Maktab Sharif

Python | Main course

Session 8

- Files
- Pickling
- Logging
- Typing

by Mohammad Amin H.B. Tehrani - Reza Yazdani

www.maktabsharif.ir

Files

Maktab Sharif

Intro

Maktab Sharif

One of the most common tasks that you can do with Python is reading and writing files. Whether it's writing to a simple text file, reading a complicated server log, or even analyzing raw byte data, all of these situations require reading or writing a file.

A file is a contiguous set of bytes used to store data.This data is organized in a specific format and can be anything as simple as a text file or as complicated as a program executable. In the end, these byte files are then translated into binary 1 and 0 for easier processing by the computer.

examples

Akbar.jpg
m78.txt
main.py
script.sh

File paths

Maktab Sharif

When you access a file on an operating system, a file path is required. The file path is a string that represents the location of a file. It's broken up into three major parts:

~/Maktab78/ my_file . my_ext

1. **Folder Path:** the file folder location on the file system where subsequent folders are separated by a forward slash / (Posix [Unix, Linux, BSD]) or backslash \ (NT [Windows])
2. **File Name:** the actual name of the file
3. **Extension:** the end of the file path prepended with a period (.) used to indicate the file type

File handling in python

Maktab Sharif

File handling is an important part of any web application.
Python has several functions for creating, reading, updating, and deleting files.

open() function

1. The key function for working with files in Python is the **open()** function.
2. The **open**() function takes two parameters; filename, and mode.
3. There are **four different methods (modes)** for opening a file:

Syntax:

```
file = open(file_path , mode)
...
file.close()
```

Example:

```
file = open('my_file', 'r')
print(file.read())
file.close()  # ?
```

open() function

Maktab Sharif

key	mode	Description
`"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

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

Reading files

Maktab Sharif

Open mode: **'r'**

read() method

The **open()** function returns a file object, which has a **read()** method for reading the content of the file:

example.txt :

```
Hello World!
```

Example:

```
f = open('example.txt')  # mode = 'r'
x = f.read(3)
print(x, type(x)) #??
print(f.read())  #???
```

Reading files

Maktab Sharif

Open mode: **'r'**

read() method

The **open()** function returns a file object, which has a **read()** method for reading the content of the file:

Example:

```
f = open('example.txt')  # mode = 'r'
x = f.read(3)
print(x, type(x)) #??
print(f.read())  #???
```

example.txt :

```
Hello World!
```

output:

```
Hel <class 'str'>
lo World!
```

readline(...) method

Maktab Sharif

readline(limit=...)

The **readline()** method returns one line from the file.

You can also specified how many bytes from the line to return, by using the size parameter.

By default it returns all of the current line.

Example:

```
f = open('example.txt', 'r')
print(f.readline())
print(f.readline(7))
print(f.readline())
print(f.readline())
print(f.readline())
```

example.txt :

```
Test test:
> Hello World!
> Date: 4/14/2021
> Time: 1:18 pm
```

readlines(...) method

Maktab Sharif

readlines(hint=...)

The **readlines()** method returns a list containing each line in the file as a **list** item.
Use the hint parameter to limit the number of lines returned. If the total number of bytes returned exceeds the specified number, no more lines are returned.

Example:

```
f = open('example.txt', 'r')
print(f.readlines(15))
print(f.readlines())
```

example.txt :

```
Test test:
> Hello World!
> Date: 4/14/2021
> Time: 1:18 pm
```

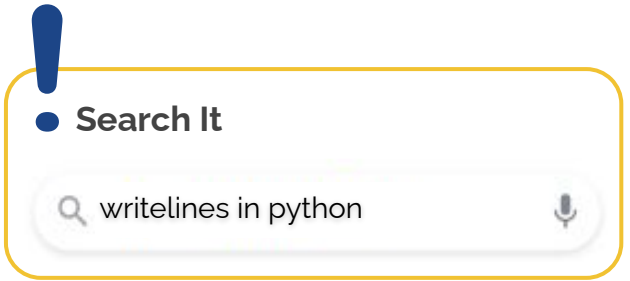
Writing files

Maktab Sharif

Write to an Existing File:

'a' : Append - will **append** to the end of the file

'w' : Write - will **overwrite** any existing content



Create a New File:

'x' : Create - Creates the specified file, **returns an error if the file exists**

'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

write() method

The **write()** method writes a specified text to the file.

