Introduction to Regular Expressions

Faculty of Information Technology
Monash University

FIT5196 Week 2

- A regular expression is a set of symbols that describes a text patten.
  - \d{4}-\d{2}-\d{2}
  - wrangling
- Why regular expressions?
  - Regular expressions are useful in finding, replacing and extracting information from text, such as log files, HTML/XML files, and other documents
    - Search a document for color or neighbor with or without u
    - Covert a tab-delimited file to a comma-delimited file
    - Find duplicated words in a text
    - Search and replace “Bob” and “Bobby” with “Robert”
  - Regular expressions are useful in verifying whether input fits into the text pattern, such as verifying
    - phone numbers: Does a phone number have the right number of digits?
    - emails: Is an email address in a valid format?
    - date: Is a date in the right format? Does the month exceed 12?

MONASH University

```
r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
```
[1]

[1] http://emailregex.com/

Regular Expressions: validate emails

r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"[1]

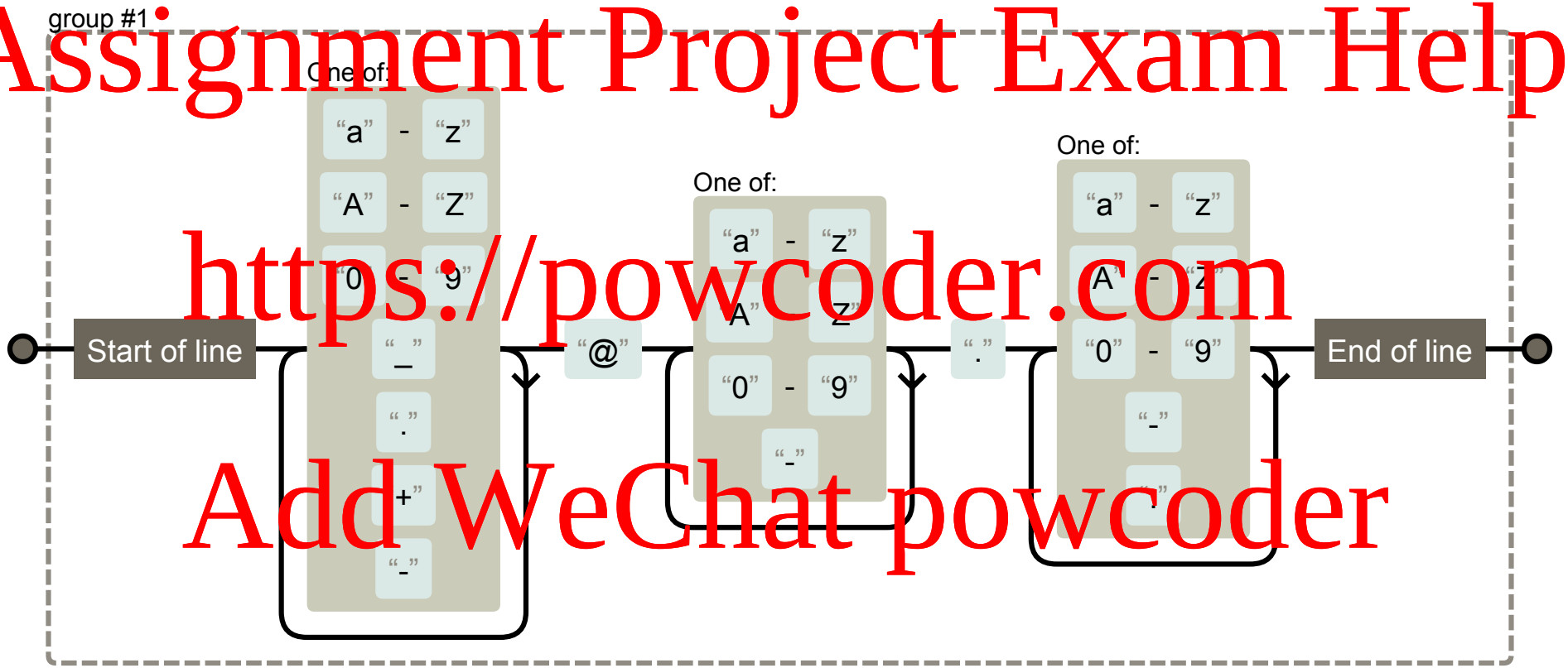


Figure: Figure generated by https://regexper.com/

[1] http://emailregex.com/

Outline

Matching String Literals

The most obvious feature of regular expressions is matching strings with one or more literal characters, called string literals.

