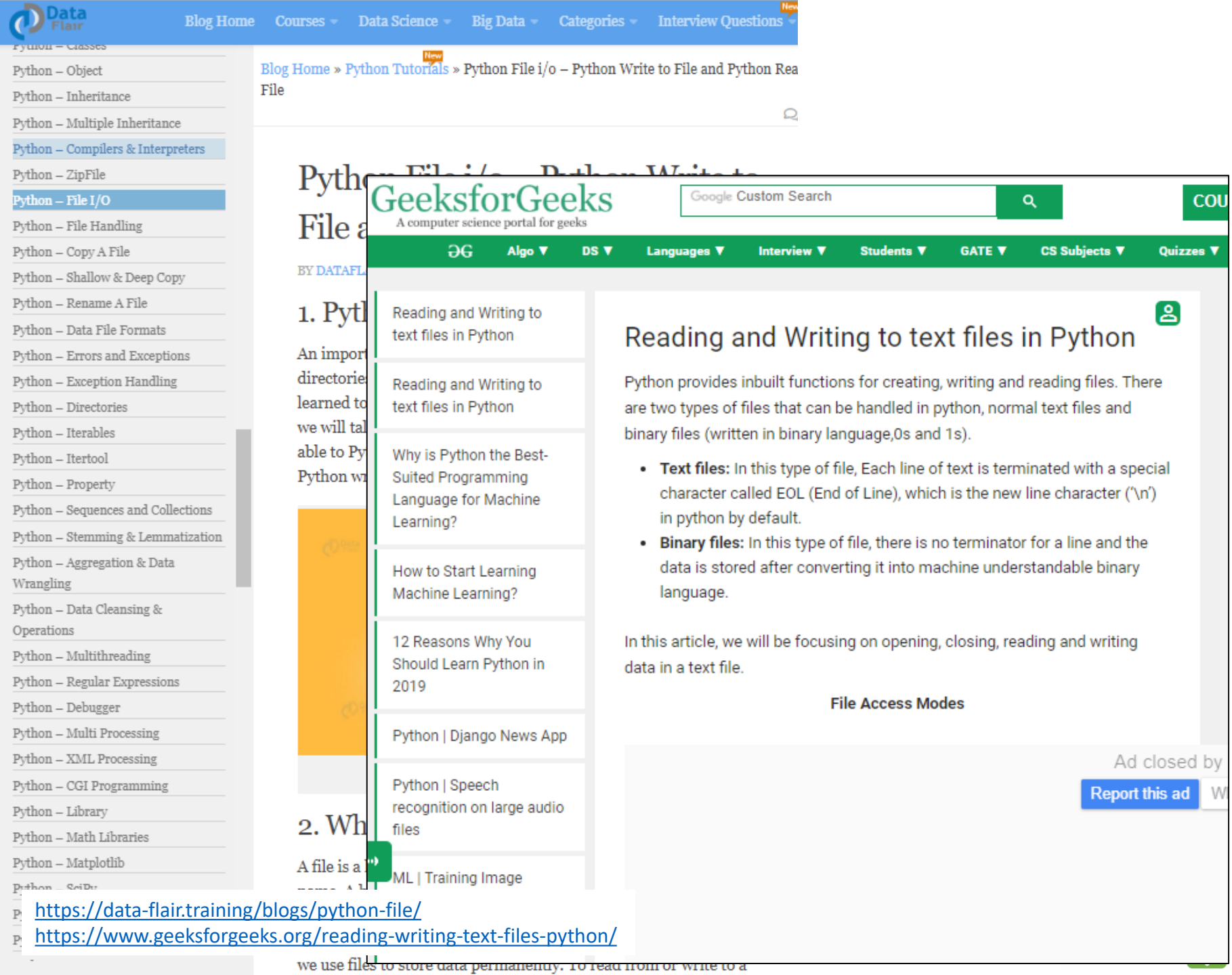
Week 07 - support

on file handling i.e. read and write

File:

https://data-flair.training/blogs/python-file/



https://data-flair.training/blogs/python-file/
https://www.geeksforgeeks.org/reading-writing-text-files-python/

What is a file?

