

Week 8 : ~~Boba~~ Regex



07131 GPI Lecture

All boba related artwork done by Amy Liu for GPI. Do not reuse without crediting.

Announcements

- Dotfiles Extratation this weekend
- Also daylight saving ends this Sunday for those in US!



• Pet tax!

Brought to you
by Kish-Log



[vlog] started to raise [#kitten](#) and [#puppy](#) together - first 7 days

28,416,086 views • Sep 15, 2019

👍 281K 💬 11K ➦ SHARE ≡+ SAVE ⋮



KISH-Log
310K subscribers

SUBSCRIBE

What we want to achieve

Create financial statements from 1988 to 2019

Glob/Range

\$ touch .financial_statement

What we want to achieve

Create financial statements from 1988 to 2019

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country?.mp4
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip [?] .financial_statement
```


What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher ?scsPerson
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher *.scsPerson
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher
... except Tom and Veronica

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher *.scsPerson
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher
... except Tom and Veronica

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher *.scsPerson
```

```
???? (impossible to do efficiently)
```

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher
... except Tom and Veronica

Vouchers for Tom and Veronica should allow for unlimited boba! We want something that will accept/match "", "boba", "bobaboba", "bobabobaboba"... etc (boba only for now... not enough budget)

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher *.scsPerson
```

???? (impossible to do efficiently)

What we want to achieve

Create financial statements from 1988 to 2019

Remove all big_cat_country episodes

Zip up financial statements between 2010 and 2019, if any

Send every person in scs 1x free boba voucher
... except Tom and Veronica

Vouchers for Tom and Veronica should allow for unlimited boba! We want something that will accept/match "", "boba", "bobaboba", "bobabobaboba"... etc (boba only for now... not enough budget)

Glob/Range