- Everything is essentially a character in regular expressions.
  - cat matches "cat".
  - cat matches the first three characters of "cattle" and "catfish".
- It is similar to searching in word processing program
- Matching is case-sensitive.
  - cat does not match "Cat".
- How does regular expression engine work?

cat

The cow, camel and cat communicated.

Character sets: [...]

Assume that we are going to match the following two words:

grey gray

What the regular expression should be?

Character sets: [...]

Assume that we are going to match the following two words:

grey gray

What the regular expression should be?

- [...] indicate a set of characters
  - Matches any one of several characters in the set, but only one
  - The order of characters does not matters.

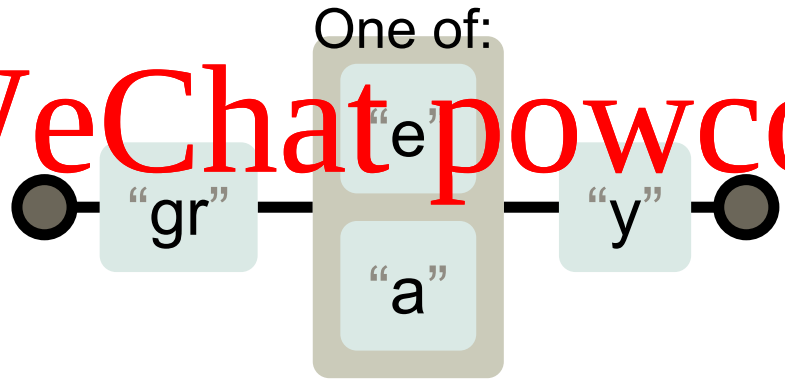
Character sets: [...]

Assume that we are going to match the following two words:

grey gray

What the regular expression should be?

- [...] indicate a set of characters
  - Matches any one of several characters in the set, but only one
  - The order of characters does not matters.
  - The regular expression is gr[ea]y



  - gr[ea]y does not match grAy, graay, and graey.

Character ranges: $[a-zA-Z]$ and $[0-9]$

Assume that we are going to match victory car plate numbers, for example

XRA 000, 1AA 1AA

Note the letters can be from A to Z, and the numbers can be from 0 to 9. What the regular expression should be?

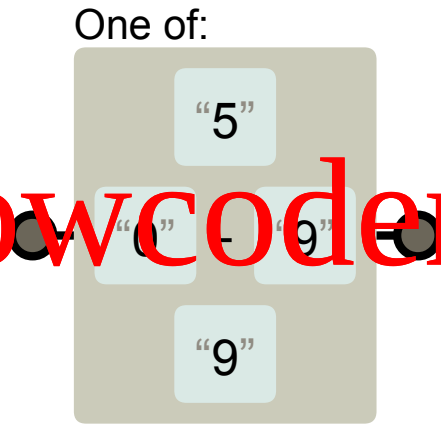
Character ranges: $[a-zA-Z]$ and $[0-9]$

Assume that we are going to match victory car plate numbers, for example

XRA 000, 1AA 1AA

Note the letters can be from A to Z, and the numbers can be from 0 to 9. What the regular expression should be?

- Character ranges can be indicated by giving two characters and separating them by a '-'.
- Example:
  - $[0-9]$
  - $[a-z]$ or $[A-Z]$
- Caution
  - $[50-99]$ is not all numbers from 50 to 99, it is the same as $[0-9]$.



