

DEALING WITH /REGEXION/

DETROIT SOFTWARE GUILD
MIKE SCHUTTE
AUGUST 20, 2019

> @TMIKESCHU (☕️🎤👞)

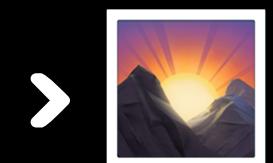
> @TMIKESCHU (☕️🎤👞)

> 🏠

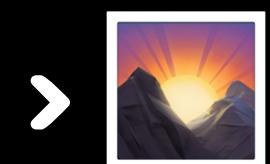
> @TMIKESCHU (☕️🎤👞)



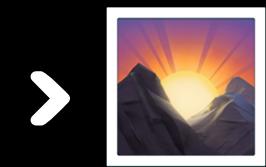
> @TMIKESCHU (☕️🎤👞)



> @TMIKESCHU (☕️🎤👞)



> @TMIKESCHU (☕️🎤👞)



> **TED**

FRIENDGINEERS WILL BE ABLE TO...

WBAI

> FEEL AT PIECE WITH THE CRAZINESS OF REGEX

- › FEEL AT PIECE WITH THE CRAZINESS OF REGEX
- › HAVE A FEW STRATEGIES FOR DECIDING WHEN TO USE REGEX

- › FEEL AT PIECE WITH THE CRAZINESS OF REGEX
- › HAVE A FEW STRATEGIES FOR DECIDING WHEN TO USE REGEX
- › BE LIKE "WOW CAPTURE GROUPS ARE AMAZING"

