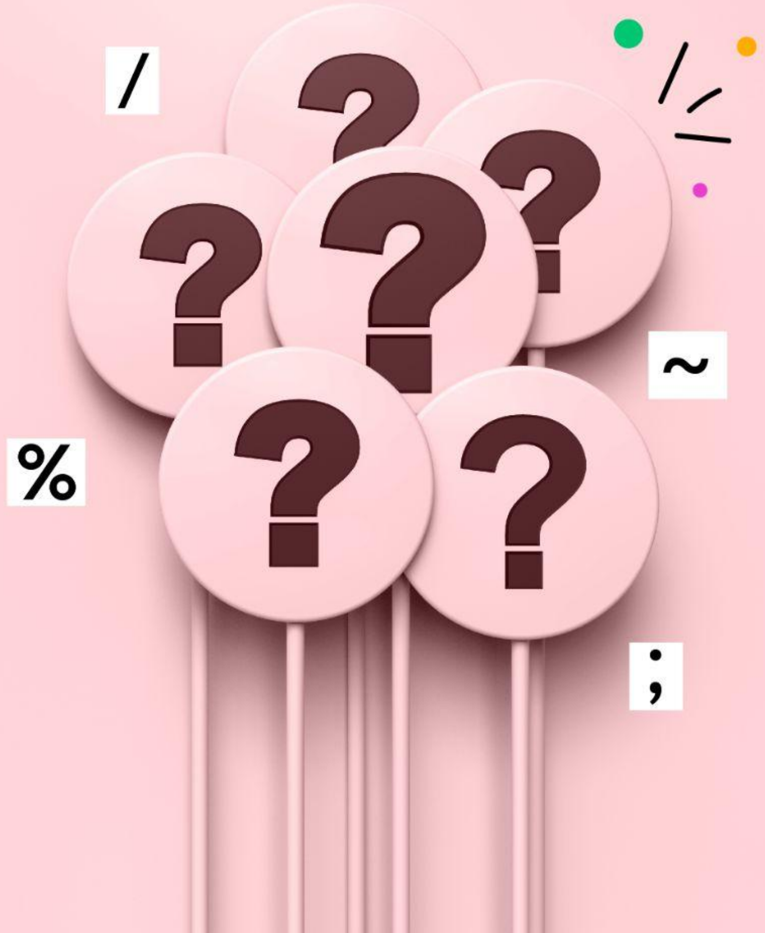
platform.sh

# RegEx: demystifying the hieroglyphics



SymfonyCon
BRUSSELS 2023

## Follow along

- https://github.com/mupsi/regex-presentation
- https://regex101.com/

## Disclaimers

- You don’t need to be a maths geek to understand (I’m not!)
- Practice makes perfect, so try it yourself at home!

## An introduction to Regular Expressions

## Regular Expressions

- Originated in 1951 by mathematician Stephen Cole Kleene

## Regular Expressions

- **Originated in 1951 by mathematician Stephen Cole Kleene**
- **Describe regular languages in a formal language theory**

## Regular Expressions

- Originated in 1951 by mathematician Stephen Cole Kleene
- Describe regular languages in a formal language theory
- Are an algebraic way to describe languages

```
/ ^(?=(?!(.)\1)([^\DO:105-93+30])(?-1)(?<!\d(?<=(?![5-90-3])\d))).[^\WHY?]$
```

## Regular Expressions

"...everything is essentially a character, and we are writing patterns to match a specific sequence of characters."
**- Faisal Shahbaz**

