Hedy Khalatbari
August 14, 2021
Foundations of Programming: Python
Assignment07
GitHub repo URL: https://github.com/Seattle15/IntroToProg-Python-Mod07
GitHub webpage URL: https://seattle15.github.io/IntroToProg-Python-Mod07/

Programming with Python: Module 7

Introduction

Step 1 – Watch the video for Module 7

1) **working with text files**

   a. **read, write, and append modes**: can define a function for each in your script

      - if you try reading a file that does not exist you will get an error message; this is not the case in write and append modes
      - use 'for loop' with read mode to iteratively read rows of data and append it to a list (or tuple or string)

   b. **reading data options**

      - **.readline()**: reads one row at a time and then advances to the next row
        - use repeatedly to read more lines, for example with a while loop

      - **.readlines()**: reads all lines of data and returns a list. **.read()** reads all lines and returns a string
        - everything is pulled in from file and stored in memory; rather that the option of iterating thru lines and only saving the data you wanted

   c. **combining reading and writing**

      - you can write a function that contains options for all three (read, write, and append) modes (i.e., a custom wrapper function)
        - or have separate custom functions for each
        - note that the concept of write and append modes may be more clearly presented as overwrite and write to the user; so that they understand that with the 'write mode' they would be losing all prior data

2) **working with binary files**

a. **pickling**: saving data in a binary format (instead of plain text). This can obscure the file's content (but not encrypt it) and may reduce the file's size

b. **import pickle, pickle.dump(), pickle.load()**

3) **structured error handling (try-except)**: trap errors due to interactions of humans with your code in a try-except block. Other languages may call if a try-catch block. Gives the programmer more control over the error handling messages that users will see

a. **using the exception class**: a class is used to create objects. Python creates an Exception object when an error occurs that includes the information about the error. The except block allows you to capture a variable containing the error message generated the error

- in the except block, you must specify the type of 'Exception as e' – which is counter to how Python usually works. You can then print out more data about 'e' (Box 1)

Box 1 – Try-except block

Try- except block script	Running

`try:`
`  quotient = 5/0`
`  print(quotient)`
`except Exception as e:`
`  print("There was an error! << Custom Message")`
 	
`  print("Built-In Pythons error info: ")`
`  print(e)`
`  print(type(e))`
`  print(e.__doc__)  # class attribute __doc__ of object`
`  print(e.__str__())  # base exception; return str(self)`
	C:\Python39\python.exe
	C:/_PythonClass/ModDemos/test_Mod07.py
	There was an error! << Custom Message
	
	Built-In Pythons error info:
	division by zero
	<class 'ZeroDivisionError'>
	Second argument to a division or modulo operation was zero.
	division by zero

b. **catching specific exceptions**: Exception class can catch any type of error, but you can catch specific errors using more specific exception classes. These are illustrated in Box 2. Note that blocks 1-3 of code are executed (and corresponding running section demonstrated) for the first to third lines of the try block respectively (with the other lines commented out). They are included in this Box all together for demonstration purposes.
The generic exception block should always be the last one, otherwise, it will catch all errors and the error will never reach the more specific exceptions

- Built-in Exceptions — Python 3.9.6 documentation(external site) There is class hierarchy for built-in exceptions and the first two tiers of this hierarchy are summarized in Box 3

Box 2 – Try-except block with specific exceptions

Note that blocks 1-3 correspond to lines 1 to 3 in try block- with other lines commented out

Try- except block script	Running

try:
  quotient = 5/0
  f = open('SomeFile.txt', 'r+') *# the read plus option gives an error if file does not exist*
  f.write(quotient) *# causes an error if the file does not exist*
	C:\Python39\python.exe
	C:/_PythonClass/ModDemos/test_Mod07.py

except ZeroDivisionError **as** e: *# block 1*
  print("Please do no use Zero for the second number!")
  print("Built-In Python error info: ")
  print(e, e.__doc__, type(e), sep='\n')
	Please do no use Zero for the second number! *# block 1*
	Built-In Python error info:
	division by zero
	Second argument to a division or modulo operation was zero.
	<class 'ZeroDivisionError'>

except FileNotFoundError **as** e: *# block 2*
  print("Text file must exist before running this script!")
  print("Built-In Python error info: ")
  print(e, e.__doc__, type(e), sep='\n')
	Text file must exist before running this script! *# block 2*
	Built-In Python error info:
	[Errno 2] No such file or directory: 'SomeFile.txt'
	File not found.
	<class 'FileNotFoundError'>

except Exception **as** e: *# block 3*
  print("There was a non-specific error!")
  print("Built-In Python error info: ")
  print(e, e.__doc__, type(e), sep='\n')
	There was a non-specific error! *# block 3*
	Built-In Python error info:
	name 'f' is not defined
	Name not found globally.
	<class 'NameError'>

