

Review – Collection of Data

- List: []
 - Ordered elements, mutable elements, mutable container
- Set: {} set()
 - Unordered elements, immutable elements, mutable container
- Tuple: ()
 - Ordered elements, mutable elements, immutable container
- Dictionary: {}
 - Unordered elements, immutable keys and mutable values, mutable container

File IO – Basics

Lecture 7

Hyung-Sin Kim



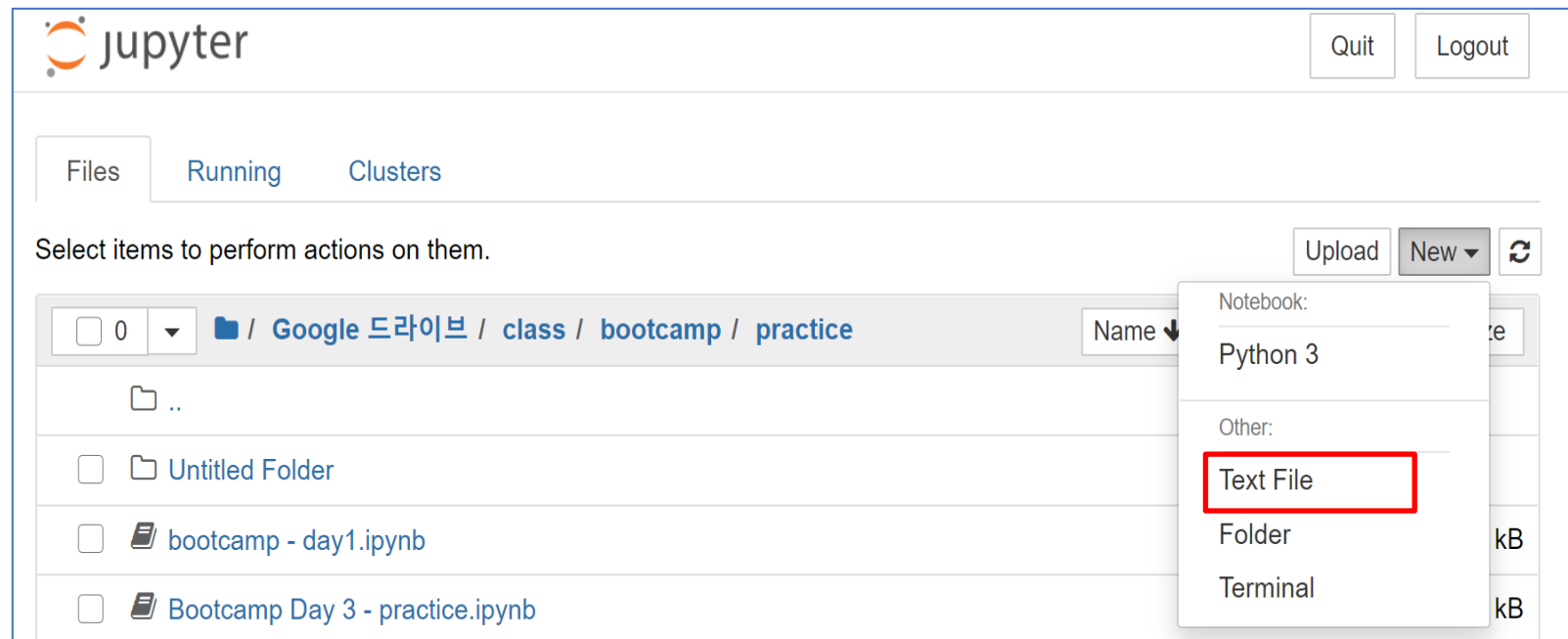
SNU Graduate School of Data Science

Introduction

- So far, we have made our own data by using various structures, string, list, tuple, set, and dictionary
- In the real world, there are lots and lots of data in files
- Once you can read and write data from/to files, you can work on real data!
- Among many kinds of files, text, music, video, ppt, word, etc., we will focus on **text files**

Opening a File – Do it!

- Make a directory, perhaps called “practice”
- In jupyter, select New->Text File and type the following:
 - First line of text
 - Second line of text
 - Third line of text



- Save this file in your current directory under the name **file_example.txt**

Opening a File – Do it!

