Managing Data I

MSBA7001 Business Intelligence and Analytics
HKU Business School
The University of Hong Kong

Instructor: Dr. DING Chao

About this course

Managing Data	Web Scraping	Data Visualization
txt, csv, json	Requests	Tableau
Regular Expressions	Beautiful Soup	Matplotlib
NumPy	Selenium	
Pandas	Using APIs	

Agenda

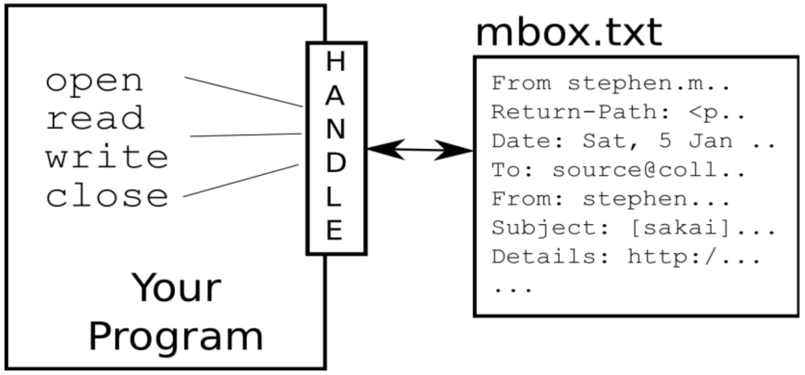
- Opening & closing a file
- Reading from and writing to files
  - ➢txt
  - ➢csv
  - ➢json
- Regular Expressions (Regex)