Box 3 - Class hierarchy for built-in exceptions – the top two tiers

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
     +-- StopIteration
     +-- StopAsyncIteration
     +-- ArithmeticError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- EOFError
     +-- ImportError
     +-- LookupError
     +-- MemoryError
     +-- NameError
     +-- OSError
     +-- ReferenceError
     +-- RuntimeError
     +-- SyntaxError
     +-- SystemError
     +-- TypeError
     +-- ValueError
     +-- Warning
```

c. **raising custom errors**: use 'raise Exception' and you can print out a custom error message (Box 4)

- you can use the basic exception class or other classes
- you can use 'raise Exception' without a try-except block (last block row of code in Box 4)

Box 4 – Raising custom errors

Try-except bloc with raise Exception script

```
try:
   new_file_name = input("Enter the name of the file you want to make: ")
   if new_file_name.isnumeric():
      raise Exception('Do not use numbers for the file\'s name')
except Exception as e:
   print("There was a non-specific error!")
   print("Built-In Python error info: ")
   print(e, e.__doc__, type(e), sep='\n')
```

Running

```
C:\Python39\python.exe
C:/_PythonClass/ModDemos/test_Mod07.py
Enter the name of the file you want to make: 123
There was a non-specific error!
Built-In Python error info:
Do not use numbers for the file's name
Common base class for all non-exit exceptions.
<class 'Exception'>
```

Raise exception without a try-except block

```
new_file_name = input("Enter the name of the file you want to make: ")
if new_file_name.isnumeric():
   raise Exception("Do not use numbers for the file's name")
```

Running