Files on most modern file systems are composed of three main parts:

1. **Header:** metadata about the contents of the file (file name, size, type, and so on)
2. **Data:** contents of the file as written by the creator or editor
3. **End of file (EOF):** special character that indicates the end of the file

There are three different categories of file objects:

- Text files
- Buffered binary files
- Raw binary files

Most of the focus

https://realpython.com/read-write-files-python/

Python Tutorial
Python 3 - About
Python 3 - Installation
Python 3 - Environment
Python 3 - Hello World
Python 3 - Variables
Python 3 - User Input
Python 3 - Strings
Python 3 - Lists
Python 3 - Tuples
Python 3 - Dictionary
Python 3 - Numbers
Python 3 - Operators
Python 3 - If Statements
Python 3 - While Loops
Python 3 - For Loops
Python 3 - Functions
Python 3 - Modules
Python 3 - Classes
Python 3 - Read/Write Files

Python Reference
Python 3 - Operators
Python 3 - Escape Sequences
Python 3 - String Methods

Python Read Write Files

← Python Classes

About Python →

Python has various methods for performing operations against files. Here's an overview.

Open a File

When you read or write a file, the first thing you need to do is open it (or create it). Python provides the `open()` method that is used to open a file. It also creates the file if it doesn't already exist.

The `open()` syntax is like this:

```
open(filename, mode)
```

It takes two parameters. The first one provides the name of the file, and the second parameter specifies the mode to be used. A mode of `r` means "read", means "append". You can append the mode with `b` to specify binary mode. You can also do stuff like `r+` to make it read

http://python-ds.com/python-read-write-files

Create a File

Here's an example of creating a new file:

```
# Create the file in 'write' mode
f = open("hello.txt", "w")

# Write some text to the file
f.write("Hello World!")

# Close the file
f.close()
```

Read a File

Now that we've created a file and added some content to it, we can read it. Here's how:

```
# Open the file in 'read' mode
f = open("hello.txt", "r")

# Put the contents of the file into a variable
f_contents = f.read()

# Close the file
f.close()

# Print the file's contents
print(f_contents)
```

http://python-ds.com/python-read-write-files

Need to close or use with…

- Close is to leave the file alone

```
reader = open('dog_breeds.txt')
reader.close()
```

- Using with the close is automatic – after doing what is inside with

```
with open('dog_breeds.txt') as reader:
    # Further file processing goes here
```

https://realpython.com/read-write-files-python/

"With … as …"

Read

```
with open('data.txt', 'r') as f:
    data = f.read()
```

Write

```
with open('data.txt', 'w') as f:
    data = 'some data to be written to the file'
    f.write(data)
```

https://realpython.com/working-with-files-in-python/

File Modes in Python

Mode	Description
'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
'+'	This will open a file for reading and writing (updating)

https://www.guru99.com/reading-and-writing-files-in-python.html

Writing and reading…

```
#writing something in a new file “guru99.txt”
f= open("guru99.txt","w+")
for i in range(10):
        f.write("This is line %d\r\n" % (i+1))
f.close()


#Open the file back and read the contents
f=open("guru99.txt", "r")
#option 1: reads all aa slist
contents =f.read()
print (contents)
#option 2 : readlines reads the individual line into a list
fl =f.readlines()
for x in fl:
print(x)
```

https://www.guru99.com/reading-and-writing-files-in-python.html

Read a Single Line

You can use `readline()` to read a single line in the file:

```
f = open("hello.txt", "r")
line1 = f.readline()
line2 = f.readline()
print("Line 1:", line1)
print("Line 2:", line2)
f.close()
```

Looping over the Lines

You can use a `for` loop to loop over each line in the file. Here's how that works:

```
f = open("hello.txt", "r")
for line in f:
    print("Line:", line, end="")
f.close()
```

http://python-ds.com/python-read-write-files

Read Multiple Lines as a List

You can use the `readlines()` or the `list()` methods to return all lines in a list. Like this:

```
# Using 'readlines()'
f = open("hello.txt", "r")
f_content = f.readlines()
print(f_content)
f.close()

# Using 'list()'
f = open("hello.txt", "r")
f_content = list(f)
print(f_content)
f.close()
```

http://python-ds.com/python-read-write-files

Reading

Method	What It Does
`.read(size=-1)`	This reads from the file based on the number of `size` bytes. If no argument is passed or `None` or `-1` is passed, then the entire file is read.
`.readline(size=-1)`	This reads at most `size` number of characters from the line. This continues to the end of the line and then wraps back around. If no argument is passed or `None` or `-1` is passed, then the entire line (or rest of the line) is read.
`.readlines()`	This reads the remaining lines from the file object and returns

```
>>> f = open('dog_breeds.txt')
>>> f.readlines()  # Returns a list object
['Pug\n', 'Jack Russell Terrier\n', 'English Springer Spaniel\n', 'German Shepherd\n
```

```
>>> with open('dog_breeds.txt', 'r') as reader:
>>>     # Read & print the entire file
>>>     print(reader.read())
```

https://realpython.com/read-write-files-python/

Writing... (text files)

