Parsing

Parsers

Jenna Zeigen

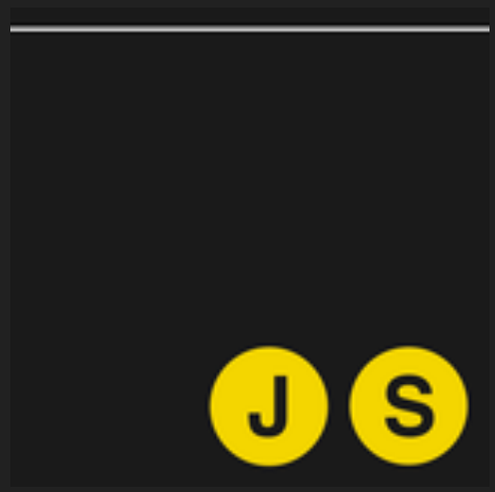
JSConf Hawaii

2/5/2020

Senior Frontend Engineer
at Slack

Organizer of EmpireJS

Organizer of BrooklynJS



@zeigenvector

jenna.is/at-jsconfhi

@zeigenvector
jenna.is/at-jsconfhi

parsing

parsers!

@zeigenvector
jenna.is/at-jsconfhi

1. abcs of language

@zeigenvector
jenna.is/at-jsconfhi

1. abcs of language
2. hmm, actually, let's just step through a (small) parser

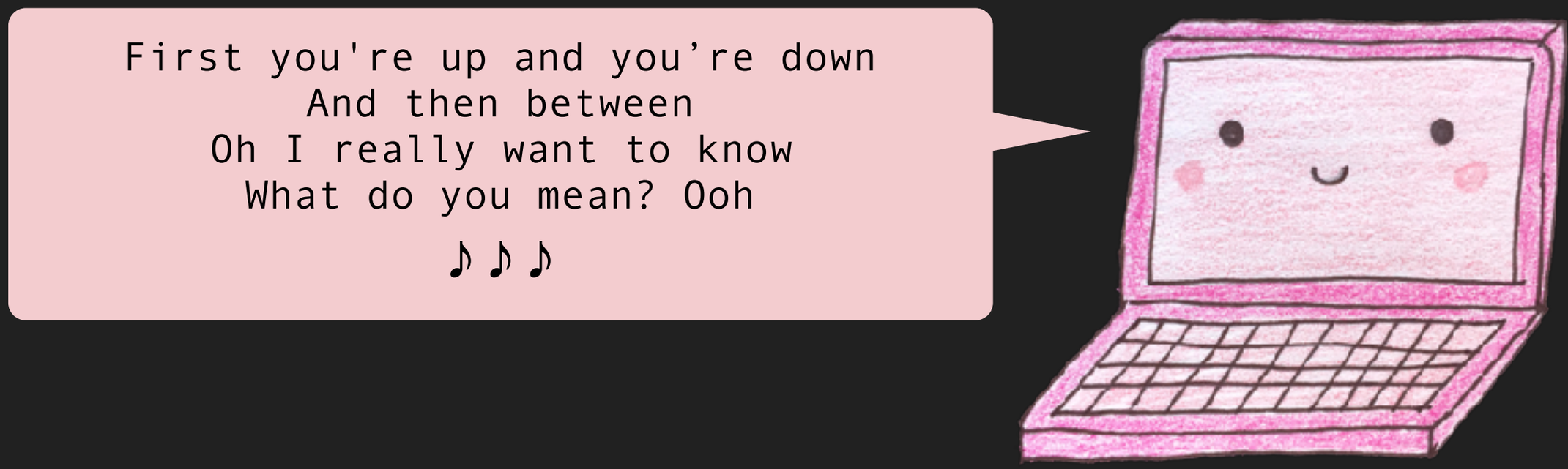
@zeigenvector
jenna.is/at-jsconfhi

the abcs of language

@zeigenvector
jenna.is/at-jsconfhi

the abcs of language

"language" is a structured system of communication

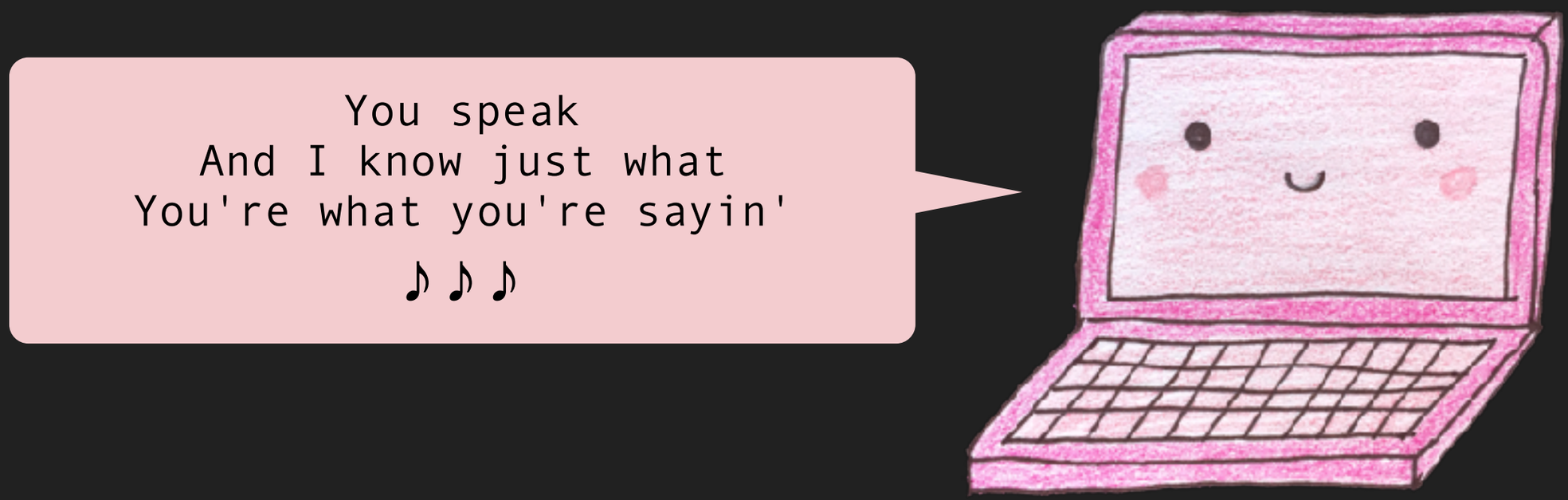


Justin Bieber - What Do You Mean?
https://en.wikipedia.org/wiki/Language

@zeigenvector
jenna.is/at-jsconfhi

the abcs of language

"natural language" is a naturally evolved system that humans use to communicate with each other