**Sequence of characters that specifies a search pattern in text**

## What Regular Expressions are not

- **A programming language**

## What Regular Expressions are not

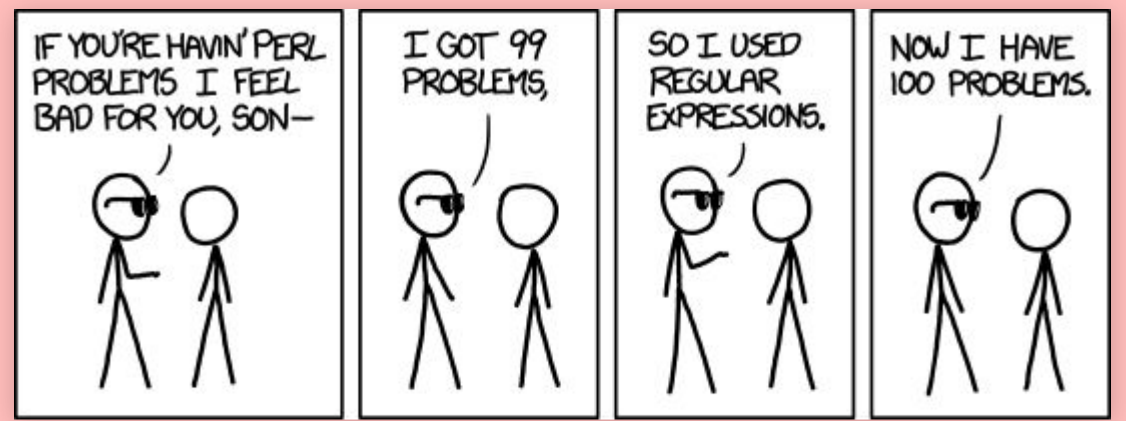
- A programming language
- Unlearnable

“There is nothing regular about Regular Expressions.”

— A former attendee

## What Regular Expressions are not

- A programming language
- Unlearnable
- The solution to every problem


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.

Source: https://xkcd.com/1171/

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

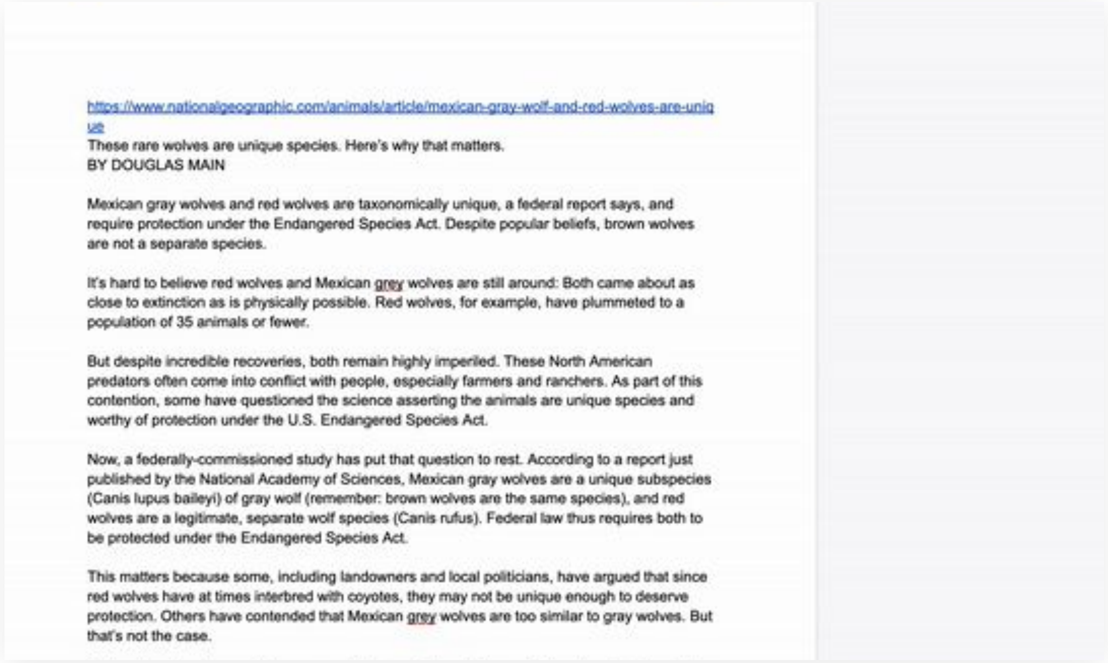
Source:https://xkcd.com/208/

## So what can I use them for?

## Regular Expression uses

- **Finding text**

# Reg Expressions in Word/Google Docs



https://www.nationalgeographic.com/animals/article/mexican-gray-wolf-and-red-wolves-are-unique
These rare wolves are unique species. Here's why that matters.
BY DOUGLAS MAIN

Mexican gray wolves and red wolves are taxonomically unique, a federal report says, and require protection under the Endangered Species Act. Despite popular beliefs, brown wolves are not a separate species.