```
C:\Python39\python.exe
C:/_PythonClass/ModDemos/test_Mod07.py
Enter the name of the file you want to make: 123
Traceback (most recent call last):
  File "C:\_PythonClass\ModDemos\test_Mod07.py", line 3, in <module>
    raise Exception("Do not use numbers for the file's name")
Exception: Do not use numbers for the file's name
```

d. **creating custom exception classes**: you can create your own custom classes with more features

4) creating advanced GitHub pages: creating a Markdown file and formatting the page. I supplemented my learning with the course 'How to deploy a website' on Code Academy

- use Jekyll to create a markdown (md) file; Jekyll engine converts markdown language to HTML
- some tips for formatting with Jekyll are summarized in Table 1
- GitHub Pages (public webpages that are hosted and published through GitHub) is used to deploy a website created with the Jekyll markdown (Figure 1)
- create a new public repository -> create an index.md file (via Add file -> Create new file -> docs/ -> index.md)
- you an edit the index.md file with markdown language in the 'edit file' tab and click 'preview' to see what it looks like

- I tried out the markdown language on GitHub (Figure 2) and published it (Figure 3)
  - publish steps: Settings ->GitHub pages -> Source (select main and docs and then save)
  - my GitHub page URL: https://seattle15.github.io/ITFnd100-Mod07/
  - I checked that all the features (including the image) were performing as expected – I had to make minor changes to my list (ensuring that spaces were at the end of each entry to indicate a new line)

Table 1- Jekyll markdown

Page component	Format in markdown
Headers	#, ##, ###, …
Italics	**italics** or _italics_
Bold	****bold**** or __bold__
Strikethough	~~~~strikethough~~~~
Bold and nested italic	****bold and nested _italic_****
All bold and italic	******all bold and italic******
Ordered list	use numbers
Unordered list	use *, + or –
Inline link	[website_name](URL) or
Links	URL and <URL> will automatically get turned into links
Images	Upload images by dragging and dropping, selecting from a file browser, or pasting; resize it to desired dimensions before uploading
Code blocks	fence blocks of code with ``` (triple back ticks) or indent with four spaces
Quote inline code	written as `*inline code here*` (encased with back ticks)
Tables	constructed using '|' and '– ' and use ':' for text alignment
Block quotes	use >
Horizontal line across page	use three or more sequential ---, ***, or underscores
New line	put two or more spaces at the end of the line
Line breaks	hit enter twice

Sources Basic writing and formatting syntax - GitHub Docs(external site)
Organizing information with tables - GitHub Docs (external site)

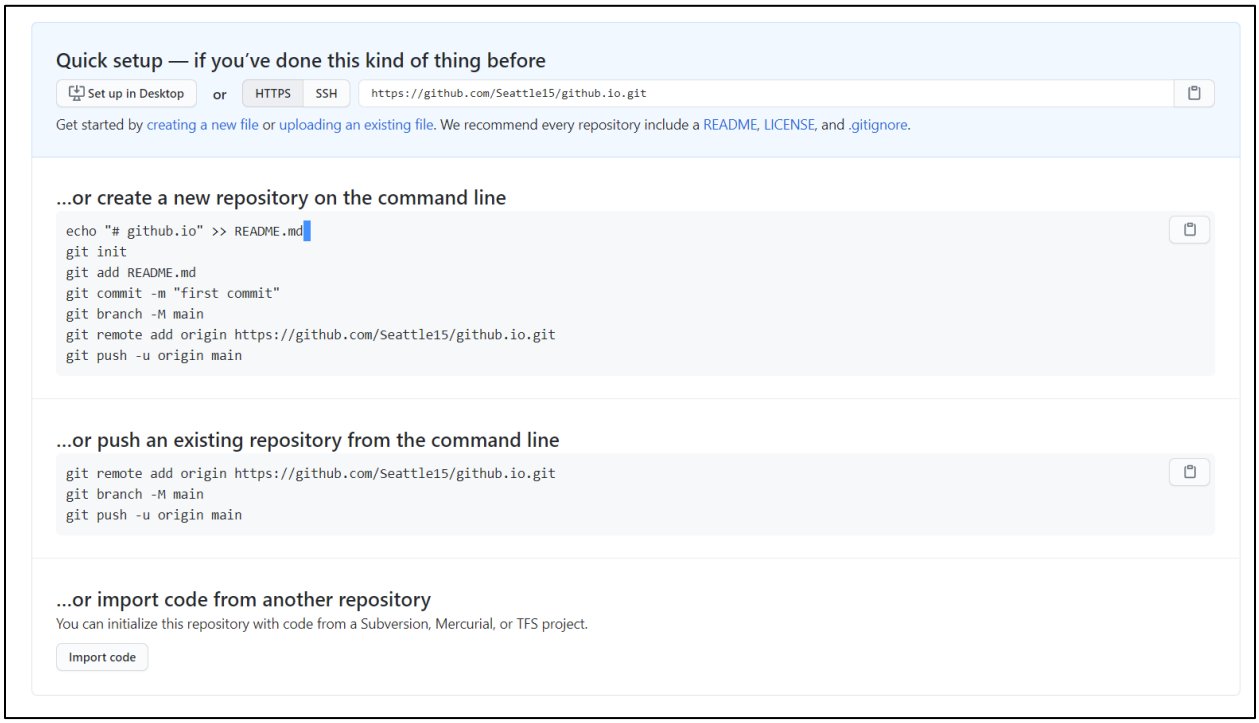

Figure 1. Screen capture of instructions on GitHub for setting up a repository

Figure 2. Screen captures of the 'Edit File' and 'Preview' contents demonstrating experimentation with markdown language on GitHub

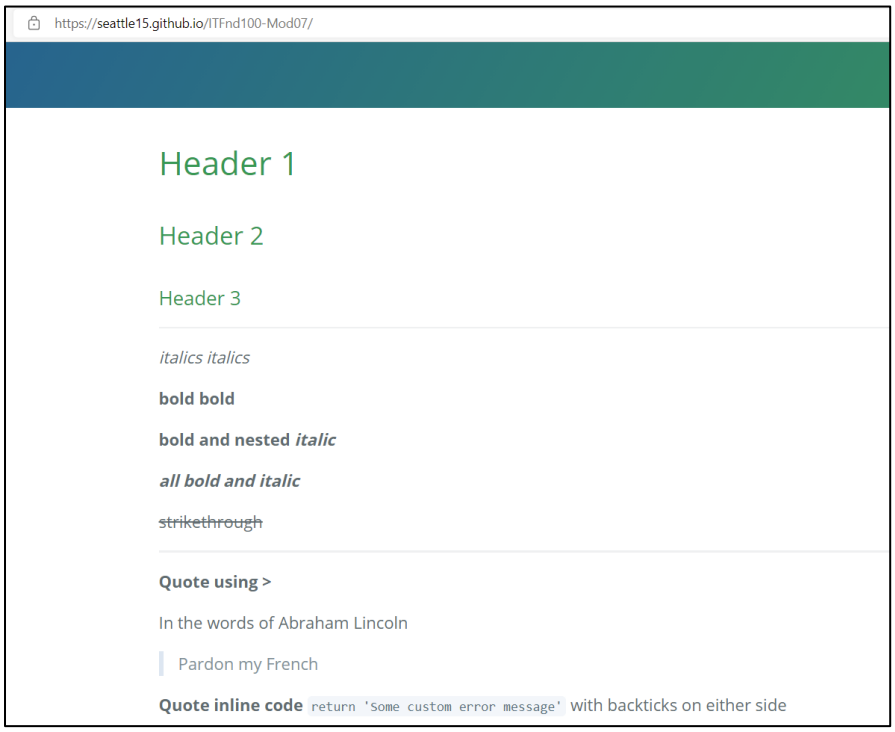

Figure 3. Screen capture of published GitHub page illustrating the result of using the markdown language

5) Files in Python - additional reading

Reading and writing to text files

- with keyword invokes a context manager for the file and automatically closes the file at the end of the block, i.e., there is no need to use obj_file.close(). It replaces older ways of file access

- general syntax – read() # reads all document; 'r' is default argument