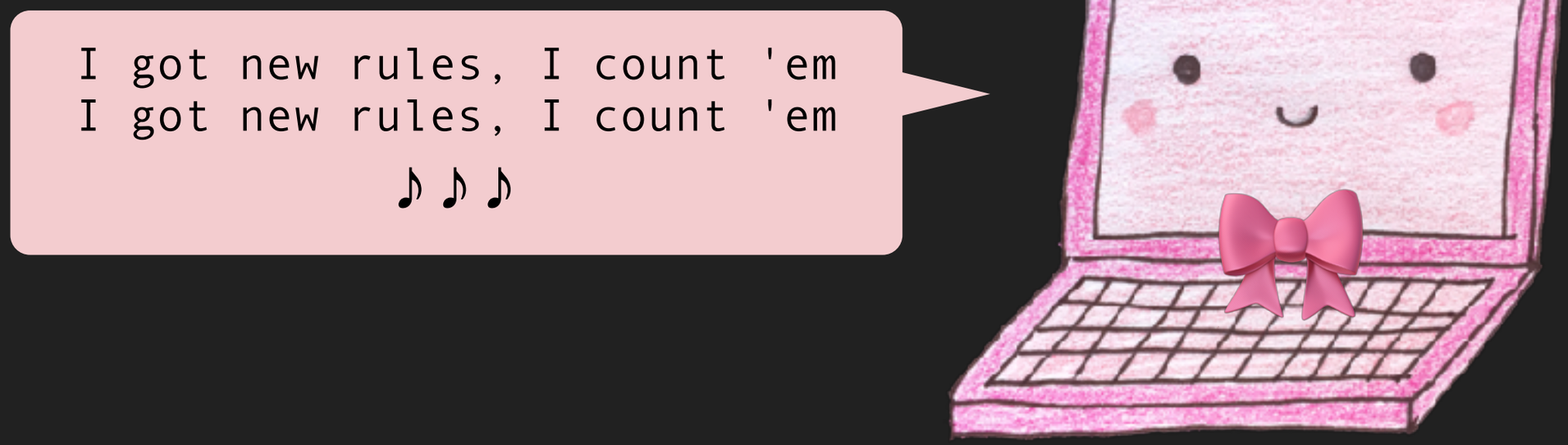


No Doubt - Don't Speak
https://en.wikipedia.org/wiki/Language

@zeigenvector
jenna.is/at-jsconfhi

the abcs of language

"formal languages" have an alphabet and words, which can be combined correctly based on specific rules



Dua Lipa - New Rules
https://en.wikipedia.org/wiki/Formal_language

@zeigenvector
jenna.is/at-jsconfhi

grammar school

a language's grammar is the set of rules for that language

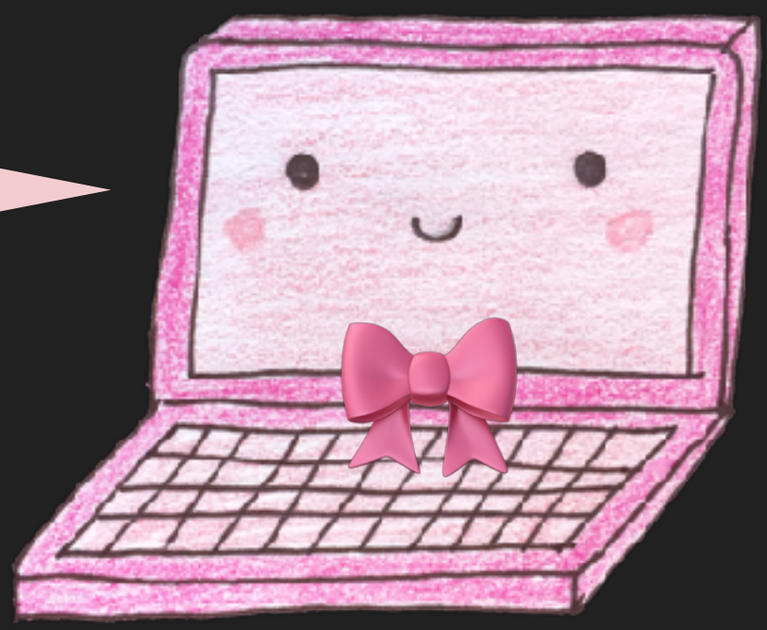
MC Hammer - U Can't Touch This
https://en.wikipedia.org/wiki/Formal_grammar

@zeigenvector
jenna.is/at-jsconfhi

grammar school

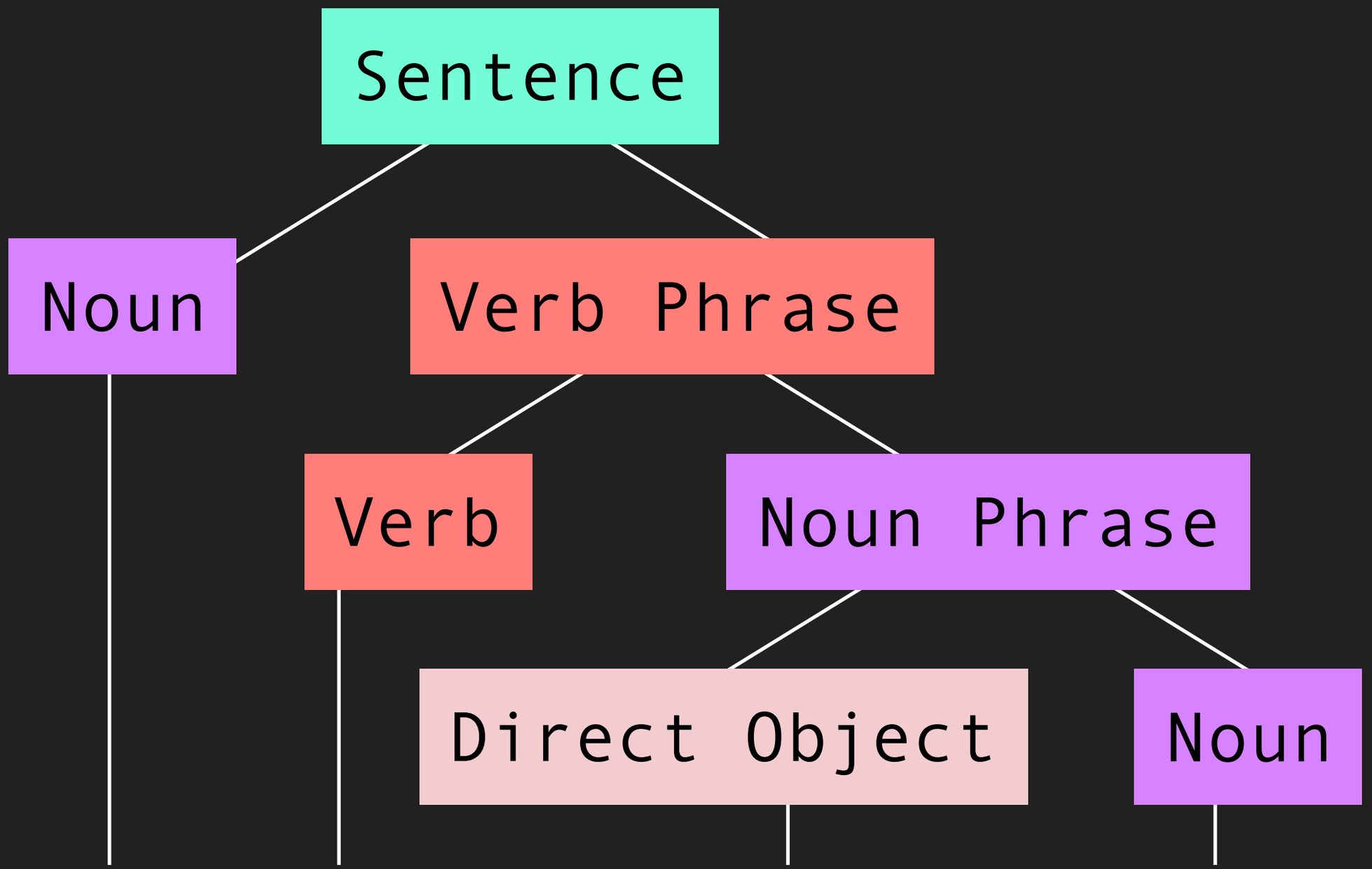
"formal grammars" put these rules in terms of replacement