Character ranges: $[a-zA-Z]$ and $[0-9]$

Assume that we are going to match victory car plate numbers, for example

XRA 000, 1AA 1AA

Note the letters can be from A to Z, and the numbers can be from 0 to 9. What the regular expression should be?

- [A-Z0-9][A-Z][A-Z]\s[0-9][A-Z0-9][A-Z0-9]

Negative character sets: [^ . . .]

Assume that we are going write a regular expression that matches only the live animals

Question: what is the regular expression?

Negative character sets: [^...]

Assume that we are going write a regular expression that matches only the live animals

npg dog bog

Question: what is the regular expression?

- [^...]: If the first character of the set is ^, all the characters that are not in the set will be matched.
  - [^bdg]og matches "hog" and "dog", but not "bog".
  - Caution:
    - Does see[^mn] match "see"?
    - Does see[^mn] match "see "?

Try the regular expression in Pythex (http://pythex.org/)!

Metacharacters inside character sets: [.+]

Assume that we are going to match the following two strings:

var(9), var[0]

Now, we need to match () and [ ]. How can we do that?

Metacharacters inside character sets: [.+]

Assume that we are going to match the following two strings:

var(9), var[0]

Now, we need to match () and [ ]. How can we do that?

- Metacharacters inside character sets are already escaped. In other words they lose their special meaning inside sets
  - Example
    - h[ai.u]t matches “hat”, “h.t”, but not “hot”
- Exceptions:], -, ^ and \that do need to be escaped.
  - h[a-u]t → h[ai-u]t?

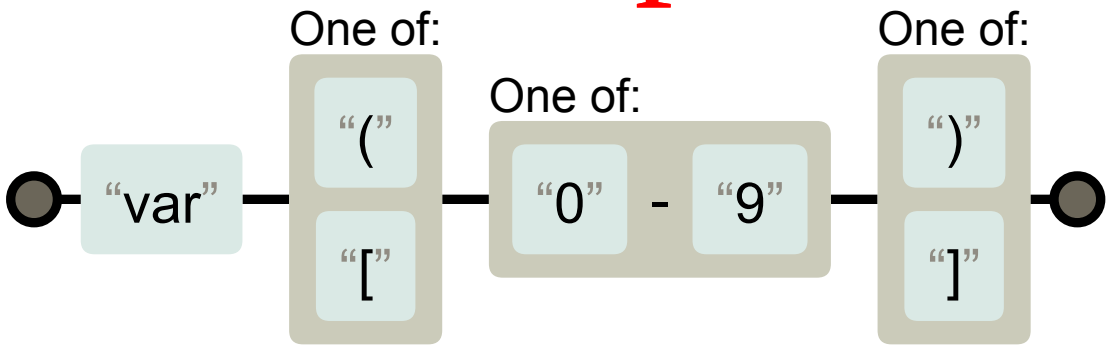
Metacharacters inside character sets: [.+]

Assume that we are going to match the following two strings:

var(9), var[0]

Now, we need to match () and [ ]. How can we do that?

- Metacharacters inside character sets are already escaped. In other words they lose their special meaning inside sets.
  - Example
    - h[ai.u]t matches "hat", "h.t", but not "hot"
- Exceptions:], -, ^ and \that do need to be escaped.
  - h[ai.u]t → h[ai]u]t?
- var[([][0-9][)\]]