```
with open('file_name.txt', 'r') as obj_file:
    file_contents = obj_file.read()
print(file_contents)
```

- general syntax – readlines() # reads all lines, iterate through lines with for loop

```
with open('file_name.txt', 'r') as obj_file:
    for line in  obj_file.readlines():
        print(line)
```

- general syntax – readlines() # reads one line, placing cursor at the start of next line

```
with open('file_name.txt', 'r') as obj_file:
first_line = obj_file.readline() ;  print(first_line)
second_line = obj_file.readline() ;  print(second_line)
```

third_line = obj_file.readline() ; print(third_line)

- general syntax – write() # writes string to file; 'w' overwrites file and 'a' appends to file
  with open('file_name.txt', 'w') as obj_file:
  obj_file.write('Python is awesome!')

Comma separated values (CSV) – from Python documentation (cited below)

- in CSV files the values are either separated by commas or other delimiters (such as tabs or semicolons)

- CSV format: import and export format for spreadsheets and databases
  - reads and writes tabular data in CSV format
  - csv module has reader and writer objects
  - use 'import csv' to use this module in your script

- **functions- reader and writer**

  csv.**reader**(*csvfile*, *dialect='excel'*, ***fmtparams*)
  - 'return a reader object which will iterate over lines in the given *csvfile*
  - If *csvfile* is a file object, it should be opened with newline = '' '. e.g.:
    with open('sample_file.csv', newline ='') as csfile

  csv.**writer**(*csvfile*, *dialect='excel'*, ***fmtparams*)
  - 'return a writer object responsible for converting the user's data into delimited strings on the given file-like object'

- **DictReader and DictWriter classes**: used to read and write data in dictionary form

  csv.**DictReader**(*f*, *fieldnames=None*, *restkey=None*, *restval=None*, *dialect='excel'*, **args*, ***kwds*)
  - DictReader: 'create an object that operates like a regular reader but maps the information in each row to a dictionary whose keys are given by the optional *fieldnames* parameter'
  - the default keys of the dictionary are the entries in the first line of the CSV file

  csv.**DictWriter**(*f*, *fieldnames*, *restval=''*, *extrasaction='raise'*, *dialect='excel'*, **args*, ***kwds*)
  - DictWriter: 'create an object which operates like a regular writer but maps dictionaries onto output rows'

- sample code

```
with open('file_name.csv') as csv_file:
   csv_file_dictionary = csv.DictReader(csv_file)
   for row in csv_file_dictionary:
      print(row['key_name'])

with open('file_name.csv', newline='') as csv_file:     # to account for \n in csv file
   csv_file_dictionary = csv.DictReader(csv_file, delimiter=';')   #   value; value
   for row in csv_file_dictionary:
      print(row['key_name'])
```

```
with open('file_name.csv', 'w') as output_csv:
    fields = ['key_1', 'key_2', 'key_3']    # define the fields
    output_writer = csv.Dictwriter(output_csv, fieldnames=fields)   # instantiate
    output_writer.writeheader()   # add headers, i.e., fieldnames
    for item in list_of_dictionaries:
        output_writer.writerow(item)   # write each dictionary row as a csv row
```

- source csv — CSV File Reading and Writing — Python 3.9.6 documentation (external site)

6) **Naming conventions in** Python – additional reading

a. proper naming conventions for variables, functions, methods, etc.:

- 'variables, functions, methods, packages, modules: this_is_a_variable
- classes and exceptions: CapWords
- protected methods and internal functions: _single_leading_underscore
- private methods: __double_leading_underscore
- constants: CAPS_WITH_UNDERSCORES'

b. indentation: use 4 spaces

- source Python Best Practices - Every Python Developer Must Know - DataFlair (data-flair.training) (external site)

LAB 7-1. Working with binary files

Listing for this lab is in pasted in the box below. The script performed as expected. I tried printing the returned value directly from the function and after assigning it to a variable. After listening to Mr. Root's explanations I added the index to the list.

LAB7-1 script	Running

`# ---------- #`
`# Title: Lab7-1`
`# Description: A simple example of storing data in a binary file`
`# ChangeLog: (Who, When, What)`
`# Hedy Khalatbari,07.28.2021,Created Script`
`# ---------- #`
`import pickle  # This imports code from another code file!`
 	
`# Data ---------- #`
`strFileName = 'AppData.dat'`
`lstCustomer = []`
 	
`# Processing ---------- #`
`def save_data_to_file(file_name, list_of_data):`
`  objFile = open(file_name, "ab")`
`  pickle.dump(list_of_data, objFile)`
`  objFile.close()`
	C:\Python39\python.exe C:/_PythonClass/ModDemos/LAB7-1.py
	
	Enter an Id: 100
	Enter a Name: Hedy
	[100, 'Hedy']
	Data saved is:
	[100, 'Hedy']
	Hedy
	
	Process finished with exit code 0