To the left, to the left
To the left, to the left (Mmm)
To the left, to the left
Non-terminals in the spot to the left
To the left, to the left
The grammar tells us for what symbols
They are replaceable
♪♪♪



Beyoncé - Irreplaceable
https://en.wikipedia.org/wiki/Formal_grammar

grammar school



Jenna gave the talk

@zeigenvector
jenna.is/at-jsconfhi

grammar school

Sentence = Noun + Verb Phrase

Verb Phrase = Verb + Noun Phrase

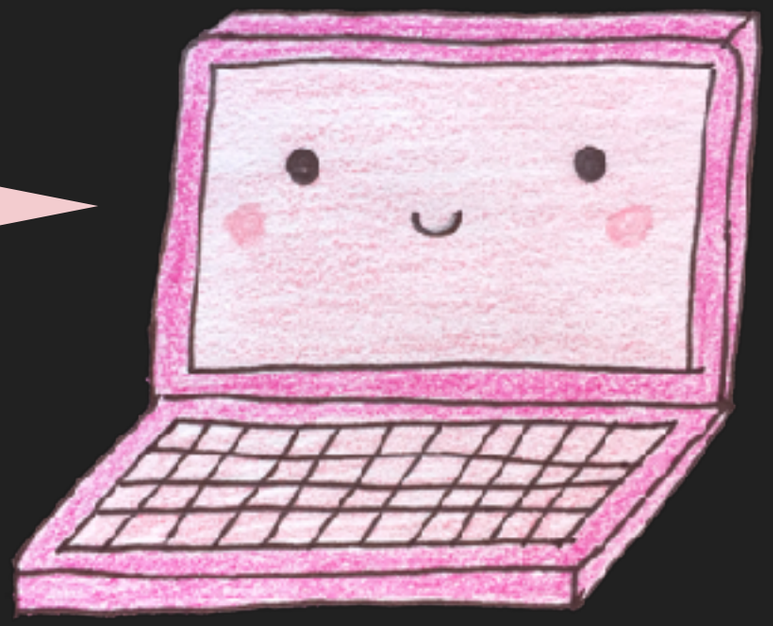
Noun Phrase = Direct Object + Noun

@zeigenvector
jenna.is/at-jsconfhi

grammar school

Programming language grammars are defined in their spec

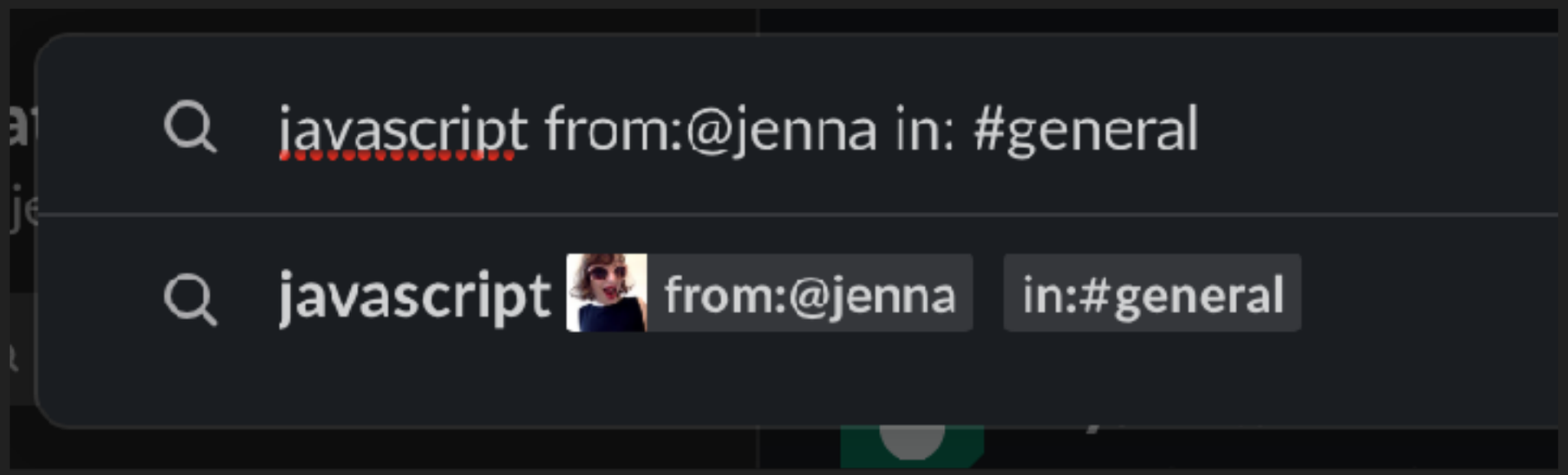
thank u, spec
thank u, spec
thank u, spec
I'm so very grateful for my spec
♪ ♪ ♪



Ariana Grande - thank u, next
https://www.ecma-international.org/ecma-262/10.0/index.html

