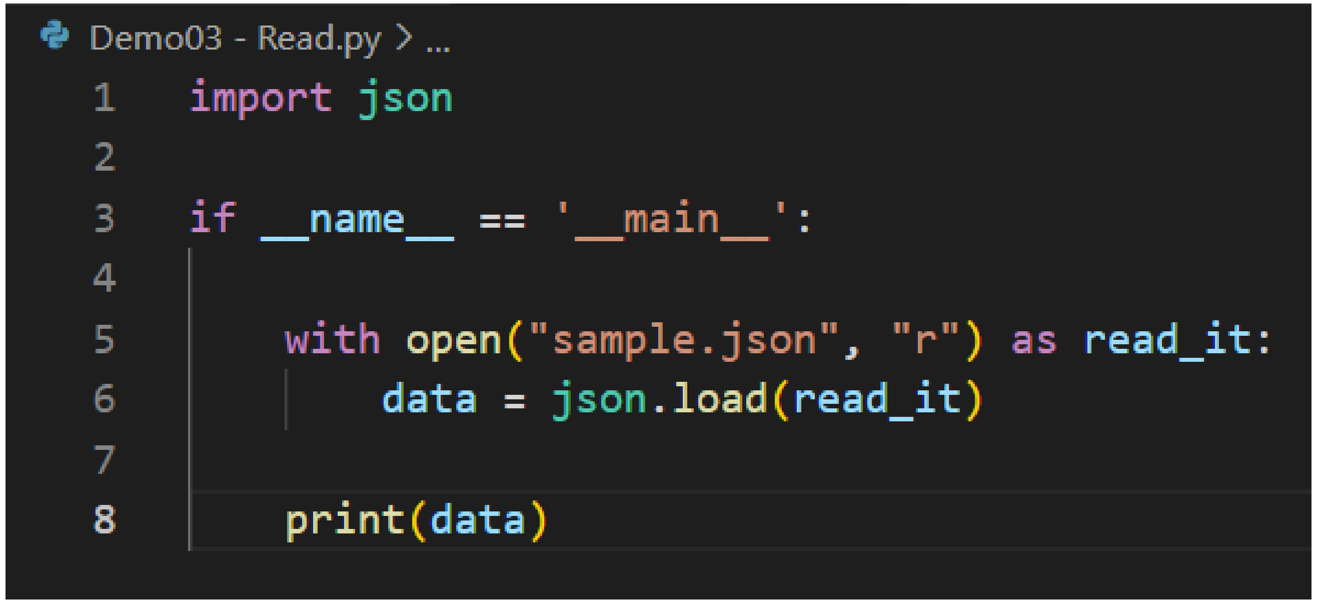
```json
{"width": 10, "height": 20, "depth": 30}
```

We should see "sample.json" file in current directory

Write to File

(Serializing JSON)

