GTK CYBER

GET ON THE FAST TRACK

Regular Expressions Overview

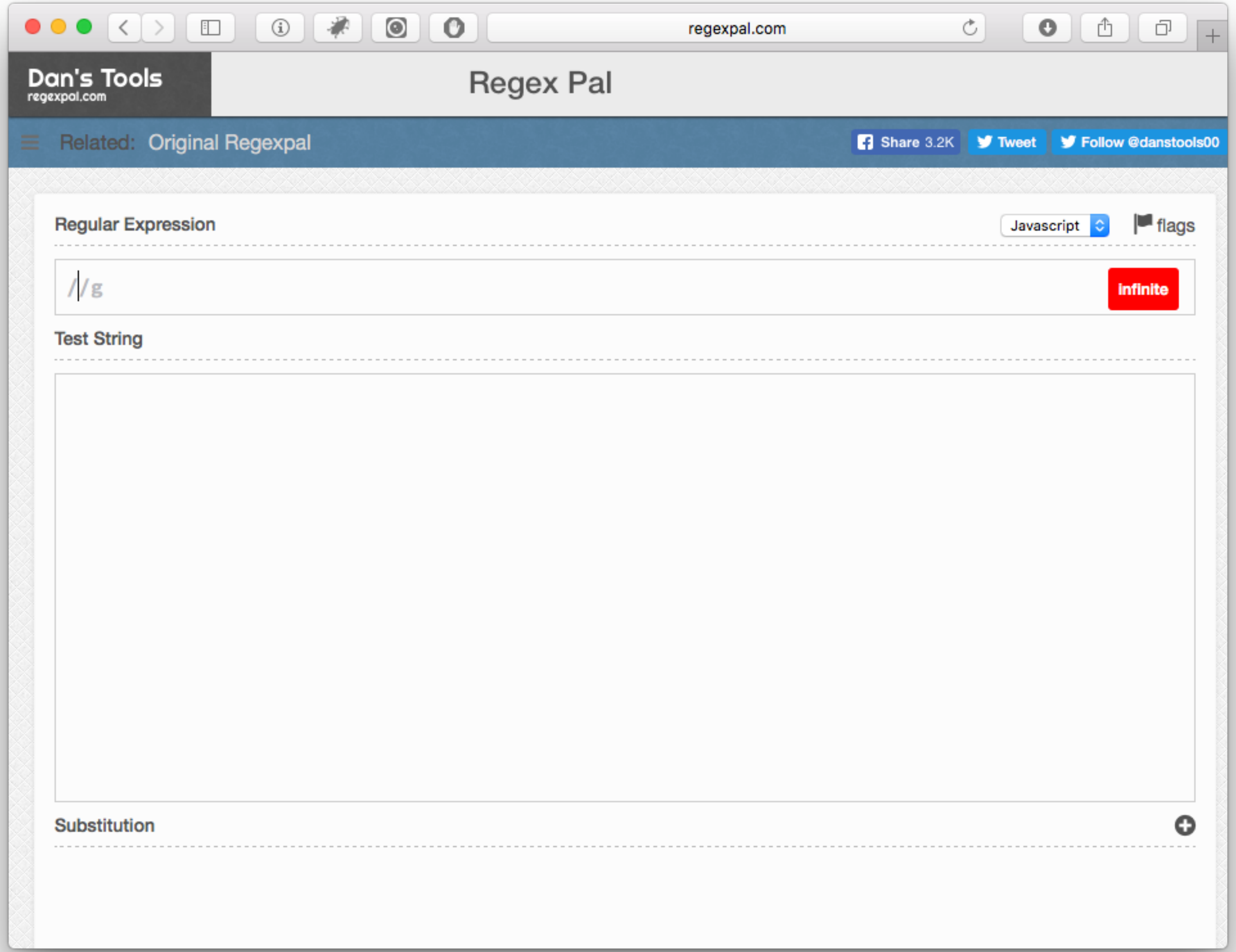
What is a Regular Expression?

A regular expression defines a pattern of characters.

Can be used for:
Validation
Data Extraction
Data Cleaning

One pattern can match one or many sets of characters

English	Pattern	Matches	Does Not Match
4 numbers in a row	\d\d\d\d or \d{4}	1234 2222 3333	a1234 AAsaaaa 123
2 numbers, a slash, two numbers, a slash, 4 numbers	\d\d/\d\d/\d\d\d\d or \d{2}/\d{2}/\d{4}	11/01/2013 10/22/2015 23/45/2222	11/1/2013 1/11/2015 aa/aa/aaaa dsifjosdijfoas