@zeigenvector
jenna.is/at-jsconfhi

syntax city



@zeigenvector
jenna.is/at-jsconfhi

syntax city

javascript "front end"
in:#random in: #general
from:@jenna

@zeigenvector
jenna.is/at-jsconfhi

syntax city

```
javascript "front end" in:#random
in: #general from:@jenna
```

Query → Term

Query → Term Query

Query → Filter

Query → Filter Query

@zeigenvector
jenna.is/at-jsconfhi

syntax city

javascript | "front end"

in:#random | in: #general

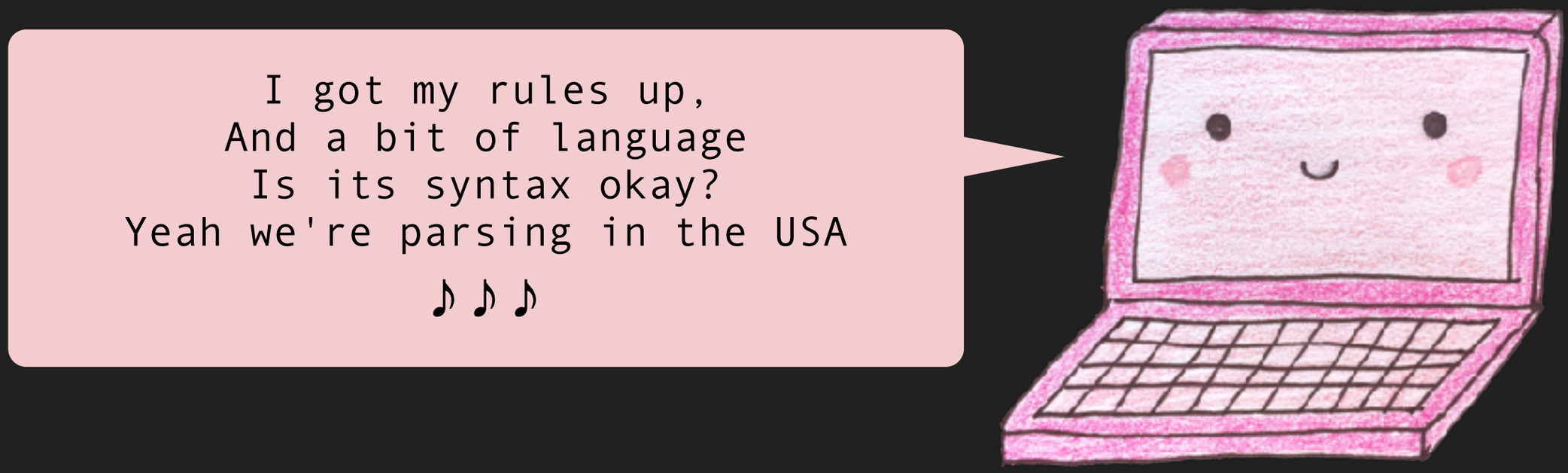
from:@jenna

ok, now parsers

@zeigenvector
jenna.is/at-jsconfhi

moving parse

the process of analyzing language against the rules of its grammar



Miley Cyrus - Party in the USA
https://en.wikipedia.org/wiki/Parsing

moving parse

a function that takes raw input and returns meaningful data created from the input, or an error

All the beautiful inputs
Are very, very meaningful
You know, space is my favorite delimiter
I felt so symbolic yesterday

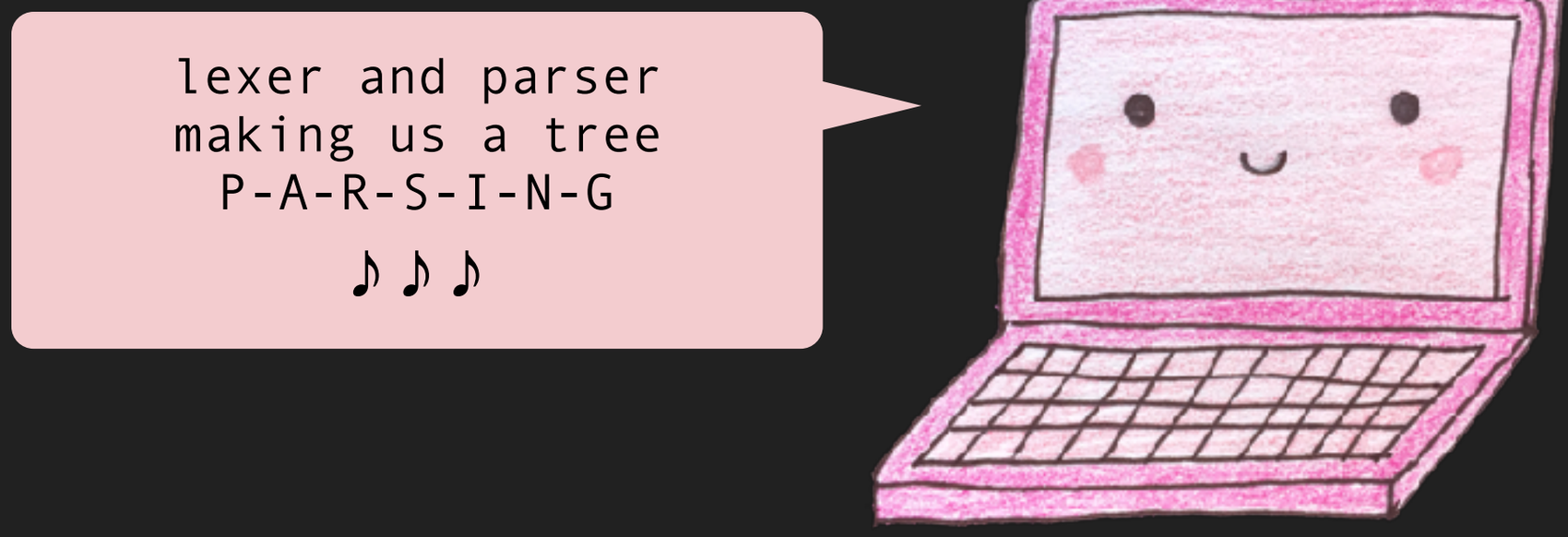
♪ ♪ ♪

Counting Crows - Mr. Jones
https://dev.to/yelouafi/a-gentle-introduction-to-parser-combinators-21a0

@zeigenvector
jenna.is/at-jsconfhi

moving parse

parsers usually have two parts: the lexer and the parser



https://en.wikipedia.org/wiki/Parsing

lex go

the lexer takes the text and breaks it down into meaningful units, called "tokens"

Old Crow Medicine Show - Wagon Wheel
https://tomassetti.me/guide-parsing-algorithms-terminology/

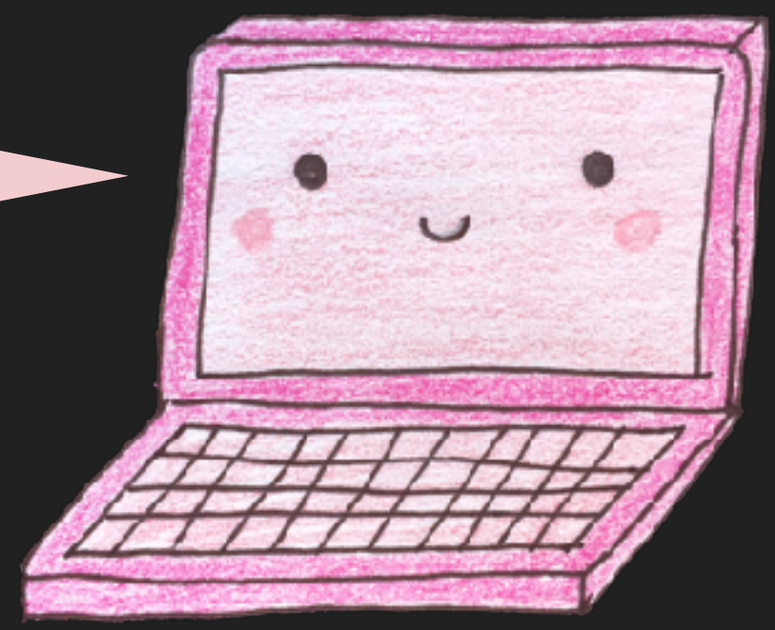
@zeigenvector
jenna.is/at-jsconfhi

lex go

first, the "scanner" goes through and breaks the string of characters into the proper chunks, or "lexemes"