JSON Convert Python ⇔ JSON

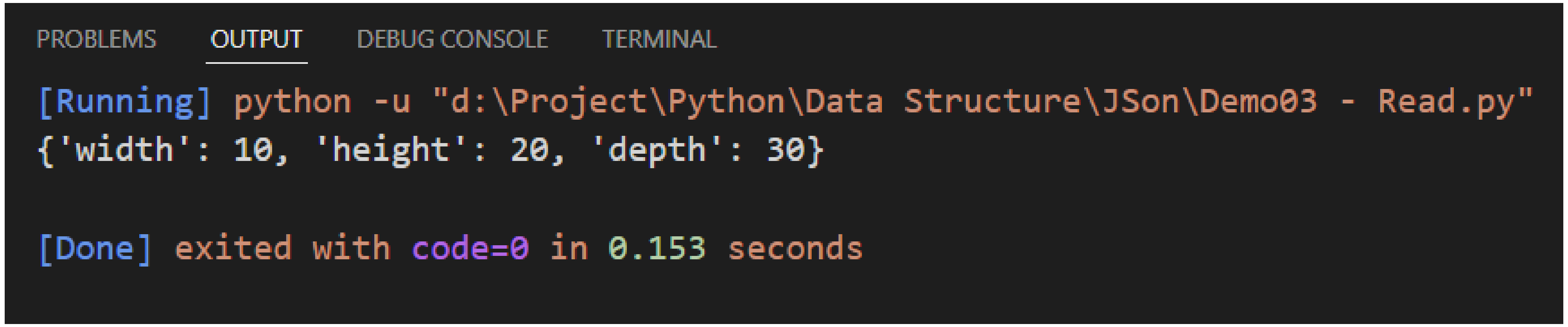


Demo03 - Read.py > ...

```python
import json

if __name__ == '__main__':

    with open("sample.json", "r") as read_it:
        data = json.load(read_it)

    print(data)
```