Method	What It Does
`.write(string)`	This writes the string to the file.
`.writelines(seq)`	This writes the sequence to the file. No line endings are appended to each sequence item. It's up to you to add the appropriate line ending(s).

```
with open('dog_breeds.txt', 'r') as reader:
    # Note: readlines doesn't trim the line endings
    dog_breeds = reader.readlines()

with open('dog_breeds_reversed.txt', 'w') as writer:
    # Alternatively you could use
    # writer.writelines(reversed(dog_breeds))

    # Write the dog breeds to the file in reversed order
    for breed in reversed(dog_breeds):
        writer.write(breed)
```

https://realpython.com/read-write-files-python/

Writing text files: don't forget "\n"

```
new_file=open("D:\\new_dir\\newfile.txt",mode="w",encoding="utf-8")
```

```
new_file.write("Writing to a new file\n")
new_file.write("Writing to a new file\n")
new_file.write("Writing to a new file\n")
new_file.close()
```

https://www.datacamp.com/community/tutorials/reading-writing-files-python

Print to text file using “file” parameter

```
# Code for printing to a file
sample = open('samplefile.txt', 'w')

print('GeeksForGeeks', file = sample)
sample.close()
```

```
print( "{:6s}\t{:^40s}\t{:2.1f}\n".format( numero, nome, notaFinal(r)), file=fout )
```

```
# Code for printing to STDERR
import sys

print('GeeksForGeeks', file = sys.stderr)
```

https://www.geeksforgeeks.org/python-file-parameter-print/

Work with two files

```
d_path = 'dog_breeds.txt'
d_r_path = 'dog_breeds_reversed.txt'
with open(d_path, 'r') as reader, open(d_r_path, 'w') as writer:
    dog_breeds = reader.readlines()
    writer.writelines(reversed(dog_breeds))
```

Work with two files

```
d_path = 'dog_breeds.txt'
d_r_path = 'dog_breeds_reversed.txt'
with open(d_path, 'r') as reader, open(d_r_path, 'w') as writer:
    dog_breeds = reader.readlines()
    writer.writelines(reversed(dog_breeds))
```

Some notes on formats

Don't Re-Invent the Snake

There are common situations that you may encounter while working with files. Most of these cases can be handled using other modules. Two common file types you may need to work with are `.csv` and `.json`. *Real Python* has already put together some great articles on how to handle these:

- Reading and Writing CSV Files in Python
- Working With JSON Data in Python

Additionally, there are built-in libraries out there that you can use to help you:

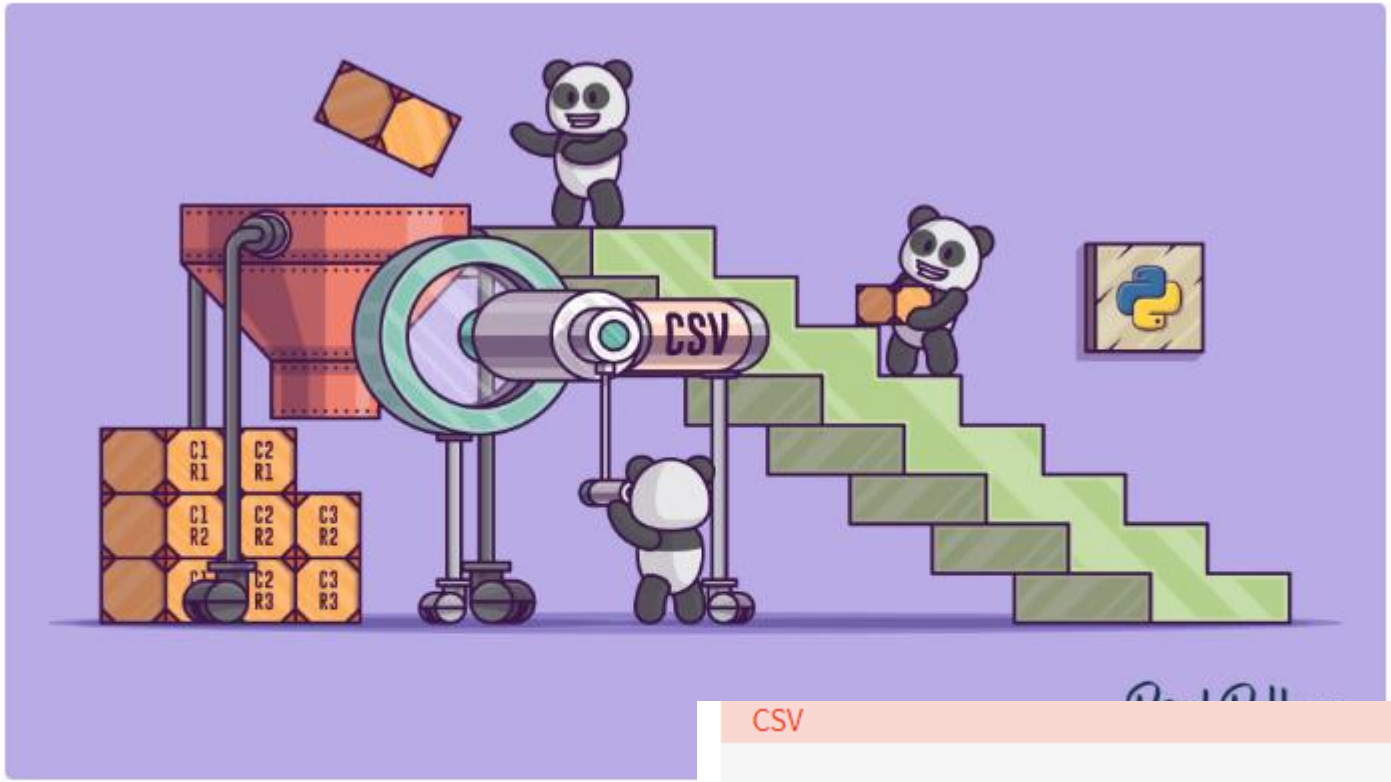
- **`wave`**: read and write WAV files (audio)
- **`aifc`**: read and write AIFF and AIFC files (audio)
- **`sunau`**: read and write Sun AU files
- **`tarfile`**: read and write tar archive files
- **`zipfile`**: work with ZIP archives
- **`configparser`**: easily create and parse configuration files
- **`xml.etree.ElementTree`**: create or read XML based files
- **`msilib`**: read and write Microsoft Installer files
- **`plistlib`**: generate and parse Mac OS X `.plist` files

There are plenty more out there. Additionally there are even more third party tools available on PyPI. Some popular ones are the following:

- **`PyPDF2`**: PDF toolkit
- **`xlwings`**: read and write Excel files
- **`Pillow`**: image reading and manipulation

https://realpython.com/read-write-files-python/
https://realpython.com/python-json/
https://realpython.com/python-csv/