```
def read_data_from_file(file_name):
    objFileData = ''
    objFile = open(file_name, "rb")
    objFileData = pickle.load(objFile)  # load() only loads one row of data
    objFile.close()
    return objFileData   # this is a list object

# Presentation ---------------------------------- #
# Get ID and NAME From user, then store it in a list object
intId = int(input("Enter an Id: "))
strName = str(input("Enter a Name: "))
lstCustomer = [intId, strName]

# store the list object into a binary file
save_data_to_file(strFileName, lstCustomer)

# Read the data from the file into a new list object and display the
contents
print(read_data_from_file(strFileName))
cust_id_name = read_data_from_file(strFileName)  # assign return value
to a variable
print(' Data saved is:')
print(cust_id_name)                    # print the variable
print(cust_id_name[1])                  # variable is a list & can use index
```

Step 2 - Read a book chapter

I read chapter seven in the course textbook.

- selected text file access modes: 'r', 'w', 'a', 'r+', 'w+', 'a+'
- selected file object methods: close(), read([size]), readline([size]), readlines(), write(ouput), writelines(*output*)
    - readlines() returns lines in a file as elements in a list
    - writelines(*output*) writes the strings in the list *output* to a file
- selected binary file access modes: 'rb', 'wb', 'ab', 'rb+', 'wb+', 'ab+'
- **pickle module**: used to pickle and store more complex data in a fila
- pickle.dump(data_to_pickle, file_to_store_to)
- pickle.load(file_to_load_from); sequential pickle.load() functions read the pickled objects in order of dumping from the list
- **shelve module**: used to store and randomly access pickled objects in a file. Works like a dictionary
- *create a shelf* shelf_name = shelve.open(pickled_file, [access mode]) ->
  add lists shelf_name ['key_list_name'] = [*list elements*]
  ensure data is written to file shelf_name.sync() *# changes written to a buffer and then periodically to the file*
  .
  close shelf file shelf_name.close() *# this also updates the file*
  retrieve & print pickled data print(shelf_name ['key_list_name']) *# prints list elements*
- shelve module access modes
    - 'r' and 'w': read and write access modes
    - 'c': open a file for reading or writing; creates the file if it does not exist. Default access mode (i.e., used then not specified)
    - 'n': create a new file for reading or writing; if it exists its contents are overwritten

- selected exception types: IOError, IndexError, KeyError, NameError, SyntaxError, TypeError, ValueError, ZeroDivisionError
- try-except and try-except-else blocks

Step 3 - Research Exception Handling in Python

- I have documented my reading and research n exception handling in Step 1:
  - The two top tiers of hierarchy for exception handling in Python are summarized on Box 3 (on page 3)
  - Box 2 and box 4 (on paged 3 and 4) demonstrate sample scripts for exception handling and custom errors

- Python - Exceptions Handling - Tutorialspoint (external site; difficulty level: beginner). I have directly quoted excerpts from the website:
  - 'Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them: exception handling and assertions'
  - **EOFError** is defined as ,' Raised when there is no input from either the raw_input() or input() function and the end of file is reached' – I use this in my assignment script (based on a script that I saw on Stack Overflow)
  - **exception:** 'an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions'
  - **handling an exception:** 'If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible'; refer to Box 5.
    - you can use one *except* statement to handle multiple exceptions
  - **else clause:** 'After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception. The else-block is a good place for code that does not need the try: block's protection.'

- Built-in Exceptions — Python 3.9.6 documentation (external site; difficulty level: hard)

Box 5 – Syntax of try except blocks