I was born to lex (Yes)
According to the spec
What amazing tech,
Having this effect (Woo)
And soon the parser will turn
These strings into objects (Money)

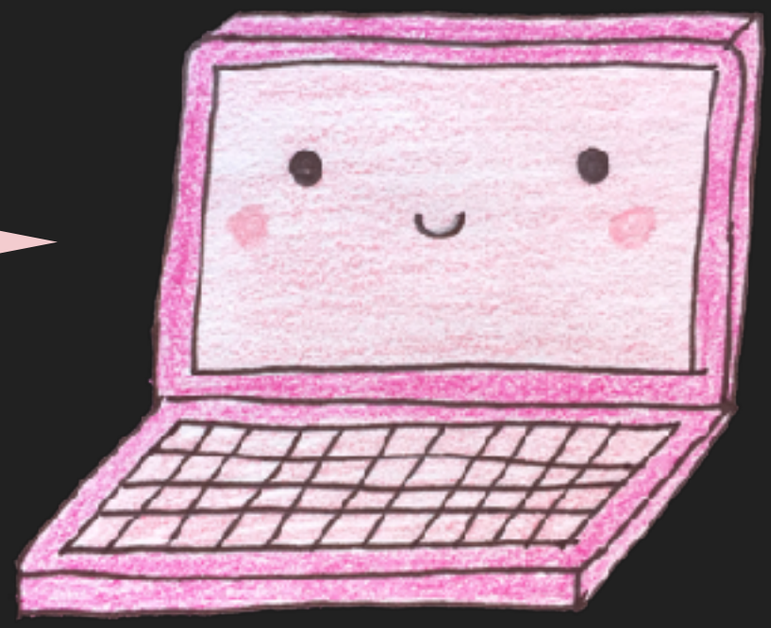
♪ ♪ ♪



Cardi B - Money
https://en.wikipedia.org/wiki/Lexical_analysis

lex go

coding time
♪ ♪ ♪



Semisonic - Closing Time
https://github.com/jennazee/sparse/blob/master/sparse.js
https://jenna.is/sparse/sparse.html

@zeigenvector
jenna.is/at-jsconfhi

lex go

```
const lexemes =
'Jenna gave the
talk'.split(' ');
```

@zeigenvector
jenna.is/at-jsconfhi

lex go

const	lexemes	=
"Jenna gave the talk"		

.	split	
(	' '	)
;		

@zeigenvector
jenna.is/at-jsconfhi

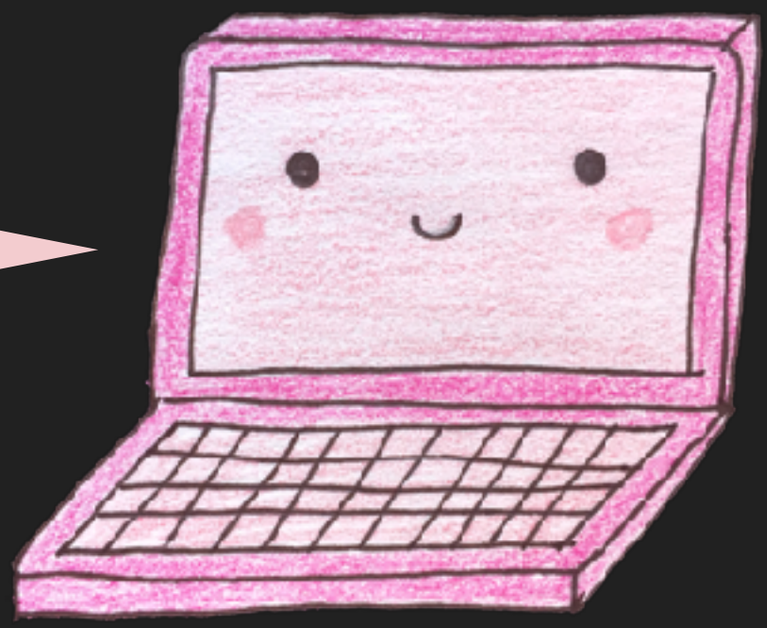
lex go

then, the "evaluator"
combines the lexeme's type
with its value to create

the "token"

I then begin to encounter with my parse,
To split the text apart
Break it down into sections
Tokens from the lexemes

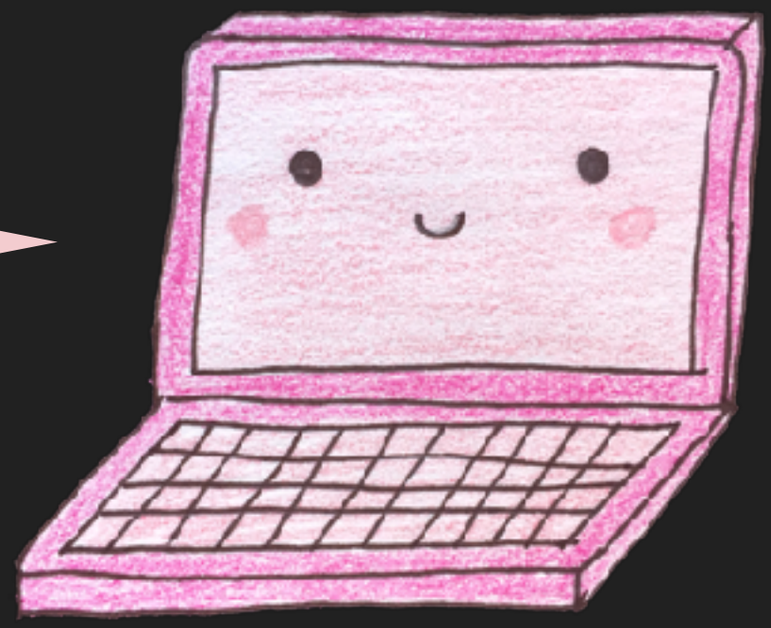
♪ ♪ ♪



The Notorious B.I.G. – Sky's The Limit
https://en.wikipedia.org/wiki/Lexical_analysis

lex go

coding time
♪ ♪ ♪



Semisonic - Closing Time
https://github.com/jennazee/sparse/blob/master/sparse.js
https://jenna.is/sparse/sparse.html

@zeigenvector
jenna.is/at-jsconfhi

lex go

const	lexemes	=
"Jenna gave the talk"		

.	split	
(	' '	)
;		