Real Python Start Here Learn Python Store More



Reading and Writing

by Jon Fincher 57 Comments data-scie

Tweet Share Email

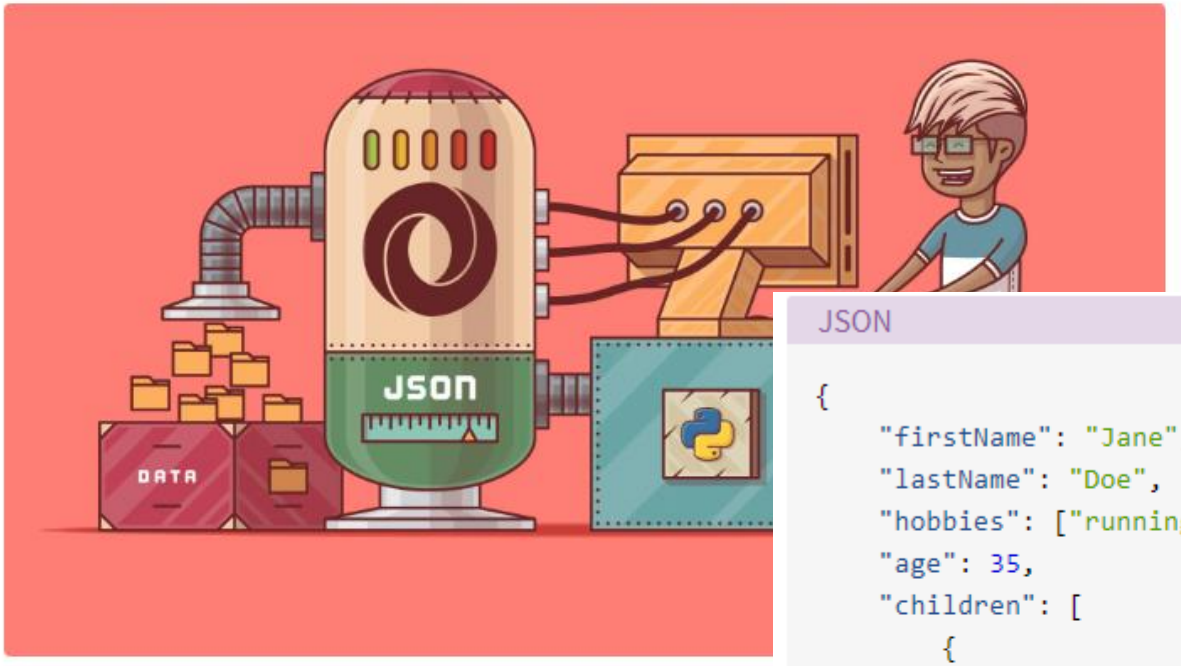
CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

Table of Contents

https://realpython.com/python-csv/

Real Python Start Here Learn Python Store More



Working With JSON Data ir

by Lucas Lofaro 34 Comments intermediate python

Tweet Share Email

Table of Contents

- A (Very) Brief History of JSON
- Look, it's JSON!
- Python Supports JSON Natively!
  - A Little Vocabulary
  - Some Useful Keyword Arguments

JSON

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

https://realpython.com/python-json/

Encodings

There is one more piece of crucial information: encoding. Some files may have to be read as a particular encoding type, and sometimes you need to write out a file in a specific encoding system. For such cases, the `open()` statement should include an encoding spcification, with the `encoding='xxx'` switch:

```
myfile = open('alice.txt', encoding='utf-8')     # Reading a UTF-8 file; 'r' is omitted

myfile = open('results.txt', 'w', encoding='utf-8')   # File will be written in UTF-8
```

foo.py

Mostly, you will need `'utf-8'` (8-bit Unicode), `'utf-16'` (16-bit Unicode), or `'utf-32'` (32-bit), but it may be something different, especially if you are dealing with a foreign language text. Here is a full list of encodings.

https://www.pitt.edu/~naraehan/python3/reading_writing_methods.html
https://docs.python.org/3/library/codecs.html#standard-encodings

Encodi

There is one more piece of c ar encoding type, and sometimes you ne n() statement should include an

```
myfile = open('alice
myfile = open('result
```

tted

-8

foo.py

Mostly, you will need 'utf-8 may be something different, especia dings.

Codec	Aliases	Languages
ascii	646, us-ascii	English
big5	big5-tw, csbig5	Traditional Chinese
big5hkscs	big5-hkscs, hkscs	Traditional Chinese
cp037	IBM037, IBM039	English
cp273	273, IBM273, csIBM273	German *New in version 3.4.*
cp424	EBCDIC-CP-HE, IBM424	Hebrew
cp437	437, IBM437	English
cp500	EBCDIC-CP-BE, EBCDIC-CP-CH, IBM500	Western Europe
cp720		Arabic
cp737		Greek
cp775	IBM775	Baltic languages
cp850	850, IBM850	Western Europe

...

utf_32	U32, utf32	all languages
utf_32_be	UTF-32BE	all languages
utf_32_le	UTF-32LE	all languages
utf_16	U16, utf16	all languages
utf_16_be	UTF-16BE	all languages
utf_16_le	UTF-16LE	all languages
utf_7	U7, unicode-1-1-utf-7	all languages
utf_8	U8, UTF, utf8, cp65001	all languages
utf_8_sig		all languages

https://www.pitt.edu/
https://docs.python.or

Encodings: Some notes

Example: `f = open(fname, encoding="latin-1")`

Note

While the Windows `cp1252` encoding is also sometimes referred to as "latin-1", it doesn't map all possible byte values, and thus needs to be used in combination with the `surrogateescape` error handler to ensure it never throws `UnicodeDecodeError`. The `latin-1` encoding in Python implements ISO_8859-1:1987 which maps all possible byte values to the first 256 Unicode code points, and thus ensures decoding errors will never occur regardless of the configured error handler.

