Python 10: (Plain) Data Files

Introduction: This document presents learning steps for Python 10. In Python 10, you will get introduced with plain data files in Python.

Note: In this phase, it is expected the learner can divide the program into smaller learning steps. The goal and direction of the topics will be provided. The learning must take smaller steps towards the goals such that can implement solutions to the given problems and product(s).

Materials:

The activities are designed based on these following references:

- **BRef-01**: Book, Bill Lubanovic; "Introducing Python: Modern Computing in Simple Packages"; Available here
- **ORef-01**: Online Tutorial; Charles Severance; "Python for Everybody"; Available here

Path:

Step: Files and Content.

Goals:

```
After taking this step, you will be able to:
        1. interpret and implement Python programs using plain data files:
encoding / decoding, ascii and utf-8 encodings, binary vs text files, reading /
modifying / wrting binary / text data files, searching content using regular
expressions.
```

What to Learn?

1. Using **BRef-01: Chapter 12**, **BRef-01: Chapter 14**, **ORef-01: Chapter 7** and **ORef-01: Chapter 11** answer and experiment the following questions:
    1. What is encoding and decoding?
       Encoding is transmitting a sequence of characters into a specific format or efficient transmission or storage.
       Decoding is the opposite of it, it converts the decoded sequence into the original sequence
    2. How can we encode a string to bytes in Python? How can we decode some bytes to a string?
       In python, the string object has the method encode(), and the object bytes has the method decode()
    3. What are text data and binary data? Experiment with some examples.
       Text data looks like this, binary data looks like this: 01010101010101010
    4. Given an existing text file, how can we open and read the content in Python? What about a binary file?
       With open(file, “r”, “b”), “b” opens it in binary.

5. Given an existing text file, how can we open and add some new content? What about a binary file?
   With open(file, “W”, “b”) W is for write. “b” opens it in binary
6. Given an existing text file, how can we open and add search for an exact value? What about a binary file? We can open the file with read, and with a for loop loop through each line and look for the exact value.
7. Given an existing text file, how can we open and add search for a pattern using regular expression? Same as before, but the search value is a variable with the regex.

Exercises:

1. Design at least ten different exercises of your own. They should improve understanding topics of this step. Share your exercises with your learning community and practice.

Learning Activities:

Code Analysis

There is a code provided below by a programmer. Looking to the initializer of the class, it seems the code starts by reading data from a file. Unfortunately, we don't have access to the file, but we can analyze the code and understand how the content of the file is structured.

- Read the body of `load_data(self)`. Try to guess how the data columns are structured.
  I don’t understand the code written.
- Reading the method `load_data(self)` we may understand the structure of the data columns and data types, but still we don't know the meaning of data values. This is something we can extract from the method `construct_temprature_list(self)`. Read this method carefully and try to explain what will be result of this method?
- Manually write some lines within a text file and try to run the code using your own data values. Does it print what you expect?

```
class TemperatureDataAnalyzer:
    def __init__(self, file_path):
        self.file_path = file_path
        self.temperature_data = []
    # Method to open the file and load lines as an attribute
    def load_data(self):
        with open(self.file_path, 'r') as file:
            data = [line.strip().split() for line in file]
            self.temperature_data = [list(map(int,d[:-1]))+[float(d[len(d)-1])]
for d in data]
    # Method to perform the analysis and construct the list
    def construct_temperature_list(self):
        temperature_list = []
        for data in self.temperature_data:
            month, day, year, temperature = data[:]
            if year not in [item[0] for item in temperature_list]:
                temperature_list.append((year, {}))
            if month not in temperature_list[-1][1]:
                temperature_list[-1][1][month] = 0.0
            temperature_list[-1][1][month] = max(temperature ,
temperature_list[-1][1][month])
        return temperature_list
```

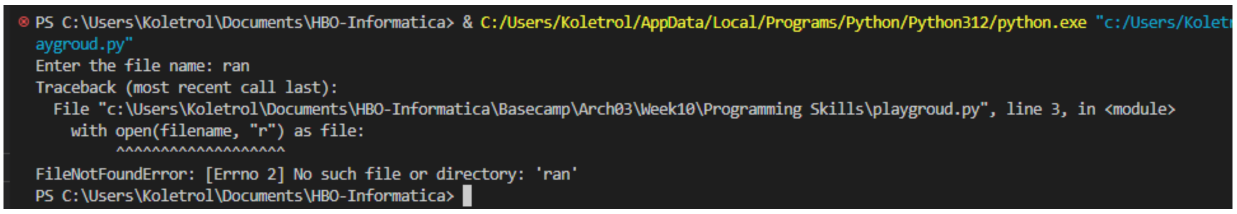
```
def main():
    file_path = './temps.txt'
    analyzer = TemperatureDataAnalyzer(file_path)
    analyzer.load_data()
    temperature_list = analyzer.construct_temperature_list()
    print(temperature_list)

if __name__ == '__main__':
    main()
```