ROADMAP

ROADMAP

> MY SOAPBOX: WHY REGEX SHOULD BE MESSY

ROADMAP

- > MY SOAPBOX: WHY REGEX SHOULD BE MESSY
- > BRIEF OVERVIEW OF REGULAR EXPRESSIONS

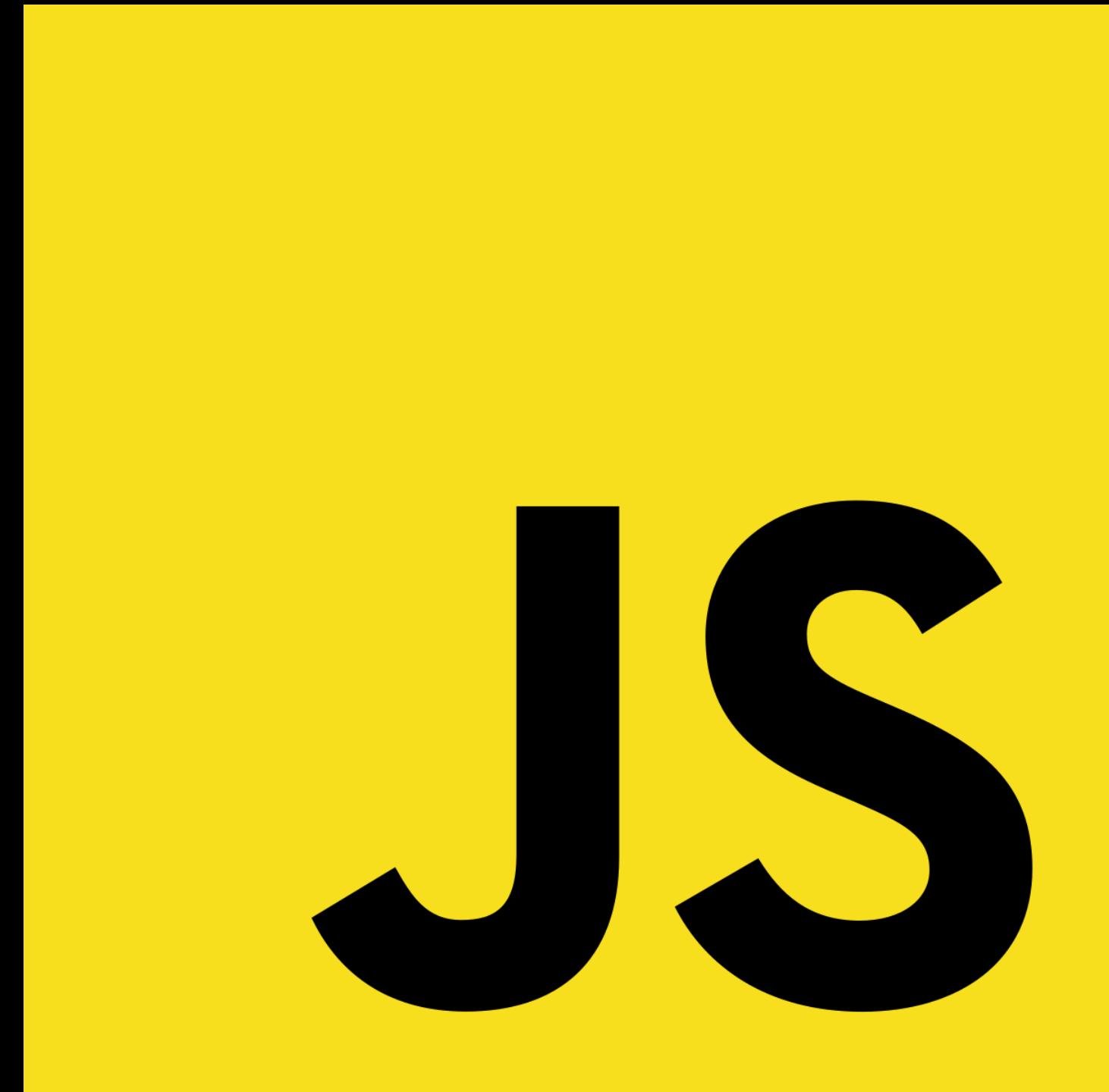
ROADMAP

- > MY SOAPBOX: WHY REGEX SHOULD BE MESSY
- > BRIEF OVERVIEW OF REGULAR EXPRESSIONS
- > WHEN TO USE REGEX

ROADMAP

- > MY SOAPBOX: WHY REGEX SHOULD BE MESSY
- > BRIEF OVERVIEW OF REGULAR EXPRESSIONS
- > WHEN TO USE REGEX
- > CAPTURE GROUPS

DISCLAIMER



A black and white photograph of a person running on a sandy beach. The runner is positioned on the right side of the frame, moving towards the left. The background shows the ocean waves crashing onto the shore under a hazy sky.

