codecademy

Files

Python File Object

A Python file object is created when a file is opened with the `open()` function. You can associate this file object with a variable when you open a file using the `with` and `as` keywords. For example:

```
with open('somefile.txt') as
file_object:
```

You can then print the content of the file object, `file_object` with `print()`.

```
print(file_object)
```

You might see something like this on the output terminal:

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

Python Readline Method

To read only one line instead of multiple lines in a Python file, use the method `.readline()` on a file object that is returned from the `open()` function. Every subsequent `.readline()` will extract the next line in the file if it exists.

```
with open('story.txt') as
story_object:
  print(story_object.readline())
```

will print only the first line in `story.txt`.

Python Append To File

Writing to an opened file with the `'w'` flag overwrites all previous content in the file. To avoid this, we can append to a file instead. Use the `'a'` flag as the second argument to `open()`. If a file doesn't exist, it will be created for append mode.

```
with open('shopping.txt', 'a') as
shop:
  shop.write('Tomatoes, cucumbers,
celery\n')
```

Python Write To File

By default, a file when opened with `open()` is only for reading. A second argument `'r'` is passed to it by default. To write to a file, first open the file with write permission via the `'w'` argument. Then use the `.write()` method to write to the file. If the file already exists, all prior content will be overwritten.

```
with open('diary.txt','w') as
diary:
  diary.write('Special events for
today')
```

Python Readlines Method

Instead of reading the entire content of a file, you can read a single line at a time. Instead of `.read()` which returns a string, call `.readlines()` to return a list of strings, each representing an individual line in the file. Calling this code:

```
with open('lines.txt') as
file_object:
  file_data =
file_object.readlines()
print(file_data)
```

returns a list of strings in `file_data`:

```
['1. Learn Python.\n', '2. Work
hard.\n', '3. Graduate.']
```

Iterating over the list, `file_data`, and printing it:

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

outputs:

```
1. Learn Python.

2. Work hard.

3. Graduate.
```

Class csv.DictWriter

In Python, the `csv` module implements classes to read and write tabular data in *CSV* format. It has a class `DictWriter` which operates like a regular writer but maps a dictionary onto output rows. The keys of the dictionary are column names while values are actual data.

The `csv.DictWriter` constructor takes two arguments. The first is the open file handler that the CSV is being written to. The second named parameter, `fieldnames`, is a list of field names that the CSV is going to handle.

```
# An example of csv.DictWriter
import csv

with open('companies.csv', 'w') as
csvfile:
  fieldnames = ['name', 'type']
  writer = csv.DictWriter(csvfile,
fieldnames=fieldnames)
  writer.writeheader()
  writer.writerow({'name': 'Codecademy',
'type': 'Learning'})
  writer.writerow({'name': 'Google',
'type': 'Search'})

"""
After running the above code,
companies.csv will contain the following
information:

name,type
Codecademy,Learning
Google,Search
"""
```

Python Read Method

After a file is opened with `open()` returning a file object, call the `.read()` method of the file object to return the entire file content as a Python string. Executing the following Python code:

```
with open('mystery.txt') as
text_file:
  text_data = text_file.read()
print(text_data)
```

will produce a string containing the entire content of the read file:

```
Mystery solved.
Congratulations!
```