Example: `f = open(fname, encoding="utf-8")`

Consequences:

- `UnicodeDecodeError` may be thrown when reading such files (if the data is not actually in the specified encoding)
- `UnicodeEncodeError` may be thrown when writing such files (if attempting to write out code points which have no representation in the target encoding).
- the `surrogateescape` error handler can be used to be more tolerant of encoding errors if it is necessary to make a best effort attempt to process files that contain such errors instead of rejecting them outright as invalid input.

http://python-notes.curiousefficiency.org/en/latest/python3/text_file_processing.html

Processing lines

strip

The **strip()** method returns a copy of the string in which all chars have been stripped from the beginning and the end of the string (default whitespace characters).

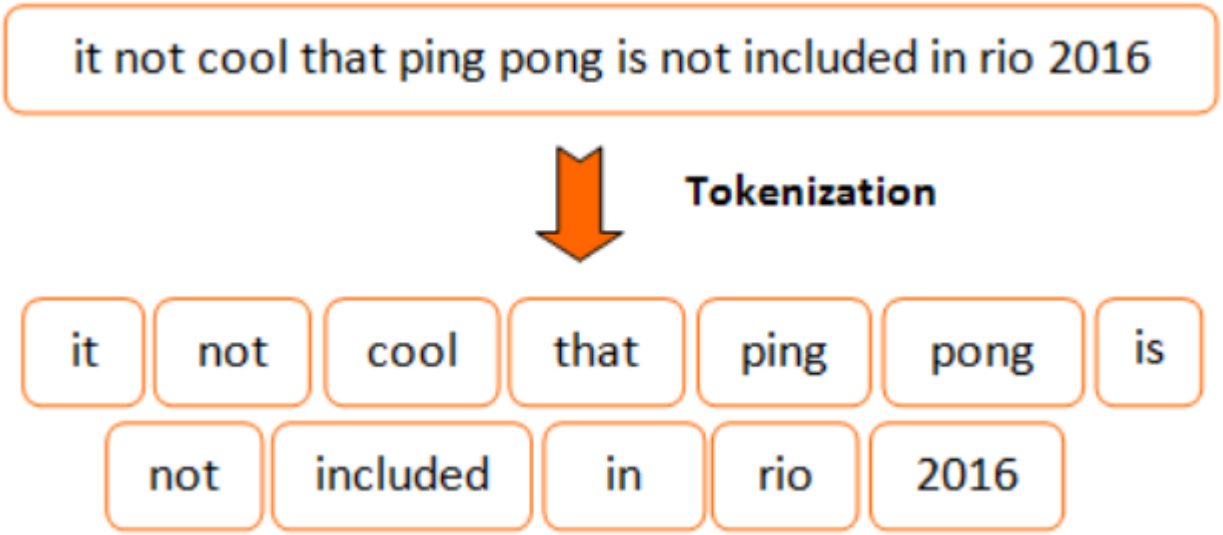
```
#!/usr/bin/python3

str = "*****this is string example....wow!!!*****"
print (str.strip( '*' ))
```

```
this is string example....wow!!!
```

split

The **split()** method returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified), optionally limiting the number of splits to num.



https://www.tutorialspoint.com/python3/string_split.htm

split

The **split()** method returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified), optionally limiting the number of splits to num.

```
['this', 'is', 'string', 'example....wow!!!']
['th', 's is string example....wow!!!']
['this is string example....', 'o', '!!!']
```

```
#!/usr/bin/python3

str = "this is string example....wow!!!"
print (str.split( ))
print (str.split('i',1))
print (str.split('w'))
```

https://www.tutorialspoint.com/python3/string_split.htm

Typical processing

```
with open( …filename …) as f:
  # get line in f
   for line in f:
      # take out “empty” in beginning and end - strip
      # and split in list using space (default) p
       words = line.strip().split()
       # do something...
```

```python
import sys

src = sys.argv[1]     # The input source file
dst = sys.argv[2]     # The file to output to

with open(src, 'r') as fin, open(dst, 'a') as fout:      # Open in append mode

    student = fin.readline().strip()     # Strip the newline character
    while student:

        student_data = student.split()   # ['Liam', '84'] for example

        name = student_data[0]
        mark = int(student_data[1])

        if mark < 40:
            grade = "FAIL"
        else:
            grade = "PASS"

        fout.write('{:s} {:s}\n'.format(name, grade))      # Write to the output file

        student = fin.readline().strip()      # Get the next student
```

https://www.slitherintopython.com/book/chapter_10/chapter_10.html

File I/O

Overview

Teaching: 10 min
Exercises: 5 min

Questions
- How can I read data from a file?