ME USING REGEX

HOW TO DEAL WITH /REGEXION/

STRINGS AND LANGUAGE

STRINGS AND LANGUAGE

> TYPOS

STRINGS AND LANGUAGE

- > TYPOS
- > DUPLICATION

STRINGS AND LANGUAGE

- > TYPOS
- > DUPLICATION
- > PATTERNS

STRINGS AND LANGUAGE

- > TYPOS
- > DUPLICATION
- > PATTERNS
- > FORMATS

STRINGS AND LANGUAGE

- > TYPOS
- > DUPLICATION
- > PATTERNS
- > FORMATS
- > REPETITION

ACCEPT IT

ACCEPT IT

> REGEX IS MESSY

ACCEPT IT

- > REGEX IS MESSY
- > BECAUSE STRING DATA IS MESSY

ACCEPT IT

- > REGEX IS MESSY
- > BECAUSE STRING DATA IS MESSY
- > BECAUSE LANGUAGE IS MESSY

REGULAR EXPRESSIONS: AN OVERVIEW

REGULAR EXPRESSIONS: AN OVERVIEW

> BORN IN 1951

REGULAR EXPRESSIONS: AN OVERVIEW

- > BORN IN 1951
- > POPULARIZED IN 1968

REGULAR EXPRESSIONS: AN OVERVIEW

- > BORN IN 1951
- > POPULARIZED IN 1968
- > TEXT EDITOR SEARCH

REGULAR EXPRESSIONS: AN OVERVIEW

- > BORN IN 1951
- > POPULARIZED IN 1968
 - > TEXT EDITOR SEARCH
 - > LEXICAL ANALYSIS

REGULAR EXPRESSIONS: AN OVERVIEW

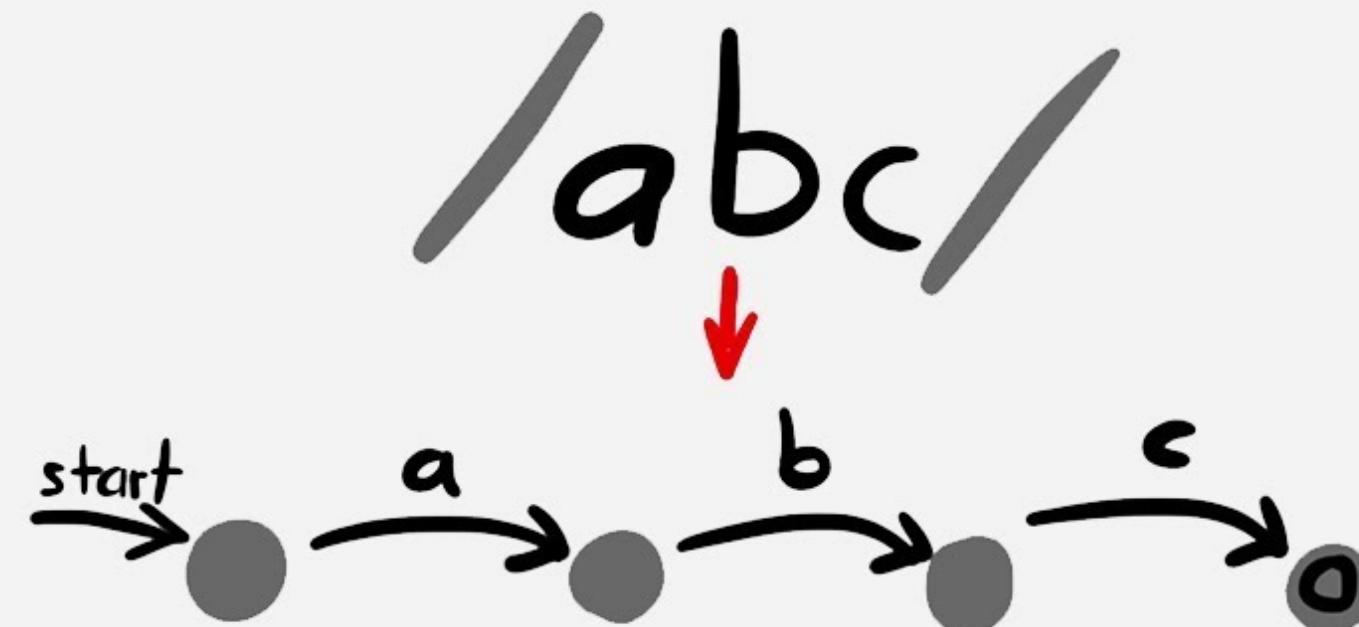
- > BORN IN 1951
- > POPULARIZED IN 1968
 - > TEXT EDITOR SEARCH
 - > LEXICAL ANALYSIS
- > POSIX. PERL. PCRE

REGULAR EXPRESSIONS: AN OVERVIEW

- > BORN IN 1951
- > POPULARIZED IN 1968
 - > TEXT EDITOR SEARCH
 - > LEXICAL ANALYSIS
- > POSIX. PERL. PCRE
- > FINITE STATE MACHINE

Regex

short



[HTTPS://WWW.YOUTUBE.COM/WATCH?V=HPRXXJHQVFQ](https://www.youtube.com/watch?v=HPRXXJHQVFQ)

CODE THAT
WANTS TO BE
REGEXED

**IF YOU ASK MORE THAN
ONE QUESTION ABOUT A
STRING...**

DITCH && AND || FOR //

```
- someString.startsWith(":") &&
-   someString.split("").some(char => Boolean(Number(char)))
+ /^.*\d/.test(someString)

- someString.includes("someWord") || someString.includes("someOtherWord");
+ /someWord|someOtherWord/.test(someString)
```