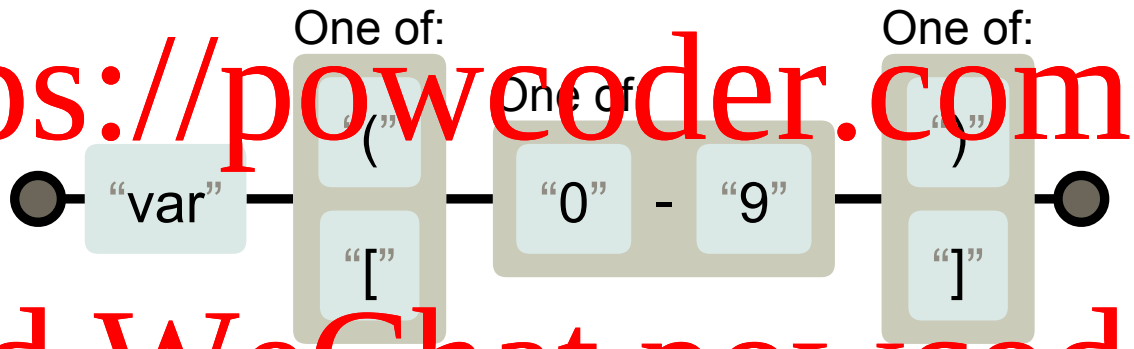
Metacharacters inside character sets: [.+]

Assume that we are going to match the following two strings:

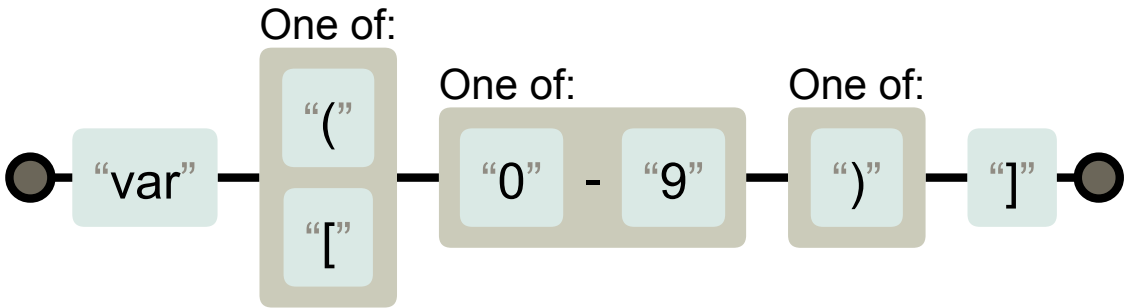
var(9);var[0]

Now, we need to match () and [ ]. How can we do that?

- var[([][0-9][)\]]



- var[([][0-9][)]]



Shorthand character sets

Shorthand	meaning	Equivalent
\d	matches any decimal digit from 0 to 9	[0-9]
\w	matches any word character	[a-zA-Z0-9_]
\s	matches any white space character	[\t\n\r]
\D	matches any non-digit character;	[^0-9]
\W	matches any non-alphanumeric character	[^a-zA-Z0-9_]
\S	matches any non-whitespace character	[^\t\n\r]

Shorthand character sets

Shorthand	meaning	Equivalent
\d	matches any decimal digit from 0 to 9	[0-9]
\w	matches any word character	[a-zA-Z0-9_]
\s	matches any white space character	[\t\n\r]
\D	matches any non-digit character;	[^0-9]
\W	matches any non-alphanumeric character	[^a-zA-Z0-9_]
\S	matches any non-whitespace character	[^\t\n\r]

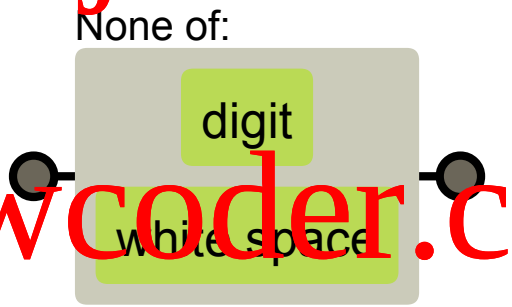
- Examples:
  - \d\d\d\d matches four-digit numbers, such as "2018", but not text.
  - \w\w\w matches three word characters, such as "abc", "123" and "d_b"
  - \w\w\s\w matches "ab c" but not "a bc".
  - [\w]-[\w] matches two characters separated by a hyphen.
  - [^\d] is the same as [\D]

Shorthand character sets

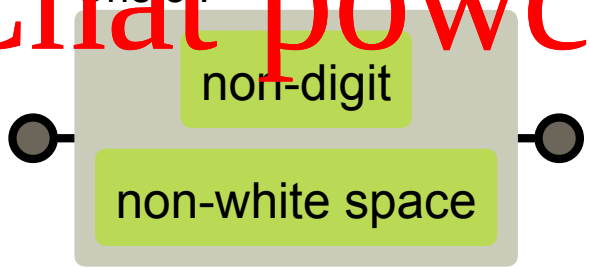
- Caution:
  - Is [^\d\s] the same as [\D\S]?