It's hard to believe red wolves and Mexican grey wolves are still around: Both came about as close to extinction as is physically possible. Red wolves, for example, have plummeted to a population of 35 animals or fewer.

But despite incredible recoveries, both remain highly imperiled. These North American predators often come into conflict with people, especially farmers and ranchers. As part of this contention, some have questioned the science asserting the animals are unique species and worthy of protection under the U.S. Endangered Species Act.

Now, a federally-commissioned study has put that question to rest. According to a report just published by the National Academy of Sciences, Mexican gray wolves are a unique subspecies (Canis lupus baileyi) of gray wolf (remember: brown wolves are the same species), and red wolves are a legitimate, separate wolf species (Canis rufus). Federal law thus requires both to be protected under the Endangered Species Act.

This matters because some, including landowners and local politicians, have argued that since red wolves have at times interbred with coyotes, they may not be unique enough to deserve protection. Others have contended that Mexican grey wolves are too similar to gray wolves. But that's not the case.

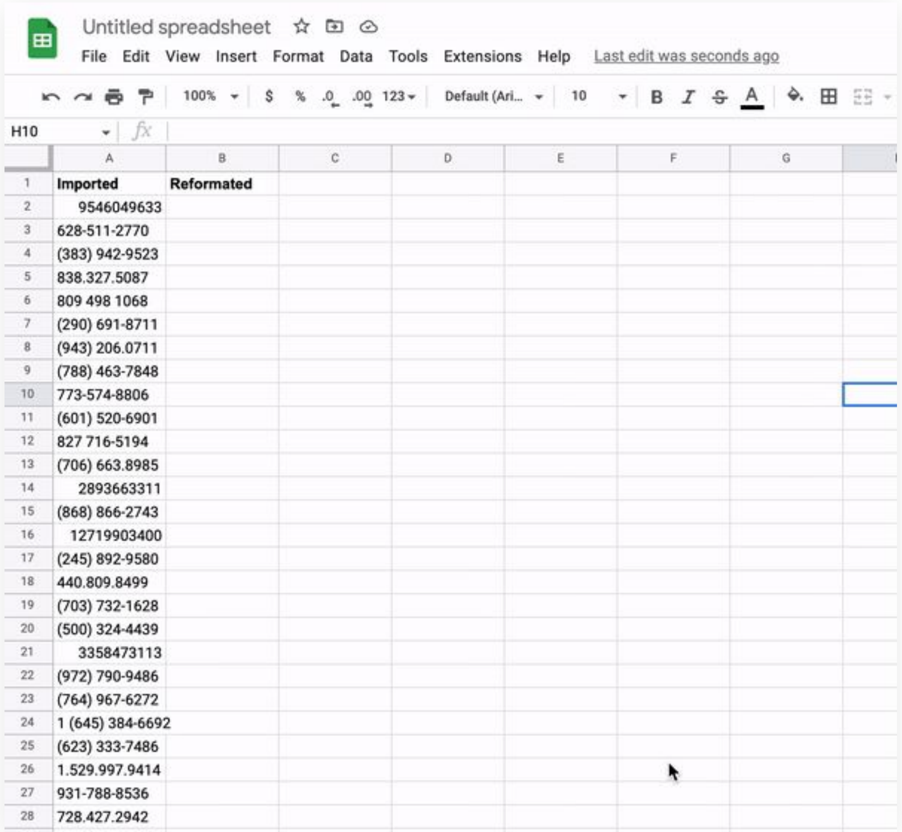
## Regular Expression uses

- Finding text
- Validating text

## Regular Expression uses

- Finding text
- Validating text
- String manipulation

# Reg Expressions in Excel/Google Sheets



Untitled spreadsheet
File Edit View Insert Format Data Tools Extensions Help
Last edit was seconds ago
100%
Default (Ari...
10
H10