(ARRAY OF CHARACTERS).CONTEXT < (STRING).CONTEXT

**REGEX FORCES YOU TO CONSIDER THE
STRING AS (MORE OF) A WHOLE**

METHODS TO USE

> CHANGE FORMAT

- > CHANGE FORMAT
 - > `STRING.prototype.replace (=> STRING)`

- > CHANGE FORMAT
 - > `STRING.PROTOTYPE.REPLACE (=> STRING)`
- > GET SUBSTRING(S)

- > CHANGE FORMAT
 - > `STRING.PROTOTYPE.REPLACE` (\Rightarrow STRING)
- > GET SUBSTRING(S)
 - > `STRING.PROTOTYPE.MATCH` (\Rightarrow ARRAY)

- > CHANGE FORMAT
 - > `STRING.prototype.replace` (\Rightarrow STRING)
- > GET SUBSTRING(S)
 - > `STRING.prototype.match` (\Rightarrow ARRAY)
- > ASSERT STRING QUALITIES

- > CHANGE FORMAT
 - > `STRING.PROTOTYPE.REPLACE` (\Rightarrow STRING)
- > GET SUBSTRING(S)
 - > `STRING.PROTOTYPE.MATCH` (\Rightarrow ARRAY)
- > ASSERT STRING QUALITIES
 - > `REGEX.PROTOTYPE.TEST` (\Rightarrow BOOLEAN)

- > CHANGE FORMAT
 - > `STRING.PROTOTYPE.REPLACE` (\Rightarrow STRING)
- > GET SUBSTRING(S)
 - > `STRING.PROTOTYPE.MATCH` (\Rightarrow ARRAY)
- > ASSERT STRING QUALITIES
 - > `REGEX.PROTOTYPE.TEST` (\Rightarrow BOOLEAN)
- > STATEFUL SEARCH

- > CHANGE FORMAT
 - > `STRING.PROTOTYPE.REPLACE` (\Rightarrow STRING)
- > GET SUBSTRING(S)
 - > `STRING.PROTOTYPE.MATCH` (\Rightarrow ARRAY)
- > ASSERT STRING QUALITIES
 - > `REGEX.PROTOTYPE.TEST` (\Rightarrow BOOLEAN)
- > STATEFUL SEARCH
 - > `REGEX.PROTOTYPE.EXEC` (\Rightarrow ARRAY)

YOU KNOW WHAT THOSE PARENTHESES IN REGULAR EXPRESSIONS ARE. RIGHT?

`/(\d+)/;`



CAPTURE GROUPS: KEEP IT TOGETHER

/()/*

> IS FAMILIARITY WORTH RIGIDITY?

- > IS FAMILIARITY WORTH RIGIDITY?
- > IS DIFFICULTY WORTH FLEXIBILITY?

IS DIFFICULTY WORTH FLEXIBILITY? 

TASK

CREATE A FUNCTION THAT

TASK

CREATE A FUNCTION THAT

> TAKES IN A NAME IN First Last FORMAT

TASK

CREATE A FUNCTION THAT

- › TAKES IN A NAME IN First Last FORMAT
- › AND RETURNS THE NAME IN Last, First FORMAT

```
const albus = "Albus Dumbledore";  
  
function lastFirst(name) {  
    // TODO  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Albus"
```

APPROACH #1: SPLIT

```
function lastFirst(name) {  
    return name  
        .split(" ")  
        .reverse()  
        .join(", ");  
}  
  
console.log(lastFirst(albus)); // => "Dumbledore, Albus"
```

APPROACH #2: REGEX

```
function lastFirst(name) {  
  const reFirstLast = /(\w+)\s(\w+)/;  
  return name.replace(reFirstLast, "$2, $1");  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Albus"
```

-

```
someString.replace(/(cats)(dogs)/, (full, group1, group2) => {  
    // do stuff with the groups  
});
```

