Regular expressions

Regular expressions are a very powerful tool to do string matching and processing.

Allows you to do things like:
- Tell me if a string starts with a lowercase letter, then is followed by 2 numbers and ends with “ing” or “ion”
- Replace all occurrences of one or more spaces with a single space
- Split up a string based on whitespace or periods or commas or …
- Give me all parts of the string where a digit is proceeded by a letter and then the “#” sign

Regular expressions: literals

We can put any string in a regular expression
- /test/ matches any string that has “test” in it
- /this class/ matches any string that has “this class” in it
- /Test/ case sensitive: matches any string that has “Test” in it

http://xkcd.com/208/
Regular expressions: character classes

A set of characters to match:
- put in brackets: []
- [abc] matches a single character a or b or c

What would the following match?
/\[Tt\]est/ any string with “Test” or “test” in it

Can use - to represent ranges
- [a-z] is equivalent to \[abcdefghijklmnopqrstuvwxyz\]
- [A-D] is equivalent to \[ABCD\]
- [0-9] is equivalent to \[0123456789\]

For example:
/\[0-9][0-9][0-9][0-9]/
matches any four digits, e.g. a year

Can also specify a set NOT to match:
^ means all characters EXCEPT those specified
- [^a] all characters except ‘a’
- [^0-9] all characters except numbers
- [^A-Z] ???
**Regular expressions: character classes**

For example:

```
/\[0-9][0-9][0-9][0-9]/
```

matches any four digits, e.g. a year

Can also specify a set NOT to match:

- `^` means all characters EXCEPT those specified
  - `[^a]` all characters except ‘a’
  - `[^0-9]` all characters except numbers
  - `[^A-Z]` not an upper case letter (be careful, this will match any character that’s not uppercase, not just letters)

**Meta-characters (not always available)**

- `\w` word character (a-zA-Z_0-9)
- `\W` non word-character (i.e. everything else)
- `\d` digit (0-9)
- `\s` whitespace character (space, tab, endline, …)
- `\S` non-whitespace
- `\b` matches a word boundary (whitespace, beginning or end of line)
- `.` matches any character

---

**What would the following match?**

- `/19\d\d/`
  - would match any 4 digits starting with 19

- `/\s\s/`
  - matches anything with two adjacent whitespace characters (spaces, tabs, etc)

- `/\[aeiou]\s/`
  - any three letter word that starts with a vowel

---

**Regular expressions: repetition**

- `*` matches zero or more of the preceding character

  `/\w*\w/`
  - matches any string with:
    - red
    - bleed
    - bleed

- `/A.*A/`
  - matches any string starts and ends with A

- `+` matches one or more of the preceding character

  `/\w+\w/`
  - matches any string with:
    - red
    - bleed
    - bleed
Regular expressions: repetition

? zero or 1 occurrence of the preceding
/fights?/  
matches any string with “fight” or “fights” in it

{n,m} matches n to m inclusive
/ba{3,4}d/  
matches any string with

- baad
- baaaad

Regular expressions: repetition revisited

What if we wanted to match:
This is very interesting
This is very very interesting
This is very very very interesting

Would /This is very+ interesting/ work?
- No… + only corresponds to the ‘y’
- /This is (very )+interesting/

Repetition operators only apply to a single character.
Use parentheses to group a string of characters.

Regular expressions: beginning and end

^ marks the beginning of the line
$ marks the end of the line
/test/ test can occur anywhere
/^test/ must start with test
/test$/ must end with test
/^test$/ ???
### Regular expressions: disjunction

**We want to match:**
- I like cats
- I like dogs

**Does `/^I like (cats|dogs)$/` work?**

No!

**Matches:**
- I like cats
- I like dogs

**Solution?**

---

### Some examples

- All strings that start with a capital letter
- IP addresses
  - `255.255.122.122`
- Matching a decimal number
- All strings that end in 'ing'
- All strings that end in 'ing' or 'ed'
- All strings that begin and end with the same character
Some examples

All strings that start with a capital letter
/^[A-Z]/

IP addresses
/\b(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/b/

Matching a decimal number
/\[\+]?[0-9]+/.