Where the specified text will be inserted depends on the file mode and stream position.

Writing files Example

Maktab Sharif

```
def print_example():
    f = open('example.txt')
    print(f.read())
    f.close()
```

```
content = """Test test:
> Hello World!
> Date: 4/14/2021
> Time: 1:18 pm"""
```

Output in the next page:

```
f = open('example.txt', 'x')
f.write(content)
print_example('\nBefore close:')  # ???
f.close()
print_example('\nAfter close:')   # ???

f = open('example.txt', 'a')
f.write('\n---FILE APPEND---')
f.close()
print_example('\nAfter append:')  # ???

f = open('example.txt', 'w')
f.write('\n---FILE WRITE---')
f.close()
print_example('\nAfter overwrite:')  # ???

f = open('example.txt', 'x')  # ???
```

Example: output

Maktab Sharif

```
Before close:

After close: Test test:
> Hello World!
> Date: 4/14/2021
> Time: 1:18 pm

After append: Test test:
> Hello World!
> Date: 4/14/2021
> Time: 1:18 pm
---FILE APPEND---

After overwrite:
---FILE WRITE---

FileExistError:...
```

Example: Writing files using print

Maktab Sharif

```
user_information = {
   'id': 34,
   'first_name': 'Akbar',
   'last_name': 'Mohammadi',
   'phone': '9342224445',
   'email': 'ak.moh@gmail.com',
}

with open('akbar.info', 'w') as f:
   print(user_information, file=f)
```

akbar.info

```
{'id': 34, 'first_name': 'Akbar', 'last_name': 'Mohammadi', 'phone':
'9342224445', 'email': 'ak.moh@gmail.com'}
```

With statement (files)

Maktab Sharif

When we using files in python, we can use **with** statement, **it will automatically close the file.** The with statement provides a way for ensuring that a clean-up is always used.

syntax:

```
with open(...) as <file_var>:
    ...
```

Example:

```
with open('example.txt', 'r') as f:
  print(f.read())
```

Equals to:

```
f = open('example.txt')
print(f.read())
f.close()  # DON'T FORGET
```

Example:

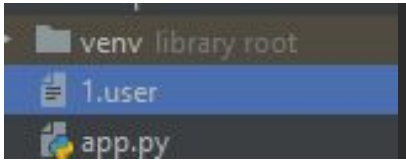
Maktab Sharif

```python
class User:
    def __init__(self, id, name, phone):
        self.id = id
        self.name = name
        self.phone = phone

    def save(self):
        file_name = f'{self.id}.user'
        with open(file_name, 'w') as f:
            f.write(f'{self.id=}\n{self.name=}\n{self.phone=}')
        return file_name

    @classmethod
    def from_file(cls, file_path):
        self = User.__new__(User)
        with open(file_path, 'r') as f:
            file_content = f.read()
            exec(file_content)
        return self
```

Try it:

```python
u = User(1, 'akbar', '9129000111')
file_name = u.save()
print(file_name)  #???
```

Then:




```
self.id=1
self.name='akbar'
self.phone='9129000111'
```

File ‘1.user’ created.

