Python File I/O Cheat Sheet

Until now, you have been reading and writing to the ***standard input*** and ***standard output***. Python also provides functions to read from and write to data files. Most require a ***file object***.

`open(file_name [, access_mode])`: returns a file object connected to the file named **`file_name`**. The access mode determines if the file will be used for input or output or both; **`access_mode`** can be one of the following strings:

- **`'r'`** –for reading (input); the default. The file must already exist and it must be readable.
- **`'w'`** –for writing (output). If the file exists, it is emptied; if not, the file is created.
- **`'a'`** –for appending (output). If the file exists, writing starts at the end of the file; if not, the file is created.

See The Python Tutorial Section 2.7 for other access modes. These are the only 3 we will use.

You typically assign the file object to a variable in order to later be able to read from or write to the file, e.g.
`file_obj = open('myFile.txt')`.

`file_obj.readline()`: Returns the next line of the file connected to **`file_obj`** as a string, if the file is open for reading; or raises an exception, if not. The return string includes the **`'\n'`** at the end of the line.

`file_obj.read()`: Returns the contents of the file connected to **`file_obj`** as a string, if the file is open for reading; or raises an exception, if not. ***Not recommended for reading large files!***

`file_obj.write(str_exp)`: Writes **`str_exp`** to the file connected to **`file_obj`**, if the file is open for writing; or raises an exception, if not. Requires *a single string argument* and does *not* add a newline character (**`'\n'`**) to the string written.

`print(exp`... **`file=file_obj)`**: Writes the strings returned by calling **`str()`** on each **`exp`** to the file connected to **`file_obj`**, if the file is open for writing; or raises an exception, if not. The values of **`sep`** and **`end`** determine what separates values and what is added to the end.

`file_obj.close()`: Closes the file connected to **`file_obj`**; flushes any buffers associated with **`file_obj`**; and breaks the connection to the file.

A common Python pattern for reading and processing data from a file, one line at a time:

```
inp_obj = open('inFile.txt')
for line_str in inp_obj:
    # process line_str, the string of chrs up to and including
    # the next '\n' in the file attached to inp_obj

input_obj.close()
```

A common Python pattern for processing data and writing it to a file:

```
out_obj = open('outFile.txt', 'w')
while not_done_processing_data:
    # calculate the next string, next_str, to write to the file
    out_obj.write(next_str)

out_obj.close()
```

A common Python pattern for reading and processing data from one file, and writing processed data to another, one line at a time:

```
inp_obj = open('inFile.txt')
out_obj = open('outFile.txt', 'w')
for line_str in inp_obj:
    # process line_str (characters up to and including
    # the next '\n' in the file attached to inp_obj)

    # calculate the next string, next_str, to write to the file
    out_obj.write(next_str)

input_obj.close()
out_obj.close()
```

In the last two patterns, you could replace the call to the **`write`** method with:
`print(next_str, file=out_obj)`

(The former will not add a `'\n'` whereas the latter will.)