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **Imported** | **Reformatted** | | | | | |
| 2 | 9546049633 | | | | | | |
| 3 | 628-511-2770 | | | | | | |
| 4 | (383) 942-9523 | | | | | | |
| 5 | 838.327.5087 | | | | | | |
| 6 | 809 498 1068 | | | | | | |
| 7 | (290) 691-8711 | | | | | | |
| 8 | (943) 206.0711 | | | | | | |
| 9 | (788) 463-7848 | | | | | | |
| 10 | 773-574-8806 | | | | | | |
| 11 | (601) 520-6901 | | | | | | |
| 12 | 827 716-5194 | | | | | | |
| 13 | (706) 663.8985 | | | | | | |
| 14 | 2893663311 | | | | | | |
| 15 | (868) 866-2743 | | | | | | |
| 16 | 12719903400 | | | | | | |
| 17 | (245) 892-9580 | | | | | | |
| 18 | 440.809.8499 | | | | | | |
| 19 | (703) 732-1628 | | | | | | |
| 20 | (500) 324-4439 | | | | | | |
| 21 | 3358473113 | | | | | | |
| 22 | (972) 790-9486 | | | | | | |
| 23 | (764) 967-6272 | | | | | | |
| 24 | 1 (645) 384-6692 | | | | | | |
| 25 | (623) 333-7486 | | | | | | |
| 26 | 1.529.997.9414 | | | | | | |
| 27 | 931-788-8536 | | | | | | |
| 28 | 728.427.2942 | | | | | | |

## So how do I actually use it?

## Syntax

- **Literal characters**

## Syntax

- Literal characters
- Special characters

## Syntax

- Literal characters
- Special characters
- Character classes

## Syntax

- Literal characters
- Special characters
- Character classes
- Shorthand character classes

## Syntax

- Literal characters
- Special characters
- Character classes
- Shorthand character classes
- Characters everywhere!

## Literal characters

`foo` is a valid regular expression

## Delimiters

**Character that defines the boundaries of your Regular Expression**

- `/` ← most common
- `~`
- `%`
- `#`
- `@`
- `;`
- `` ` ``

## Regular Expression engine

- **Software that can process regular expressions**
- **Sometimes called "flavors"**

## Warning

## Warning

- **Not every engine is the same**

## Warning

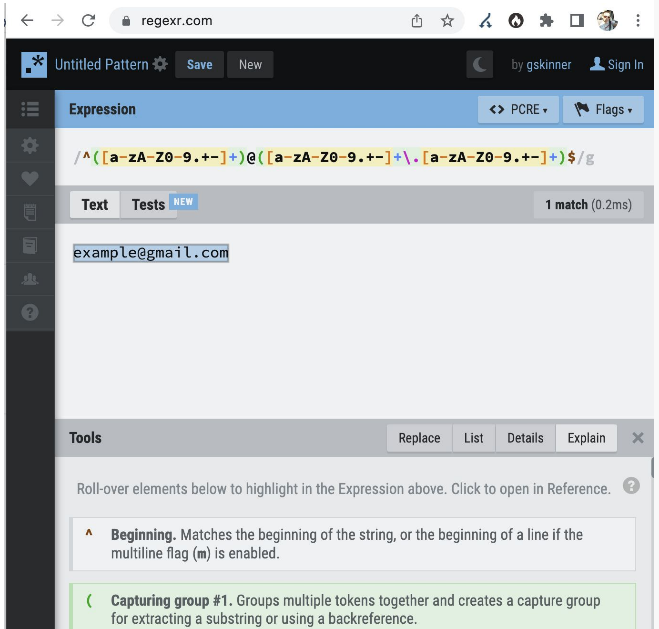
- Not every engine is the same
- Standards are "loose"

## Warning

- Not every engine is the same
- Standards are "loose"
- Always test in a RegEx tool!