Opening & Closing a File

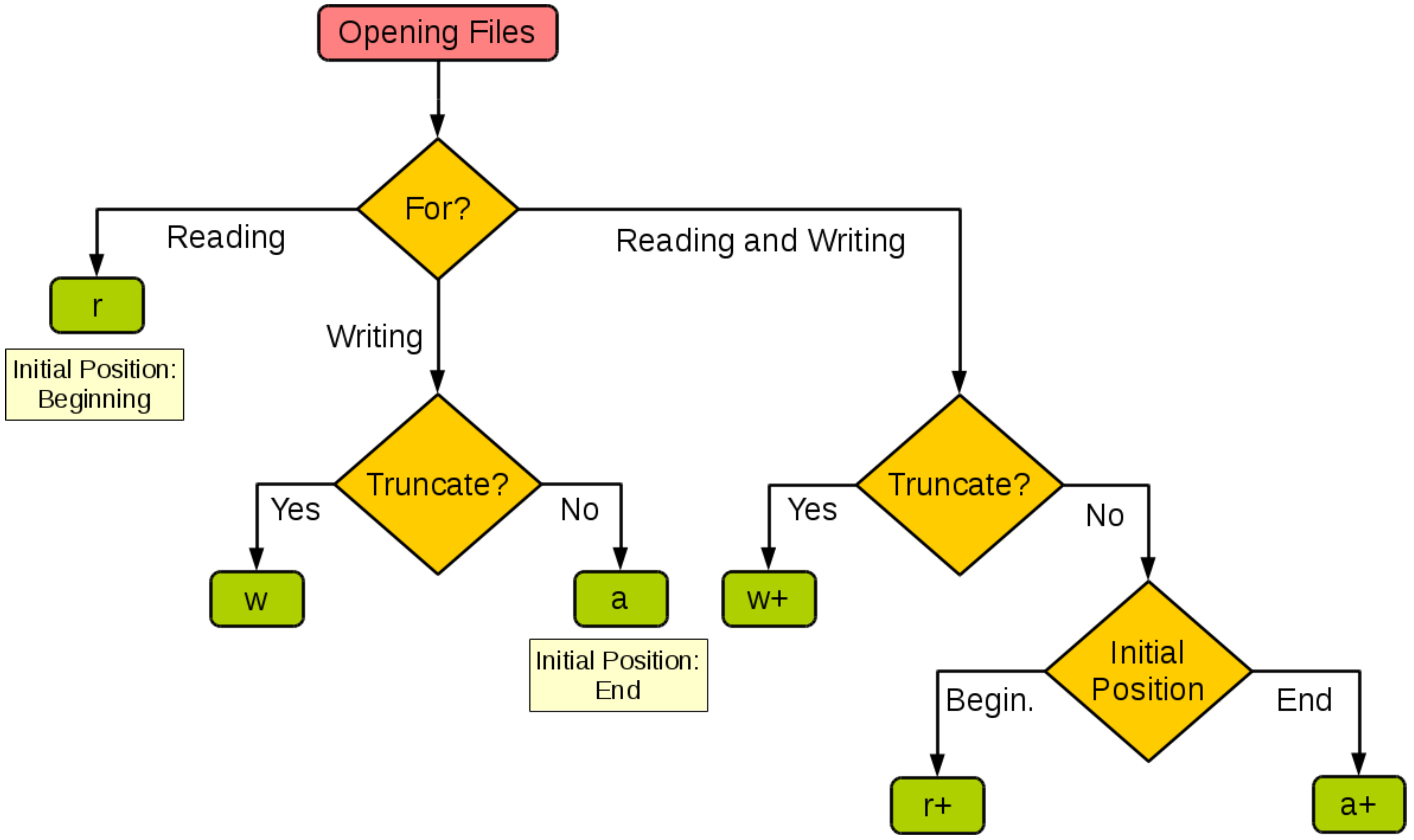
Opening a File

- Before we can read the contents of the file (txt or csv or any Python supported files), we must tell Python which file we are going to work with.
- `open` returns a "file handle" - a variable used to perform operations on the file.

```
open(filepath/filename, mode)
```



About Mode


Opening Files
For?
Reading
Writing
Reading and Writing
r
Initial Position: Beginning
Truncate?
Yes
No
w
a
Initial Position: End
Truncate?
Yes
No
w+
Initial Position
Begin.
End
r+
a+

About Mode

- There is also a binary mode **b**, which forces the data to be binary.

	r (default)	r+	w	w+	a	a+
Read	✓	✓	✗	✓	✗	✓
Write	✗	✓	✓	✓	✓	✓
Creates file	✗	✗	✓	✓	✓	✓
Erases file	✗	✗	✓	✓	✗	✗
Initial position	Start	Start	Start	Start	End	End

Filenames and Paths

- If the file is in the same directory with your code, simply type the file name.

```
handle = open('text_file.txt', 'r')
```

- If it’s in the parallel directory, then use relative path.

```
handle = open('../data/text_file.txt', 'r')
```

- Or, you can also use the absolute path, by using the full path of the file such as “/home/data/mbox.txt”.
- Use `try` and `except` to deal with potential bad filenames or paths.

Attributes of the File Object

- File objects have multiple attributes.
- Append attribute to the file handle, without parentheses.

Attribute	Description
name	Returns name of the file.
encoding	Returns the encoding this file uses, such as UTF-8.
mode	Returns access mode with which file was opened.
closed	Returns true if file is closed, false otherwise.

Closing a File

- After working with the file, we should always close it with the **close** method.
- Or, a shortcut by using the **with** statement. It will take care of closing the file.

```
>>> handle = open('text_file.txt','r')
>>> do something with the file
>>> handle.close()
```

Equivalent

```
with open('text_file.txt','r') as handle:
    do something with the file
```

File Methods (Partial)

Method	Description
close()	Closes the file
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

TXT (Text)

Text File Structure

- A file handle opens for read can be treated as a sequence of strings where each line in the file is a string in the sequence.
- There is a newline at the end of each line.

```
Two roads diverged in a yellow wood,\n
And sorry I could not travel both\n
And be one traveler, long I stood\n
And looked down one as far as I could\n
To where it bent in the undergrowth; \n
```

This is an excerpt from text_file.txt

- We can use the `for` loop to traverse the sequence.

```
handle = open('text_file.txt', 'r')
for line in handle:
    print(line)
```

Methods to Read and Write

Method	Description
read()	Returns the file content as one string
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

CSV

CSV File Structure

- CSV: **C**omma **S**eparated **V**alues
- It is a readable text file.
- Every row is considered as a list or a tuple.
- So, the file content is a list of lists (tuples).
- The first row is usually a header row.