regexpal.com

Dan's Tools
regexpal.com

Regex Pal

Related: Original Regexpal

Share 3.2K
Tweet
Follow @danstools00

Regular Expression

Javascript
flags

```
//g
```

Infinite

Test String

Substitution

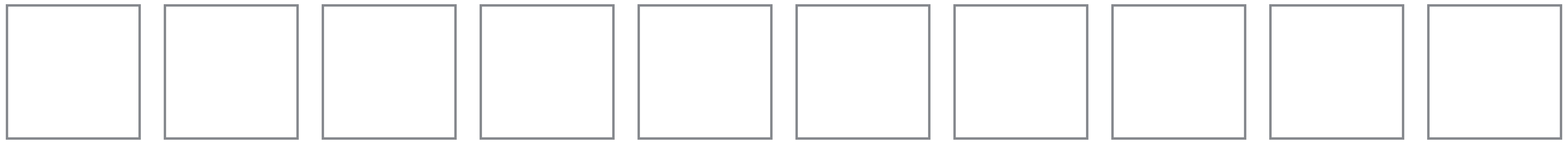
regexpal.com

Challenge 1

Let’s write a pattern that matches a date. Such as…

07/30/2016

Each Regex Character Represents a Character in a String



10 boxes for 10 characters

0	7	/	3	0	/	2	0	1	6

But this will only
match our one date.

Literal vs Special Characters

Literal Characters

0	7	/	3	0	/	2	0	1	6

Character Sets

\d	\d	/	\d	\d	/	\d	\d	\d	\d

Wildcards

.	.	/	.	.	/	.	.	.	.

Literal Characters

- Escape certain characters that have special meaning
  - \ can define a character set or escape a special character (\d or \. or \\)

Character Sets

- Can explicitly define a set of characters
  - [aeoiu]

- Can define a range of characters
  - [a-z0-9]

- Can represent a set of characters
  - \d
  - \w

- Can represent *not* characters
  - [^aeiou]
  - \D

Shorthand for Character Sets

There are shortcuts for commonly used character sets:

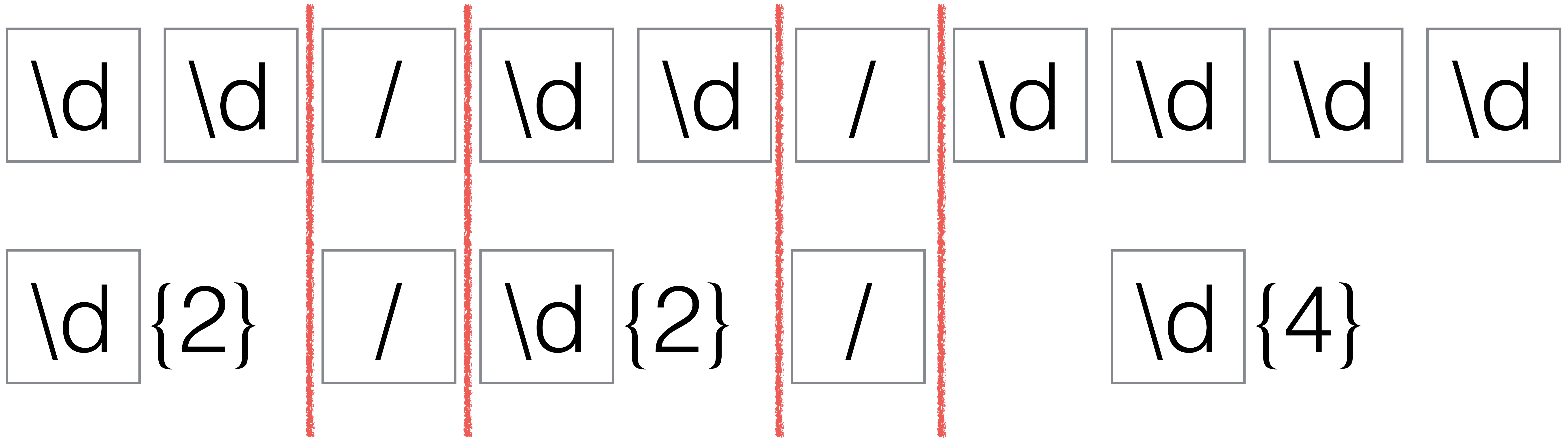
Shortcut	Definition	Example
\s	Any whitespace character	/a\sb/ matches: a b
\S	Any non-whitespace character	/a\Sb/ matches : abb
\d	Any digit	\d\d-\d matches 12-3
\D	Any non-digit	/a\Db/ matches aBc or abc
\w	Any alpha-numeric character	
\W	Any non-alpha-numeric character	

Wildcard

.