# RegEx builders

- **Online Options**
  - + https://regex101.com/
  - + https://regexr.com/
  - + https://rubular.com/



```
/^([a-zA-Z0-9.+-]+)@([a-zA-Z0-9.+-]+\.[a-zA-Z0-9.+-]+)$/g
```

```
example@gmail.com
```

Roll-over elements below to highlight in the Expression above. Click to open in Reference.

^ **Beginning.** Matches the beginning of the string, or the beginning of a line if the multiline flag (**m**) is enabled.

( **Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

## RegEx builders

- Native/installable options
  - RegexBuddy (Windows)
  - Expressions (macOS)

## Special characters

AKA MetaCharacters

**12 Special Characters**

## Special characters

AKA MetaCharacters

### 12 Special Characters

- `\`
- `^`
- `$`
- `[`
- `.`
- `|`
- `?`
- `*`
- `+`
- `{`
- `(`
- `)`

## Anchors

- `^`

`^` is an anchor. Specifically, the start of a string or line

```
/^bar/
```

## Anchors

- `^`
- `$`

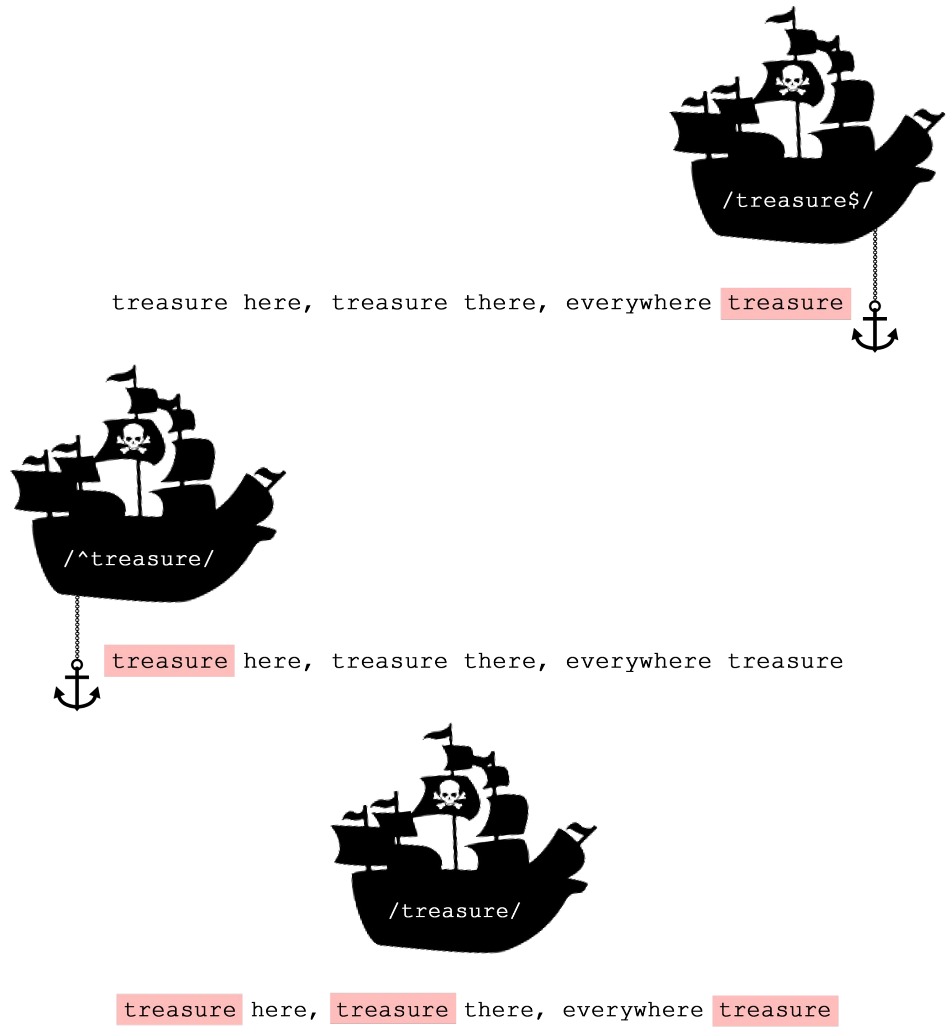
`$` is also an anchor, but for the end of a string or line

```
/bar$/
```

