

# BREAKING THE ICE

WITH  
REGULAR EXPRESSIONS



# Contents

---

- 01 The String Story
- 02 Crew Emails
- 03 Confirmative
- 04 Multi-line Strings
- 05 Capture Groups



# Why Regular Expressions?

Imagine we run a website that asks people to enter their Orlando-area phone number to receive local restaurant offers.

407-555-1212

First 3 numbers have to be  
either 407 or 321 (Orlando  
area code)



# Why Regular Expressions?

Imagine we run a website that asks people to enter their Orlando-area phone number to receive local restaurant offers.

407-555-1212



Next, there could be a dash,  
but it's optional



# Why Regular Expressions?

Imagine we run a website that asks people to enter their Orlando-area phone number to receive local restaurant offers.

407-555-1212



Followed by 3 more numbers



# Why Regular Expressions?

Imagine we run a website that asks people to enter their Orlando-area phone number to receive local restaurant offers.

407-555-1212



Another optional dash



# Why Regular Expressions?

Imagine we run a website that asks people to enter their Orlando-area phone number to receive local restaurant offers.

407-555-1212

And finally, 4 more numbers

what would the code to  
validate this look like?

# Validating Input Without Regular Expressions

```
if
  ((chars[0] == 4 && chars[1] == 0 && chars[2] == 7)
   ||
   (chars[0] == 3 && chars[1] == 2 && chars[2] == 1))
  &&
  chars[3] == "-" || chars[3] == "" && ...
```

... and that's just the first few numbers!

The same validation using a regular expression



```
if (chars.match(/regular expression/))
```

FYI, the syntax can be different per language



# Understanding a Regular Expression

---

```
if (chars.match(/regular expression/))
```

## Subject String

The string where we're looking for a match is called the subject string

## aka, regex

A set of characters that represent rules for searching or matching text in a string in a concise, predictable way



# Our First Regular Expression

Does the phone number have a 407 area code?

```
if (chars.match(/407/))
```

Literally matches the characters 407

number

*subject string*

✓ ✓ ✓  
**407-555-1212**

True!



# Not Finding a Match

---

Does the phone number have a 407 area code?

```
if (chars.match(/x407/))
```

number

*subject string*

<sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup><sup>x</sup>  
**321-555-1212**

**False!**

Does the phone number have a 407 or 321 area code?



# Using the OR / Alternation Operator

Does the phone number have a 407 or 321 area code?

```
if (chars.match(/407|321/))
```

Takes left-most match first

Matches the characters on the left or right

number

*subject string*



**321-555-1212**

True!



# The Parts of a Regular Expression

---

/ 407 | 321 /

The pattern starts after a slash

The characters in the middle are called a "pattern"

The pattern ends before this slash

Writing regex is really like asking the question,  
"Does a group of characters match a specific pattern?"



# Not Just Numbers

---

Regular Expression

`/boat/`

Matches "boat"

Subject

boat



Successful match

ship



Does not match the word "ship"



# Number OR letters

Regular Expression  Matches "boat" or "ship"

`/boat|ship/`

Subject

boat 

 Successful match

ship 

 Successful match

`/407|321/`

 Just like our number pattern

**BREAKING THE ICE**  
WITH  
REGULAR EXPRESSIONS



# What Are Regular Expressions Used For?

Common uses include:

## Validations

Phone numbers  
Email  
Passwords  
Domain names



## Searching

Words in a sentence  
Unwanted characters  
Extracting sections  
Replacing/cleaning/formatting






# Section 2

## The String Story



# Our First Problem: Collecting Names

We are assembling a crew!



