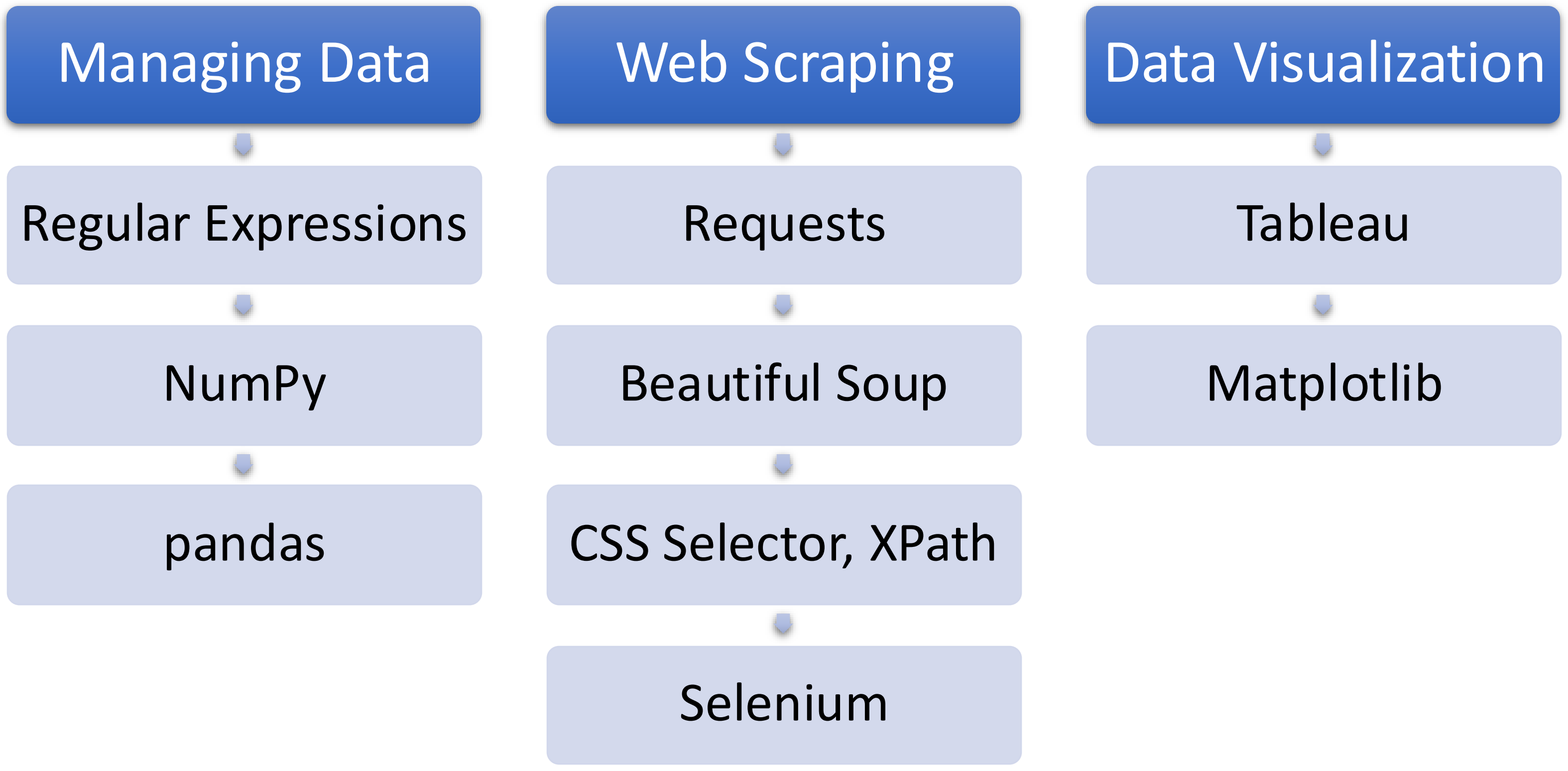
Managing Data I

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

Instructor: Dr. DING Chao

Course Roadmap



Regular Expressions (RegEx)

Regular Expressions

- Our data file may include millions of lines, we want to extract a specific selection of data:
  - ✓ date and time
  - ✓ email addresses
  - ✓ Names
  - ✓ URLs
- Regular Expressions (also called RegEx) provide a great way to match and parse text patterns.

```
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.

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

- Match object 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