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace#Specifying_a_function_as_a_parameter

 **CHANGE ALERT** 

```
const albus = "Albus Percival Dumbledore";  
fullName(albus); // => "Dumbledore, Albus Percival"
```

> ..MIDDLE NAMES TOO.

```
const albus = "Albus Percival Dumbledore";  
  
fullName(albus); // => "Dumbledore, Albus Percival"
```

APPROACH #1: SPLIT

```
function lastFirst(name) {  
  return name  
    .split(" ")  
    .reverse()  
    .join(", ");  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Percival, Albus"
```



```
function lastFirst(rawName) {  
  const names = rawName.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  return `${last}, ${rest.join(" ")}`;  
}  
  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival"
```

APPROACH #2: REGEX

```
function lastFirst(name) {  
  const reFirstLast = /(\w+)\s(\w+)/;  
  return name.replace(reFirstLast, "$2, $1");  
}  
console.log(lastFirst(albus)); // => "Percival, Albus Dumbledore"
```



```
function lastFirst(name) {  
  const reFirstLast = /(\w+\s*\w*)\s+(\w+)/;  
  return name.replace(reFirstLast, "$2, $1");  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival"
```

- $/(\backslash w^+) \backslash s(\backslash w^+)/;$
- + $/(\backslash w^+ \backslash s^* \backslash w^*) \backslash s(\backslash w^+)/;$

COMPARISON

SPLIT

COMPARISON

SPLIT

> CALCULATING INDICES

COMPARISON

SPLIT

- > CALCULATING INDICES
- > ACCOMMODATING FOR ZERO-BASED COUNTING

COMPARISON

SPLIT

- > CALCULATING INDICES
- > ACCOMMODATING FOR ZERO-BASED COUNTING
- > ARRAY.PROTOTYPE METHODS

COMPARISON

SPLIT

- > CALCULATING INDICES
- > ACCOMMODATING FOR ZERO-BASED COUNTING
- > ARRAY.PROTOTYPE METHODS
- > STRING INTERPOLATION

COMPARISON

REGEX

COMPARISON

REGEX

> PATTERNS

COMPARISON

REGEX

- > PATTERNS
- > THERE IS A FIRST BIT

COMPARISON

REGEX

- > PATTERNS
- > THERE IS A FIRST BIT
- > AND A LAST BIT

COMPARISON REGEX

- > PATTERNS
- > THERE IS A FIRST BIT
- > AND A LAST BIT
- > AND SOMETIMES EXTRA MIDDLE BITS IN THE FIRST BIT

THAT IS MESSY

THAT IS OKAY

THAT IS AWESOME

 **CHANGE ALERT** 

```
const albus = "Albus Percival Wulfric Brian Dumbledore";
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian"
```

> ...MIDDLE NAMES TOO

```
const albus = "Albus Percival Wulfric Brian Dumbledore";
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian"
```

- > ...MIDDLE NAMES TOO
- > ...MULTIPLE MIDDLE NAMES

```
const albus = "Albus Percival Wulfric Brian Dumbledore";
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian"
```

APPROACH #1: SPLIT

```
function lastFirst(rawName) {  
  const names = rawName.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  return `${last}, ${rest.join(" ")}`;  
}  
  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival Wulfric Brian"
```



APPROACH #2: REGEX

```
function lastFirst(name) {  
  const reFirstLast = /(\w+\s*\w*)\s(\w+)/;  
  return name.replace(reFirstLast, "$2, $1");  
}  
console.log(lastFirst(albus)); // => "Wulfric, Albus Percival Brian Dumbledore"
```



```
- const reFirstLast = /(\w+\s*\w*)\s(\w+)/;
+ const reFirstLast = /(\w+(\s\w+)*)\s(\w+)/;
```

```
function lastFirst(name) {  
  const reFirstLast = /(\w+(\s\w+)* )\s(\w+)/;  
  return name.replace(reFirstLast, "$3, $1");  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival Wulfric Brian"
```

