Module 5 - Overview

Module Learning Outcomes

After successful completion of this module, you will be able to do the following:

1. Read, write, and append text files.
2. Use the pickle module to pickle and un-pickle data.
3. Use the json module to write to and read from a JSON file.

Key questions:

* What's the difference between text files and binary files?
* What are the differences between using pickle and using json for object persistence?

Explorations

Use the pages within this module to explore the following concepts:

* Exploration: [File handling, pickling, JSON](https://canvas.oregonstate.edu/courses/1915078/pages/exploration-file-handling-pickling-json) (MLOs 1-3)
* Video Demo: [File Handling, Pickle, JSON](https://canvas.oregonstate.edu/courses/1915078/pages/video-demo-file-handling-pickle-json) (MLOs 1-3)
* [Module 5 exercise solutions](https://canvas.oregonstate.edu/courses/1915078/pages/module-5-exercise-solutions)

Optional Resources

* [*Think Python* Chapter 14Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2015.html)

Task List

Complete the following assignments and other tasks:

* Read the Exploration pages and do the interactive exercises on those pages (MLOs 1-3).
* Do [Assignment 5](https://canvas.oregonstate.edu/courses/1915078/assignments/9227005), which gives you practice with reading and writing text files, and with using pickle and json for object persistence (MLOs 1-3).
* Take [Quiz 5](https://canvas.oregonstate.edu/courses/1915078/quizzes/2859160) (MLOs 1-3).

# Exploration: File handling, pickling, JSON

## File handling

Files are ubiquitous in computing. Storing data in some kind of file is how we are able to have it persist and be retrievable in the future, even if the computer gets turned off. There are two main kinds of files, text files and binary files. Text files are composed of characters and are usually human readable. Binary files are not restricted to characters - they can be arbitrary sequences of bits - and are usually not human readable. For example, the .py files you write are text files, but .jpg files are binary files. In this section, we'll look at how to read and write text files.

First, you need to open a file. The first parameter of the open command is the name of the file. If the file is not in the same directory as your program, you need to include the path name (e.g. 'C:\Program Files\python\hello\_world.py'). The second parameter of the open command is 'r' if you're reading from a file, or 'w' if you're writing to a file.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The readline() function reads an entire line as a string, up to and including the newline character at the end that separates lines of the file. If you enter this in the lower pane of the code editor:

line

You'll see that it prints the first line of dogs.txt with the "\n" newline character I mentioned.

If the code in the example had called readline() again (before closing the file), it would have read the second line of the file, and so on. The '\n' is a newline character. If we don't want that, we can get rid of whitespace at the ends of a string by using strip(). You can see the results by entering this in the code editor:

line\_stripped

And you'll notice that the newline character is gone. You can see that I close the file via close() at the end. We could alternatively use a try...finally here to make sure the file gets closed:

infile = open('dogs.txt', 'r')  
try:  
 line = infile.readline()  
finally:  
 infile.close()

However, as mentioned in the lesson on exception handling, using finally this way has been superseded by the **with** statement, which automatically closes the file. Using a with statement looks like this:

with open('dogs.txt', 'r') as infile:  
 text = infile.readline()

We can iterate over all the lines in a file with a for loop like this:

with open('dogs.txt', 'r') as infile:  
 for line in infile:  
 print(line.strip())

This opens dogs.txt for reading, prints out each line, and then closes the file. However, what if the file is missing, or what if we typed its name wrong? That would raise the exception FileNotFoundError. We can use a try...except to handle that exception:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Now if the file isn't there, instead of crashing, your program prints out a helpful message.

### **Writing to a file**

Here's an example that opens a file for writing and iterates through a list, writing each element to the file. The with statement will automatically close the file. When you run this code example, you'll see how the 'cats.txt' file is now listed along with 'main.py'.

Graphical user interface, text, application

Description automatically generated

Concatenating with the newline character advances us to the next line of the file. We don't have to worry about handling a FileNotFoundError. If the file doesn't exist, it will be created. If the file does exist, it will be **overwritten** by the new contents.

The argument passed to the write method must be a string. If we want to write numbers to the file, they must first be converted to strings. The simplest way is to cast them with str().

Graphical user interface, text, application

Description automatically generated

To append to an existing file instead of overwriting it, use 'a' instead of 'w' as the second argument for open(). When appending to a file, you may need to handle a FileNotFoundError.

## Pickling

It's easy enough to convert numbers to strings, but what if we want to save lists, sets, dictionaries, etc.? The **pickle** module provides a way for us to do that. The file produced by pickling is not a text file and is not human readable. To write an object to a file, we use dump(), which takes as parameters the object and the file to write to, which means we need to open a binary file for writing. Since it's a binary file, we use 'wb' instead of 'w'.

