Introduction to python for AI

Jan Carbonell

December 3, 2020

Strive School

Strings

Strings I

Strings in python are very similar to a list of chars but they have their own functionality and utility methods.

You can index a string just like you would index a list `[ ]`. You can also check for the presence of another string by using the same interface as with lists `in`.

```
s ="Hello"

s[0] # "H"
s[0:2] # He
"el" in s # True
```

Strings II

String interpolation is one of the things that come really handy one digging into code. Python 3.6 brings the possibility of using a special kind of strings, the **f-strings** that come with a very nice interface.

```
a =1
s =f"A is {a}"
s # A is 1

s ="A is " +str(a) # we need to cast the integer
```

Strings III

Usually we will be dealing with strings, python provides with a set of methods to treat them, some of them are very useful.

```
s ="Hello world! "
s.split(" ") # ["Hello", "world!", ""]

s.strip().split(" ") # ["Hello", "world!"]
```

Module structure

Every python file creates a module, a module is a set of function or classes that can be imported from other files. This lets us divide our logic into semantically similar functionalities.

Typically we will implement a class or a set of methods per file and then import them from other files.

```
#----------------------file1
def f(a, b):
    return a *b
#----------------------file2
from file1 import f
f(2, 2) # 4
```

Files

Handling files

Python lets us read, write and append files easily with the `open(filename, mode)` command. Depending on the mode we tell the OS how we want to open the file.

- **r** Reading.
- **r+** Reading and *Writing* with position at *beginning*.
- **w** Writing erasing file.
- **w+** Writing and Reading erasing file. Creates if file doesn't exist
- **a** Appending to the file.
- **a+** Appending the file.

```
with open("filename", "r+") as file_handler:
    file_handler.readlines()
```

General tip: Don't panic

1. google is your friend
2. worst case scenario, ask for help (literally)

```
help(set)
```

Okay, now how the fuck do I leave help?

```
q
ctrl-z kills python
\textbf{panic method}: close and open the terminal
```

Let's start again: Handling files

1. In Python, there is no need for importing external library to read and write files.
2. Python provides an inbuilt function for creating, writing and reading files.

Today we will learn:

1. Why this is needed
2. How to Create a Text File
3. How to Append Data to a File
4. How to Read a File
5. How to Read a File line by line
6. File Modes in Python

Why files are needed

What happens when we execute a python file or we type an operation to the terminal?

1. Having prints is not enough
2. What if we want to store the results of the program but not the process? (It may take a while)
3. Let's look at a somewhat complex program.

Solution: Files! Text, csv, etc.

How to Create a Text File (1)

With Python you can create a .text files (strive.txt) by coding! Let's see how it is done:

```
f= open("strive.txt","w+")
```

What we have done:

1. Variable f opens a file given 2 arguments, the file and the operation
2. We used w to write and the + to create a new file if it does not exist.
3. We could also "r" read or "a" append.

How to Create a Text File (2)

```
for i in range(10):
   f.write("This is line %d\n" % (i+1)
```

What we have done:

1. We have a for loop that runs over a range of 10 numbers.
2. Using the write function to enter data into the file.
3. The result is a file that tells us which number of line it is.
4. notice the special characters
   n *newline*
5. If you don't understand what something does, **remove it** and see what happens

How to Create a Text File (3)

```
f.close()
```

This will close the instance of the *strive.txt* file (and automatically save it)

How to Create a Text File (4)

Adding all of this together, we have a basic program to write text in files!

```
def create():
    f= open("strive.txt","w+")
    for i in range(10):
        f.write("This is line %d\n" % (i+1))
    f.close()
```

How to Append Data to a File (1)

Sometimes we don't want to erase what is already in a file. We can then append a new text to the already existing file or the new file.

```
f= open("strive.txt","a+")
```

With the + sign it would still create it but in our case we already have the file, so we are not required to create a new file.

How to Append Data to a File (2)

```
for i in range(2):
    f.write("Appended line %d\n" % (i+1))
```

This will write data into the file in append mode.

