By: Michelle Morales

How To Handle Plain Text Files in Python 3

Updated October 13, 2016 135.3k PYTHON DEVELOPMENT

Introduction

Python is a great tool for processing data. It is likely that any program you write will involve reading, writing, or manipulating data. For this reason, it's especially useful to know how to handle different file formats, which store different types of data.

For example, consider a Python program that checks a list of users for access control. Your list of users will likely be stored and saved in a text file. Perhaps you are not working with text, but instead have a program that does financial analysis. In order to do some number crunching, you will likely have to input those numbers from a saved spreadsheet. Regardless of your application, it is almost guaranteed that inputting or outputting data will be involved.

This tutorial will briefly describe some of the format types Python is able to handle. After a brief introduction to file formats, we'll go through how to open, read, and write a text file in Python 3.

When you're finished with this tutorial, you'll be able to handle any text file in Python.

Prerequisites

For this tutorial, you should have Python 3 installed as well as a local programming environment set up on your computer. If this is not the case, you can get set up by following the appropriate installation and set up guide for your operating system:

- Ubuntu 16.04 or Debian 8
- CentOS 7
- Mac OS X
- Windows 10

Background

Python is super accommodating and can, with relative ease, handle a number of different file formats, including but not limited to the following:

File type	Description
txt	Plain text file stores data that represents only characters (or strings) and excludes any structured metadata
CSV	Comma-separated values file uses commas (or other delimiters) to structure stored data, allowing data to be saved in a table format
HTML	HyperText Markup Language file stores structured data and is commonly used with most websites
JSON	JavaScript Object Notation is a simple and efficient format, making it one of the most commonly used formats to store and transfer data

This tutorial will focus on the **txt** file format.

Step 1 — Creating a Text File

Before we can begin working in Python, we need to make sure we have a file to work with. To do this, we'll open up a text editor and create a new txt file, let's call it `days.txt`.

days.txt

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Next, save your file and make sure you know where you put it. In our example, our user **sammy**, saved the file here: `/users/sammy/days.txt`. This will be very important in later steps, where we open the file in Python.

Now that we have a txt file to process, we can begin our code!

Step 2 — Opening a File

Before we can write our program, we have to create a Python programming file, so create the file `files.py` with your text editor. To make things easy, save it in the same directory as our `days.txt` file: `/users/sammy/`.

To open a file in Python, we first need some way to associate the file on disk with a variable in Python. This process is called *opening* a file. We begin by telling Python where the file is. The location of your file is often referred to as the file *path*. In order for Python to open your file, it requires the path. The path to our `days.txt` file is: `/users/sammy/days.txt`. In Python, we will create a string variable to store this information. In our `files.py` script, we will create the `path` variable and set the variable to the days.txt path.

files.py

```
path = '/users/sammy/days.txt'
```

We will then use Python's `open()` function to open our `days.txt` file. The `open()` function requires as its first argument the file path. The function also allows for many other parameters. However, most important is the optional *mode* parameter. Mode is an optional string that specifies the mode in which the file is opened. The mode you choose will depend on what you wish to do with the file. Here are some of our mode options:

- `'r'` : use for reading
- `'w'` : use for writing
- `'x'` : use for creating and writing to a new file
- `'a'` : use for appending to a file
- `'r+'` : use for reading and writing to the same file

In this example, we only want to read from the file, so we will use the `'r'` mode. We will use the `open()` function to open the `days.txt` file and assign it to the variable `days_file`.

files.py

```
days_file = open(path,'r')
```

After we have opened the file, we can then read from it, which we will do in the next step.

Step 3 — Reading a File

Since our file has been opened, we can now manipulate it (i.e. read from it) through the variable we assigned to it. Python provides three related operations for reading information from a file. We'll show how to use all three operations as examples that you can try out to get an understanding of how they work.