## Bonus!

## Bonus

Whenever possible,
ANCHOR!



treasure here, treasure there, everywhere treasure

treasure here, treasure there, everywhere treasure

treasure here, treasure there, everywhere treasure

## Character classes

- `[`

`[` allows us to define a character class

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

## Character classes

AKA Character Sets

- **Match a single literal character from a list of literal characters**

## Character classes

AKA Character Sets

- Match a single literal character from a list of literal characters
- Also allow us to define a range of literal characters

## Character classes

`AKA Character Sets`

- **Match a single literal character from a list of literal characters**
- **Also allow us to define a range of literal characters**
- `]` **is not a special character unless used with** `[` **to create a character class**

## Character classes

`AKA Character Sets`

- Match a single literal character from a list of literal characters
- Also allow us to define a range of literal characters
- `]` is not a special character unless used with `[` to create a character class
- Inside a character class you do not need to escape special characters, except for `]`, `\`, `^`, and `-`.

## Negation

- `^`

`^` is the negation character

`/^bar/`

## Wait... what?

## Negation

**When placed after a `[`, the `^` symbol negates the character class**

## Shorthand character classes

`AKA Special Sequences`

**A `\` followed by one of several literal characters, that stands in for a larger character class**

## Shorthand character classes

AKA Special Sequences

**Examples**

- `\d` **is shorthand for** `[0-9]`
- `\w` **is shorthand for** `[A-Za-z0-9_]`
- `\s` **is shorthand for all whitespace characters, or** `[ \t\r\n\f]`
- **Plus about 25 more**

## The weird one

- `.`

`.` matches any single character (except for line breaks)

`/bar./`

## Alternation

- `|`

`|` creates a branch for the regular expression engine to follow. Similar to an OR statement in programming

```
/bar|foo/
```

## Warning

## Warning

```
The RegEx Engine
always returns the
leftmost match
```

**Example**

`/cat|cats/`

There were many cats near the bowl, with one cat by the door

## Quantifiers

- `?`

`?` makes the preceding token in the regular expression optional (zero or once)

```
/foo?bar/
```

## Warning

## Warning: greediness

By default, a quantifier tells the engine to match *as many* instances of its quantified token or subpattern as possible.

Given the text "It's raining cats and dogs" a regex pattern of `/cats?/` will **always** match "cats" instead of just "cat"

## Quantifiers

- `?`
- `*`

`*` matches the preceding token in the regular expression *zero* or more times

`/foo*bar/`

## Quantifiers

- `?`
- `*`
- `+`

`+` matches the preceding token in the regular expression *one* or more times

`/foo+bar/`

## Quantifiers

- `?`
- `*`
- `+`
- `{`

`{` combined with `}` allows us to specify the number of times the previous token should be matched

```
/fo{2,3}bar/
```

## Quantifiers

- **Syntax is** `{min,max}`
  + `min` is zero or a positive number indicating the minimum number of matches of the previous token
  + `max` is an integer equal to or greater than `min` indicating the maximum number of matches

## Quantifiers

- `{0,1}` is equivalent to `?`

## Quantifiers

- `{0,1}` is equivalent to `?`
- `{0,}` is equivalent to `*`

## Quantifiers

- `{0,1}` is equivalent to `?`
- `{0,}` is equivalent to `*`
- `{1,}` is equivalent to `+`

## Quantifiers

- `{0,1}` is equivalent to `?`
- `{0,}` is equivalent to `*`
- `{1,}` is equivalent to `+`
- Omitting both the comma and `max` tells the engine to repeat the token exactly `min` times.

## Quantifiers

- `{0,1}` is equivalent to `?`
- `{0,}` is equivalent to `*`
- `{1,}` is equivalent to `+`
- Omitting both the comma and `max` tells the engine to repeat the token exactly `min` times.
- `}` is not a special character unless used with `{`

## Grouping

- `(`
- `)`

Placing a pattern between `(` and `)` allows you to group parts of a regular expression together.

```
/theat(er|re)/
```

```
/foo(bar){2}/
```

## Capturing

- `(`
- `)`

....NOT AGAIN!!!