Use open to open fi

- Arguments are a path and:
  - 'r' for reading
  - 'w' for writing (immediately erases ex
  - 'a' for appending
- Result is an object with methods for rea
  - `file.read()` reads the entire file.
  - `file.read(N)` reads up to that many
- Close files with `file.close`.

```
reader = open('myfile.txt', 'r')
data = reader.read()
```

thispointer.com

Python Pandas C++11 Tutorials C++ Tutorials STL Multi

Design Patterns java Datastructure Subscribe

Home » FileHandling » Python » You are reading »

5 Different ways to read a file lin line in Python

Varun September 29, 2018 FileHandling, Python No Comment

Python.

our python script. Let's see

it's contents line by line.

https://thispointer.com/5-different-ways-to-read-a-file-line-by-line-in-python/

https://swcarpentry.github.io/python-second-language/12-file-io/

Handling directories & files

Just some examples… explore the links

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods : The *file* object provides functions to manipulate files.
- OS Object Methods : This provides methods to process files as well as directories.

https://www.tutorialspoint.com/python/python_files_io.htm

Paths & directories

Details out of scope

https://data-flair.training/blogs/python-directory/
https://realpython.com/read-write-files-python/

Handling directories

```
>>> os.listdir('my_directory/')
['sub_dir_c', 'file1.py', 'sub_dir_b', 'file3.txt', 'file2.csv', 'sub_dir']
```

```
from pathlib import Path

entries = Path('my_directory/')
for entry in entries.iterdir():
    print(entry.name)
```

```
sub_dir_c
file1.py
sub_dir_b
file3.txt
file2.csv
sub_dir
```

```
import os

# List all files in a directory using os.listdir
basepath = 'my_directory/'
for entry in os.listdir(basepath):
    if os.path.isfile(os.path.join(basepath, entry)):
        print(entry)
```

```
my_directory/
|
├── sub_dir/
|   ├── bar.py
|   └── foo.py
|
├── sub_dir_b/
|   └── file4.txt
|
├── sub_dir_c/
|   ├── config.py
|   └── file5.txt
|
├── file1.py
├── file2.csv
└── file3.txt
```