Supporting Topics

Introduction: In Python, exceptions are unexpected events that can happen during the execution of a program, disrupting the normal flow of code. These can include errors like attempting to access a non-existent file. To handle these exceptions and prevent the program from crashing, Python provides a mechanism called the `try-except` block. The code that might raise an exception is placed inside the `try` block. If an exception occurs within the `try` block, Python immediately jumps to the corresponding `except` block, where the specific exception can be caught and handled gracefully. This allows developers to predict potential errors and define appropriate responses, ensuring the program can continue running smoothly even when unexpected issues arise. Using `try-except` blocks enhances the robustness of Python programs by providing a way to manage errors and prevent them from causing the entire program to terminate abruptly.

- Perform a research about exception handling in Python.
  Done.
- Implement a simple example with an intentional error, like trying to open a file with a wrong name. Run the program and see what will be the result of the execution.



```
PS C:\Users\Koletrol\Documents\HBO-Informatica> & C:/Users/Koletrol/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/Kolet
aygroud.py"
Enter the file name: ran
Traceback (most recent call last):
  File "c:\Users\Koletrol\Documents\HBO-Informatica\Basecamp\Arch03\Week10\Programming Skills\playgroud.py", line 3, in <module>
    with open(filename, "r") as file:
         ^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'ran'
PS C:\Users\Koletrol\Documents\HBO-Informatica>
```

```
filename = input("Enter the file name: ")
with open(filename, "r") as file:
    print("It worked!")
```

- Improve your program by adding `try-except` block:

```
filename = input("Enter the file name: ")
try:
    with open(filename, "r") as file:
        print("It worked!")
except FileNotFoundError:
    print("File not found")
```

- Raised exceptions are object instances. They expose methods to detect their types and messages. Use these object instances to print proper error messages to the user when an exception occurs.
- Improve the code given above for code analysis with exception handling.

```
class TemperatureDataAnalyzer:
    def __init__(self, file_path):
        self.file_path = file_path
        self.temperature_data = []
    # Method to open the file and load lines as an attribute
    def load_data(self):
        with open(self.file_path, "r") as file:
            data = [line.strip().split() for line in file]
            self.temperature_data = [list(map(int, d[:-1])) + [float(d[len(d) - 1])] for d in data]
    # Method to perform the analysis and construct the list
    def construct_temperature_list(self):
        temperature_list = []
        for data in self.temperature_data:
            month, day, year, temperature = data[:]
            if year not in [item[0] for item in temperature_list]:
                temperature_list.append((year, {}))
            if month not in temperature_list[-1][1]:
                temperature_list[-1][1][month] = 0.0
            temperature_list[-1][1][month] = max(temperature, temperature_list[-1][1][month])
        return temperature_list
def main():
    file_path = "./temps.txt"
    analyzer = TemperatureDataAnalyzer(file_path)
    analyzer.load_data()
    temperature_list = analyzer.construct_temperature_list()
    print(temperature_list)
if __name__ == "__main__":
    try:
        main()
    except FileNotFoundError:
        print("File not found")
```

This are the questions our team made to be answered:

Exception handling

1 – What is exception handling?

2 – Which exceptions have you encountered?

3 – What kinds of exceptions are there?

This are the questions we got from Team Zero:

1) Wat is exception handling

Exception handling is het proces van reageren op uitzonderlijke omstandigheden die speciale verwerking vereisen tijdens de uitvoering van een programma.

2) Wat is het nut van exception handling

Het nut van exception handling is om de normale uitvoering van een programma te onderbreken en fouten op een gecontroleerde manier af te handelen, zodat het programma niet abrupt stopt.

3) Wat zijn de verschillende type exceptions in python

Enkele verschillende types van exceptions in Python zijn:

MemoryError: Geheugenfout

IndexError: Index buiten bereik

KeyError: Sleutel niet gevonden

TypeError: Verkeerd datatype gebruikt

ValueError: Verkeerde waarde gebruikt

ZeroDivisionError: Delen door nul.

Sources:

BRef-01: Book, Bill Lubanovic; "Introducing Python: Modern Computing in Simple Packages";

ORef-01: Online Tutorial; Charles Severance; "Python for Everybody";

https://docs.python.org/3/library/exceptions.html