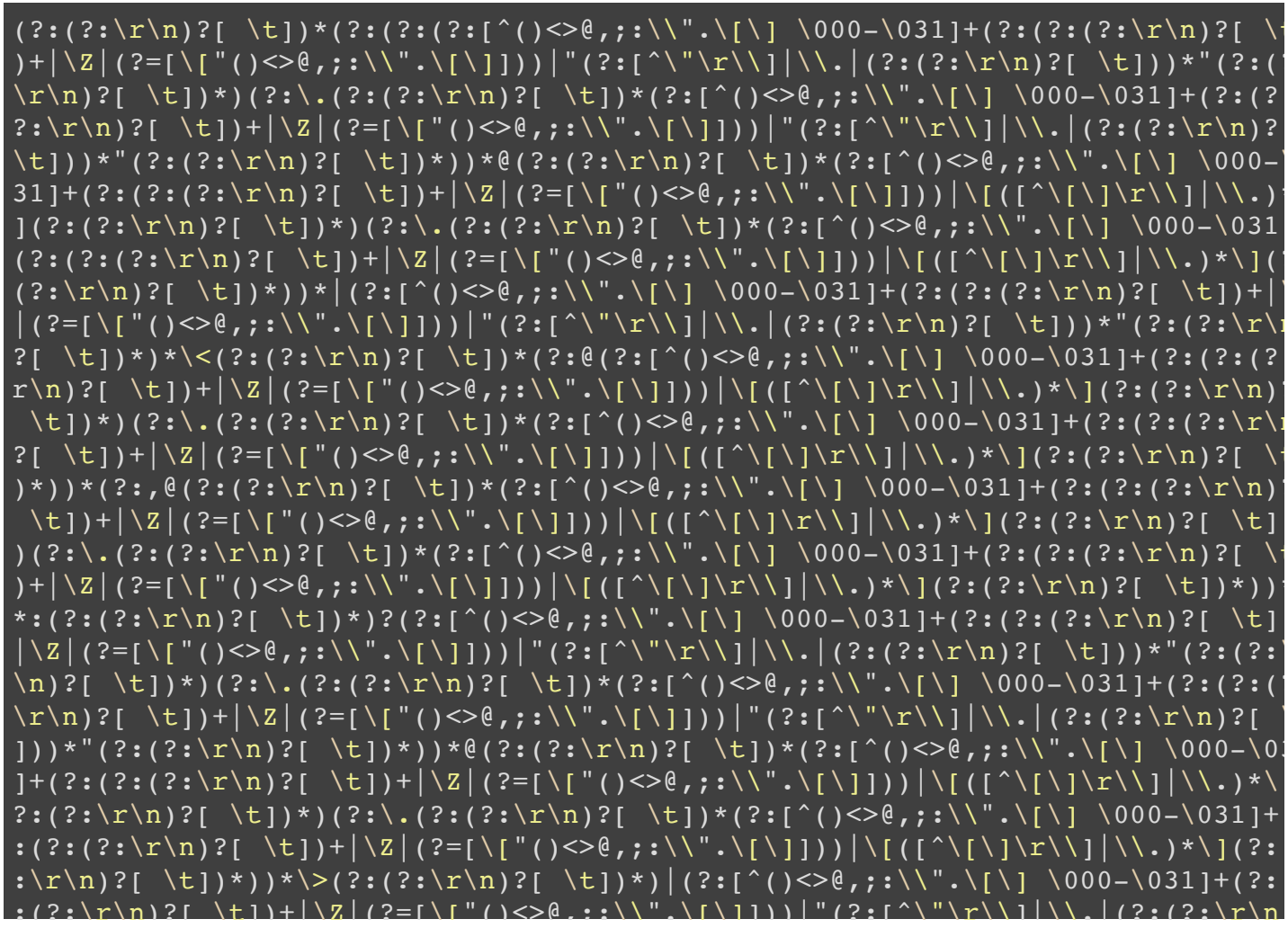
Regular Expressions



What are they?