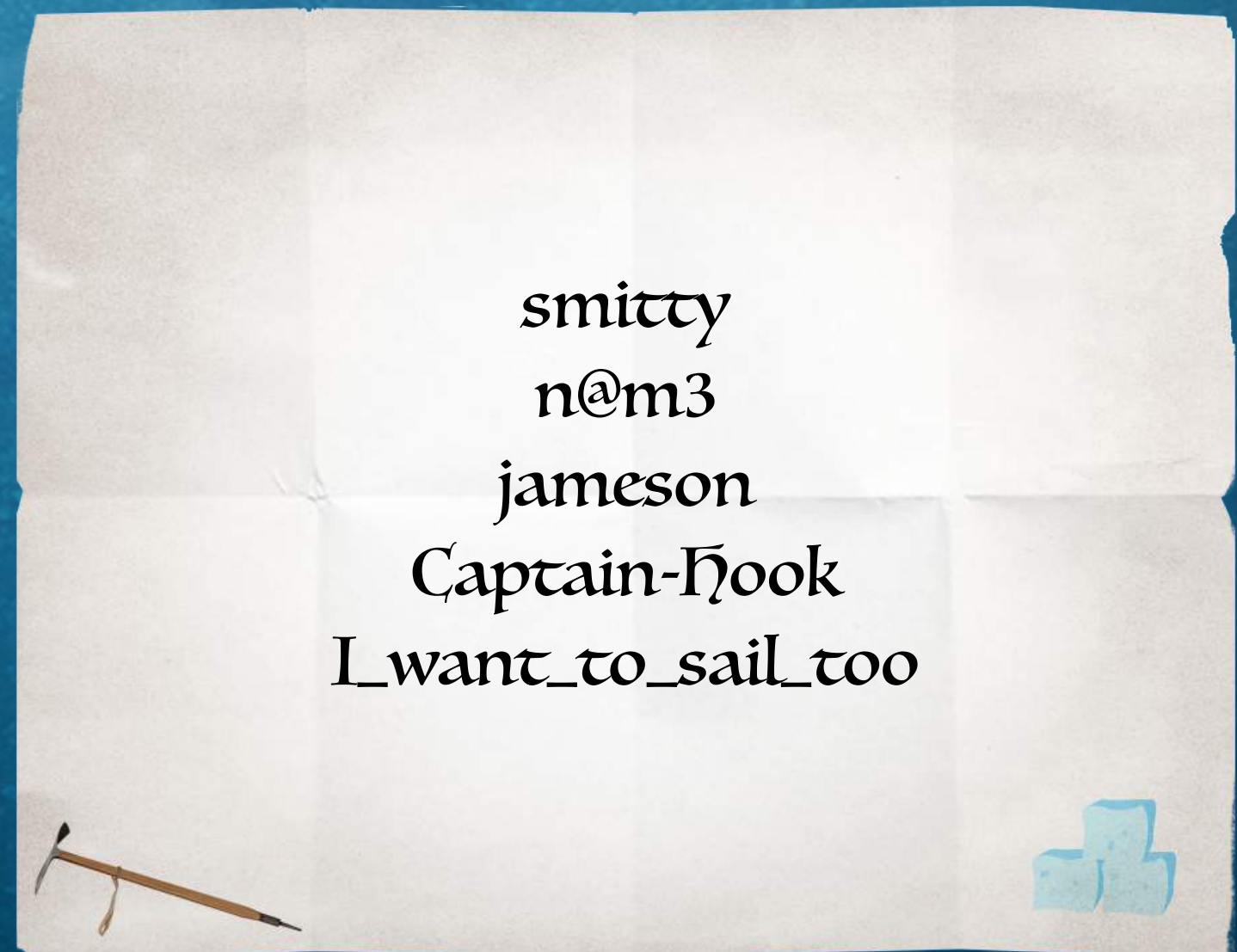
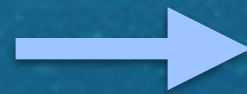
**Register**

Username

Password

☐ Remember me

**Register**





# We Need to Check Multiple Usernames

---

Regular Expression

/smitty/

Matches the name  
"smitty"

Subject

smitty





# How Can We Match on Multiple Versions of Names?

Regular Expression

/smitty|james/

Match "smitty" - if no match,  
then match "james"

Subject

smitty



james



OR / alternation operator allows  
match on "smitty" or "james"



# Problem: We Need to Repeat the “R”

Regular Expression

`/ar/`

Subject

ar



arr



arrr



This works, but is inefficient



This was a partial match



This was a partial match

Regular expressions look for matches anywhere in their subject



# Problem: Long Expression Repeating

Regular Expression

```
/ar|arr|arrrr|arrrrr/
```

This is going to grow too large

Subject

ar



arr



arrrr



There are just too many versions to account for

This works, but is inefficient



# Solution: The Plus Operator

Regular Expression

`/ar+ /`

Matches "r" 1 or more times  
until no longer matched

aka quantifier

Subject

ar



arr



arrr



Matches 1 "a" and or more "r"  
until no longer matched



This means, "Look for the previous character 1 or more times."