Opened in text editor

```
no, Name, City
1, Michael, New Jersey
2, Jack, California
3, Donald, Texas
```

Opened in Excel

SN	Name	City
1	Michael	New Jersey
2	Jack	California
3	Donald	Texas

Methods to Read and Write

- We need to use the `csv` module.
- First create a reader/writer object and then read/write via the reader/writer object.

```
import csv

reader = csv.reader(handle)
writer = csv.writer(handle)
```

These are methods of the writer object

Method	Description
writerow()	Writes the specified list (tuple) to the file
writerows()	Writes a list of lists (tuples) to the file

JSON

JSON File Structure

- JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON is a syntax for storing and exchanging data.
- It is still considered text, also called JSON strings.
- It is easy for machines to parse and generate.

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

JSON Syntax Rules

- Data is in key/value pairs, like a dictionary in Python.
- keys must be strings, written with double quote.
- values must be one of the following data types:

➢ a string	`{ "name":"John" }`
➢ a number	`{ "age":30 }`
➢ an object	another JSON object
➢ an array	`{"employees":[ "John", "Anna", "Peter"]}`
➢ a boolean	`{ "sale":true }`
➢ null	`{ "middlename":null }`

JSON Strings vs. Python Objects

- There is a one-on-one conversion between JSON strings and Python objects.

JSON String	Python Object
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

Methods to Read and Write

- We need to use the `json` module.

```
import json
```

Method	Description
load()	Returns the file content as a Python dictionary.
loads()	Converts a JSON string into a Python object.
dump()	Writes a Python dictionary to the file.
dumps()	Converts a Python dictionary into a JSON string.

Summary

- Opening and closing a file
- Reading from and writing to a txt file
- Reading from and writing to a csv file
- Reading from and writing to a json file

Regular Expressions (Regex)

Regular Expressions

- Our data file may include millions of lines, we want to extract a specific section of data, e.g., the date and time the email was sent, or the email addresses.
- Regular Expressions (also called RegEx) provide a great way to match and parse text patterns.
- RegEx can be quite mysterious at first.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

The RegEx Module

- There are a number of common methods for regular expression objects.

```
import re
```

Method	Description
findall()	Returns a list containing all matches
search()	Returns a match object if there is a match anywhere in the string, None on failure
split()	Returns a list where the string has been split at each match
sub()	Replaces one or many matches with a string
compile()	Returns a RegEx pattern

Searching Characters in a String

- **search** returns a match object if there is a match anywhere in the string, or None on failure.
- This is similar to the string method **find**.

```
>>> text = 'HKU Business School'
>>> if re.search('HKU', text): print('yes')
yes
```

Equivalent

```
  if text.find('HKU') >= 0: print('yes')
```

Match Object

- A Match Object is an object containing information about the search and the result.
- It has properties and methods used to retrieve information about the search, and the result.

Method	Description
span()	Returns a tuple containing the start and end positions of the match
start()	Returns the start position of the match
end()	Returns the end position of the match
string	Returns the string passed into the method
group()	Returns the part of the string where there was a match
groups()	Returns a tuple containing all the subgroups of the match

Compiling a RegEx Object

- We can use the `compile` method to create a RegEx object, which can be used with RegEx methods.

```
text = 'HKU Business School'
pattern = re.compile('HKU')
if pattern.search(text): print('yes')
```

$\Updownarrow$ Equivalent

```
text = 'HKU Business School'
if re.search('HKU', text): print('yes')
```

Matching and Parsing Text

- Use **findall** method to match a pattern and return a list of all matched substrings.
- If there is no match, then return an empty list.

```
>>> text = 'HKU Business School'
>>> result = re.findall('s', text)
>>> print(result)
['s', 's', 's']
```

Equivalent

```
>>> text = 'HKU Business School'
>>> pattern = re.compile('s')
>>> result = pattern.findall(text)
>>> print(result)
['s', 's', 's']
```

Metacharacters

- Metacharacters are characters with a special meaning.