 **CHANGE ALERT** 

```
const albus = "Albus Percival Wulfric Brian Dumbledore, Jr.";
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian, Jr."
```

> ..MIDDLE NAMES TOO

```
const albus = "Albus Percival Wulfric Brian Dumbledore, Jr.";  
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian, Jr."
```

- > ..MIDDLE NAMES TOO
- > ..MULTIPLE MIDDLE NAMES

```
const albus = "Albus Percival Wulfric Brian Dumbledore, Jr.";  
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian, Jr."
```

- > ...MIDDLE NAMES TOO
- > ...MULTIPLE MIDDLE NAMES
- > ...SUFFIXES TOO

```
const albus = "Albus Percival Wulfric Brian Dumbledore, Jr.";  
lastFirst(albus); // => "Dumbledore, Albus Percival Wulfric Brian, Jr."
```



APPROACH #1: SPLIT

```
function lastFirst(rawName) {  
  const names = rawName.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  return `${last}, ${rest.join(" ")}`;  
}  
  
console.log(lastFirst(albus)); // => "Jr., Albus Percival Wulfric Brian Dumbledore,"
```



```
function lastFirst(rawName) {  
  const [name, suffix] = rawName.split(", ");  
  const names = name.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  const output = `${last}, ${rest.join(" ")}`;  
  if (suffix) {  
    return `${output}, ${suffix}`;  
  }  
  return output;  
}  
  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival, Jr."
```

APPROACH #2: REGEX

```
function lastFirst(name) {  
  const reFirstLast = /(\w+(\s\w+)* )\s(\w+)/;  
  return name.replace(reFirstLast, "$3, $1");  
}  
console.log(lastFirst(albus)); // => "Dumbledore, Albus Percival, Jr."
```





 **CHANGE ALERT** 

```
const albus = "Albus";
lastFirst(albus); // => "Albus"
```

> ...MIDDLE NAMES TOO

```
const albus = "Albus";
lastFirst(albus); // => "Albus"
```

- > ...MIDDLE NAMES TOO
- > ...MULTIPLE MIDDLE NAMES

```
const albus = "Albus";
lastFirst(albus); // => "Albus"
```

- > ...MIDDLE NAMES TOO
- > ...MULTIPLE MIDDLE NAMES
- > ...SUFFIXES TOO

```
const albus = "Albus";
lastFirst(albus); // => "Albus"
```

- > ...MIDDLE NAMES TOO
- > ...MULTIPLE MIDDLE NAMES
- > ...SUFFIXES TOO
- > ...JUST FIRST NAME IS OKAY ^_^(ツ)_/^-^

```
const albus = "Albus";
lastFirst(albus); // => "Albus"
```

APPROACH 1: SPLIT

```
function lastFirst(rawName) {  
  const [name, suffix] = rawName.split(", ");  
  const names = name.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  const output = `${last}, ${rest.join(" ")}`;  
  if (suffix) {  
    return `${output}, ${suffix}`;  
  }  
  return output;  
}  
  
console.log(lastFirst(albus)); // => "Albus,"
```



```
function lastFirst(rawName) {  
  const [name, suffix] = rawName.split(", ");  
  const names = name.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex);  
  
  const output = `${last}, ${rest.join(" ")}`;  
  if (suffix) {  
    return `${output}, ${suffix}`;  
  }  
  if (output.endsWith(",")) {  
    return output.slice(0, output.length - 1);  
  }  
  return output;  
}
```



```
const output = `${last}, ${rest.join(" ")}`;
if (suffix) {
  return `${output}, ${suffix}`;
}
+ if (output.endsWith(",,")) {
+   return output.slice(0, output.length - 1);
+
return output
```

```
function lastFirst(rawName) {  
  const [name, suffix] = rawName.split(", ");  
  const names = name.split(" ");  
  const maxIndex = names.length - 1;  
  const last = names[maxIndex];  
  const rest = names.slice(0, maxIndex).join(" ");  
  
  let output = last;  
  if (Boolean(rest)) {  
    output += `, ${rest}`;  
  }  
  
  if (suffix) {  
    output += `, ${suffix}`;  
  }  
  return output;  
}
```



```
- const output = `${last}, ${rest.join(" ")}`;
- if (suffix) {
-   return `${output}, ${suffix}`;
- }
- if (output.lastIndexOf(",") === output.length) {
-   return output.slice(0, output.length - 1);
- }
+ let output = last;
+ if (Boolean(rest)) {
+   output += `, ${rest}`;
+ }

+ if (suffix) {
+   output += `, ${suffix}`;
+ }
return output;
```