- Open a python cell and type this program:
 - `file = open('file_example.txt', 'r')`
 - `contents = file.read()`
 - `file.close()`
 - `print(contents)`
- Do you see something? 😊

Opening a File

- `file = open(<<file_name>>, mode)`
 - Open a file and return an object that knows how to get information from the file
 - This object also keeps track of **current location**: how much you've read and which part of the file you are about to read next
 - **File cursor**: The marker that keeps track of the current location in the file (initially at the beginning of the file and moving to the end as you read or write data)
 - Mode: **'r'** is for reading, **'w'** is for writing, and **'a'** for appending
- `file.read()`
 - Read the entire file, make the contents into a string, and return the string
- `file.close()`
 - Release all resources associated with the open file object

Opening a File – With Statement

- A common programming pattern
 - (1) Get access to a resource, (2) do something with the resource, and (3) tidy up and release the resource
 - `open()` – access, `read()` – do something, `close()` – release
 - But it is possible for `close()` to **not be executed** due to some errors
- A safer way is to implement same function by using ‘**with**’ statement
 - with `open('file_example.txt', 'r')` as file:
 - `contents = file.read()`
 - `print(contents)`
 - Resource is released when the program leaves the ‘with’ statement

File Path

- **Absolute path:** If file is in the following absolute path
 - `/User/hskim/Desktop/practice/file_example.txt`
- **Relative path:** Starting from the current working directory
 - If you are at `/User/hskim/Desktop`, just type “`practice/file_example.txt`”
 - If you are at `/User/hskim/Desktop/temp`, type “`../practice/file_example.txt`”
 - `../` lets you go up!
- Some functions about directory in module “`os`”
 - `>>> import os`
 - `>>> os.getcwd()` `# show the current working directory`
 - `>>> os.chdir("/User/hskim/Desktop")`

Reading Contents in Files

- When you want to read **characters** and put them into a single string
 - with `open("file_example.txt", 'r')` as file:
 - `contents = file.read()`
 - Read from the current cursor location to the end of file, the file cursor moves to the end of file
 - `contents = file.read(10)`
 - Read 10 characters from the current cursor location, the file cursor moves to the 11-th character
 - If you want to read what you have already read again, close and open the file again
- When you want to make a list of strings containing **individual lines**
 - `lines = file.readlines()`
 - Output example: `['First line of text.\n', 'Second line of text.\n', 'Third line of text.\n']`
 - Output contains whitespace characters. To remove them, you need to use strip method: `lines.strip()`
 - Now that you have a list of strings, you can do `reversed(lines)` or `sorted(lines)`

Reading Contents in Files

- When you want to do something for a **single** line
 - with `open("file_example.txt", 'r')` as file:
 - `line = file.readline()`
 - `<<do something with the single line>>`
- When you want to **repeat** something for every line
 - with `open("file_example.txt", 'r')` as file:
 - `for line in file:`
 - `<<do something with the single line>>`

Reading Contents in Files

- When you work on a file over Internet!
 - `import urllib.request`
 - `url = “https://robjhyndman.com/tsdldata/ecology1/hopedale.dat”`
 - `with urllib.request.urlopen(url) as file:`
 - `for line in file:`
 - `line = line.strip()`
 - `line = line.decode(“utf-8”)`
 - `print(line)`
- Since there are so many data types in a file in the Internet, you don’t know if the file has text or music...
 - Python’s `read` or `readline` methods return a **bytes** type, instead of string
 - You need to decode bytes type to use it as string

Reading Contents in Files

- Bit
 - To a computer, information is nothing but bits (0 or 1)
 - All data (characters, sounds, and pixels) are represented as sequences of bits
- Byte
 - A larger unit of data - 8 bits
 - Programming languages interpret bytes for users to think them as integers, strings, functions, and documents...
- Since Byte is the fundamental data unit, regardless of data types, Python reads information on a webpage as bytes

Writing Files

- with `open('file_example.txt', 'w')` as `output_file`:
- `output_file.write('Programming for Data Science')`
 - Write characters to a file and returns the number of characters written
 - In the write mode ('w'), write from the beginning of the file
- with `open('file_example.txt', 'a')` as `output_file`:
- `output_file.write(' is fun...?')`
 - In the append mode ('a'), write from the end of the file
- You need to manually add “\n” to change the line

Reading and Writing Together

- with `open('file_input.txt', 'r')` as `input_file`, `open('file_output.txt', 'w')` as `output_file`:
- `<<read input_file, do something, and write to output_file>>`