Now try it:

```python
other_u = User.from_file(file_name)
print(other_u.id, other_u.name, other_u.phone)  #???
```

Pickling

Maktab Sharif

Intro

Maktab Sharif

The pickle module implements binary protocols for serializing and de-serializing a Python object structure.
Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

```
data = [
  {
     'key1': 1,
     'key2': 2,
     'key-name': 'akbar',
  },
  (1, 2, 3, 4),
  'A String!'
]
```

```
import pickle

pickled = pickle.dumps(data)
print('PICKLED:', pickled)  # ???

unpickled = pickle.loads(pickled)
print('UN-PICKLED:', unpickled) #???
```

Example


```python
import pickle


class User:
    def __init__(self, id, name, phone):
        self.id = id
        self.name = name
        self.phone = phone

u = User(1, 'akbar', '09123456789')
pickled = pickle.dumps(u)
print('PICKLED user:', pickled)

unpickled = pickle.loads(pickled)
print('UN-PICKLED user:', unpickled)
```

```
PICKLED user:
b'\x80\x04\x95I\x00\x00\x00\x00\x00\x00\
x00\x8c\x08__main__\x94\...'
UN-PICKLED user: <__main__.User object
at 0x0000025AD85FD190>
```

pickle.dumps & pickle.loads

Maktab Sharif

dump() method

This function is used to write a pickled representation of obj to the open file object given in the constructor.

dumps() method

This function is equivalent to Pickler(file, protocol).dump(obj). This is used to write a pickled representation of obj to the open file object file.

load() method

This function is used to read a pickled object representation from the open file object file and return the reconstituted object hierarchy specified.

loads() method

This function is equivalent to Unpickler(file).load(). This function is used to read a pickled object representation from the open file object file and return the reconstituted object hierarchy specified.

Example:

Maktab Sharif

```python
import pickle

class User:
    def __init__(self, id, name, phone):
        self.id = id
        self.name = name
        self.phone = phone

    def save(self):
        file_name = f'{self.id}.user'
        with open(file_name, 'wb') as f:
            pickle.dump(self, f)
        return file_name

    @classmethod
    def from_file(cls, file_path):
        with open(file_path, 'rb') as f:
            return pickle.load(f)
```

Try it:

```python
u = User(1, 'akbar', '9129000111')
file_name = u.save()
print(file_name)  #???
```

1.user:

```
��I      �__main__��User���)��}�
(�id�K�name��akbar��phone��
09123456789�ub.
```

Now try it:

```python
other_u = User.from_file(file_name)
print(other_u.id, other_u.name, other_u.phone)  #???
```

Logging

Maktab Sharif

Intro

Logging is a means of tracking events that happen when some software runs. The software's developer adds logging calls to their code to indicate that certain events have occurred.

module:

```
import logging
```

Logging level

The log level corresponds to the "importance" a log is given: an "**error**" log should be more urgent then than the "**warn**" log, whereas a "**debug**" log should be useful only when debugging the application.

Logging levels

Maktab Sharif

Level Name	Level	Usage	Example
CRITICAL	50	To log high importance issues	Internet connection,...
FATAL	50	To log high importance issues	Attacks, ...
ERROR	40	To log normal issues (errors)	Invalid data, ...
WARNING (WARN)	30	An indication that something unexpected happened, or indicative of some problem in the near future	disk space low, ...
INFO	20	Confirmation that things are working as expected.	OS specs
DEBUG	10	Detailed information, typically of interest only when diagnosing problems.	Trace program

Logging methods

Maktab Sharif

```python
import logging

logging.log(logging.DEBUG, "A Debug log")
logging.log(logging.INFO, "An Info log")
logging.log(logging.WARNING, "A Warning log")
logging.log(logging.ERROR, "An Error log")
logging.log(logging.FATAL, "A Fatal log")
logging.log(logging.CRITICAL, "A Critical log")

logging.debug("A Debug log")
logging.info("An Info log")
logging.warning("A Warning log")
logging.error("An Error log")
logging.fatal("A Fatal log")
logging.critical("A Critical log")
```

```
WARNING:root:A Warning log
ERROR:root:An Error log
CRITICAL:root:A Fatal log
CRITICAL:root:A Critical log

WARNING:root:A Warning log
ERROR:root:An Error log
CRITICAL:root:A Fatal log
CRITICAL:root:A Critical log
```

Debug & Info???

