|  |  |
| --- | --- |
| Name | Radoslaw Hankiewicz |
| Date | 11/28/2021 |
| Course | Foundation of Programming |
| Title | Assignment 7 |

# **Introduction**

The purpose of this document is to present what I learned in the seventh module of the course.

The seventh module of the course presents some new information about working with text files: read, append modes, reading data options, the readline() function and using it with a while loop, the readlines() function and using a for loop, the with…as option. We also talked about working with binary files, structured error handling and using the exception class, ways to catch specific exceptions and raising custom errors, creating custom exception classes and custom GitHub readme files.

**Assignment\_07**

**GitHub link**: <https://github.com/Radohan/Assignment_07>

In the Assignment\_07 we were asked to follow the instructions and modify our previous script and add structured error handling around the areas where there is user interaction, type casting (string to int) or file access operations.

A screenshot of a computer

Description automatically generated

Figure 1 – Assignment\_07 in Spyder

Text

Description automatically generated

Figure 2 – Assignment\_07 in Anaconda

**Working with text files**

We have already talked about a couple of different ways of using text files:

* **write mode**

fileObj = open(file\_name, ‘w’)

fileObj.write(data)

fileObj.close()

* **read mode**

fileObj = open(file\_name, ‘r’)

data = fileObj.read()

fileObj.close()

* **append mode**

fileObj = open(file\_name, ‘a’)

fileObj.write(data)

fileObj.close()

# **Reading data options**

* **readline() function** 🡪 returns a string

On each call of thisfunction, the next line (one line) of data will be returned.

fileObj = open(file\_name, ‘r’)

data = fileObj.readline()

fileObj.close()

It can be used with a loop to get multiple lines of data.

* **readlines() function** 🡪 returns a list

This function reads all the lines in a file.

fileObj = open(file\_name, ‘r’)

data = fileObj.readlines()

fileObj.close()

* **Using a for loop**

The for loop can be used to iterate over the data in the file.

for line in open(file\_name, ‘r’):

data.append(line)

# **Using the with…as option**

The with statement in Python is used for resource management and exception handling. It simplifies the write process to just two lines.

with open(file\_name, ‘w’) as fileObj:

fileObj.write(data)

# **Working with binary files**

When it is not necessary to change the data in a readily readable format, the information can be saved as it is in memory – this is the binary code. It can be a good solution for saving states of the application or settings the user made.

To save binary information the pickle module should be used to take the information associated with an object and serialize it (or de-serialize it) it in a way that it can easily be stored as binary information.

* **write**

with open (file\_name, ‘wb’) as fileObj:

pickle.dump(data, fileObj)

* **read**

with open (file\_name, ‘rb’) as fileObj:

data = pickle.load(fileObj)

* **append**

with open (file\_name, ‘ab’) as fileObj:

pickle.dump(data, fileObj)

**Structured error handling**

The structured error handling helps the program to handle crashes and protects its parts whenever there is a foreseeable possibility that something could go wrong.

It uses a **try-except** block to handles this type of situations.

# **Using the Exception class**

In Python, we use a class **Exception** to hold information about an error.

Every error that occurs creates an object of this Exception class.

This object will be filled with information about the error that caused this exception.

# **Catching specific Exceptions**

The Exceptions class can catch any type of error.

Python offers multiple **built-in Exceptions** that can be used to catch specific Exceptions.

**Raising custom errors**

In Python, it is possible to **raise** errors based on custom conditions.

**Creating custom exception classes**

In Python, it's possible to define custom exceptions by creating a new class.

This exception class must be derived from the built-in Exception class.

**Creating custom GitHub readme files**

The GitHub readme.md file can be customized.

Styling in the readme file is simplified by using markdown code.

# **Summary**

The module no. 7 presented some new information about working with text files: read, append modes, reading data options, the readline() function and using it with a while loop, the readlines() function and using a for loop, the with…as option. We also talked about working with binary files, structured error handling and using the exception class, ways to catch specific exceptions and raising custom errors, creating custom exception classes and custom GitHub readme files.