**National University of Computer and Emerging Sciences**



**Lab Manual 05 CL461-Artificial Intelligence Lab**

|  |  |
| --- | --- |
| Course Instructor | Dr Mubashir baig |
| Lab Instructor (s) | Nimra  Saima Akhtar |
| Section | H |
| Semester | Spring 2022 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

**Objectives**

After performing this lab, students shall be able to understand Python data structures which include:

* Python file handling
* Python iterators

* **Task Distribution**

|  |  |
| --- | --- |
| **Total Time** | **170 Minutes** |
| Python File Handling | 20 Minutes |
| Python Iterators | 10 Minutes |
| Exercise | 90 Minutes |

* **Python File Handling**

Python allows users to handle files by supporting to read and write files, along with many other file handling options. More details can be learnt [HYPERLINK "https://towardsdatascience.com/knowing-these-you-can-cover-99-of-file-operations-in-python-84725d82c2df"here HYPERLINK "https://towardsdatascience.com/knowing-these-you-can-cover-99-of-file-operations-in-python-84725d82c2df"](https://towardsdatascience.com/knowing-these-you-can-cover-99-of-file-operations-in-python-84725d82c2df)

* **Open & Close a file**

When you want to read or write a file, the first thing to do is to open the file. Python has a builtin function **open** that opens the file and returns a file object. To return a file object we use open() function along with two arguments, that accepts file name and the mode, whether to read or write.

The syntax is given below:

**open(filename, mode)**

* **Kinds of modes**

There are three basic types of modes in which files can be opened in Python.

|  |  |
| --- | --- |
| **mode** | **meaning** |
| r | open for reading (default) |
| r+ | open for both reading and writing (file pointer is at the beginning of the file) |
| w | open for writing (truncate the file if it exists) |
| w+ | open for both reading and writing (truncate the file if it exists) |
| a | open for writing (append to the end of the file if exists & file pointer is at the end of the file) |

Always keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “ **r** ” by default.

Let’s look at this program and try to analyze how the read mode works:

|  |
| --- |
| # a file named "book", will be opened with the reading mode.  file = open('book.txt', 'r')  # This will print every line one by one in the file for each in file: |

print (each)

* **Working of read() mode**

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use **file.read()**. The full code would work like this:

# Python code to illustrate read() mode file = open("file.text", "r") print (file.read())

Another way to read a file is to call a certain number of characters like in the following code the interpreter will read the first five characters of stored data and return it as a string:

# Python code to illustrate read() mode character wise file = open("file.txt", "r") print (file.read(5))

* **Working of write() mode**

Let’s see how to create a file and how write mode works:

To manipulate the file, write the following in your Python environment:

|  |
| --- |
| # Python code to create a file file = open('book.txt','w')  file.write("This is the write command")  file.write("It allows us to write in a particular file") file.close() |

The close() command terminates all the resources in use and frees the system of this particular program.

* **Working of append() mode**

# Python code to illustrate append() mode file = open('book.txt','a') file.write("This will add this line") file.close()

* **Python Iterators**

An iterator is an object that contains a countable number of values. It is an object that can be iterated upon, meaning that you can traverse through all the values. Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods iter() and next().

Every time you ask an iterator for the **next** item, it calls its \_\_next\_\_method. If there is another value available, the iterator returns it. If not, it raises a StopIteration exception. More information about iterators can be found here.

This behavior (only returning the next element when asked to) has two main advantages:

* Iterators need less space in memory. They remember the last value and a rule to get to the next value instead of memorizing every single element of a (potentially very long) sequence.
* Iterators don’t check how long the sequence they produce might get. For instance, they don’t need to know how many lines a file has or how many files are in a folder to iterate through them.

(One important note: don’t confuse iterators with iterables. Iterables are objects that can create iterators by using their \_\_iter\_\_ method)

* **Building Custom Iterators**

|  |  |  |
| --- | --- | --- |
| \_\_iter\_\_() | and the | \_\_next\_\_() |

Building an iterator from scratch is easy in Python. We just have to implement the methods.

|  |
| --- |
| \_\_iter\_\_() |

The method returns the iterator object itself. If required, some initialization can be performed.

|  |
| --- |
| \_\_next\_\_() |

The method must return the next item in the sequence.

Return an iterator from a tuple, and print each value:

|  |
| --- |
| mytuple = ("apple", "banana", "cherry") myit = iter(mytuple)  print(next(myit)) print(next(myit)) print(next(myit)) |

To iterate the characters of a string:

mystr = "banana"

for x in mystr:

print(x)

The for loop actually creates an iterator object and executes the next() method for each loop.

* **Exercise (50 marks)**

## Even/Odd using Lambda function

Write a Python program to filter a list of integers using Lambda.  
Original list of integers:  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Even numbers from the said list:  
[2, 4, 6, 8, 10]  
Odd numbers from the said list:  
[1, 3, 5, 7, 9]

## Reading text from a file and storing it in reversed order

Design a code which reads text from the file “Alphabets.txt” and stores its data in reverse order in another file. For this you may upload the given text file on Google Collab’s session and define the path as:

file\_path= ‘/Alphabets.txt’

The same convention can be followed for defining path of the resultant file (reversed text file).

**1.3 Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):**

## 1.4 Create a class for rectangle shape that calculates its area based upon the length and width. Make a sub class of triangle called Trapezium, such that it inherits the functionality of rectangle class and implements area method of its own. Length and width should be defined in the constructor of rectangle class.

Area

Area of Rectangle =Length+Width

Area of Trapezium=½\*(l+w)\*h

After creation of the class, define the relevant attributes. Define a function for area computation and then a function for displaying area. Incorporate your knowledge of class and objects here.

**1.5 Write a function AMCount() in Python, which should read each character of a text file STORY.TXT, should count and display the occurrence of alphabets A and M (including small cases a and m too).**

**For Example:**

**If the file content is as follows:**

**“Updated information**

**As simplified by official websites.”**

**The EUCount() function should display the output as:**

**A or a:4**

**M or m :2**