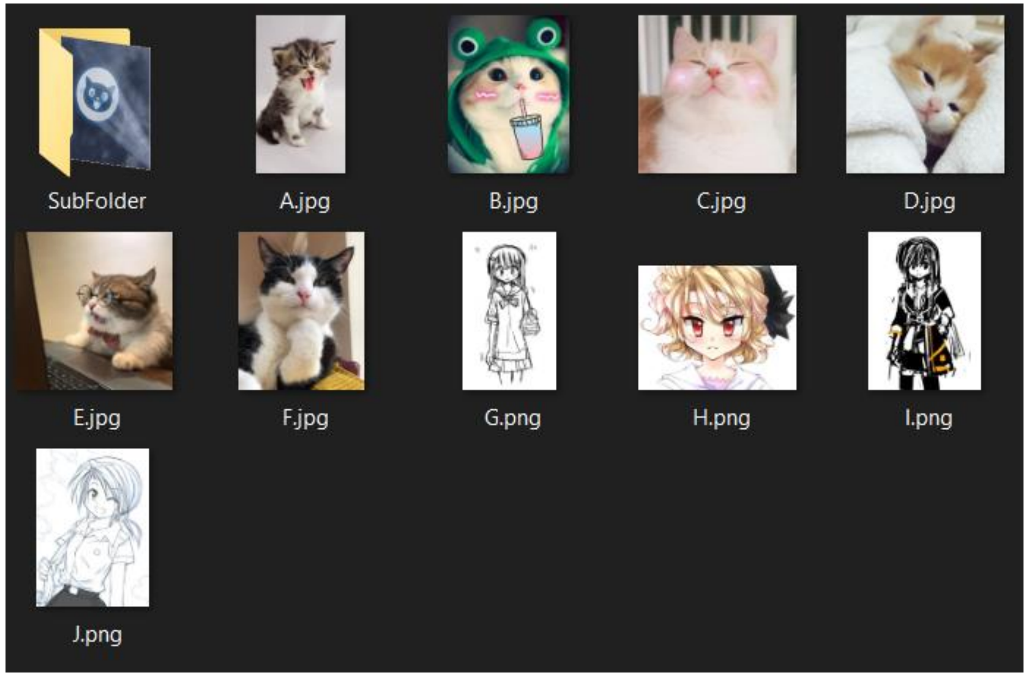
Read from File

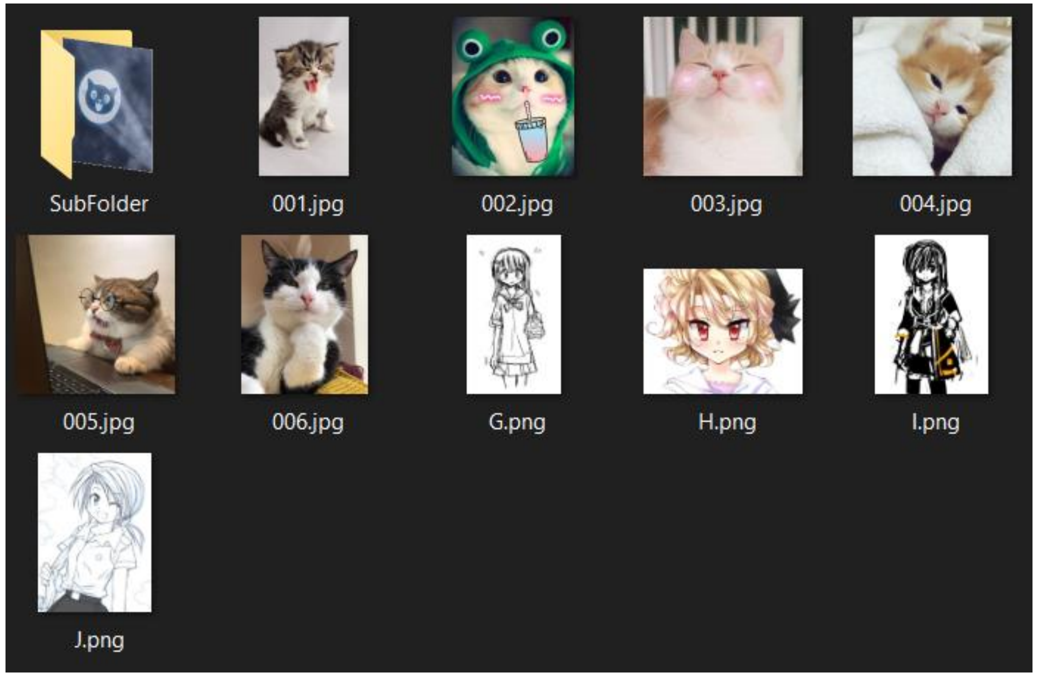
(Deserializing JSON)



```
[Running] python -u "d:\Project\Python\Data Structure\JSon\Demo03 - Read.py"
{'width': 10, 'height': 20, 'depth': 30}

[Done] exited with code=0 in 0.153 seconds
```

Workshop





สร้าง Function สำหรับ Rename file ชนิดที่กำหนด (เช่น .jpg) ใน Directory

ให้กลายเป็นตัวเลขเรียงลำดับ 001.jpg, 002.jpg, 003.jpg,...

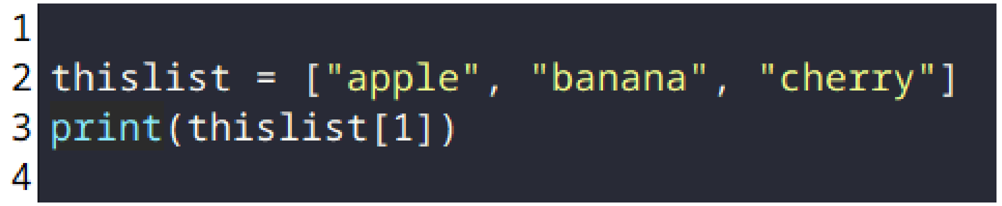
List

List เป็นโครงสร้างข้อมูลที่ใช้เก็บข้อมูลหลาย Items ไว้ในตัวแปรเพียงตัวแปรเดียว

- ❑ List เป็น **1 ใน 4 โครงสร้างข้อมูลพื้นฐาน**ที่ติดมากับภาษา Python
- ❑ อีก 3 โครงสร้างพื้นฐานที่เหลือคือ **Tuple, Set, Dictionary**
- ❑ ข้อมูลที่อยู่ภายใน List **สามารถซ้ำกันได้** โดยแต่ละตัวมีตำแหน่งคงที่เสมอ
- ❑ ข้อมูลที่อยู่ภายใน List **สามารถเปลี่ยนแปลงได้ (เพิ่ม / ลบ / แก้ไข)**
- ❑ ข้อมูลแต่ละตัวภายใน List **สามารถมีชนิดของข้อมูล (Type) ต่างกันได้**

List

การเข้าถึงข้อมูลภายใน List (Access Items)