```
$ touch {1988..2019}.financial_statement
```

```
$ rm big_cat_country*.mp4
```

```
$ zip fin_statements.zip 201?.financial_statement
```

```
$ ./sendFreeBobaVoucher *.scsPerson
```

???? (impossible to do efficiently)

???? (impossible)

Glob's kinda useless ... Do we have something better?



Introducing ... REGEX!

- “Regular Expression”
 - Patterns that match against certain strings
 - Different from globs!
 - Compatible with many applications
- But why are they called regular expressions?
 - For interesting theoretical reasons (that you will learn later)
- Example (that we’ll dissect later)
 - `(\d{3}-?){2}\d{4}`

Introducing ... REGEX!

- Normal characters
- Quantifiers
- Character classes
- Groups
- Other special expressions

Parts of a regular expression

- Normal characters

- boba – matches "boba"



- Character classes

- [sml] – matches "s", "m", or "l" (no, not standard meta language, it's small, medium, or large)
- \d – matches a digit
- . – matches any character



Character Classes

“Boba is my personality”

Class	Matches
[abc]	a or b or c
[a-z]	a lowercase letter
\s	whitespace
\d	digit
\w	alphanumeric char
.	any single character (ie, wildcard)

More Character Classes

Class	Matches	Class	Matches
[abc]	a or b or c	[^abc]	Not any of a, b or c
[a-z]	a lowercase letter	[^A-Za-z]	Not a character
\s	whitespace	\S	Not a whitespace
\d	digit	\D	Any non-digit
\w	alphanumeric char	\W	Any non-alphanumeric char
.	any single character (ie, wildcard)	\.	Just a period

Parts of a regular expression

- Up until now, we've seen character classes that only match to 1 single character. How do we specify ~quantity~?

- Quantifiers

- boba? - matches "bob" or "boba"
- boba* – matches "bob", "boba", " bobaaaa", etc.



or



More Quantifiers

“How many cups of boba
do you want?”
“yes”



Quantifier	Matches
<code>a?</code>	Zero or one (ie. optional)
<code>a*</code>	Zero or more
<code>a+</code>	One or more
<code>a{3}</code>	Exactly 3
<code>a{3,}</code>	3 or more
<code>a{3,6}</code>	Between 3 and 6 (inclusive)

Parts of a regular expression

- So far, the quantifiers only acts on the “character” immediately before it.
- Use parentheses for grouping
 - `boba*` - matches “bob”, “boba”, “bobaaa”...
 - `(boba)*` - matches “”, “boba”, “bobaboba”, “bobabobaboba” ...
 - `boba?` - matches “bob” or “boba”
 - `(boba)?` - matches exactly “” or “boba”

Parts of a regular expression

- So far, the quantifiers only acts on the “character immediately before it.
- Use parentheses for grouping
 - `boba*` - matches “bob”, “boba”, “bobaaa”...
 - `(boba)*` - matches “”, “boba”, “bobaboba”, “bobabobaboba” ...
 - `boba?` - matches “bob” or “boba”
 - `(boba)?` - matches exactly “” or “boba”



boba?



(boba)?

Note that pattern matching does NOT have to start from the beginning!

- Pattern: “\d” (will match any digit)
- Matches: (almost like a search-and-find)
 - abc**1**xyz
 - **1**abcxyz
 - abcxyz**1**

... but you might get false positives

- Pattern: “free boba”
- Matches:
 - “**free boba** with the purchase of 10 bobas”
 - “**free boba** when you spend \$1000 or more”
 - “Breaking news: care**free boba**-lover spends \$1000”

Other special expressions

- \wedge - Start of string/line (not to be confused with \wedge in $[\wedge abc]$)
 - **$\wedge(\text{free})$** matches only “free” at beginning of the string/line
- $\$$ - End of string/line
 - **$(\text{boba})\$$** matches only “boba” at the end of the string/line

Other special expressions

- `^` - Start of string/line (not to be confused with `^` in `[^abc]`)
 - **`^(free)`** matches only “free” at beginning of the string/line
- `$` - End of string/line
 - **`(boba)$`** matches only “boba” at the end of the string/line

`^` - jump to the first non-blank character of the line
`$` - jump to the end of the line



Other special expressions

- (this|that)
 - Don't put spaces unless you want to match a space
 - Put as many as you want: **(this|that|here|there)**
 - Can nest or use other character classes! Ex. you can express the above as **(th(is|at)|t?here)**



Class	Matches	Quantifier	Matches
[abc]	a or b or c	a?	Zero or one (ie. optional)
[a-z]	a lowercase letter	a*	Zero or more
\s	whitespace	a+	One or more
\d	digit	a{3}	Exactly 3
\w	alphanumeric char	a{3,}	3 or more
.	any single character	a{3,6}	Between 3 and 6 (inclusive)

[^abc]

Not any of a, b or c

[^A-Za-z]

Not a character

\S

Not a whitespace

\D

Any non-digit

\W

Any non-alphanumeric char

\.

Just a period

Other expressions

^(start)

Matches

“start” at the beginning of line/string

(end)\$

“end” at the end of the line/string

(this|that)

“this” or “that”

Example - boba shop phone numbers

- Multiple possible strings
 - 123-456-7890
 - 1234567890
 - 456-789-1234
- But the formats follow a few patterns
 - ###-###-####
 - #####

Example - boba shop phone numbers

(\d{3}-?){2}\d{4}

Example - boba shop phone numbers

`(\d{3}-?){2}\d{4}`

Matches any digit

Example - boba shop phone numbers

(\d{3}-?){2}\d{4}

Matches any 3 digits

Example - boba shop phone numbers

(\d{3}-?)\d{2}\d{4}

Matches an optional hyphen

Example - boba shop phone numbers

Ex:

123-456-

123456-

123456

`(\d{3}-?){2}\d{4}`

Matches 2 groups of 3 digits

Example - boba shop phone numbers

`(\d{3}-?){2}\d{4}`

Matches 2 groups of 3 digits, then
4 more digits

Example

`(small|medium|large) milk(green)? tea(
with (((no|less|half|extra)
(sugar|ice))|(milk foam)))*`

Match your own favorite drink order!



<https://regex101.com/>

REGULAR EXPRESSION 2 matches, 182 steps (~1ms)

```
i / (small|medium|large) milk (green)? tea (with (((no|less|half|extra) (sugar|ice))) (milk foam)))? / gm
```

TEST STRING