```
try:
   You do your operations here;
   ......................
except Exception_1:
   If there is Exception_1, then execute this block.
except Exception_2:
   If there is Exception_2, then execute this block.
   ......................
```

else: If there is no exception, then execute this block.

Step 4 - Research Pickling in Python

- **Web tutorials:**

1. Python Pickle Module for saving objects (serialization) - Bing video (external site)
    a. names for pickling in other languages: flattening, serialization, and marshalling

2. Using the Python pickle Module - Bing video (external site)

3. Pickling Data With Python! - Bing video (external site)

4. Python3 Advanced Tutorial 11 - Serialization with Pickle - Bing video (external site)
    a. This was an advanced tutorial, and the first half was relevant to this class

- **Web pages:**

1. Python Pickling (tutorialspoint.com) (external site; difficulty level: beginner) and Python Pickle Example - JournalDev (external site; difficulty level: beginner)
    a. **Why do we pickle?** pickling and unpickling allow us to easily transfer data from one server or system to another and then store the data in a file or database
    b. import pickle module; to write and read as a pickled file use the 'wb' and 'rb' (writing-binary and reading-binary respectively bytes)
    c. pickle.dump() function: used to store object data to the file. Takes three arguments: object data, file object and key-value argument
    d. pickle.load() function: used to restore object data from the file

2. pickle — Python object serialization - GeeksforGeeks (external site; difficulty level: hard). Quoted from webpage:
    a. 'The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.
    b. **Pickling**: It is a process where a Python object hierarchy is converted into a byte stream.
    c. **Unpickling**: It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.'
    d. **Functions**: *dump()* and *load()* are used to serialize and se-serialize respectively

3. Python Pickle Tutorial - DataCamp (external site: difficulty level: intermediate)

4. pickle — Python object serialization — Python 3.9.6 documentation (external site; difficulty level:hard)

- problem solving: I was unable to pickle.load more than one object from my pickled file. After an exhaustive (literally five hours plus) search I finally found a script in Stack Overflow that worked python - How to read pickle file? - Stack Overflow (external site)

Box 6 – Pickling script samples for dumping loading more than one object (from Stack Overflow)

```
def add_to_pickle(path, item):                  # addend to file (not write)
   with open(path, 'ab') as file:
      pickle.dump(item, file, pickle.HIGHEST_PROTOCOL)


objects = []                                    # read all objects into a list
with (open("myfile", "rb")) as openfile:
   while True:
      try:
         objects.append(pickle.load(openfile))
      except EOFError:
         break
```

Step 5 - Apply your knowledge

I tried numerous times to load several objects dumped into a pickled file (with sequential load statements) but was unsuccessful. I have documented my quest in the for finding the solution to this in Step 4 above – Mr. Root will probably be mentioning the solution when he gives us a preview of the assignment for this Module, but I find it much more fulfilling (and challenging) to find the answers (albeit imperfect) on my own.
I settled for a simple demonstration of error handling where the unpickle_from_file function uses a while loop to load the sequential objects and once there is no more data to load the EOFError result in a break.
The script is simple, and I provide the input – rather than asking for user input- as my main goal was to demonstrate successful pickling and unpickling.

Box 7 – Assignment for pickling and exception handing

Script

```
# ------------------------------------------------ #
# Title: Demo script for pickling and error handling
# Description: write a script that demonstrate pickling
# ChangeLog: (Who, When, What)
#  Hedy Khalatbari, 08/13/2021, Created Script
# ------------------------------------------------ #

# -- Data -- #
import pickle

list_steps_daily = [1000, 1500, 2000, 2500, 3000, 3000,
2500, 2000, 1500, 1000]
list_dates = ['080121', '080221', '080321', '080421',
'080521', '080621', '080721',
          '080821', '080921', '081021']
```

Running script

```
C:\_PythonClass\Assignment07\venv\Scripts\python.e
xe C:/_PythonClass/Assignment07/Assignment07.py
The number of daily steps documented in the Health
App in my iPhone in the first ten days of August are:
['080121', '080221', '080321', '080421', '080521',
'080621', '080721', '080821', '080921', '081021']
[1000, 1500, 2000, 2500, 3000, 3000, 2500, 2000,
1500, 1000]

Program will pickle these lists.
Click ENTER to continue...
Object was pickled
Program will now unpickle these lists.
Click ENTER to continue...
```

```
file_name = 'pickled.txt'            # text file to save the
pickled data in
pickled_input = open(file_name, 'wb')    #
creates/opens file in write mode (erases content)
pickled_input.close()

# -- Processing -- #
# function for pickling an object and saving it to the file
def pickle_to_file(obj_to_pickle):
   with open(file_name, 'ab') as pickled_input:
      pickle.dump(obj_to_pickle,pickled_input,
pickle.HIGHEST_PROTOCOL)
   message = 'Object was pickled'
   return message