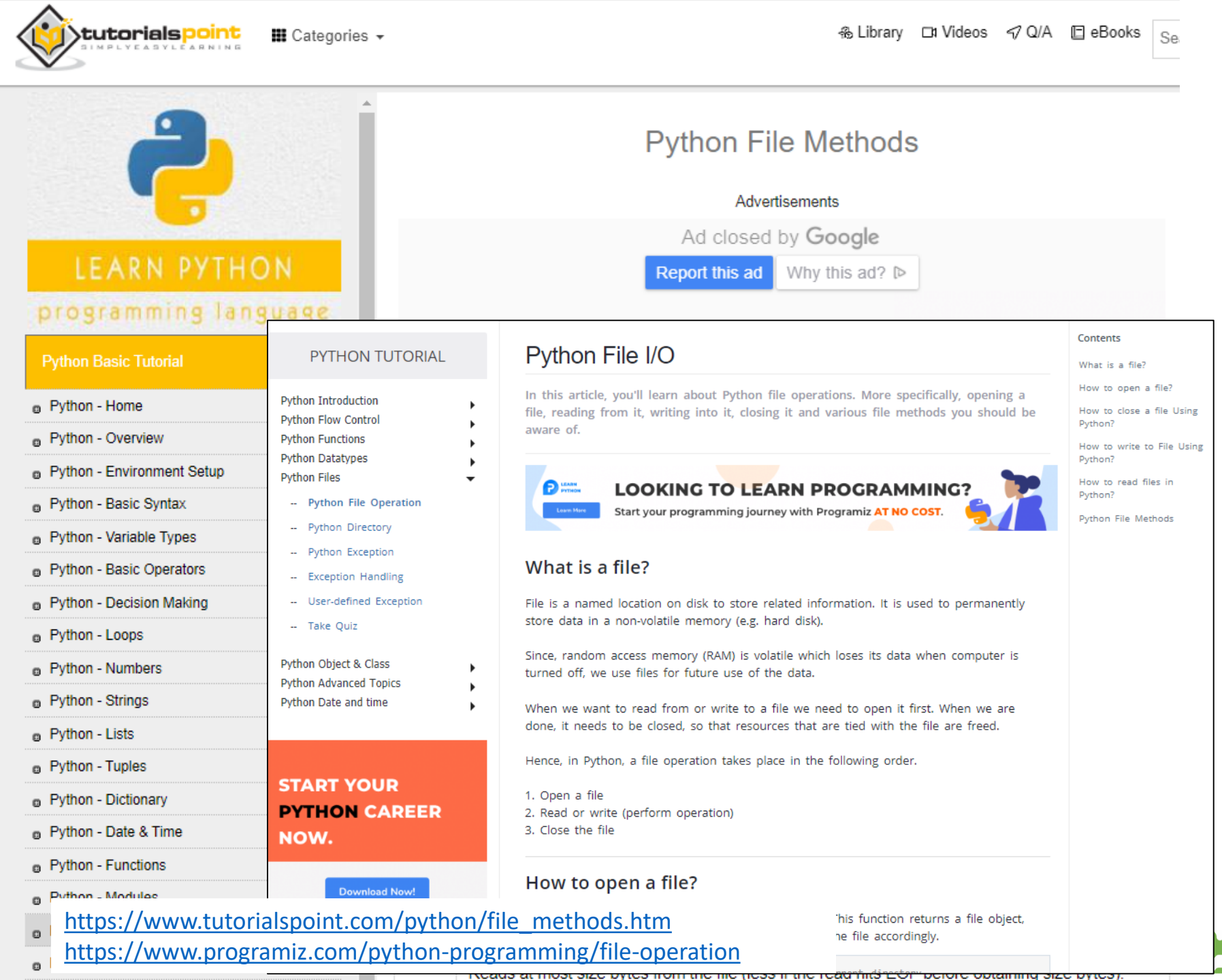
https://realpython.com/working-with-files-in-python/

Handling directories

```
import os

# List all subdirectories using os.listdir
basepath = 'my_directory/'
for entry in os.listdir(basepath):
    if os.path.isdir(os.path.join(basepath, entry)):
        print(entry)
```

https://realpython.com/working-with-files-in-python/
https://www.programiz.com/python-programming/directory
http://net-informations.com/python/file/directory.htm
https://www.tutorialspoint.com/python/os_file_methods.htm



https://www.tutorialspoint.com/python/file_methods.htm
https://www.programiz.com/python-programming/file-operation

Python OS File/Directory Methods

Python Basic Tutorial

- Python - Home
- Python - Overview
- Python - Environment Setup
- Python - Basic Syntax
- Python - Variable Types
- Python - Basic Operators
- Python - Decision Making
- Python - Loops
- Python - Numbers
- Python - Strings
- Python - Lists
- Python - Tuples
- Python - Dictionary
- Python - Date & Time
- Python - Functions
- Python - Exceptions

PYTHON TUTORIAL

- Python Introduction
- Python Flow Control
- Python Functions
- Python Datatypes
- Python Files
  - Python File Operation
  - Python Directory
  - Python Exception
  - Exception Handling
  - User-defined Exception
  - Take Quiz
- Python Object & Class
- Python Advanced Topics
- Python Date and time

Python Directory and Files Management

In this article, you'll learn about file and directory management in Python, i.e. creating a directory, renaming it, listing all directories and working with them.

What is Directory in Python?

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the `os` module, which provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the `getcwd()` method.

This method returns the current working directory in the form of a string. We can also use the `getcwdb()` method to get it as bytes object.

```
>>> import os

>>> os.getcwd()
'C:\\Program Files\\PyScripter'

>>> os.getcwdb()
b'C:\\Program Files\\PyScripter'
```

Contents

- What is Directory in Python?
- Get Current Directory
- Changing Directory
- List Directories and Files
- Making a New Directory
- Renaming a Directory or a File
- Removing Directory or File

https://www.programiz.com/python-programming/directory
https://www.tutorialspoint.com/python/os_file_methods.htm

The END