Objects?? Methods??

You know them already without knowing what they were...

In [ ]:

```
msg = 'Hej all!'
print(msg.startswith('Hej'))
```

In [ ]:

```
print(msg.find('all'))
```

Reading from a File

A 'file' is what your computer can save and read as **data**.

An incredible amount of data is available in files.

Files can contain weather data, traffic data, socioeconomic data, literary works, and more. Reading from a file is particularly useful in data analysis applications, but it’s also applicable to any situation in which you want to analyze or modify information stored in a file.

File types

You probably noticed that files are typically of the form `name.suffix`. The `suffix` part tells us about the type of the file. Here are some file ending examples:

- `.txt` Plain text files
- `.html` Website files
- `.csv` Comma-separated values
- `.rtf` Rich-text formatting
- `.doc` Document files
- `.pdf` Pdf files

We will get back to this later. For now let's look at a plain text file (`.txt`):

In [1]:

```
%%bash
head bones_in_london.txt
```

```
The Project Gutenberg EBook of Bones in London, by Edgar Wallace

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.net

Title: Bones in London
```

Text files can either be read from or written to. Let's have a look at reading first.

Reading an Entire File

When you want to work with the information in a text file, the first step is to **open** it.

```
with open('bones_in_london.txt') as file:
    ...
```

This will give you an **object** in the `file` variable. You can use this object to both read and write. But let's look at reading first.

In [2]:

```
with open('bones_in_london.txt') as file:
    print(file)
```

```
<_io.TextIOWrapper name='bones_in_london.txt' mode='r' encoding='UTF
-8'>
```

file.read()

In [4]:

```
with open('bones_in_london.txt') as file:
    content = file.read()
    print(content)
```

```
The Project Gutenberg EBook of Bones in London, by Edgar Wallace

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.net


Title: Bones in London

Author: Edgar Wallace

Release Date: December 13, 2008 [EBook #27525]

Language: English


*** START OF THIS PROJECT GUTENBERG EBOOK BONES IN LONDON ***
```

Challenge: print only the first 700 lines

In [6]:

```
with open('bones_in_london.txt') as file:
    content = file.read()
    print(content)
```

```
The Project Gutenberg EBook of Bones in London, by Edgar Wallace

Thi
```

Let's start by looking at the `open() function`. To do any work with a file, even just printing its contents, you first need to open the file to access it. The `open()` function needs one argument: the name of the file you want to open. Python looks for this file in the directory where the program that is currently being executed is stored.

The `open()` function returns an object representing the file. Here, `open(file_name)` returns an object representing `a_study_in_scarlet.txt`. Python stores this object in `file_object`, which we will work with later in the program.

The keyword `with` denotes a *Context Manager*, which essentially wraps a block of code and performs an action at the end of the block, no matter how it exits. In this case, it closes the file once access to it is no longer needed. Notice how we call `open()` in this program but not `close()`. You could open and close the file by calling `open()` and `close()`, but if a bug in your program prevents the `close()` statement from being executed, the file may never close. This may seem trivial, but improperly closed files can cause data to be lost or corrupted. And if you call `close()` too early in your program, you'll find yourself trying to work with a closed file (a file you can't access), which leads to more errors. It is not always easy to know exactly when you should close a file, but with the structure shown here, Python will figure that out for you. All you have to do is open the file and work with it as desired, trusting that Python will close it automatically when the time is right.

Once we have a file object representing `a_study_in_scarlet.txt`, we use the `read()` method in the second line of our program to read the entire contents of the file and store it as one long string in contents. When we print the value of contents, we get the **entire** text file back.

The only difference between this output and the original file is the extra blank line at the end of the output. The blank line appears because `read()` returns an empty string when it reaches the end of the file; this empty string shows up as a blank line. If you want to remove the extra blank line, you can use `rstrip()`.

Reading Line by Line

When you are reading a file, you will often want to examine each line of the file. You might be looking for certain information in the file, or you might want to modify the text in the file in some way.

You can use a `for` loop on the file object to examine each line from a file one at a time.

In [7]:

```
with open('bones_in_london.txt') as file:
    for line in file:
        print(line)
```

```
The Project Gutenberg EBook of Bones in London, by Edgar Wallace



This eBook is for the use of anyone anywhere at no cost and with

almost no restrictions whatsoever.  You may copy it, give it away or

re-use it under the terms of the Project Gutenberg License included

with this eBook or online at www.gutenberg.net




Title: Bones in London
```

Making a List of Lines from a File

When you use `with`, the file object returned by `open()` is only available inside the `with` block that contains it. If you want to retain access to a file's contents outside the with block, you can store the file's lines in a list inside the block and then work with that list.