```
medium milk tea with no sugar with half ice with milk foam  
large milk green tea with half sugar with no ice
```

EXPLANATION

- ▼ / (small|medium|large) milk (green)? tea (with (((no|less|half|extra) (sugar|ice))) (milk foam)))? / gm
 - ▼ 1st Capturing Group (small|medium|large)
 - ▼ 1st Alternative small
small matches the characters **small** literally (case sensitive)
 - ▼ 2nd Alternative medium
medium matches the characters **medium** literally (case sensitive)
 - ▼ 3rd Alternative large
large matches the characters **large** literally (case sensitive)
 - milk matches the characters **milk** literally (case sensitive)
 - ▼ 2nd Capturing Group (green)?
 - ▼ Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (**greedy**)

MATCH INFORMATION

Match 1

Group	Start	End	Match
Full match	0	58	medium milk tea with no sugar with half ice with milk foam
Group 1.	0	6	medium
Group 3.	43	58	with milk foam
Group 4.	49	58	milk foam
Group 5.	35	43	half ice
Group 6.	35	39	half

QUICK REFERENCE

Search reference	Description	Regex
⊙ All Tokens	A single character of: a, b or c	[abc]
★ Common Tokens	A character except: a, b or c	[^abc]
⊙ General Tokens	A character in the range: a-z	[a-z]
⚓ Anchors	A character not in the range: a-z	[^a-z]
⊕ Meta Sequences	A character in the range: a-z or A-Z	[a-zA-Z]
✳ Quantifiers	Any single character	.
⊙ Group Constructs	Any whitespace character	\s
■ Character Classes	Any non-whitespace character	\S
	Any digit	\d

Quiz!

Matches

Regex

bobabobaboba or bobaboba

Quiz!

Matches

bobabobaboba or **bobaboba**

boba any number of times

Regex

`bobaboba(boba)?` or `(boba){2,3}`

Quiz!

Matches	Regex
bobabobaboba or bobaboba	bobaboba(boba)? or (boba){2,3}
boba any number of times	(boba)*
[any letter][any number] ex: A4	

Quiz!

Matches	Regex
bobabobaboba or bobaboba	bobaboba(boba)? or (boba){2,3}
boba any number of times	(boba)*
[any letter][any number] ex: A4	[A-Za-z]\d
example.com website.com etc.	

Quiz!

Matches	Regex
bobabobaboba or bobaboba	bobaboba(boba)? or (boba){2,3}
boba any number of times	(boba)*
[any letter][any number] ex: A4	[A-Za-z]\d
example.com website.com etc.	[a-z]+\.\com

• It's ~confusion~ time



Alright let's put glob back into our brain...

Regex vs Globbs and ranges

Regex	Glob/Range equivalent
.	?
IWant[1-7]CupsOfBoba\.please	IWant{1..7}CupsOfBoba.please
[sm1]	{s,m,1}
(this that)	{this,that}
.*	*
(ab)*	Not possible

When to use what?

Regex

- Grep, sed, vim
- Useful for parsing or validating data like an email, phone number, password, credit card num etc.
- Searching with regex is also supported in VSCode, Google Sheets, etc

Glob

- In terminal command line, used by shells for matching file and directory names using wildcards. The capabilities of globbing depend on the shell.

GREP

- Bash terminal command!
- Usage: `$ grep [flags] pattern [file...]`
- Globally search for a **regular expression** and **print** matching lines
- `$ grep 'needle' haystack.txt`
 - Searches haystack.txt for "needle"s.
- `$ grep -r 'boba' path/to/directory`
 - Searches recursively for "boba"s. 🍫
- grep is fast for theoretical reasons you will learn in the future

● *GREP - note!*

- If you have quotes in your pattern, make sure to either escape OR put the whole pattern in another quotation
 - **\$ grep "" file or \$ grep \" file**
 - **\$ grep "" file or \$ grep \' file**
- If you have spaces in your pattern, put the whole pattern in quotes!
 - **\$ grep 'free boba voucher' file**

Remember that in general, put quotes around your shell command arguments if it has quotes!

GREP

- Useful flags (usage: `$ grep [flags] pattern [file...]`)
 - `-i` (ignores case when searching)
 - `-c` (counts up the number lines that contains a match)
 - `-v` (inverse; print out lines that do NOT match)
 - `-n` (also print out the line numbers)
 - `-F` (interpret the pattern verbatim/literally, no regex involved)
 - Read the man page for more! <https://linux.die.net/man/1/grep>

GREP

- There are a lot of flags and edge cases and when-to-use-what (basic vs extended vs perl), so when stuff don't work, try googling it first!
- Example:
 - **\$ grep (this|that) file** doesn't work ...
 - Need to use egrep (or -E flag) **\$ grep -E (this|that) file**
 - Same issue when using braced quantifiers like {3}
- Example:
 - **\$ grep \d file** to look for a digit doesn't work... What should I do?
 - Use **\$grep [[:digit:]] file** or **\$grep -P '\d' file**

SED - stream editor

- Can perform find and replace on a file
- `$ sed 's/find/replace/g' path/to/file`
 - Prints result of replacement to the command line, leaving input untouched
- `$ sed -i 's/find/replace/g' path/to/file`
 - `-i` for "In place"
 - Edits the file



ZombieLab Tips

- Be careful with escaping correctly
- Don't forget to do `$ chmod +x script.sh` and add `#!/bin/bash`