File IO – Reading Techniques

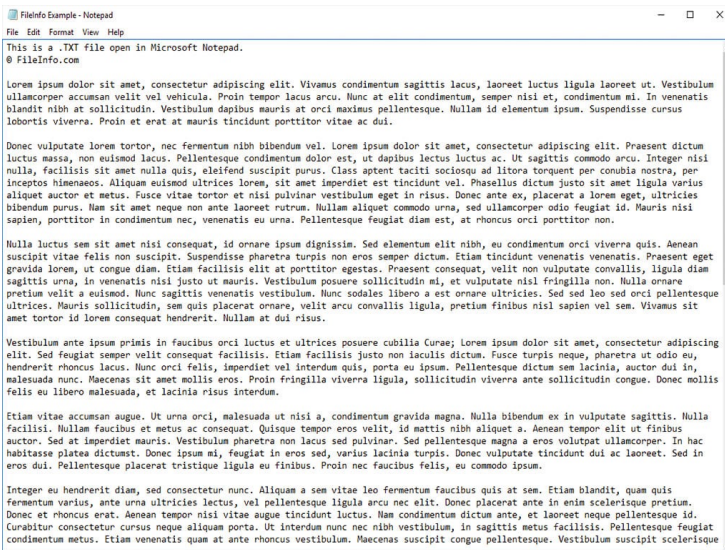
Lecture 7

Hyung-Sin Kim



SNU Graduate School of Data Science

Now let's extract data from a file and organize them as data structures we've learned!



File Info Example - Notepad

File Edit Format View Help

This is a .TXT file open in Microsoft Notepad.

© FileInfo.com

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus condimentum sagittis lacus, laoreet luctus ligula laoreet ut. Vestibulum ullamcorper accusan velit vel vehicula. Proin tempor lacus arcu. Nunc at elit condimentum, semper nisi et, condimentum mi. In venenatis blandit nibh at sollicitudin. Vestibulum dapibus mauris at orci malesuada pellentesque. Nullam id elementum ipsum. Suspendisse cursus lobortis viverra. Proin et erat at mauris tincidunt porttitor vitae ac dui.

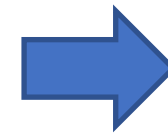
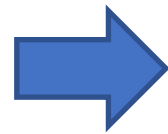
Donec vulputate lorem tortor, nec fermentum nibh bibendum vel. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent dictum luctus massa, non euismod lacus. Pellentesque condimentum dolor est, ut dapibus lectus luctus ac. Ut sagittis comodo arcu. Integer nisi nulla, facilisis sit amet nulla quis, eleifend suscipit purus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam euismod ultrices lorem, sit amet imperdiet est tincidunt vel. Phasellus dictum justo sit amet ligula varius aliquet auctor et metus. Fusce vitae tortor et nisi pulvinar vestibulum eget in risus. Donec ante ex, placerat a lorem eget, ultricies bibendum purus. Nam sit amet neque non ante laoreet rutrum. Nullam aliquet comodo urna, sed ullamcorper odio feugiat id. Mauris nisi sapien, porttitor in condimentum nec, venenatis eu urna. Pellentesque feugiat diam est, at rhoncus orci porttitor non.

Nulla luctus sem sit amet nisi consequat, id ornare ipsum dignissim. Sed elementum elit nibh, eu condimentum orci viverra quis. Aenean suscipit vitae felis non suscipit. Suspendisse pharetra turpis non eros semper dictum. Etiam tincidunt venenatis venenatis. Praesent eget gravida lorem, ut congue diam. Etiam facilisis elit at porttitor aegestas. Praesent consequat, velit non vulputate convallis, ligula diam sagittis urna, in venenatis nisi justo ut mauris. Vestibulum posuere sollicitudin mi, et vulputate nisl fringilla non. Nulla ornare pretium velit a euismod. Nunc sagittis venenatis vestibulum. Nunc sodales libero a est ornare ultricies. Sed sed leo sed orci pellentesque ultrices. Mauris sollicitudin, sem quis placerat ornare, velit arcu convallis ligula, pretium finibus nisl sapien vel sem. Vivamus sit amet tortor id lorem consequat hendrerit. Nullam at dui risus.

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed feugiat semper velit consequat facilisis. Etiam facilisis justo non laculis dictum. Fusce turpis neque, pharetra ut odio eu, hendrerit rhoncus lacus. Nunc orci felis, imperdiet vel interdum quis, porta eu ipsum. Pellentesque dictum sem lacinia, auctor dui in, malesuada nunc. Maecenas sit amet mollis eros. Proin fringilla viverra ligula, sollicitudin viverra ante sollicitudin congue. Donec mollis felis eu libero malesuada, sit lacinia risus interdum.

Etiam vitae accumsan augue. Ut urna orci, malesuada ut nisl a, condimentum gravida magna. Nulla bibendum ex in vulputate sagittis. Nulla facilisi. Nullam faucibus et metus ac consequat. Quisque tempor eros velit, id mattis nibh aliquet a. Aenean tempor elit ut finibus auctor. Sed at imperdiet mauris. Vestibulum pharetra non lacus sed pulvinar. Sed pellentesque magna a eros volutpat ullamcorper. In hac habitasse platea dictumst. Donec ipsum mi, feugiat in eros sed, varius lacinia turpis. Donec vulputate tincidunt dui ac laoreet. Sed in eros dui. Pellentesque placerat tristique ligula eu finibus. Proin nec faucibus felis, eu comodo ipsum.