**BREAKING THE ICE**  
WITH  
REGULAR EXPRESSIONS



# How Can We Match on Multiple Versions of Names?

Regular Expression

`/smitty|james|ar+/  
Multiple versions of "arr"`

Subject

smitty



james



ar



arrr





# Matching on Multiple Versions of Names

## Regular Expression

```
/smitty|james|ar+/?
```

## Subject

james



jameson



Last 2 letters are not matched



This was a partial match

Regular expressions look for matches  
anywhere in their subject and take the  
left-most match first



# Matching All Characters in the Alphabet Using Ranges

## Regular Expression

```
/a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z/
```

This works, but is inefficient ❌

Square brackets  
create a character set  
aka character class

Start range here

[a-z]

End range here

## NOTES

- A range only works in a character set
- This character set represents 1 character



# Using Our Character Set in a Pattern

Regular Expression

From a-z for 1  
character

/[a-z]/

Subject

smitty



Matches the first character represented by our character set.

How can we change this to match all 6 characters?



# Matching Multiple Characters With Multiple Character Sets

Regular Expression

`/[a-z][a-z][a-z][a-z][a-z][a-z]/`

From a-z for 6 characters

Subject

smitty



james



Does not match

How can we change this pattern  
to accept any amount of letters?



# Checking for 1 Character Set Multiple Times

Regular Expression

`/[a-z]+/`

Plus operator



This means, "Look for the previous character 1 or more times."

Now they all match!

Subject

smitty



james



jameson



ar



arr



arrr





# Problem: Capital Letters

---

Regular Expression

```
/[a-z]+/
```

Subject

Blackbeard



Problem: Our pattern is ignoring capital letters



# Matching Uppercase and Lowercase Characters

Regular Expression

From a-z, A-Z for 1 or more times

`/[a-zA-Z]+/`

Subject

Blackbeard



Now we're matching capital letters!



# Simplifying Patterns With a Casing Modifier

Regular Expression

```
/[a-z]+/i
```

Letters after final slash  
are called “modifiers”

The **i** modifier means “case insensitive,” which will match uppercase and lowercase characters.

Subject

Blackbeard





# Problem: Names With 2 Words Not Matching

Regular Expression

```
/[a-z]+/i
```

Subject

Captain hook



Problem: Not matching whitespace



# How Can We Match Whitespace?

using a literal space is hard to read - you might miss it!

/Captain hook/



/Captain\shook/



\s

is the same as saying  
"a whitespace character"

Whitespace can include:

- Spaces
- Tabs
- New lines



# Matching Spaces in a Character Set

Regular Expression

`/[a-z\s]+/i`

`\s`

is the same as saying  
“a whitespace character”

Subject

Captain hook



NOTE: The order of characters doesn't matter here!

`[\sa-z]+`

is the same thing!



# Problem: How Can We Match Numbers?

Regular Expression

Does not account for numbers

```
/[a-z\s]+/i
```

Subject

Captain hook



Long John the 3rd



Problem: we do not match any numbers



# Matching Numbers in a Range

## Regular Expression

`/[a-z0-9\s]+/i`

From a-z, A-Z, 0-9, whitespace  
for 1 or more times

using a range to match numbers

## Subject

Captain hook

Long John the 3rd

we now match on numbers 0-9



# Refactoring With the Word Metacharacter

`\w` is the same as `[a-zA-Z0-9]`

Regular Expression

`/[a-z0-9\s]+/i` 

These have the same result,  
but the shortcut is easier to read.

`/[\w\s]+/` 

Also includes underscore