Regular Expressions

CS 246

Introduction

- Regular expressions can be used to see if a string conforms to a pattern
- It can be used to see if a string is a legitimate ...
  - phone number
  - date
  - zip code
  - ssn
  - password
- It can be used to search a string for a particular pattern
- It can be fed into split() to split a string into pieces (e.g., dd/mm/yy ==> [dd,mm,yy])
- It can be used to split an infix expression into pieces, e.g., (4+5) => ['(', '4', '+', '5', ')']

Regular Expressions

- A regular expression (regex) is a string pattern matched against a string
- It can be defined with a RegExp object or as a literal between /'s
  - let re = new RegExp('ban');
  - let re = /ban/; // yes, from the same people who brought you unshift() and **, 😍
- 'ban' matches any string that contains 'ban'
  - e.g., 'abandon' and 'urban' would match 'ban'; 'tuba needs tuning' would not
- The RegExp method **test(str)** returns true if str matches the regex
  - re.test('abandon') ==> true
  - re.test('Banana') ==> false // JS is case-sensitive

Special Characters

- Regular expressions use special characters to create more sophisticated patterns
- e.g., **+** means match the preceding character ≥ 1 times
- /ab+c/ would match abc, abbc, abbbbbbc but not ac
- **( )** are used in a regex to designate a group of characters, and *, + and ? apply to the entire group
- /a(bc)+d/ would match abcd, abcbcd, but not ad or abcbd
- /a(bc)?d/ would match ad, abcd, but not abcbcd
- /a(bc)*d/ would match ad, abcd, abcbcd, abcbcbcd, etc.

Selected Special Characters

- To use any of these characters in a pattern, they need to be escaped. e.g., to match who?, use /who\?/

Symbol	Description	Example	Mnemonic
*	Match the preceding character 0 or more times.	`/ab*c/` matches "abc", "abbbbc", and "ac".	* looks like a 0, so 0 or more
+	Match the preceding character 1 or more times.	`/ab+c/` matches "abc" and "abbbbc" but not "ac".	+ looks like a 1 (1 vertical, 1 horizontal), so 1 or more
?	Match the preceding character 0 or 1 time.	`/ab?c/` matches "abc" and "ac", but not "abbc".	? is binary - 0 or 1
^	Match at the beginning.	`/^ab/` matches "abc" but not "cab".	We should **start** car(at)ing about each other more 🙄
$	Match at the end.	`/ab$/` matches "cab" but not "abc".	"The buck **stops** here" 🙄
|	Match string on the left OR string on the right.	`/ab|cd/` matches "abc" and "bcd".	None needed

Exercises

- Match hello, helloo, hellooo, etc. at the beginning of a string
- Match red or green
- Match reading or Reading at the end of a string (with as short a regex as possible)
- Match flp or flap
- Match flp, flap, flaap, flaaap, etc.
- Match (?)

Exercises - Solutions

- Match hello, helloo, hellooo, etc. at the beginning of a string /^hello+/
- Match red or green /red|green/
- Match reading or Reading at the end of a string (with as short a regex as possible) /(R|r)eading$/ or (as we'll see in a moment on the next slide) /[Rr]|eading$/
- Match flp or flap /fla?p/
- Match flp, flap, flaap, flaaap, etc. /fla*p/
- Match (?) /\(\?\)/

Character Ranges

- Use **[ ]** to match any single character in a range
- /[aeiou]/ matches any (lowercase) vowel
- /[0-9]/ matches any digit
- ^ negates a range (when it appears inside [ ])
- /[^aeiou]/ matches anything that is not a vowel

Exercises

- Match any year in the 21st century
- Match a phone number of the form (920)ddd-dddd
- Match a postal code, e.g., R3M 3P5
- Match any word surrounded by spaces that is ALLCAPITALS

Exercises - Solutions

- Match any year in the 21st century /2[0-9][0-9][0-9]/
- Match a phone number of the form (920)ddd-dddd
- /\([1-9][0-9][0-9]\)[1-9][0-9][0-9]-[0-9][0-9][0-9][0-9]/
- Match a postal code, e.g., R3M 3P5
- /[A-Z][0-9][A-Z] [0-9][A-Z][0-9]/
- Match any word that is ALLCAPITALS
- /\s[A-Z]+\s/

N.B. Note that w is any alphanumeric and underscore character, not just an alphabetic one. [The letter *ought* to be \a, but, alas, it is not.]

Metacharacters

- A meta character is a character or character sequence that matches a class of characters in a regular expression

- e.g., . matches any single character (except newline)

- /he.p/ matches help, he p, he#p

Metacharacter	Description	Example
`.`	Match any single character except newline.	`/a.b/` matches "aZb" and "a b".
`\w`	Match any word character (alphanumeric and underscore).	`/a\wb/` matches "aAb" and "a5b" but not "a b".
`\W`	Match any non-word character.	`/a\Wb/` matches "a-b" and "a b" but not "aZb".
`\d`	Match any digit.	`/a\db/` matches "a2b" and "a9b", but not "aZb".
`\D`	Match any non-digit.	`/a\Db/` matches "aZb" and "a b", but not "a2b".
`\s`	Match any whitespace character (space, tab, form feed, line feed).	`/a\sb/` matches "a b" but not "a4b".
`\S`	Match any non-whitespace character.	`/a\Sb/` matches "a!b" but not "a b".

What matches?