Integer eu hendrerit diam, sed consectetur nunc. Aliquam a sem vitae leo fermentum faucibus quis at sem. Etiam blandit, quam quis fermentum varius, ante urna ultricies lectus, vel pellentesque ligula arcu nec elit. Donec placerat ante in enim scelerisque pretium. Donec et rhoncus erat. Aenean tempor nisi vitae augue tincidunt luctus. Nam condimentum dictum ante, et laoreet neque pellentesque id. Curabitur consectetur cursus neque aliquam porta. Ut interdum nunc nec nibh vestibulum, in sagittis metus facilisis. Pellentesque feugiat condimentum metus. Etiam venenatis quam at ante rhoncus vestibulum. Maecenas suscipit congue pellentesque. Vestibulum suscipit scelerisque



List
Set
Tuple
Dictionary

Reading Techniques – Whitespace-delimited

- A file example for student grading
 - Inhoe 2021-1111 A+
 - Jaewook 2021-2222 A+
 - ...
- A line can contain many different information, which are divided by **whitespace**

One line has **three different types** of information:
name, number, and grade

We want to handle each information separately!

Reading Techniques – Whitespace-delimited

- Whitespace handling

- with **open**('file_grade.txt', 'r') as students:
- process_file(students)

A separate function for processing a file

from typing import TextIO

- def **process_file**(input_file: TextIO) -> None:
- line = input_file.readline()
- while line:
- line = line.strip()
- **value = line.split()**
- names.append(value[0])
- numbers.append(value[1])
- grades.append(value[2])
- line = input_file.readline()

Remove "\n" at the end of the line

Break a line into a list of words
['inhoe', '2020-1111', 'A+']

Reorganize the words as three
different lists

One more step into the dirty real world

Reading Techniques – Skipping the Header

- A file example for student grading
 - # This file is for a GSDS course “computing foundations for data science.”
 - # This file is created by Hyung-Sin Kim
 - # This file contains students’ grades.
 - Inhoe 2021-1111 A+
 - Jaewook 2021-2222 A+
 - ...
- Many files include **header** to describe what they are, which is not data

We do not want to read the header!

Reading Techniques – Skipping the Header

- A solution for skipping the header
 - `def process_file(input_file: TextIO) -> None:`
 - `line = skip_header(input_file) # The first useful line after skipping header`
 - `while line:`
 - `line = line.strip()`
 - `value = line.split()`
 - `names.append(value[0])`
 - `numbers.append(value[1])`
 - `grades.append(value[2])`
 - `line = input_file.readline()`
 - `def skip_header(input_file: TextIO) -> str:`
 - `line = input_file.readline()`
 - `while line.startswith('#'):`
 - `line = input_file.readline()`
 - `return line`

Reading Techniques – Skipping the Header

- A solution that does not work
 - `def process_file(input_file: TextIO) -> None:`
 - `skip_header(input_file) # Skip header`
 - `<<Do something useful with real data>>`

- `def skip_header(input_file: TextIO) -> None:`
- `while line.startswith('#'):`
- `input_file.readline()`

- When `skip_header` ends...

Cursor is at start of
the second useful line

```
# This file is for a GSDS course "computing foundations for data science."  
# This file is created by Hyung-Sin Kim  
# This file contains students' grades.  
Inhoe 2021-1111 A+  
Jaewook 2021-2222 A+
```

One more step into the dirty real world

Reading Techniques – Handling Missing Values

- A file example for student grading
 - # This file is for a GSDS course “computing foundations for data science.”
 - # This file is created by Hyung-Sin Kim
 - # This file contains students’ grades.
 - Inhoe 2021-1111 A+
 - -
 - Jaewook 2021-2222 A+
 - ...
- Data in your file are not guaranteed to be ideal. There can be many **missing values** and **typos**

We do not want to process the weird values!

Reading Techniques – Handling Missing Values

- Ignore missing values
 - `def process_file(input_file: TextIO) -> None:`
 - `line = skip_header(input_file) # The first useful line after skipping header`
 - `while line:`
 - `line = line.strip()`
 - `if line != '-':`
 - `value = line.split()`
 - `names.append(value[0])`
 - `numbers.append(value[1])`
 - `grades.append(value[2])`
 - `line = input_file.readline()`

Summary

- Files can be read, written to, and added to
- Contents are commonly stored in list of **strings**
- Three common stages for reusability: Input, processing, and output stages
- To make the functions usable by different types of readers, the reader is opened outside the function and passed as an argument
- TextIO is used in type annotations to indicate an open file

Thanks!