## Capturing

- `(`
- `)`

Placing a pattern between `(` and `)` also allows you to capture the matched string for later reuse.

```
/^([a-zA-Z]{5})$/
```

## Capturing

- `(`
- `)`

Placing a pattern between `(` and `)` also allows you to capture the matched string for later reuse.

```
/^([a-zA-z]{5})$/
```

# Special characters

AKA MetaCharacters

**12 Special characters:**

- `\`
- `^`
- `$`
- `[`
- `.`
- `|`
- `?`
- `*`
- `+`
- `{`
- `(`
- `)`

## Bonus!

## Bonus

### Lookarounds

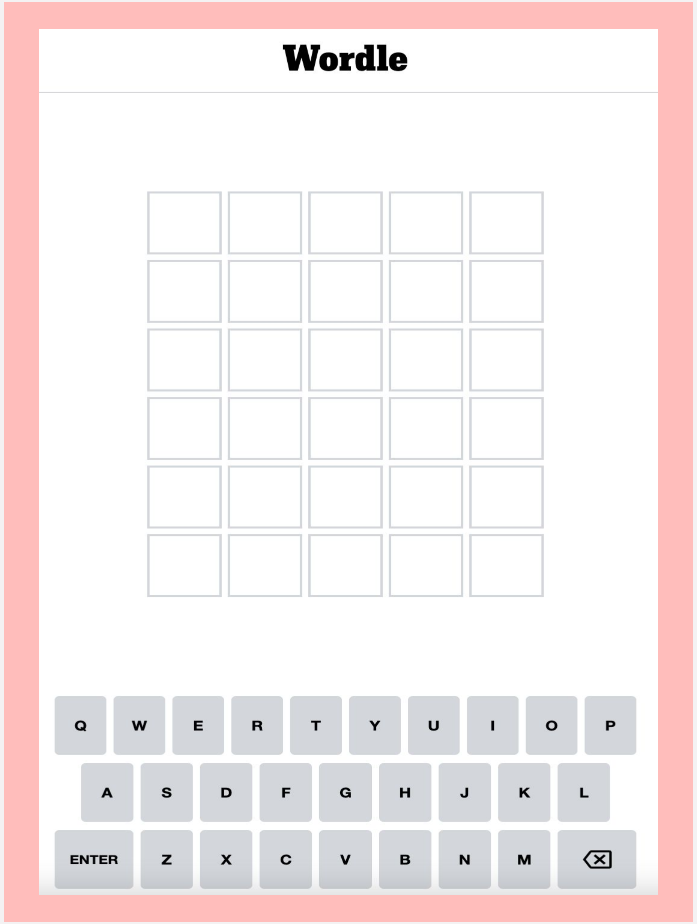
- Zero-length assertion
- Similar to `^` and `$`
- Lookahead
- Lookbehind
- Available as positive and negative

```
/(?=[a-z]{1,4}$).*/
```

## Game time!

## Game time

Build a Regular Expression to solve today's Wordle!



https://www.nytimes.com/games/wordle/

# Game time

- **Wordle clone**
  - https://wordlegame.org/
- **Crossword puzzle**
  - https://regexcrossword.com/
- **Regex golf**
  - https://alf.nu/RegexGolf/

## Resources and acknowledgements

## Resources & acknowledgements

Thank you Paul!

**Paul Gilzow**

*He/Him*

Developer Relations Engineer

paul.gilzow@platform.sh

## Resources & acknowledgements

- https://www.regular-expressions.info/
- https://en.wikipedia.org/wiki/Regular_expression
- https://www.rexegg.com/
- https://regexone.com/
- https://carlalexander.ca/beginners-guide-regular-expressions/

## Thank you!

**Marine Gandy**

DevRel Engineer, Platform.sh

marine.gandy@platform.sh

platform.sh

## Questions?