The following example stores the lines of `bones_in_london.txt` in a list inside the with block and then prints the lines outside the `with` block.

In [11]:

```
filename = 'bones_in_london.txt'

with open(filename) as file:
    lines = file.readlines()

for line in lines:
    print(line)
```

```
['\ufeffThe Project Gutenberg EBook of Bones in London, by Edgar Wal
lace\n', '\n', 'This eBook is for the use of anyone anywhere at no c
ost and with\n', 'almost no restrictions whatsoever.  You may copy i
t, give it away or\n', 're-use it under the terms of the Project Gut
enberg License included\n', 'with this eBook or online at www.gutenb
erg.net\n', '\n', '\n', 'Title: Bones in London\n', '\n', 'Author: E
dgar Wallace\n', '\n', 'Release Date: December 13, 2008 [EBook #2752
5]\n', '\n', 'Language: English\n', '\n', '\n', '*** START OF THIS P
ROJECT GUTENBERG EBOOK BONES IN LONDON ***\n', '\n', '\n', '\n', '\n
', 'Produced by Al Haines\n', '\n', '\n', '\n', '\n', '\n', '\n', '\
n', '\n', '\n', '\n', 'BONES\n', '\n', 'IN LONDON\n', '\n', '\n', 'B
y\n', '\n', 'EDGAR WALLACE\n', '\n', '\n', '\n', '\n', '\n', 'WARD,
LOCK & CO., LIMITED\n', '\n', 'LONDON AND MELBOURNE\n', '\n', '1921\
n', '\n', '\n', '\n', '\n', 'CONTENTS\n', '\n', 'CHAP.\n', '\n', '
I.--BONES AND BIG BUSINESS\n', '   II.--HIDDEN TREASURE\n', '  III.-
-BONES AND THE WHARFINGERS\n', '   IV.--THE PLOVER LIGHT CAR\n', '
V.--A CINEMA PICTURE\n', '   VI.--A DEAL IN JUTE\n', '  VII.--DETECT
IVE BONES\n', ' VIII.--A COMPETENT JUDGE OF POETRY\n', '   IX.--THE
LAMP THAT NEVER WENT OUT\n', '    X.--THE BRANCH LINE\n', '   XI.--A
```

Working with a File's Contents

After you have read a file into memory, you can do whatever you want with that data, so let's briefly explore some lines of the Sherlock Holmes story. First, we'll attempt to build a single string containing all the digits in the file with no whitespace in it.

Warning! When Python reads from a text file, it interprets all text in the file as a string. If you read in a number and want to work with that value in a numerical context, you will have to convert it to an integer using the `int()` function or convert it to a float using the `float()` function.

Python has no inherent limit to how much data you can work with; you can work with as much data as your system's memory can handle.

In [13]:

```
filename = 'bones_in_london.txt'

with open(filename) as file_pointer:
    lines = file_pointer.readlines()

story_string = ''
for line in lines[100:120]:
    story_string += line.rstrip()

print(story_string)
print(len(story_string))
```

```
Brothers, Brokers," and, beneath, "The United Merchant Shippers'
Corporation," and passed through a door which, in addition to this
declaration, bore the footnote "Private."

Here the file divided, one going to one side of a vast pedestal desk
and one to the other.  Still with their hands pushed deep into their
pockets, they sank, almost as at a word of command, each into his
cushioned chair, and stared at one another across the table.

They were stout young men of the middle thirties, clean-shaven and
ruddy.  They had served their country in the late War, and had made
many sacrifices to the common cause.  One had worn uniform and one h
ad
not.  Joe had occupied some mysterious office which permitted and,
indeed, enjoined upon him the wearing of the insignia of captain, bu
t
had forbidden him to leave his native land.  The other had earned a
little decoration with a very big title as a buyer of boots for Alli
ed
nations.  Both had subscribed largely to War Stock, and a reminder o
f
their devotion to the cause of liberty was placed to their credit ev
ery
half-year.


1077
```

A quick note on binary files

Not all files are good to read. Let's try to open an image file in Python:

In [14]:

```
1 with open('images/cabinet.gif') as file_pointer:
2     for line in file_pointer:
3         print(line)
```

```
---------------------------------------------------------------------
-------
UnicodeDecodeError                        Traceback (most recent cal
l last)
<ipython-input-14-2a3d5fff6a23> in <module>
      1 with open('images/cabinet.gif') as file_pointer:
----> 2     for line in file_pointer:
      3         print(line)

~/.virtualenvs/qsp/lib/python3.7/codecs.py in decode(self, input, fi
nal)
    320         # decode input (taking the buffer into account)
    321         data = self.buffer + input
--> 322         (result, consumed) = self._buffer_decode(data, self.
errors, final)
    323         # keep undecoded input until the next call
    324         self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe0 in position
6: invalid continuation byte
```

Uh-oh. What happened? As you know images are not text. Neither are `.pdf`, `.docx` or `.xlsx` files.

If you try to open these files as text you run into problem. They are stored as binary digits (0 or 1s) in a very specific order.

Recap

- Files and folders
- Reading from files

```
with open('file.txt') as file:
    ...
```

  - Or on the command line: `cat file.txt`
- Difference between text and binary files

Writing to a File with `file.write()`

One of the simplest ways to save data is to write it to a file. When you write text to a file, the output will still be available after you close the terminal containing your program's output.

In [ ]:

```
with open('new_file.txt', 'a') as file: # Notice the 'w' for 'write'!
    file.write('This is a test!')
```

In [ ]:

```
with open('new_file.txt', 'r') as file: # Notice the 'r' for 'read'!
    print(file.read())
```

File modes

File mode	`open` letter	Effect
Read mode	`r`	Only **read** from the file
Write mode	`w`	Only **write** to the file
Append mode	`a`	Only **append** to the file

Default mode (no `open` letter) is reading.

For more modes, see https://docs.python.org/3.7/tutorial/inputoutput.html#reading-and-writing-files (https://docs.python.org/3.7/tutorial/inputoutput.html#reading-and-writing-files)

In [ ]:

```
with open('new_file.txt', 'a') as file: # Notice the 'a' for 'append'!
    file.write('This is a test!')
```

In [ ]:

```
with open('new_file.txt', 'r') as file: # Notice the 'r' for 'read'!
    file.write('This is a test!')
```

Exercise

- Go through the program with your neighbour. What does the program do?

In [ ]:

```
# Reading
filename = 'bones_in_london.txt'
with open(filename) as file_pointer:
    lines = file_pointer.readlines()

# Processing
story_string = ''
for line in lines[100:120]:
    story_string += line

# Writing
filename = 'sherlock_copy.txt'
with open(filename, 'w') as file_object:
    file_object.write(story_string)
```

In [ ]:

```
with open('sherlock_copy.txt') as file:
    print(file.read())
```

Newlines are platform dependent

In Unix world, newlines are `'\n'`, but in Windows they are `''`:

In [ ]:

```
import platform


if platform.system() == 'Windows':
    newline = ''
else:
    newline = None

with open('sherlock_copy.txt', 'w', newline='') as file_object:
    file_object.write(story_string)
```

That is, on Windows you have to pass the newline argument when writing files.

Windows

```
with open('sherlock_copy.txt', 'w', newline='') as file_object:
    file_object.write(story_string)
```

MacOS, Linux, other Unixes

```
with open('sherlock_copy.txt', 'w') as file_object:
    file_object.write(story_string)
```

String `rstrip()`

In [15]:

```
print(help(str.rstrip))
```

```
Help on method_descriptor:

rstrip(self, chars=None, /)
    Return a copy of the string with trailing whitespace removed.

    If chars is given and not None, remove characters in chars inste
ad.

None
```

In [ ]:

```
print('Hello world \n\n\n')
```

In [18]:

```
print('    \n Hello world \n\n\n'.strip())
```

```
Hello world
```

Writing Multiple Lines

The `write()` function does not add any newlines to the text you write. So if you write more than one line without including newline characters, your file may not look the way you want it to.

In [ ]:

```
filename = 'a_study_in_scarlet.txt'
with open(filename) as file_pointer:
    lines = file_pointer.readlines()

story_string = ''
for line in lines[100:120]:
    story_string += line.strip()

filename = 'sherlock_copy.txt'
with open(filename, 'w') as file_object:
    file_object.write(story_string[:50] + '\n')
    file_object.write(story_string[50:100])
```

In [ ]:

```
with open('sherlock_copy.txt') as file:
    print(file.read())
```