Graphical user interface, text, application, email

Description automatically generated

To read an object from a file, we use load(), which takes as its single parameter the file to read from, and returns the restored object. Since we are reading from a binary file, we use 'rb' instead of 'r'.

If you print out restored\_list, you'll see it has the same elements as cat\_list, but they are two separate lists:

> cat\_list == restored\_list # true  
> cat\_list is restored\_list # false

**Don't un-pickle data from an untrusted source.**

## JSON

If you're working entirely within Python, then pickle is probably what you want. However, if you want to be able to exchange data between different computer languages, then you'll need to use something else. **JSON** stands for JavaScript Object Notation, but although it came from JavaScript, it is language independent. JSON files are text files that are human readable (at least if you're familiar with computer code). Using JSON is also more secure than using pickle. Let's try it with the same list from the pickle example:

Graphical user interface, text, application, email

Description automatically generated

Looks familiar, right? All we've changed is the module name, the file extension, and that we're writing to a text file instead of a binary file. Unlike a pickle file, we can open up and read the JSON file we just created (click on the File icon to see it in the code editor).

The mapping between Python types and JSON types is not perfectly one-to-one. If you save a tuple to a JSON file and then read it back in, you'll get a list instead. For most other types though, you'll get back the same type of object that you saved.

Let's try creating a slightly more complicated JSON file using an example borrowed from Mozilla, [Working with JSONLinks to an external site.](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON). Create a new file named SuperSquad.json and enter the following:

{  
 "squadName": "Super hero squad",  
 "homeTown": "Metro City",  
 "formed": 2016,  
 "secretBase": "Super tower",  
 "active": true,  
 "members": [  
 {  
 "name": "Molecule Man",  
 "age": 29,  
 "secretIdentity": "Dan Jukes",  
 "powers": [  
 "Radiation resistance",  
 "Turning tiny",  
 "Radiation blast"  
 ]  
 },  
 {  
 "name": "Madame Uppercut",  
 "age": 39,  
 "secretIdentity": "Jane Wilson",  
 "powers": [  
 "Million tonne punch",  
 "Damage resistance",  
 "Superhuman reflexes"  
 ]  
 },  
 {  
 "name": "Eternal Flame",  
 "age": 1000000,  
 "secretIdentity": "Unknown",  
 "powers": [  
 "Immortality",  
 "Heat Immunity",  
 "Inferno",  
 "Teleportation",  
 "Interdimensional travel"  
 ]  
 }  
 ]  
}

Then use this Python code to read the JSON file into a Python object:

with open('SuperSquad.json', 'r') as infile:  
 squad = json.load(infile)

We now have a dictionary named "squad", which contains various key-value pairs for attributes of the super squad. One of the keys is "members", whose associated value is a list of all the members of the super squad. Each element of that list is a dictionary, which contains various key-value pairs for attributes of that hero. One of the keys in that dictionary is "powers", whose associated value is a list of that hero's super abilities. To print out Madame Uppercuts third superpower, we would do this:

print(squad["members"][1]["powers"][2])

super\_heroes["members"] gives us the list of heroes, then the [1] gets us the second hero in the list, then ["powers"] gets us the list of her powers, and finally [2] gets us her third superpower (superhuman reflexes).

**When you look at a JSON file in PyCharm, it may not be formatted in a way that's easy to read. If that's the case, you can simply select the JSON code and then select Code | Reformat Code from the menu.** There are also online sites for visualizing JSON such as [this oneLinks to an external site.](http://jsonviewer.stack.hu/) you can try, to see if you like their presentation better.

## Exercises

Try these out on your computer using PyCharm:

As you work on these questions, pay attention to the process of reading a description of requirements and figuring out a specific list of tasks. This is an important skill that you can practice on these simple exercises before you try the requirements of the assignments.

1. Write code that opens a file named "Mary.txt" and writes to that file the first four lines of the nursery rhyme "Mary had a little lamb", with each line being written on a different line of the file. Now write code that opens that file and reads it, printing out each line as it goes. Use strip() to strip off the newline characters.

2. Create a dictionary that maps the numbers 1-13 to the names: William, Patrick, Jon, Tom, Peter, Colin, Sylvester, Paul, Chris, David, Matt, Peter, and Jodie. Pickle the dictionary and then un-pickle it to a dictionary called "names" and verify it's correct.

The unit test file can be downloaded from [here: File handling 2.py](https://canvas.oregonstate.edu/courses/1915078/files/98541999?wrap=1)[Download here: File handling 2.py](https://canvas.oregonstate.edu/courses/1915078/files/98541999/download?download_frd=1)

3. Do the same as #2, but instead of pickling and un-pickling the dictionary, write it to, and read it from, a JSON file.

The unit test file can be downloaded from [here: File handling 3.py](https://canvas.oregonstate.edu/courses/1915078/files/98541973?wrap=1)