@zeigenvector
jenna.is/at-jsconfhi

lex go

Keyword | Identifier | Punctuator

String

Punctuator | Identifier

Punctuator | String | Punctuator

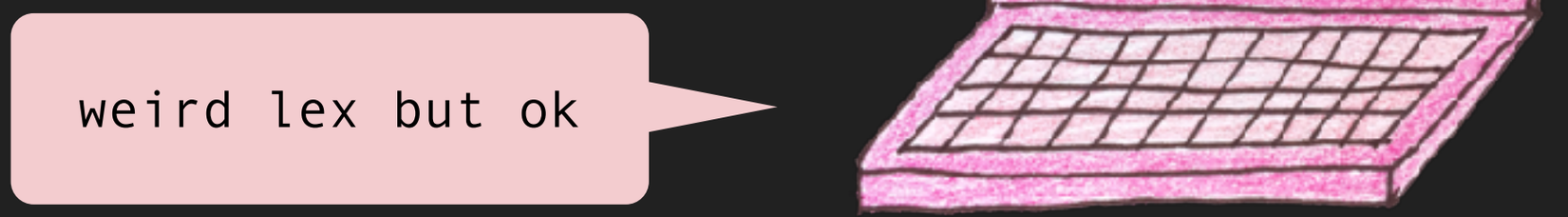
Punctuator

https://esprima.org/demo/parse.html#

@zeigenvector
jenna.is/at-jsconfhi

lex go


```
[
  { "type": "Keyword", "value": "const" },
  { "type": "Identifier", "value": "lexemes" },
  { "type": "Punctuator", "value": "=" },
  { "type": "String", "value": "'Jenna gave a talk'"
  { "type": "Punctuator", "value": "." },
  { "type": "Identifier", "value": "split" },
  { "type": "Punctuator", "value": "(" },
  { "type": "String", "value": "' '" },
  { "type": "Punctuator", "value": ")" },
  { "type": "Punctuator", "value": ";" }
]
```

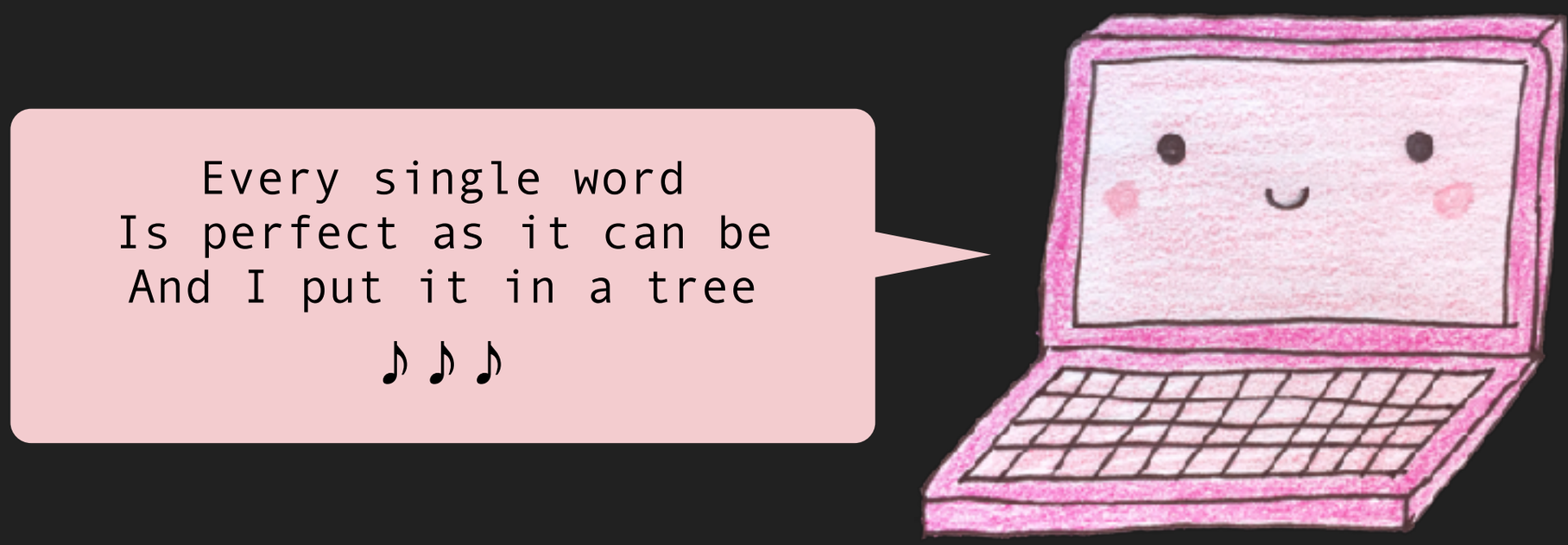



https://esprima.org/demo/parse.html#

@zeigenvector
jenna.is/at-jsconfhi

parse for the course

the parser will check that the syntax is correct while creating a structural representation

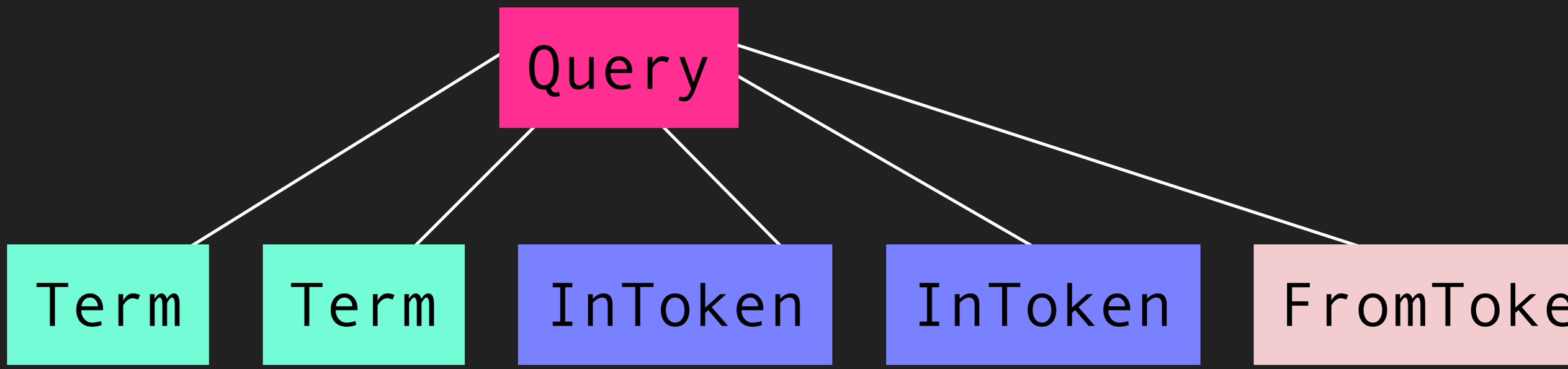


CHVRCHES ft. Marshmello - Here With Me

https://www.geeksforgeeks.org/introduction-of-parsing-ambiguity-and-parsers-set-1/