# function for unpickling all saved objects from the file
into a list
def unpickle_from_file(file):
   objects = []     # defines the list to save the objects in
   with (open(file, "rb")) as pickled_output:
      while True:
         try:
            objects.append(pickle.load(pickled_output))
         except EOFError:   #  exception handling error
raised in scenarios such as interruption of the input () function
            break
   return objects

# -- Presentation (I/O) -- #
# the presentation demonstrates the steps in running
the script and informs the user to check the pickled.txt
# file for the pickled version of the data
print('The number of daily steps documented in the
Health App in my iPhone in the first ten days of
August are:')
print(list_dates)
print(list_steps_daily)
print('\nProgram will pickle these lists.')
input('Click ENTER to continue...')
pickle_to_file(list_dates)
message = pickle_to_file(list_steps_daily)
print(message)

print('Program will now unpickle these lists.')
input('Click ENTER to continue...\n')
object_list = unpickle_from_file(file_name)
print('The unpickled lists are:')
for list in object_list:
   print(list)


print('\nYou can check the pickled version in the
```

```
The unpickled lists are:
['080121', '080221', '080321', '080421', '080521',
'080621', '080721', '080821', '080921', '081021']
[1000, 1500, 2000, 2500, 3000, 3000, 2500, 2000,
1500, 1000]