Character	Description
[]	A set of characters
\	Escape character, used to formulate special characters
.	Any character, except newline character
^	Starts with
$	Ends with
*	Zero or more occurrences
+	One or more occurrences
?	Turns greedy matching to non-greedy matching
{}	Exactly the specified number of occurrences
|	Either or
()	Capture and group

Creating More General Patterns

- A dot . is a wild card that returns a match of any one character, except for a newline (\n).
- A plus + means repeat the previous pattern at least once.
- So the combination of .+ means return a match of at least one character.

```
>>> text = 'HKU Business School'
>>> x = re.findall('B.+s', text)
>>> print(x)
['Business']
```

Greedy vs. Non-Greedy Matching

- The repeat characters (* and +) push outward in both directions (greedy) and return the largest possible substring.
- To turn greedy match off, add a ? character. Then it becomes non-greedy.

```
>>> text = 'From <chao.ding@hku.hk> Assignment 1'
>>> x = re.findall('c.+k', text)
>>> y = re.findall('c.+?k', text)
>>> print(x)
['chao.ding@hku.hk']
>>> print(y)
['chao.ding@hk']
```

Extracting a Portion of the Match

- We can determine which portion of the match is to be extracted by using parentheses.
- Parentheses are not part of the match - but they tell where to start and stop what string to extract.

```
>>> text = 'From <chao.ding@hku.hk> Assignment 1'
>>> x = re.findall('<.+@.+>', text)
>>> y = re.findall('<(.+@.+)>', text)
>>> print(x)
['<chao.ding@hku.hk>']
>>> print(y)
['chao.ding@hku.hk']
```

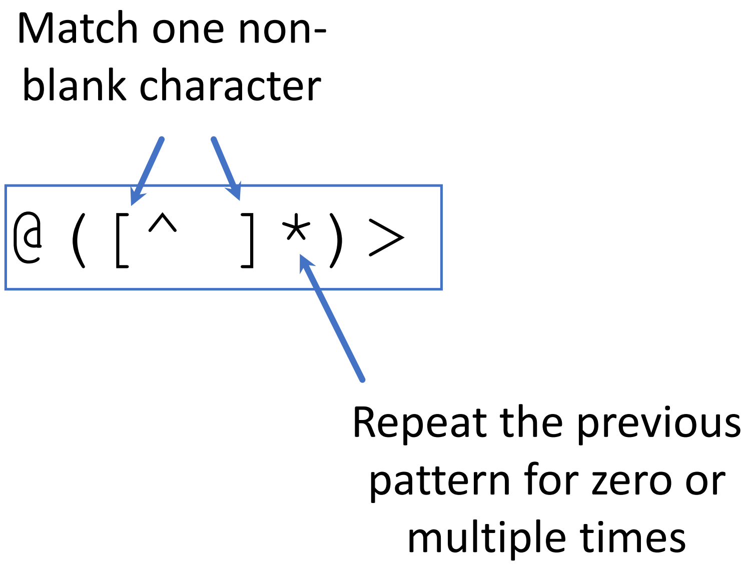
```
<(.+@.+)>
```

Only extract the portion defined in the parentheses

Extracting a Portion of the Match

- Further fine-tune the pattern to extract only the domain name hku.hk

```
>>> text = 'From <chao.ding@hku.hk> Assignment 1'
>>> x = re.findall('@([^ ]*)>', text)
>>> print(x)
['hku.hk']
```


Match one non-blank character
@([^]*)>
Repeat the previous pattern for zero or multiple times