```
re.search('HKU', text).span()
```

(0, 3)

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')
```

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)
>>> print(x)
['chao.ding@hku.hk']

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

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

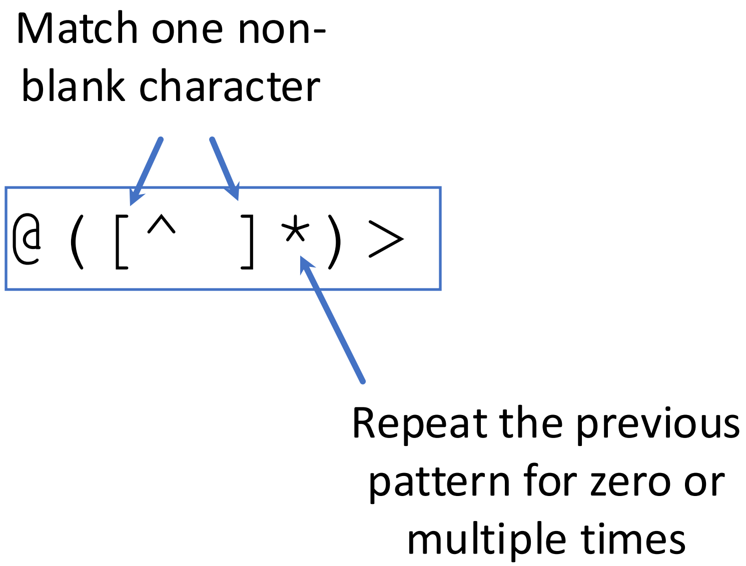
```
<(.+@.+)>
```

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

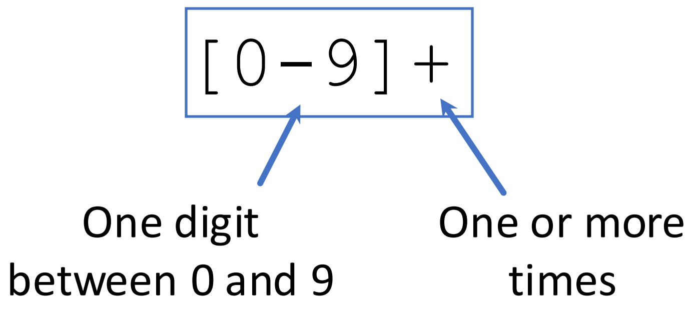
- Use square brackets to define a set of elements.

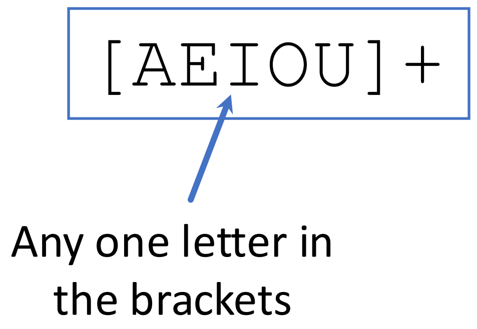
Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (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)
>>> print(x)
['2', '19', '42']

>>> y = re.findall('[AEIOU]+',text)
>>> print(y)
[]
```





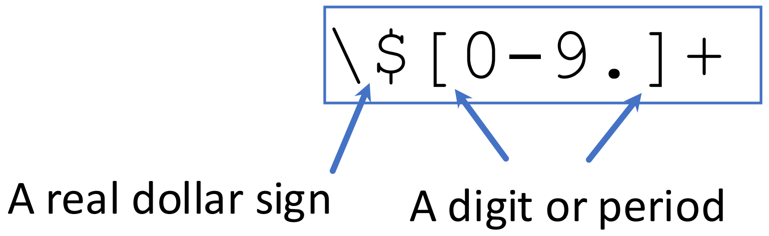
Special Characters and Escape Characters

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
```



Special Uses

Character	Description
\b	Matches with word boundary. Pattern must be in raw string.
(HK|US)D	Matches with "HKD" or "USD", returns "HK" or "US".
(?:HK|US)D	Matches and returns "HKD" or "USD". (?:) creates a non-capturing group.
x(?=y)	Matches and returns "x" only if "x" is followed by "y".
x(?!y)	Matches and returns "x" only if "x" is not followed by "y".
(?<=y)x	Matches and returns "x" only if "x" is preceded by "y".
(?<!y)x	Matches and returns "x" only if "x" is not preceded by "y"

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.