Shorthand character sets

- Caution:
  - Is [^\d\s] the same as [\D\S]?
    - [^\d\s]: Not digit OR space character



  - [\D\S]: EITHER NOT digit OR NOT space character



Try the regular expression with the following sentence: "Data Wrangling S2 2018 week 2"

Repetition Expressions: repetition meta-characters

MONASH University

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

Repetition Expressions: repetition meta-characters

MONASH University

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

oops ooops oooops ooooops

but not

ops

which regular expression(s) should we use?

1. oo*ps
2. ooo*ps
3. oo+ps
4. oo?ps

Repetition Expressions: repetition meta-characters

MONASH University

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

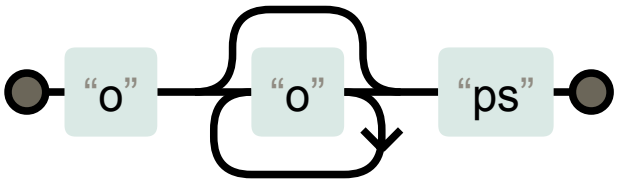
oops ooops oooooops ooooooops

but not

oops

which regular expression(s) should we use?

1. oo*ps
2. ooo*ps
3. oo+ps
4. oo?ps



Repetition Expressions: repetition meta-characters

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

oops ooops oooooops ooooooops

but not

ops

which regular expression(s) should we use?

1. oo*ps
2. ooo*ps
3. oo+ps
4. oo?ps

Repetition Expressions: repetition meta-characters

MONASH University

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

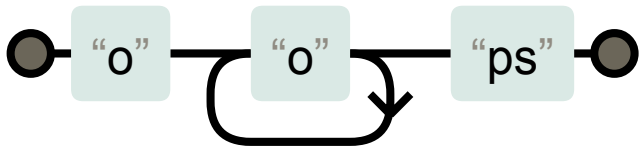
oops ooops oooops ooooops

but not

oops

which regular expression(s) should we use?

1. oo*ps
2. ooo*ps
3. oo+ps
4. oo?ps



Repetition Expressions: repetition meta-characters

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

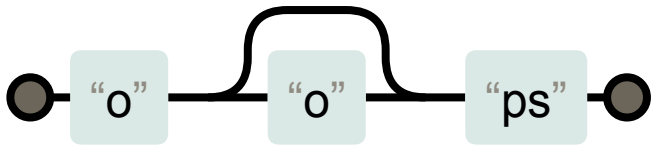
oops ooops oooooops ooooooops

but not

ops

which regular expression(s) should we use?

1. oo*ps
2. ooo*ps
3. oo+ps
4. oo?ps



Repetition Expressions: repetition meta-characters

meta-characters	meaning
*	Match 0 or more repetitions of the preceding regex
+	Match 1 or more repetitions of the preceding regex
?	Match 0 or 1 repetitions of the preceding regex

- Examples:
  - Assume we are going to match the following words

oops ooops oooops ooooops

but not

ops

The regular expressions that we can use:

ooo*ps oo+ps

Try the regular expression in Pythex!

Repetition Expressions: quantified repetitions

MONASH University

- {m, n}: matches exactly from *m* to *n* repetitions of the preceding regular expression.
  - *m* (min) and *n* (max) are positive numbers
  - *m* must be always be included, can be 0
  - *n* is optional
- Three syntax
  - \d{2} matches numbers with exactly 2 digits.
  - \d{2, 4} matches numbers with 2 to 4 digits.
  - \d{2, } matches numbers with at least 2 digits (n is infinite).