APPROACH #2: REGEX

```
function lastFirst(name) {  
  const reFirstLast = /(\w+(\s\w+)* )\s(\w+)/;  
  return name.replace(reFirstLast, "$3, $1");  
}  
  
console.log(lastFirst(albus)); // => "Albus"
```



70/81 - @TMIKESCHU



```
const reFirstLast = /( \w+ (\s\w+)* )\s(\ \w+ )/;
```

**FLEXIBILITY >
READABILITY**

...ABOUT "READABILITY"

...ABOUT "READABILITY"

> IS GERMAN READABLE?

REVIEW

REVIEW

> DESIGNED FOR ANALYZING AND SEARCHING TEXT

REVIEW

- > DESIGNED FOR ANALYZING AND SEARCHING TEXT
- > (REGEX :: STRING DATA :: LANGUAGE) == MESSY

REVIEW

- > DESIGNED FOR ANALYZING AND SEARCHING TEXT
- > (REGEX :: STRING DATA :: LANGUAGE) == MESSY
- > MORE THAN ONE QUESTION ABOUT YOUR STRING...

REVIEW

- > DESIGNED FOR ANALYZING AND SEARCHING TEXT
- > (REGEX :: STRING DATA :: LANGUAGE) == MESSY
- > MORE THAN ONE QUESTION ABOUT YOUR STRING...
- > CAPTURE GROUPS ARE GREAT FOR MANIPULATING SUBSTRINGS

GO FORTH AND PARSE YOUR STRINGS!

EMBRACE THE /PAIN/!

DON'T FEAR /REGEXION/!

RESPONSE /GRACEFULLY/ TO CHANGE 😊

THANK YOU!

RESOURCES

RESOURCES

> REPL-ish TOOL: [HTTPS://REGEXR.COM/](https://regexr.com/)

RESOURCES

- > REPL-ish TOOL: [HTTPS://REGEXR.COM/](https://regexr.com/)
- > CHEAT SHEET: [://WWW.REXEGG.COM/REGEX-QUICKSTART.HTML](http://www.rexegg.com/regex-quickstart.html)

RESOURCES

- > REPL-ish TOOL: [HTTPS://REGEXR.COM/](https://regexr.com/)
- > CHEAT SHEET: [://WWW.REXEGG.COM/REGEX-QUICKSTART.HTML](http://www.rexegg.com/regex-quickstart.html)
- > WIKI: [HTTPS://EN.WIKIPEDIA.ORG/WIKI/REGULAR_EXPRESSION](https://en.wikipedia.org/wiki/Regular_expression)

RESOURCES

- > REPL-ish TOOL: [HTTPS://REGEXR.COM/](https://regexr.com/)
- > CHEAT SHEET: [://WWW.REXEGG.COM/REGEX-QUICKSTART.HTML](http://www.rexegg.com/regex-quickstart.html)
- > WIKI: [HTTPS://EN.WIKIPEDIA.ORG/WIKI/REGULAR_EXPRESSION](https://en.wikipedia.org/wiki/Regular_expression)
- > NAMED CAPTURE GROUPS: [HTTPS://GITHUB.COM/TC39/PROPOSAL-REGEXP-NAMED-GROUPS](https://github.com/tc39/proposal-regexp-named-groups)