CHEATSHEET PYTHON 4: File Input/Output

Functions and Methods: All file methods should be performed on **file objects**, as defined using `open()`.

Command	Description	Example
`open()`	Open a file as a python object	`my_file = open("file.txt", "r")` (See below for opening modes)
`.close()`	Close a file	`my_file.close()`
`.seek()`	Place the file iterator at a certain line. *Needed* to loop over a file contents multiple times!	`my_file.seek(0)`
`.read()`	Read entire contents of a file into a string	`contents = my_file.read()`
`.readlines()`	Read file contents line-by-line. Commonly used in loops.	`lines = my_file.readlines()`
`.write()`	Write to a file	`my_file.write("I'm being written to a file!")`

csv Module functions and methods

Command	Description	Example

`csv.reader()`	Setup a csv reader on an **opened** file handle. Rows are parsed into **lists**.	`my_file = open("file.csv", "r")`
		`reader = csv.reader(my_file)`
		Note that an argument `delimiter=...` may be provided for other separators, like `\t` for tabs.

`csv.DictReader()`	Setup a csv reader on an opened file handle. Rows are parsed into **dictionaries**	`my_file = open("file.csv", "r")` `reader = csv.DictReader(my_file)`
`<file_handle>.writerow()`	Write a row to csv	reader.writerow([a,b,c,d])

File modes

Mode	Meaning
r	Read-only
w	Write-only (CAUTION: will overwrite file contents!)
a	Append (Will *not* overwrite file contents, but append content to the bottom of the file)
r+	Read and write (Mac and Linux)
rw	Read and write (PC)

re Module: All functions shown here take two arguments: `re.<function>(pattern, string)`

Function	Meaning	Example
re.search	Find instances of a regular expression in a string	`re.search(pattern, string)`
re.split	Split string by occurrences of pattern (similar to .split(), but with regex!	`re.split(pattern, string)`
re.findall	Return all non-overlapping matches of pattern in string, as a list of strings	`re.findall(pattern, string)`

CHEATSHEET PYTHON 4: File Input/Output

Regular Expressions

Symbol	Meaning
`\s`	space character
`\t`	tab character
`\n`	newline character (Mac and Linux! PCs may or may recognize this)
`\r`	newline character (PCs! Mac and Linux may or may recognize this)
`.`	wildcard
`\d`	digit (numbers only!)
`\w`	letter or number (case insensitive)
`+`	Symbol to append after a regular expression indicating "one or more of these" E.g., `\d+` means match 1 or more numbers
`*`	Symbol to append after a regular expression indicating "zero or more of these" E.g., `\d*` means match 0 or more numbers
`^`	Beginning of line character
`$`	End of line character
`()`	Use parentheses to capture matched patterns into variables
`\`	Escape regular expression characters E.g. `\.` means match an actual period
`[]`	Use brackets to define a custom regular expression E.g. `[A-Z]` matches a capital letter only. `[123]` matches the number 1,2, or 3 only.

(Note that many, **many** more exist!!)

The `with..as` command

The command `with..as` is used to create control flow blocks, and commonly is it used when performing operations on files. By using the `with..as` command, you do not need to explicitly close the file. Example:

```
with open("myfile.txt", "r") as file_handle:
  # Perform file operations
# Once the with statement is exited, the file is automatically closed
```