All strings that end in 'ing'
/ing$/

All strings that end in 'ing' or 'ed'
/ing|ed$/

Regular expressions: memory

All strings that begin and end with the same character

Requires us to know what we matched already

() used for precedence
also records a matched grouping, which can be referenced later

/^[.]*$/

d all strings that begin and end with the same character

Regular expression: memory

/\She likes \(\w+\) and he likes \1/.

What would this match?

/\She likes \(\w+\) and he likes \1/.

She likes bananas and he likes bananas
She likes movies and he likes movies
...

Regular expression: memory

/\She likes \(\w+\) and he likes \1/.

She likes bananas and he likes bananas
She likes movies and he likes movies
...
She likes \( w^+ \) and he likes \( w^+ \) \\
We can use multiple matches \\
She likes \( w^+ \) and \( w^+ \) and he also likes \( w^+ \) and \( w^+ \) \\

Most languages also allow for substitution \\
\[ s/\text{banana}/\text{apple}/ \]
substitute first occurrence banana for apple \\
\[ s/\text{banana}/\text{apple}/g \]
substitute all occurrences (globally) \\
\[ s/\text{banana}/\text{apple}/g \]
duplicate the string, separated by a space \\
\[ s/\text{s+}/g \]
substitute multiple spaces to a space \\

Java: as part of the String class \\
\[ String s = \text{"this is a test"} \]
\[ s.matches("test") \]
\[ s.matches("this\text{.} test") \]
\[ s.split("\text{.} test") \]
\[ s.replaceAll("\text{.} test", "") \]

Be careful, matches must match the whole string (i.e. an implicit ^ and $)
### Regular expressions by language

**Java:** java.util.regex

Full regular expression capabilities

Matcher class: create a matcher and then can use it

```java
String s = "this is a test"
Pattern pattern = Pattern.compile("is\s+")
Matcher matcher = pattern.matcher(s)

• matcher.matches()
• matcher.find()
• matcher.replaceAll("blah")
• matcher.group()
```

**Python:**

```python
import re
s = "this is a test"
p = re.compile("test")
p.match(s)

p = re.compile(".*test.*")
re.split("\s+", s)
re.sub("\s+", " ", s)
```

**perl:**

```
s = "this is a test"
s =~ /test/
s =~ /test$/
s =~ /this\sis .* test/
split /\s+/, $s
$s =~ s/\s+/ /g
```

### Regular expression by language

**grep**

- command-line tool for regular expressions (general regular expression print/parser)
- returns all lines that match a regular expression
- `grep "@" twitter.posts`
- `grep "http:" twitter.posts`
- can’t use metacharacters (\d, \w), use [] instead
- Often want to use “grep –E” (for extended syntax)
Regular expression by language

**sed**
- another command-line tool that uses regular expressions to print and manipulate strings
- very powerful, though we'll just play with it
- Most common is substitution:
  - `sed "s/ is a / is not a /g" twitter.posts`
  - `sed "s/ / /g" twitter.posts`
    - `sed` doesn't have `+`, but does have `*`
- Can also do things like delete all that match, etc.

Regular expression resources

**General regular expressions:**
- Ch 2.1 of the book
- [http://www.regular-expressions.info/](http://www.regular-expressions.info/)
  - good general tutorials
  - many language specific examples as well

**Java**
- See also the documentation for java.util.regex

**Python**
- [http://docs.python.org/howto/regex.html](http://docs.python.org/howto/regex.html)
- [http://docs.python.org/library/re.html](http://docs.python.org/library/re.html)

**Perl**
- [http://perldoc.perl.org/perlretut.html](http://perldoc.perl.org/perlretut.html)
- [http://perldoc.perl.org/perlre.html](http://perldoc.perl.org/perlre.html)

**grep**
- See the write-up at the end of Assignment 1

**sed**
- See the write-up at the end of Assignment 1