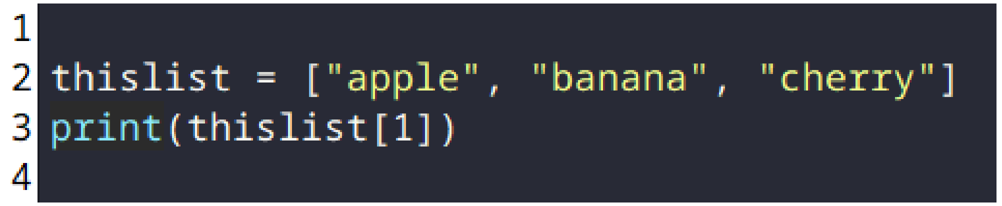
```python

thislist = ["apple", "banana", "cherry"]
print(thislist[1])

```

- ❑ ลำดับของ Item เริ่มต้นที่ 0
- ❑ ลำดับของ Item ที่มีค่าติดลบ เช่น –1 จะหมายถึงไล่ลำดับจากหลังสุดของ List
- ❑ สามารถเข้าถึงข้อมูลมากกว่าหนึ่งตำแหน่งได้ เช่น `print(thislist[2:5])` ดึงตำแหน่งที่ 3 – 5
- ❑ สามารถใช้สัญลักษณ์ (:) ในการกำหนดช่วงได้ เช่น `print(thislist[:4])`
- ❑ ตรวจสอบว่ามีข้อมูลอยู่ใน List หรือไม่ โดยใช้

```python
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

List

การเพิ่ม Item เข้าไปใน List



```python

thislist = ["apple", "banana", "cherry"]
print(thislist[1])

```

- ❑ Append ใช้เพื่อนำ item ใหม่ ไปต่อท้ายของ List
- ❑ Insert ใช้เพื่อนำ item ไปใส่ในตำแหน่งที่ระบุ