How to Append Data to a File (3)

Adding all of this together, we have a basic program to write text in files!

```
def append():
    f= open("strive.txt","a+")
    for i in range(2):
        f.write("Appended line %d\n" % (i+1))
    f.close()
```

We can now see that the output in our file has been updated with our new data below

How to Read a File (1)

Not only you can create .txt file from Python but you can also call .txt file in a "read mode"(r).

```
f= open("strive.txt","r")
```

How to Read a File (2)

We use the mode function in the code to check that the file is in open mode. If yes, we proceed ahead

```
if f.mode =="r":
```

How to Read a File (3)

Now we use f.read to read file data and store it in variable content

```
    contents =f.read()
```

How to Read a File (4)

Adding all of this together, we have a basic program to write text in files!

```
def read():
    f= open("strive.txt","r")
    if f.mode =='r':
        contents =f.read()
        print(contents)
    #No need to close here, but it is still good practice to
                                                do so.
```

Thinking

1. What if I want to do certain operations in every line? Ex: Web Scrapping? Ex2: Only store something in another file if it contains an A
2. What if I don't want to save / print the entire text? Think about big files like all the wikipedia entries?

Solution: Reading line by line!

How to Read a File Line by Line (1)

Adding all of this together, we have a basic program to write text in files!

```
def linebyline():
    f= open("strive.txt","r")
    f1 =f.readlines()
    for line in f1:
        if line =='\n':
            print("found an empty line")
        else:
            print(x)

    #No need to close here, but it is still good practice to
                                                do so.
    f1.close()
```

The power of Python

One of the best things about Python (especially if you're a data scientist) is the vast number of high-quality custom libraries that have been written for it.

Google: **python external libraries**

Also, this open-source approach is the main reason Python became the main language for Deep Learning (Researchers working together).

The standard library

Some of these libraries are in the "standard library", meaning you can find them anywhere you run Python. **It is always good to check if something is already in the standard library before trying to code it yourself**.

Ex: Can we tell what time it is now with Python?

Math module

One of the most famous libraries is the math one (useful for DL ;))

```
import math

print("It's math! It has type {}".format(type(math)))
```

Math is a module. A module is just a collection of variables (a namespace, if you like) defined by someone else. We can see all the names in math using the built-in function dir(). **This is very useful to know what any module does**

```
print(dir(math))
```

How do we access this variables?

The computer needs to know it comes from the math module. Thus, we need to declare it beforehand with math.function.

```
print("pi to 4 significant digits = {:.4}".format(math.pi))
```

But most of what we'll find in the module are functions, like math.log:

```
math.exp(32, 2)
```

Of course, if we don't know what math.log does, we can call help() on it:

```
help(math.log)
```

We can also call help() on the module itself. This will give us the combined documentation for all the functions and values in the module.

Other import syntax

If we know we'll be using functions in math frequently we can import it under a shorter alias to save some typing (though in this case "math" is already pretty short).

```
import math as mt
mt.pi
```

Now you are thinking... we only save two letters. What about this?

```
import pandas as pd
import numpy as np
import import sklearn.preprocessing.scale as prepro
```

Very often used in ML / DL to the point that some things are a standard (numpy as np, pandas...

Can we optimize more?

Wouldn't it be great if we could refer to all the variables in the math module by themselves? i.e. if we could just refer to pi instead of math.pi or mt.pi? **Good news**: we can do that.

```
from math import *
print(pi, log(32, 2))
```

import * makes all the module's variables directly accessible to you (without any dotted prefix).

Bad news: some purists might grumble at you for doing this.

I still want to optimize...

Even Worse News : they kind of have a point.

```
from math import *
from numpy import *
print(pi, log(32, 2))
```

Error! But it worked before! These kinds of "star imports" can occasionally lead to weird, difficult-to-debug situations. In this case, the math and numpy modules both have functions called log, but they have different semantics.

Because we import from numpy second, its log overwrites (or "shadows") the log variable we imported from math.

Let's compromise

A good compromise is to import only the specific things we'll need from each module:

```
from math import log, pi
from numpy import asarray
```

Submodules

We've seen that modules contain variables which can refer to functions or values. Something to be aware of is that they can also have variables referring to other modules.

```
import numpy
print("numpy.random is a", type(numpy.random))
print("it contains names such as...",
      dir(numpy.random)[-15:]
     )
```

So if we import numpy as above, then calling a function in the random "submodule" will require two dots.

```
# Roll 10 dice
rolls =numpy.random.randint(low=1, high=6, size=10)
rolls
```

Toolkit for the future

Three tools for understanding strange objects

1. type() (what is this thing?) *Ex: type(rolls)*
2. dir() (what can I do with it?) *Ex: print(dir(rolls))*
3. help() (tell me more) help(rolls.tolist())

The end

Thank you!