You can check the pickled version in the pickled.txt file
Click ENTER to exit!
```

pickled.txt file') input('Click ENTER to exit!')	

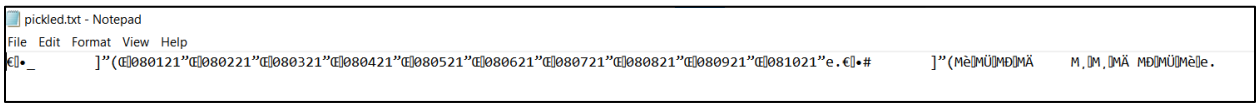

Figure 4. Screen capture of the data written to the pickled.txt file

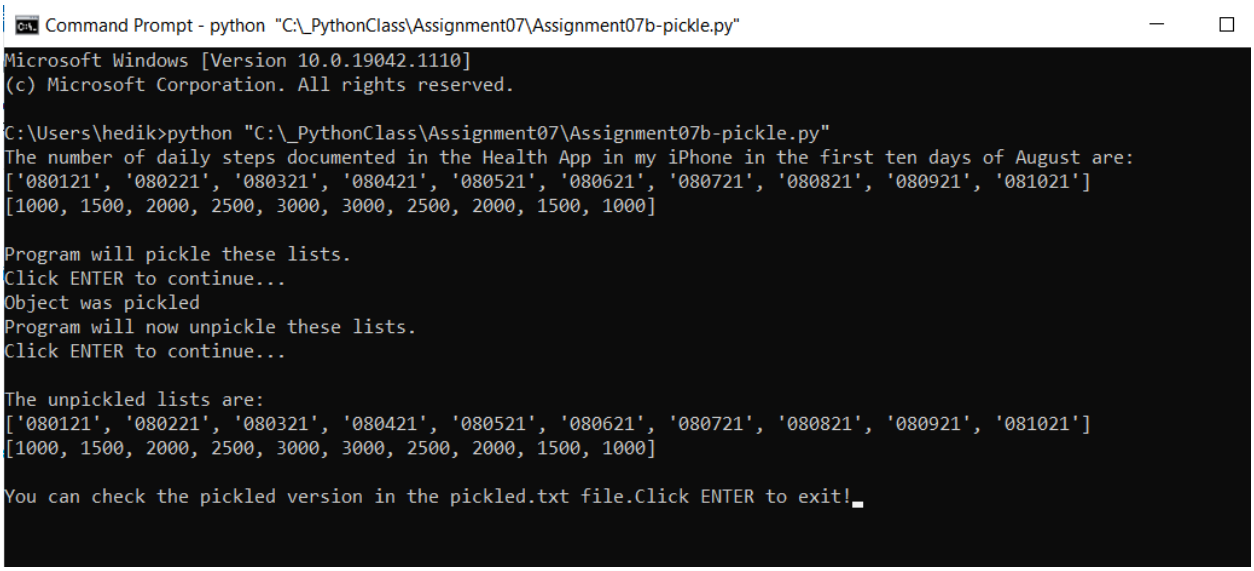

Figure 5. Screen capture of the script running in the Command console

Step 6 - Document your knowledge

I documented my learning while performing the steps in the assignment for Module 7 in this word document.

Step 7 - Post your Files to GitHub

Created a GitHub repository called "IntroToProg-Python-Mod07" under my account, uploaded my files and committed the changes. The URL is https://github.com/Seattle15/IntroToProg-Python-Mod07

Step 8 - Add a GitHub Webpage

As my assignment Word document is very lengthy, I did not add it as a GitHub page. To illustrate my understanding of this portion of the assignment I am submitting the webpage I made to practice the markdown language (Figures 2 and 3); I made further edits to the webpage and saved it in the repo for this assignment. I selected the Cayman theme for the webpage.
The URL is https://seattle15.github.io/IntroToProg-Python-Mod07/

I verified that the webpage was working as expected (Figure 4)-troubleshooting is further explained in the appropriate section in Step 1. I think it turned out quite nice! Of note, I dragged and dropped the image (a folder on my desktop) into the edit window and it showed up seamlessly on the webpage.

For further practice I added the first 6 pages of this assignment document to the GitHub webpage and verified that it was publishing correctly (Figure 5).

https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax (external site) I read the webpage suggested by Mr.Root -in parallel with watching the video - and have summarized some of the contents in the corresponding section of Step 1.

Figure 4. Screen capture of the top portion of the GitHub webpage created for the assignment. I experimented with multiple features on this page. The URL is https://seattle15.github.io/IntroToProg-Python-Mod07/

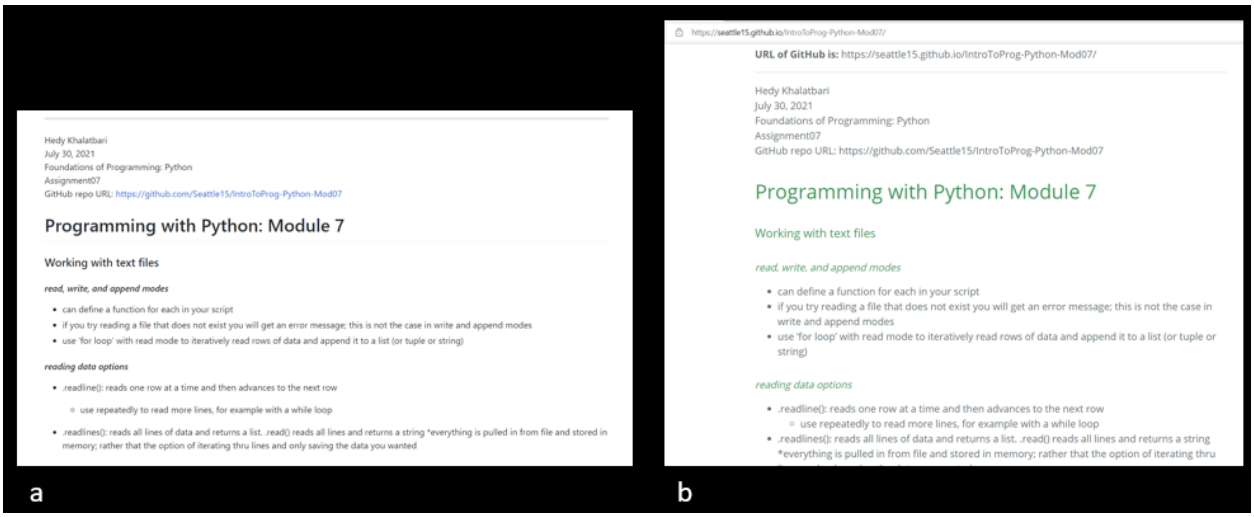

Figure 5. Screen capture of portions of the preview page (a) and the GitHub webpage (b) containing the first 6 page of this word document – this was added to the webpage shown in Figure 4. The URL is https://seattle15.github.io/IntroToProg-Python-Mod07/

Step 9 - Post a Link to GitHub

Will post a link using the Canvas discussion board in the week of Module 7.

Step 10 - Submit your work

Uploaded assignment word document and script and posted a link to my GitHub site on Canvas in the assignment textbox.

Step 11 - Perform a Peer Review (Not Graded!)

Will perform this step the week of Module 7.

Summary

In this module we learned more about working with files and GitHub. Several new concepts were also covered including exception handling and pickling. I enjoyed trying out the markdown language on GitHub – the interface was user friendly, and the student got immediate feedback on their attempts at formatting.

I have routinely been reading other resources to achieve a deeper understanding of the material in each module. In this module further reading and research on two subjects (exception handling and pickling) was part of the assignment. I was able to use Stack Overflow to find the solution to pickle dumping (with

ab) and pickle loading (with a while loop) to successfully addend and read more than one object – it was time consuming, and I took a week long break before coming back to it!