```python
thislist.append("orange")
print(thislist)
```

```python
thislist.insert(2, "orange")
print(thislist)
```

List

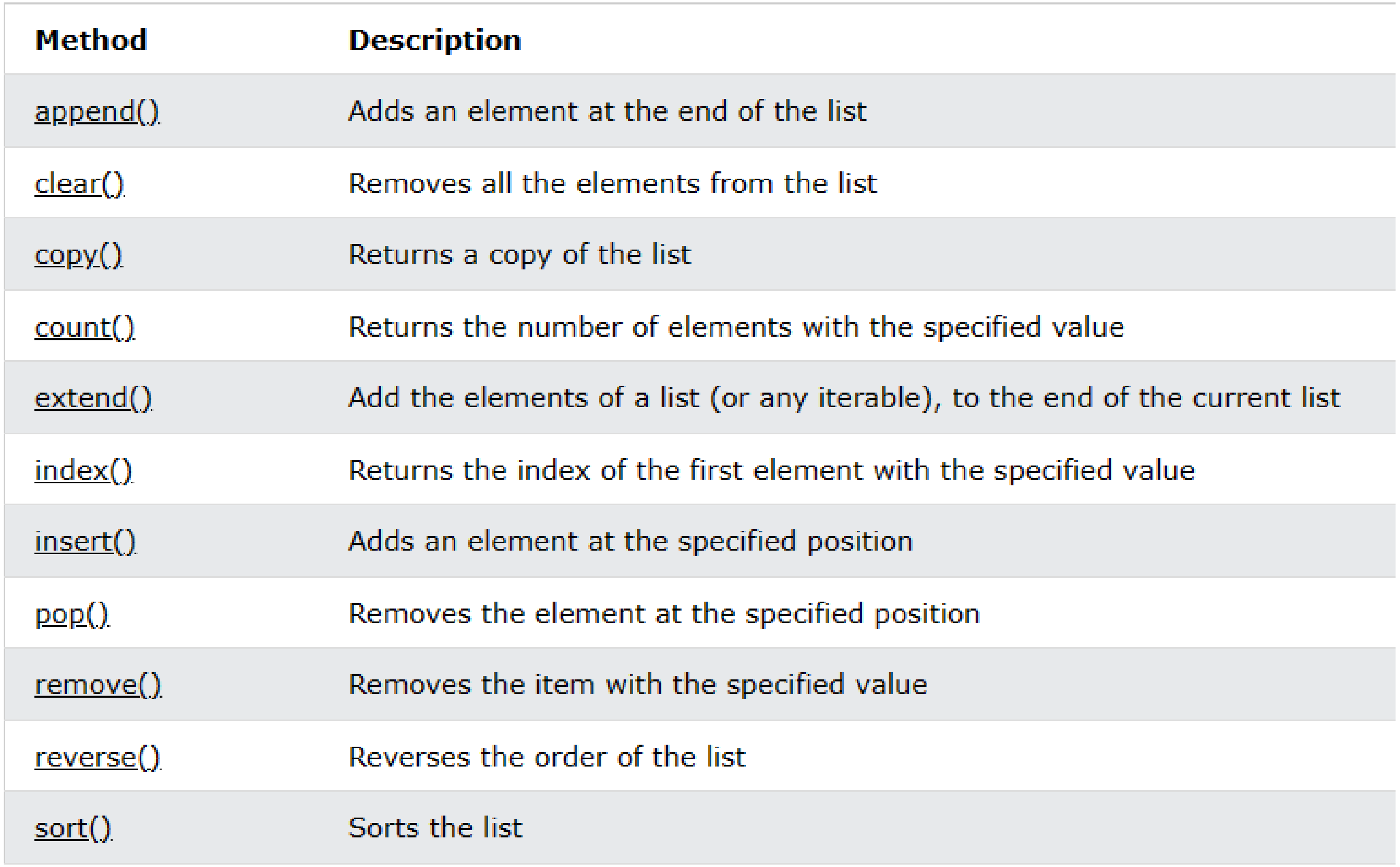
การลบหรือนำ Item ออกจาก List

- ❑ Remove ใช้เพื่อลบ item ที่**มีค่าตรงกับที่ระบุ**ออกจาก list **ถ้าซ้ำกันจะลบตัวแรกที่พบ**
- ❑ Pop ใช้เพื่อนำ item ออกจาก List โดย**ระบุตำแหน่ง**

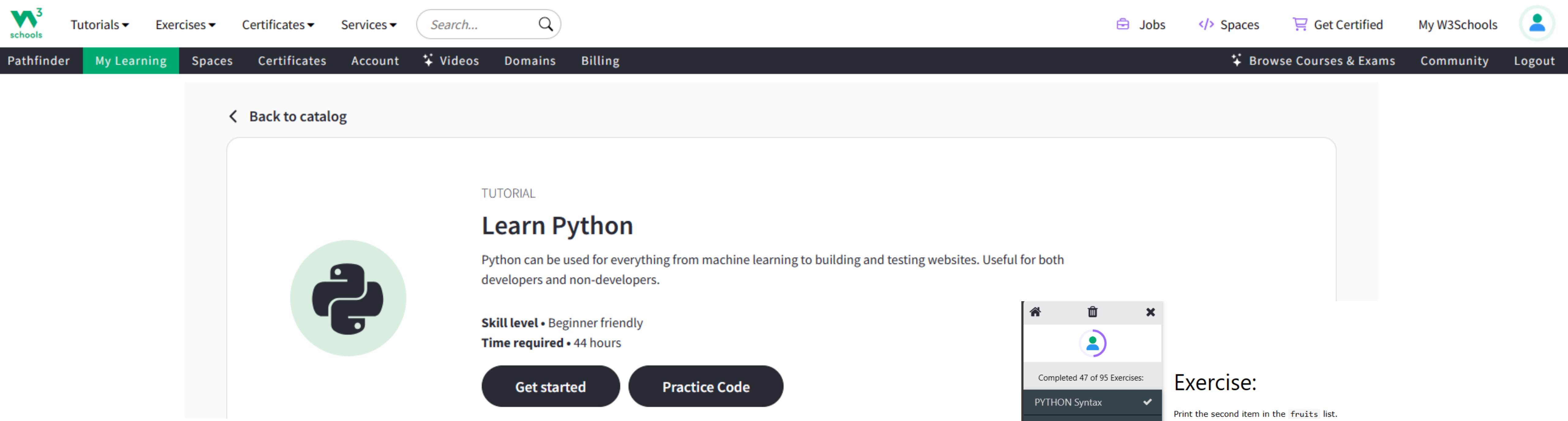
```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