Try the 'oops' example in Pythex, but with {m, n}

Repetition Expressions: quantified repetitions

- Suppose we are going to match the following

report_2018_09	assignment_2018_9
budget_8_08	assignment_18_7

but not

report_201809_39	assignment_8_9000
budget_2345678_08	assignment_000999_7

what is the regular expression?

Repetition Expressions: quantified repetitions

- Suppose we are going to match the following

report_2018_09	assignment_2018_9
budget_8_08	assignment_18_7

but not

report_201809_39	assignment_8_9000
budget_2345678_08	assignment_000999_7

what is the regular expression?

```
\w+_\d{2,4}_\d{1,2}
```

Try the regular expression in Pythex

Repetition Expressions: greedy v.s. lazy regex

MONASH University

- Greedy strategy:
  - Match as much as possible before giving control to the next regular expression part
    - Regular expressions try to match the longest possible string
  - Examples

.*\d+

number 5196

Repetition Expressions: greedy v.s. lazy regex

MONASH University

- Greedy strategy:
  - Match as much as possible before giving control to the next regular expression part.
    - Regular expressions try to match the longest possible string
  - Examples

    .*\d+

    number 5196
  - Question: Given a string like

    "data", "wrangling", "FIT5196, S2."

    What is the match of regular expression ".+", ".+"?
    1. "data", "wrangling"
    2. "wrangling", "FIT5196, S2."
    3. "data", "wrangling", "FIT5196, S2."

Repetition Expressions: greedy v.s. lazy regex

- Greedy strategy:
  - Match as much as possible before giving control to the next regular expression part.
    - Regular expressions try to match the longest possible string
  - Examples

    .*\d+

    number 5196

  - Question: Given a string like

    "data", "wrangling", "FIT5196, S2."

    what is the match of regular expression ".+", ".+"?

    1. "data", "wrangling"
    2. "wrangling", "FIT5196, S2."
    3. "data", "wrangling", "FIT5196, S2."

Repetition Expressions: greedy v.s. lazy regex

MONASH University

- Lazy strategy:
  - Match as little as possible before giving control to the next regular expression part
  - Syntax
    - *?
    - +?
    - ??
    - {m,n}?
  - Example:

.*?\d+

number 516

Repetition Expressions: greedy v.s. lazy regex

- Lazy strategy:
  - Match as little as possible before giving control to the next regular expression part
  - Syntax
    - *?
    - +?
    - ??
    - {m,n}?
  - Example:

    .*?\d+

    number 516

- Question: Given a string like

  "data", "wrangling", "FIT5196, S2."

  what is the match of regular expression ".+?", ".+?"?

  1. "data", "wrangling"
  2. "wrangling", "FIT5196, S2."
  3. "data", "wrangling", "FIT5196, S2."

Repetition Expressions: greedy v.s. lazy regex

- Lazy strategy:
  - Match as little as possible before giving control to the next regular expression part
  - Syntax
    - *?
    - +?
    - ??
    - {m,n}?
  - Example:

.*?\d+

number 516

- Question: Given a string like

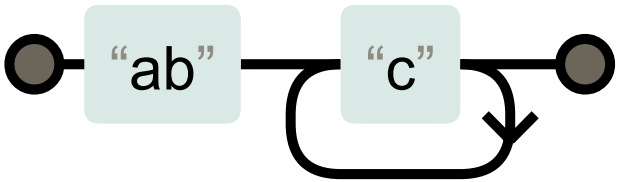
"data", "wrangling", "FIT5196, S2."