The first operation `<file>.read()` returns the entire contents of the file as a single string.

```
Output
'Monday\nTuesday\nWednesday\nThursday\nFriday\nSaturday\nSunday\n'
```

The second operation `<file>.readline()` returns the next line of the file, returning the text up to and including the next newline character. More simply put, this operation will read a file line-by-line.

```
days_file.readline()
```

```
Output
'Monday\n'
```

Therefore, once you read a line with the readline operation it will pass to the next line. So if you were to call this operation again, it would return the next line in the file, as shown.

```
days_file.readline()
```

```
Output
'Tuesday\n'
```

The last operation, `<file>.readlines()` returns a list of the lines in the file, where each item of the list represents a single line.

```
days_file.readlines()
```

```
Output
['Monday\n', 'Tuesday\n', 'Wednesday\n', 'Thursday\n', 'Friday\n', 'Saturday\n', 'Sunday\n']
```

Something to keep in mind when you are reading from files, once a file has been read using one of the read operations, it cannot be read again. For example, if you were to first run `days_file.read()` followed by `days_file.readlines()` the second operation would return an empty string. Therefore, anytime you wish to read from a file you will have to first open a new file variable. Now that we have read from a file, let's learn how to write to a new file.

Step 4 — Writing a File

In this step, we are going to write a new file that includes the title Days of the Week followed by the days of the week. First, let's create our `title` variable.

files.py

```
title = 'Days of the Week\n'
```

We also need to store the days of the week in a string variable, which we'll call `days`. To make it easier to follow, we include the code from the steps above. We open the file in read mode, read the file, and store the returned output from the read operation in our new variable `days`.

files.py

```
path = '/users/sammy/days.txt'
days_file = open(path,'r')
days = days_file.read()
```

Now that we have variables for title and days of the week, we can begin writing to our new file. First, we need to specify the location of the file. Again, we will use the directory `/users/sammy/`. We will have to specify the new file we wish to create. So, our path will actually be `/users/sammy/new_days.txt`. We provide our location information in the `new_path` variable. We then open our new file in write mode, using the `open()` function with the `'w'` mode specified.

```
new_path = '/users/sammy/new_days.txt'
new_days = open(new_path,'w')
```

Important to note, if new_days.txt already existed before opening the file its old contents would have been destroyed, so be careful when using the `'w'` mode.

Once our new file is opened, we can put data into the file, using the write operation, `<file>.write()`. The write operation takes a single parameter, which must be a string, and writes that string to the file. If you want to start a new line in the file, you must explicitly provide the newline character. First, we write the title to the file followed by the days of the week. Let's also add in some print statements of what we are writing out, which is often good practice for tracking your scripts' progress.

files.py

```
new_days.write(title)
print(title)

new_days.write(days)
print(days)
```

Lastly, whenever we are finished with a file, we need to make sure to close it. We show this in our final step.

Step 5 — Closing a File

Closing a file makes sure that the connection between the file on disk and the file variable is finished. Closing files also ensures that other programs are able to access them and keeps your data safe. So, always make sure to close your files. Now, let's close all our files using the `<file>.close()` function.

files.py

```
days_file.close()
new_days.close()
```

We're now finished processing files in Python and can move on to looking over our code.

Step 6 — Checking our Code

Before we run our code, let's make sure everything looks good. The final product should look something like this:

files.py

```
path = '/users/sammy/days.txt'
days_file = open(path,'r')
days = days_file.read()


new_path = '/users/sammy/new_days.txt'
new_days = open(new_path,'w')

title = 'Days of the Week\n'
new_days.write(title)
print(title)

new_days.write(days)
print(days)

days_file.close()
new_days.close()
```

After saving your code, open up terminal and run your Python script, like so:

Our output should look like this:

```
Output
Days of the Week

Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Now, let's double check our code fully worked by opening our new file (`new_days.txt`). If all went well, when we open our new file, it should look like this:

new_days.txt

```
Days of the Week
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Your file will look the same or similarly — you have successfully completed this tutorial!

Conclusion

In this tutorial, we went through how to handle and manipulate plain text files in Python 3. Now you can open, read, write, and close files in Python, and you can continue working with your own data in Python.

By: Michelle Morales

How To Create Django Models

How To Perform Neural Style Transfer with Python 3 and PyTorch

How To Use the Machine Learning One-Click Install Image on DigitalOcean

How To Create a Django App and Connect it to a Database

3 Comments

Leave a comment...

Log In to Comment

TheLetterM *April 26, 2017*

0 @MichelleMorales how do I go about passing the filename to the script as an argument instead of hard-coding it?

michellemorales *April 26, 2017*

1 @TheLetterM you could try something like this...

First, in your script add:

```
import sys
path = sys.argv[1]
```

Then, when you run it:

```
$ python script.py file_name.txt
```

simonchan2014 *August 25, 2017*

0 @MichelleMorales if I have a Django webapp running off DigitalOcean, can I use the Django webapp to save csv files onto the DigitalOcean droplet?