Basic Config

Maktab Sharif

You can set your target logger level in basicConfig() method.

```
import logging

logging.basicConfig(level=logging.DEBUG)  # set level

logging.log(logging.DEBUG, "A Debug log")
logging.log(logging.INFO, "An Info log")
logging.log(logging.WARNING, "A Warning log")
logging.log(logging.ERROR, "An Error log")
logging.log(logging.FATAL, "A Fatal log")
logging.log(logging.CRITICAL, "A Critical log")
```

```
DEBUG:root:A Debug log
INFO:root:An Info log
WARNING:root:A Warning log
ERROR:root:An Error log
CRITICAL:root:A Fatal log
CRITICAL:root:A Critical log
```

Default logger level = WARN

Format logs

Maktab Sharif

Full format:
"%(asctime)s — %(name)s — %(levelname)s — %(funcName)s:%(lineno)d — %(message)s"

```
import logging

# Custom log format
my_format = "%(asctime)s (%(name)s) %(levelname)s: %(message)s"
logging.basicConfig(format=my_format)

logging.log(logging.ERROR, "An Error log")
logging.log(logging.FATAL, "A Fatal log")
logging.log(logging.CRITICAL, "A Critical log")
```

```
2021-04-14 16:19:17,824 (root) ERROR: An Error log
2021-04-14 16:19:17,824 (root) CRITICAL: A Fatal log
2021-04-14 16:19:17,824 (root) CRITICAL: A Critical log
```

Log into file

Maktab Sharif

```
import logging

logging.basicConfig(filename='my-log.log', filemode='a')    # open('my-log.log', 'a')

logging.log(logging.ERROR, "An Error log")
logging.log(logging.FATAL, "A Fatal log")
logging.log(logging.CRITICAL, "A Critical log")
```

my-log.log

```
ERROR:root:An Error log
CRITICAL:root:A Fatal log
CRITICAL:root:A Critical log
```

```
import logging

logging.basicConfig()
logger = logging.getLogger('my_logger')

stream_handler = logging.StreamHandler()
file_handler = logging.FileHandler('test.log', 'a', encoding='utf-8')

log_format = logging.Formatter("%(asctime)s — %(name)s — %(levelname)s — %(funcName)s:%(lineno)d — %(message)s")

stream_handler.setLevel(logging.WARNING)
file_handler.setLevel(logging.DEBUG)
stream_handler.setFormatter(log_format)
file_handler.setFormatter(log_format)

logger.addHandler(stream_handler)
logger.addHandler(file_handler)

logger.log(logging.DEBUG, 'Debug log')
logger.log(logging.INFO, 'Info log')
logger.log(logging.WARNING, 'Warn log')
```

Advanced logging

Typing

Intro

Maktab Sharif

Python is a dynamically typed language. That means it is not necessary to declare the type of a variable when assigning a value to it.
Coding in a dynamically typed language like Python surely is more flexible, but one might want to annotate data types of objects and enforce type constraints. If a function only expects integer arguments, throwing strings into the function might crash the program.

In python we can define Types for variables and parameters , …
But it's NOT Strict Type checking yet.
It's mean than we can recommend our code user to use the type.

```
x: int = 123
y: float
b: bool = True
```

```
x: int = 12.3
y: float =
b: bool = range(1, 10)
```

Expected type 'int', got 'float' instead

Syntax

Maktab Sharif

Variable Typing:

<var_name> **: <var_type>** = value

Function Typing:

```
def func_name(arg1:type1,arg2:type2, ...) -> return_type:
    pass
```

```
def func(x: int, y: float, z: str = 'akbar'):
  res: str
  res = int(x*y)*z
  return res
```

Using class types:


```
class User:
    id: int
    name: str
    phone: str
    ...


def register(u: User):
    ...


def login(username: str, password: str) -> User:
    ...
```

‘typing’ module

Maktab Sharif

import

```
import typing
```

Some useful types:

- List
- Tuple
- Any
- Union
- Optional
- Literal
- Dict
- Set

Pre-reading
For next session

Search about:

1. *Python dill
2. Exceptions in python
3. Assertion
4. Docstring
5. *Modules in python

Maktab Sharif