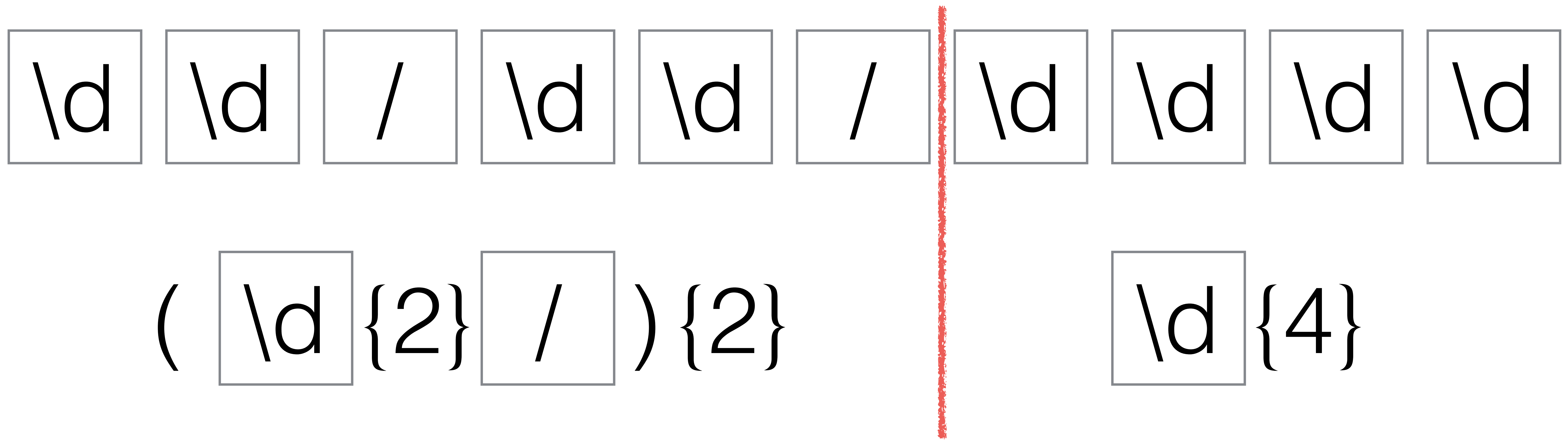
Repetition

We don't have to use 10 boxes when we have repeated characters.



Grouping

Parentheses articulate groups of characters that can be extracted or repeated.



```
\d \d / \d \d / \d \d \d \d

( \d {2} / ) {2}        \d {4}
```