parse for the course

javascript "front end" in:#random
in: #general from:@jenna



@zeigenvector
jenna.is/at-jsconfhi

parse for the course

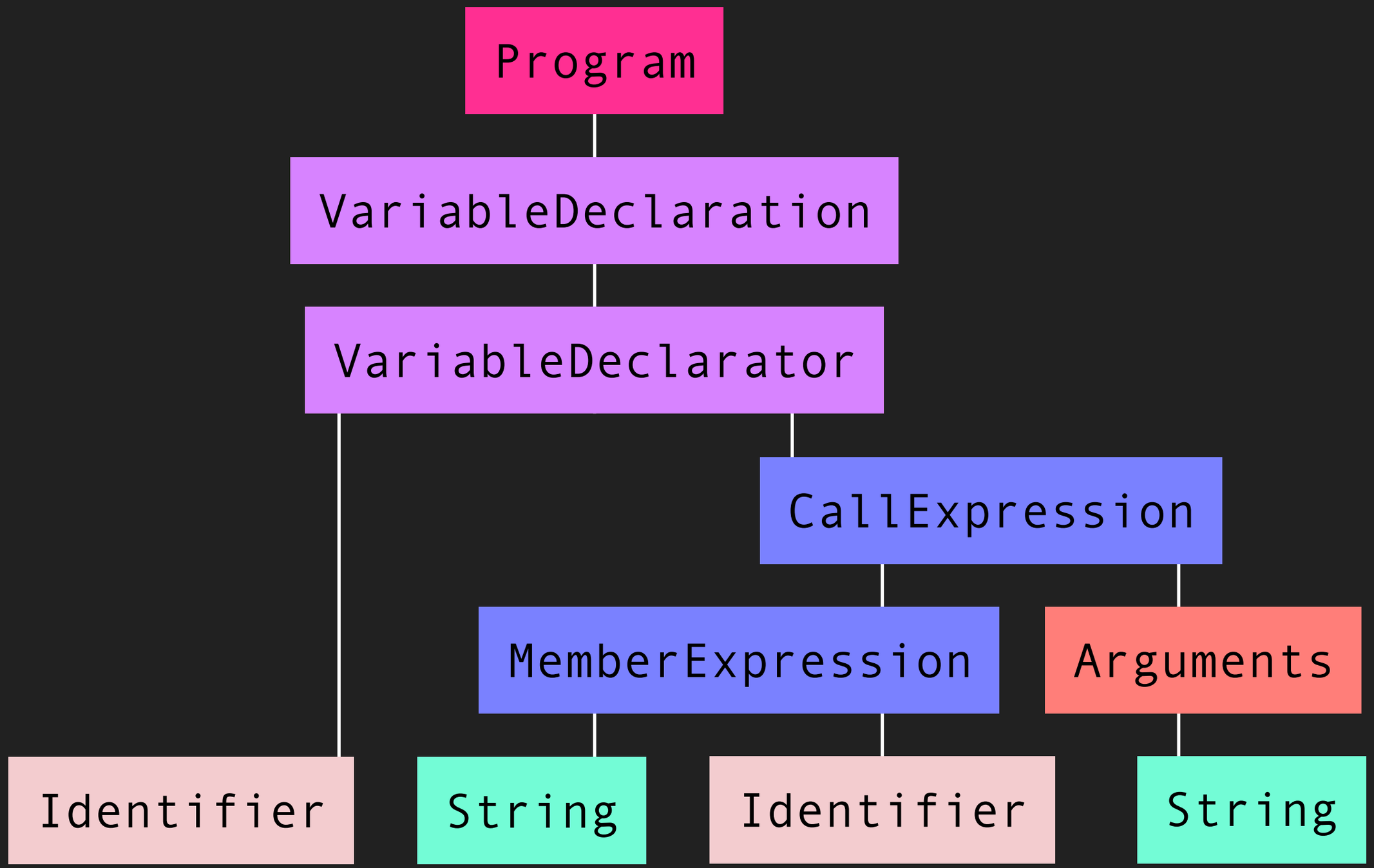
I know who I want
To read my code
(It's you!)

♪ ♪ ♪

Semisonic - Closing Time
https://github.com/jennazee/sparse/blob/master/sparse.js
https://jenna.is/sparse/sparse.html

@zeigenvector
jenna.is/at-jsconfhi

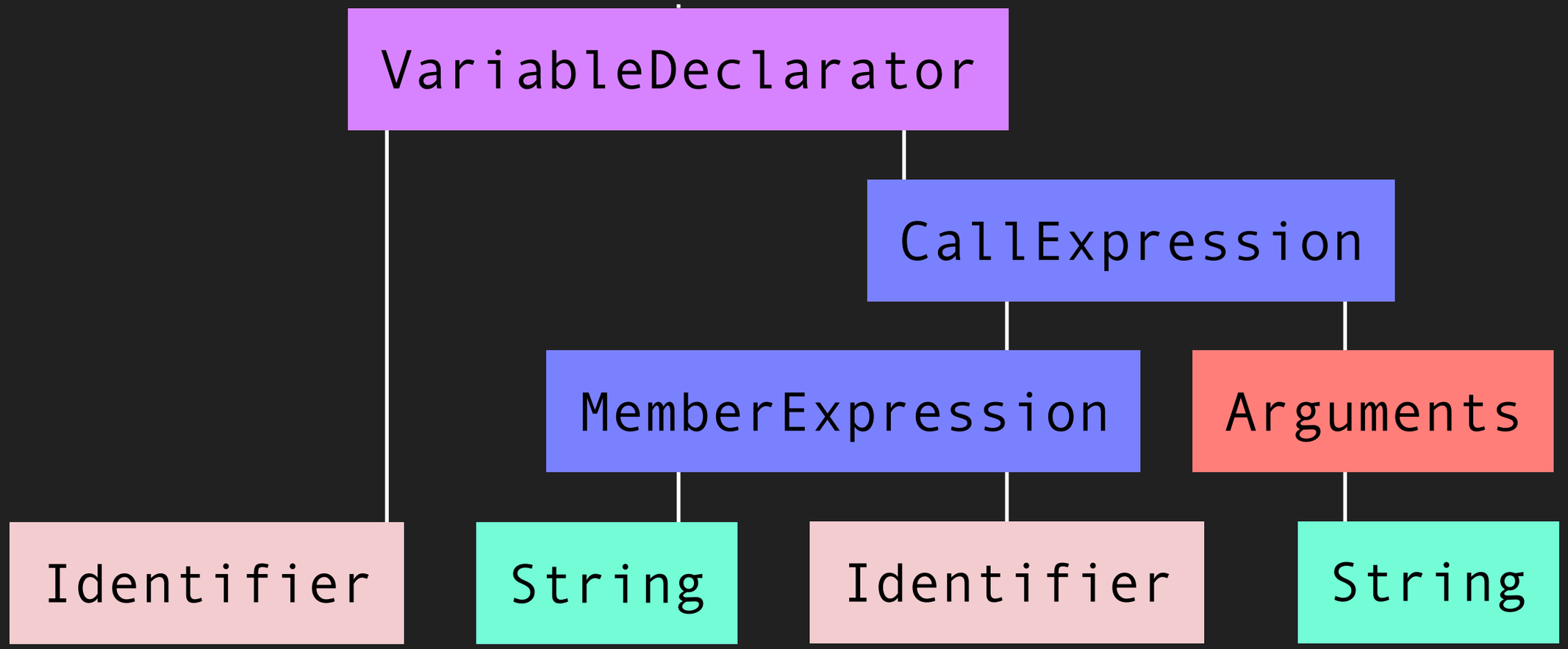
parse for the course



https://esprima.org/demo/parse.html#

@zeigenvector
jenna.is/at-jsconfhi

parse for the course




```
const lexemes = 'Jenna gave
the talk'.split(' ');
```