1.123break

2.923 break

3.break1 9

4.()break

5.5!?break

6.0.break

A `/\d\S\?/`

B `/1\s\d/`

C `/\W\W\D/`

D `/1\w+/`

E `/\d\./`

F `/9\d+/`

What matches?

1.123break

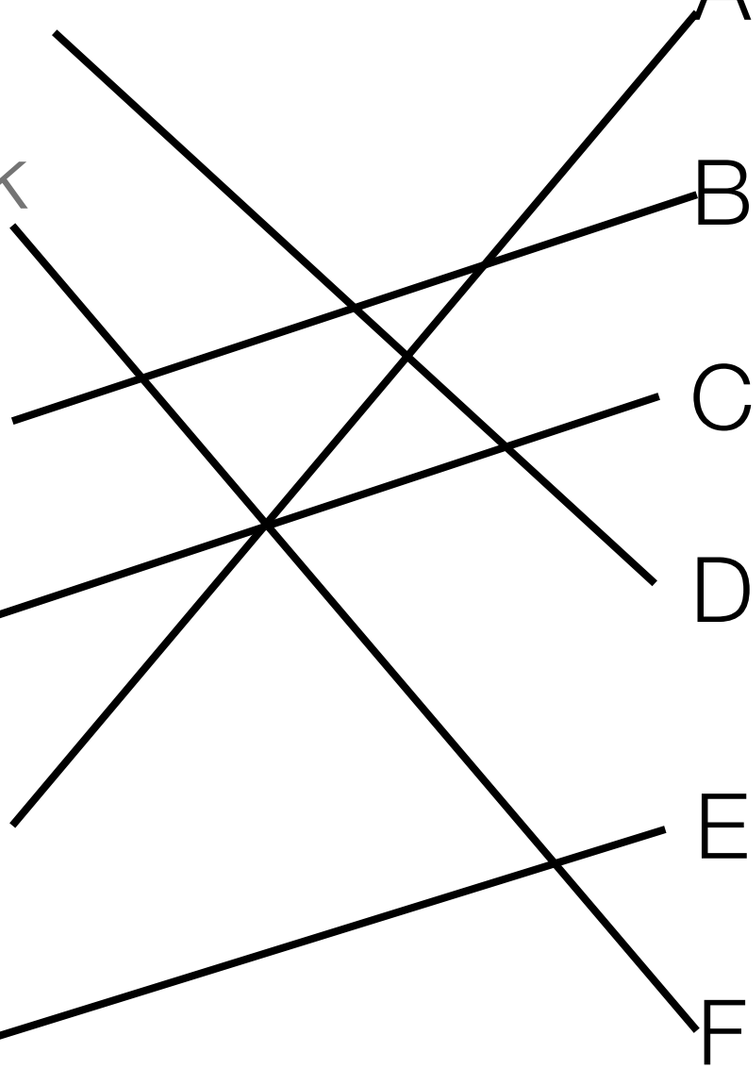
2.923 break

3.break1 9

4.()break

5.5!?break

6.0.break



A /\d\S\?/

B /1\s\d/

C /\W\W\D/

D /1\w+/

E /\d\./

F /9\d+/

More Quantifiers

- { n } matches the previous element exactly n times
- { n,} matches the previous element at least n times
- {n,m} matches the previous element n-m times (inclusive)

Exercises

- Match a String of the form Date: dd/mm/20yy where the number of spaces after the colon can be from 1-3 inclusive

Exercises

- Match a String of the form Date: dd/mm/20yy where the number of spaces after the colon can be from 1-3 inclusive

- /Date: {1,3}\d{2}\/\d{2}\/20\d\d/
  or
  /Date: {1,3}\d{2}\/\d{2}\/20\d{2}/

Mode Modifiers

- A **mode modifier** or flag changes how a regex matches,
- It appears after the closing slash in a regex, e.g., /abc*/i

Mode modifier	Description	Example
`i`	Case insensitivity - Pattern matches upper or lowercase.	`/aBc/i` matches "abc" and "AbC".
`m`	Multiline - Pattern with `^` and `$` match beginning and end of any line in a multiline string.	`/^ab/m` matches the second line of "cab\nabc", and `/ab$/m` matches the first line.
`g`	Global search - Pattern is matched repeatedly instead of just once.	`/ab/g` matches "ab" twice in "cababc".

Exercises

- Match the first occurrence of fish
- Match the last occurrence of fish
- Match all occurrences of fish
- Match all occurrences of fish using /Fish/
- 'One fish two fish red fish blue fish' ?

Exercises

- Match the first occurrence of fish /fish/
- Match the last occurrence of fish -- this requires the reg ex exec() method
- Match all occurrences of fish /fish/g
- Match all occurrences of fish using /Fish/ /Fish/gi

Other Helpful RegEx Methods

The + operator is greedy -- it doesn't give up, just like the crew of Galaxy Quest 😀. Rather, it matches as many characters as it can, subject to the rest of the pattern being matched as well. So, it returns 'give up, never', and doesn't stop at 'give' or 'give up'.

- exec() - returns a result array containing the matched portion(s)
- Example:
- let re = /give.+der/
- let result = re.exec('Never give up, never surrender - Galaxy Quest'
  - result: ['give up, never surrender', ...]
- If the regex contains multiple parts enclosed in ( ), exec() returns the entire result, then each matched part. The ( ) "remember" the matched parts
- let re = /(gi.+) (.+der)/
- result = re.exec('Never give up, never surrender - Galaxy Quest')
  - result: ['give up, never surrender', 'give up, never', 'surrender', ...]