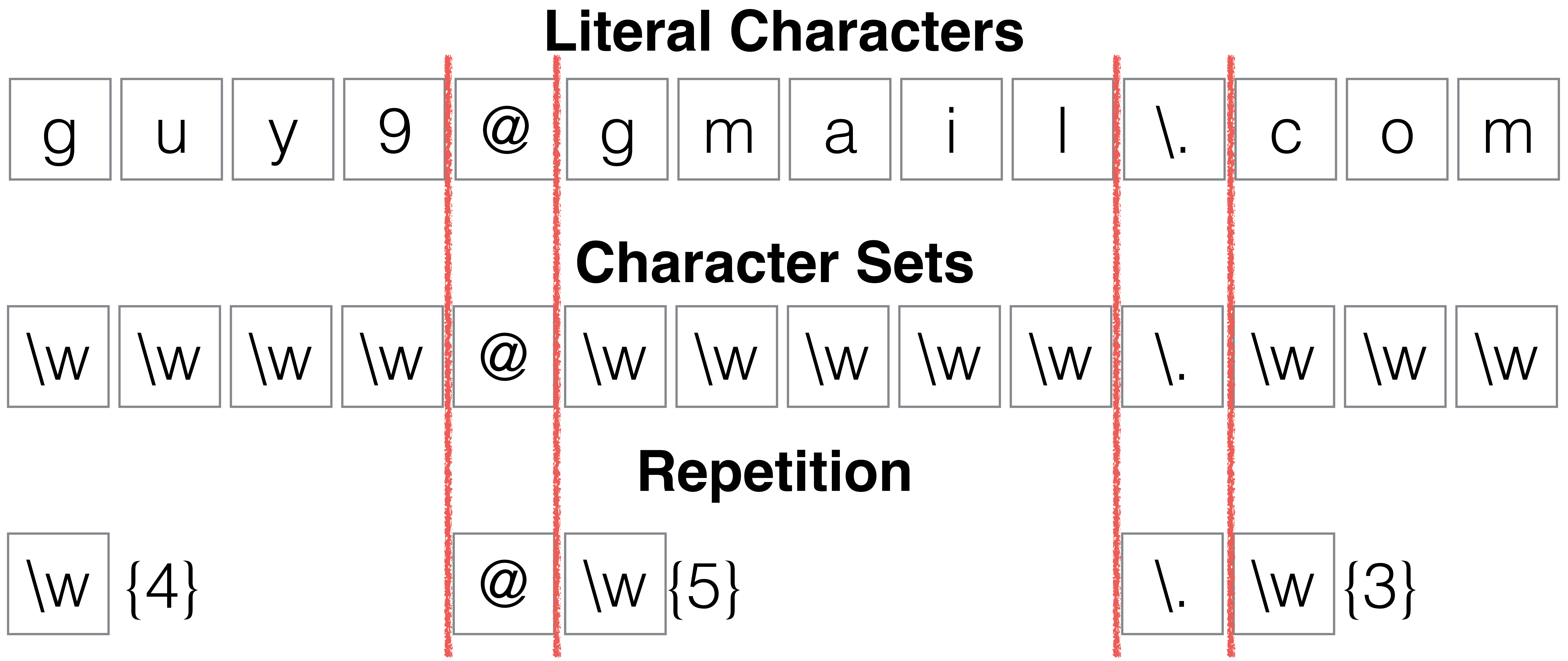
Challenge 2

Let’s write a pattern that matches an email. Such as…

guy9@gmail.com

You try! Write a pattern that uses characters sets and repetition to match the email.

Less defined repetition


Literal Characters
g u y 9 @ g m a i l \. c o m
Character Sets
\w \w \w \w @ \w \w \w \w \w \. \w \w \w
Repetition
\w {4} @ \w {5} \. \w {3}

But what if there are 6 characters in the first part of the email?

Question, Star, and Plus

? match the previous character 0 or 1 times

***** match the previous character 0 or more times

+ match the previous character 1 or more times

Greedy vs Lazy

Sometimes **.+** can match too much.

If we throw **<.+>** at *<h1>Welcome</h1>* to find opening tags, we get back the entire string when we only wanted the beginning.

Using **.+?** makes the **+** lazy, meaning it will only grab as many characters are needed in order to continue the match.

Application

```
\w {4} @ \w {5} \. \w {3}

\w  +  @ \w  +  \. \w  +
```

Exercises

Write regular expressions for the following:

- Filenames in the following format: yyyymmdd-data.xls
- IP Addresses in the format XXX.XXX.XXX.XXX
- Social Security Numbers in the format XXX-XX-XXXX
- Any 4 letter word beginning with a vowel
- Any 4 letter word with a number at the end

Regex in Python

- Python has regex support via the re module, which must be imported.
- The re module has four basic functions
  - match(<pattern>, <text>): finds the ***first*** occurrence of the pattern in the given text.
  - search(<pattern>, <text>): finds any occurrence of the pattern in the given text
  - findall(<pattern>, <text>)/finditer(<pattern>, <text>): finds all occurrences of the pattern in a given text.
  - split(<pattern>, <text>): Splits the text by the regex.
  - sub(<old>,<new>, <text>): Replaces old with the new in the given text.

Regex Option Flags

Flag	Description
re.I / re.IGNORECASE	Performs case insensitive matching
re.L / re.LOCALE	Interprets words according to locale
re.M / re.MULTILINE	Make begin consider each line
re.S / re.DOTALL	Makes a period match any character including a newline.
re.U / re.UNICODE	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X / re.VERBOSE	Allows comments in regex

Regex in Python

```
import re

text = "some 4444 text"
regex = "\d{4}"

matchObj = re.match(regex, text, re.U)

if matchObj:
   # Successful Match

else:
   #No match
```

Regex in Python

```
import re

text = "some 4444 text"
regex = "\d{4}"

matchObj = re.match(regex, text, re.U)

if matchObj:
   # Successful Match

else:
   #No match
```

Regex in Python

```
import re

text = "some 4444 text"
regex = r"\d{4}"

# Compiling Regex will improve performance
compiled_regex = re.compile(regex)
matchObj = compiled_regex.search(regex, text, re.U)

if matchObj:
   # Successful Match

else:
   #No match
```

Grouping Parentheses

- When you put parens around sections of a regex you use these to extract parts of the text
- Python uses the **`.group(n)`** function to access parts of a match
- **`group(0)`** will get you the entire matched text, whereas **`group(1)`** gets the first match.

Extracting Data with Regex

```
import re

emailAddress = "account@domain.com"
emailRegex = r"(\w+)@(\w+\.\w+)"

emailMatch = re.search(emailRegex, emailAddress)

if emailMatch:
     account = emailMatch.group(1)
     domain = emailMatch.group(2)
     completeEmail = emailMatch.group(0)
else:
   #No match
```

Back References

- Back references allow you to refer to previously matched blocks of text.
- Python uses the syntax `\1, \2, \3` in a regex to refer to previously matched groups
- Can be used in `re.sub()` to re-arrange matched parts.

In Class Exercise

Please take 20 minutes and complete

Worksheet 0.1: Regular Expressions in Python

Questions?