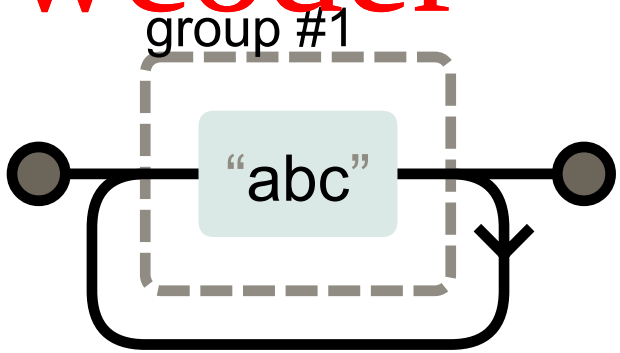
what is the match of regular expression ".+?", ".+?"?

1. "data", "wrangling"
2. "wrangling", "FIT5196, S2."
3. "data", "wrangling", "FIT5196, S2."

Grouping: (...)

- (...) matches whatever regular expression is inside the parentheses, and indicates the start and end of a group.
  - Apply repetition operators to a group of regular expressions
  - Makes regular expressions easier to read
  - Capture groups for use in matching, replacing and extraction, i.e., the contents of a group can be retrieved.
  - Cannot be used inside a character set.
- examples:
  - abc+ matches abc, abcc, abccc
  - (abc)+ matches abc, abcabc, abcabcabc





Grouping: (...)

- (...) matches whatever regular expression is inside the parentheses, and indicates the start and end of a group.
  - Apply repetition operators to a group of regular expressions
  - Makes regular expressions easier to read
  - Capture groups for use in matching, replacing and extraction, i.e., the contents of a group can be retrieved.
  - Cannot be used inside a character set.

- examples:
  - "Incident American Airlines Flight 11 involving a Boeing 767-223ER in 2001"
  - Regular expression: Incident (.*) involving

  - Try it with python script!!!

Alternation: |

- "|" is an OR operator
  - A|B will match any string that matches either A or B
  - Ordered - leftmost expression gets precedence.
  - Multiple patterns can be daisy-chained.
  - Group alternation expressions to keep them distinct.
- Examples:
  - apple|orange matches "apple" and "orange"
  - (apple|orange) juice matches "apple juice" and "orange juice"
  - w(ei|ie)rd matches both "weird" and "wierd".

The backslash plague: \

- The back slash \indicates special forms or to allow special characters to be used without invoking their special meaning.[2]

Characters	Stage
\section	text string to be matched
\\section	Escaped backslash for re.compile()
\\\\section	Escaped backslashes for a Python string literal

- So, to match a literal backslash, one has to write '\\\\' as the regular expression string
- Can we simply the expression?

[2] see https://docs.python.org/3/howto/regex.html

Raw String: r"..."

- Raw String suppress actual meaning of escape characters, and do not treat the backslash as a special character at all.

Regular Python string litera	Raw string
"\\\\section"	r"\\section"
"\\w+\\s+"	r"\w+\s+"

- Regular expressions will often be written in Python code using this raw string notation.
- Try the Python script!!!

Case study 1: validate dates

- Date samples (day, month, year):
  - 02/08/2018
  - 2/8/2018
  - 2/8/18
  - 23/08/2018
  - 23-08-2018
- See jupyter notebook!!!

Case study 2: validate credit card number

- Assume that you're given the job of implementing an order form for a company that accepts payment by credit card issued by the world's major credit card companies such as VISA, Master, and American Express.
  - All Visa card numbers start with a 4. New cards have 16 digits. Old cards have 13.
    - 4123456789012
    - 4123456789012345
  - MasterCard numbers either start with the numbers 51 through 55. All have 16 digits.
    - 5123456789012345
    - 5523456700012345
  - American Express card numbers start with 34 or 37 and have 15 digits.
    - 341234567890123
    - 371234567890123
- See jupyter notebook!!!

Summary: what to do this week

- Regular expressions are the major tool used in data parsing.
- Study materials provided in Moodle.
- Attend tutorial 2:
  - Try to finish all the materials provided in the tutorial.
- **Assessment 1** will be released at the end of this week (week 2).
- Topic for next week:
  - Parsing data stored in different file formats, CSV, JSON, XML, EXCEL, and PDF