- They are pattern matchers
- *"A sequence of symbols and characters expressing a string or pattern to be searched for within a longer piece of text"*
- They aren't just for programmers
- Often considered a write-only language

Where did they come from?

- Began with Ken Thompson with this article published in 1968
  - For the IBM 7094
  - Then the QED editor
- Lots of languages added to it
  - Perl has a great implementation of it, as does Ruby and lots of other languages

Where are they used?

- Validations
- Checking for spam
- Approximate matching
- Extracting important information
- Finding and replacing
- Automation - lots of commands use this (e.g. grep)
- The meaning of life
  - `^(?=(?!(.)\1)([^\DO:105-93+30])(?-1)(?<!\d(?<=(`

Useful sites

- Rubular
- Regexr
- Regexpr
- Ruby Docs
- MDN Regular Expressions
- Regexone
- Debuggex
- Regex101
- Regular Expression Crossword
- Regular Expression Golf
- RegexQuest
- TryRegex
- Akowalz

Approximate equality

```
"bob" == "bob"
# => true

"bob" =~ /bob/
# => 0

"Hello, my name is bob" =~ /bob/
# => 18
```

Ranges

```
"cat" =~ /[chs]at/

"bob" =~ /[Bb]ob/

"bob" =~ /[A-Z]ob/

"bob" =~ /[A-Za-z]ob/

"012345" =~ /[0-9]/

"012345" =~ /[^A-Z]/   # Anything but A-Z
```

Character classes

- **\w** - any word character
- **\W** - any non-word character
- **\d** - anything that is a digit
- **\D** - anything that isn't a digit
- **\s** - any whitespace character
- **\S** - anything but a whitespace character
- **\b** - any word boundary
- **\B** - any non-word boundary
- **.** - any character except a new line

Character Classes

```
"this has word characters" =~ /\w/

"this has no numbers" =~ /\D/

"this has numbers 012" =~ /\d/
```

Quantifiers || Qualifiers

***** - Zero or more times

+ - One or more times

? - Zero or one times (optional)

{*x*} - Exactly *x* times

{*x*,} - *x* or more times

{,*y*} - *y* or less times

{*x*,*y*} - At least *x* and at most *y* times

Repetition

```
"Arctic Arctic Arctic" =~ /(.*) \1 \1/

"oo book cook" =~ /(.*) b(\1)k c(\1)k/

"ababa" =~ /(.)(.)\1\2\1/
```

Have a go at these exercises

RegEx methods - Match

```
"Some numbers: 42" =~ /(\d+)/

puts $1
```

RegEx methods - Match

```
matches = "Get the number: 42.".match( /(\d+)/ )

matches.pre_match

matches.string

matches[1]

matches.post_match
```

Captures

```
regex = /(\d{3}).*(\d{3}).*?(\d{4})/
matches = "205-222-3219".match( regex )

matches[1]
matches[2]
matches[3]
```

Named Captures

```
matches = "Num: 42.".match( /(?<num>\d+)/ )

matches.pre_match

matches[:num]

matches.post_match
```

Scan, sub, gsub & split

```
"123 456 789".match( /\d+/ )
"123 456 789".scan( /\d+/ )

"205-222-3219".sub( /\D/, "" )
"205-222-3219".gsub( /\D/, "" )

"205-222-3219".split /-/
```

Anchors

```
"ruby      ".match( /^ruby/ )

"      ruby".match( /ruby$/ )
```

Flags

```
"RUBY" =~ /ruby/i

"205-222-3219".match(/
    (\d{3}).*   # First set of numbers: area code
    (\d{3}).*   # Second set of numbers: exchange
    (\d{4}).*   # Third set of numbers: suffix
/x)
```

Greedy vs. Reluctant

```
"RUBY" =~ /ruby/i

"205-222-3219".match(/
    (\d+).*
    (\d+).*
    (\d+).*
/x) # GREEDY!

"205-222-3219".match(/
    (\d+).*?
    (\d+).*?
    (\d+).*?
/x) # RELUCTANT
```

Have a go at these exercises