List



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list


Tutorials Exercises Certificates Services Search... Jobs Spaces Get Certified My W3Schools

Pathfinder My Learning Spaces Certificates Account Videos Domains Billing Browse Courses & Exams Community Logout

Back to catalog

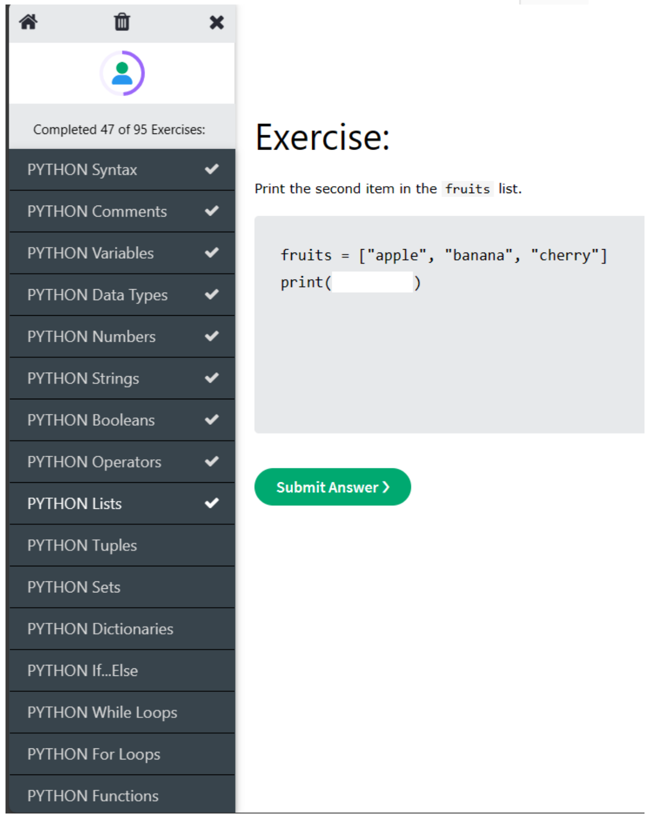
TUTORIAL

Learn Python

Python can be used for everything from machine learning to building and testing websites. Useful for both developers and non-developers.

Skill level • Beginner friendly
Time required • 44 hours

Get started Practice Code


Completed 47 of 95 Exercises:

- PYTHON Syntax ✓
- PYTHON Comments ✓
- PYTHON Variables ✓
- PYTHON Data Types ✓
- PYTHON Numbers ✓
- PYTHON Strings ✓
- PYTHON Booleans ✓
- PYTHON Operators ✓
- PYTHON Lists ✓
- PYTHON Tuples
- PYTHON Sets
- PYTHON Dictionaries
- PYTHON If...Else
- PYTHON While Loops
- PYTHON For Loops
- PYTHON Functions

Exercise:

Print the second item in the `fruits` list.

```
fruits = ["apple", "banana", "cherry"]
print(          )
```

Submit Answer ›

https://www.w3schools.com/python/exercise.asp

Login ด้วย E-mail จากนั้น...
ทำโจทย์ให้<u>ผ่าน</u>ทั้งหมด 95 ข้อ