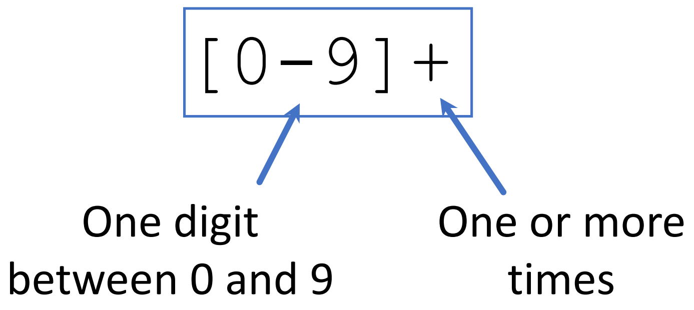
Sets

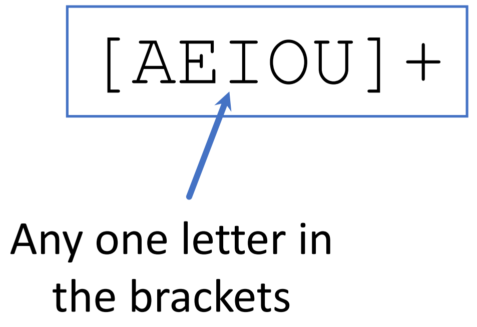
- Use square brackets to define a set of elements.

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Matching With a Set

```
>>> text = 'My 2 favorite numbers are 19 and 42'
>>> x = re.findall('[0-9]+',text)
>>> y = re.findall('[AEIOU]+',text)
>>> print(x)
['2', '19', '42']
>>> print(y)
[]
```





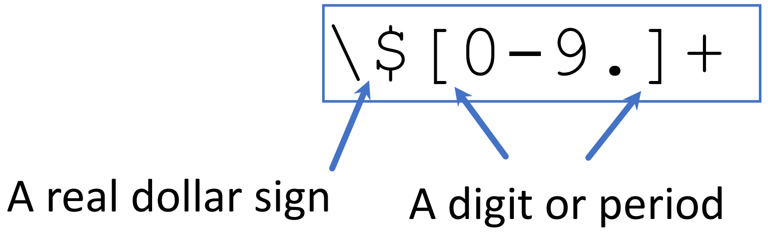
Special Characters and Escape Characters

Character	Description
\A	Returns a match if the specified characters are at the beginning of the string
\d	Returns a match where the string contains digits (numbers from 0-9)
\D	Returns a match where the string DOES NOT contain digits
\s	Returns a match where the string contains a white space character
\S	Returns a match where the string DOES NOT contain a white space character
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)
\W	Returns a match where the string DOES NOT contain any word characters
\Z	Returns a match if the specified characters are at the end of the string
\t	Returns a match with a tab
\.	Returns a match with a dot
\\\\	Returns a match with a backslash
\[	Returns a match with a left square bracket

Matching With Special Characters

- We can use the escape character \ to match with special characters.

```
>>> text = 'We just received $10.00 for cookies.'
>>> x = re.findall('\$[0-9.]+',text)
>>> y = re.findall('\$\d+’, text)
>>> print(x[0])
$10.00
>>> print(y[0])
$10
```



Regex Flags

- We use the optional flags to enable various unique features.
- For instance, ignore cases in the match.

```
>>> s = 'PYTHON is awesome'
>>> pattern = '[a-z]+'
>>> l = re.findall(pattern, s, flags = re.I)
>>> print(l)
['PYTHON', 'is', 'awesome']
```

- To add multiple flags, use | operator.

```
flags = re.I | re.M | re.X
```

Regex Flags

Flag	Alias	Meaning
re.ASCII	re.A	The re.ASCII is relevant to the byte patterns only. It makes the \w, \W,\b, \B, \d, \D, and \S perform ASCII-only matching instead of full Unicode matching.
re.IGNORECASE	re.I	perform case-insensitive matching. It means that the [A-Z] will also match lowercase letters.
re.LOCALE	re.L	The re.LOCALE is relevant only to the byte pattern. It makes the \w, \W, \b, \B and case-sensitive matching dependent on the current locale. The re.LOCALE is not compatible with the re.ASCII flag.
re.MUTILINE	re.M	The re.MULTILINE makes the ^ matches at the beginning of a string and at the beginning of each line and $ matches at the end of a string and at the end of each line.
re.DOTALL	re.S	By default, the dot (.) matches any characters except a newline. The re.DOTALL makes the dot (.) matches all characters including a newline.
re.VERBOSE	re.X	The re.VERBOSE flag allows you to organize a pattern into logical sections visually and add comments.

Summary

- What are regular expressions?
- Regex methods
- Creating Regex objects
- Matching and parsing text
- Regex flags