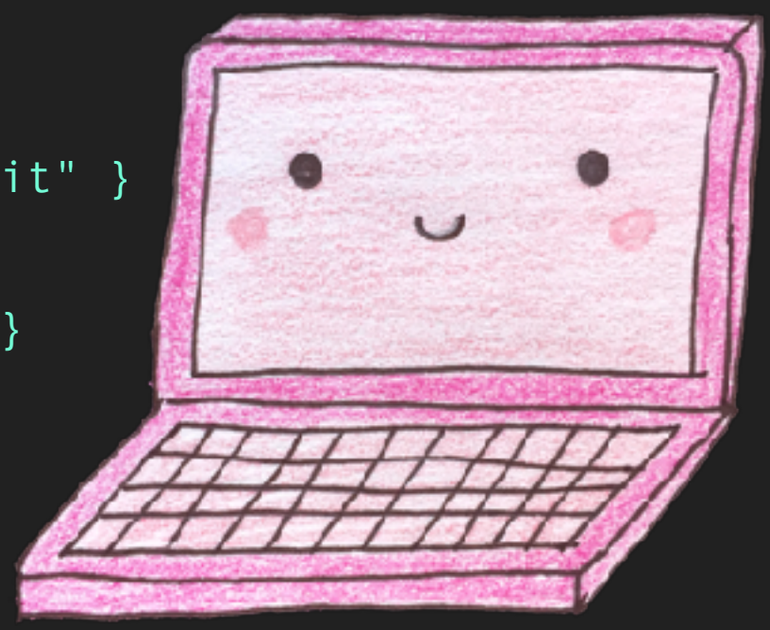
https://esprima.org/demo/parse.html#

@zeigenvector
jenna.is/at-jsconfhi

parse for the course


```
{
  "type": "Program",
  "body": [
    {
      "type": "VariableDeclaration",
      "declarations": [
        {
          "type": "VariableDeclarator",
          "id": { "type": "Identifier", "name": "lexemes" },
          "init": {
            "type": "CallExpression",
            "callee": {
              "type": "MemberExpression",
              "computed": false,
              "object": {
                "type": "Literal",
                "value": "Jenna gave the talk",
                "raw": "'Jenna gave a talk'"
              },
              "property": { "type": "Identifier", "name": "split" }
            },
            "arguments": [
              { "type": "Literal", "value": " ", "raw": "' '" }
            ]
          }
        }
      ],
      "kind": "const"
    }
```


Computers can have a little JavaScript, as a tree



https://esprima.org/demo/parse.html#

syntax city

javascript	"front end"
in: #random	in: #general

from: @jenna

@zeigenvector
jenna.is/at-jsconfhi

syntax city

javascript "front end" in:#random
in: #general from:@jenna

Query → Term

Query → Term Query

Query → Filter

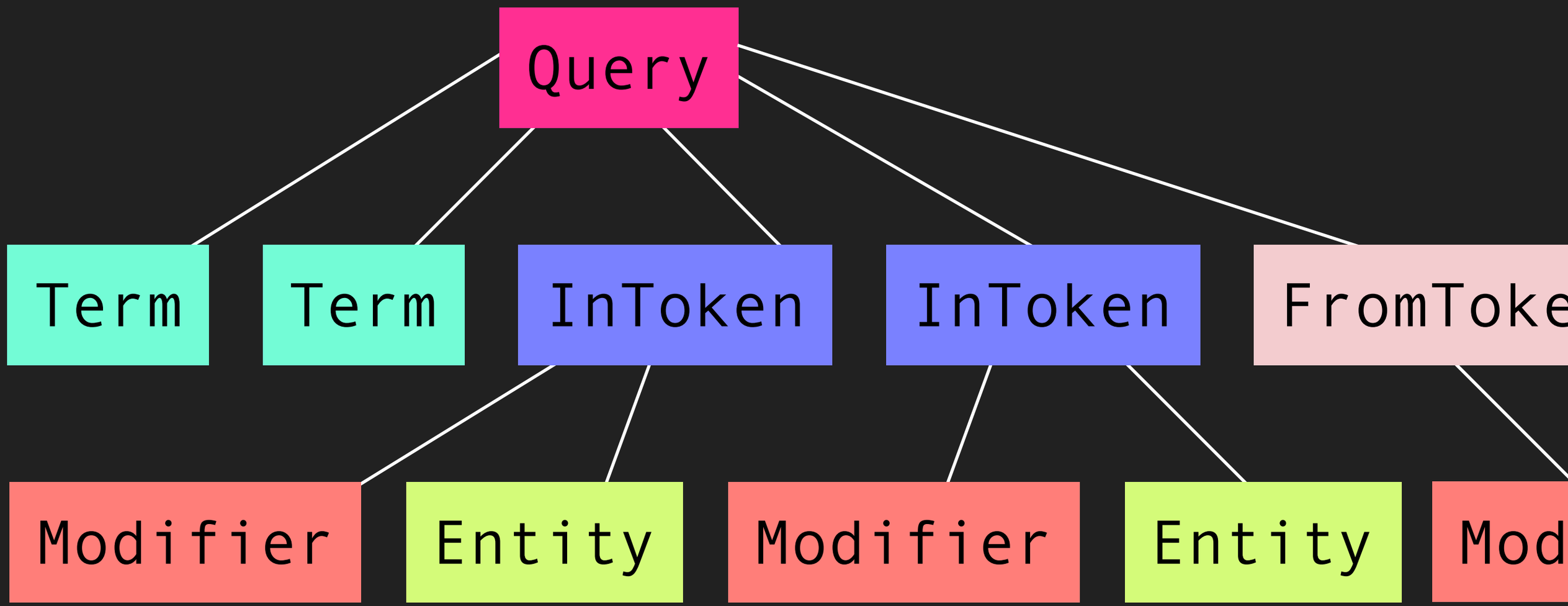
Query → Filter Query

Filter → Modifier Entity

@zeigenvector
jenna.is/at-jsconfhi

syntax city

javascript "front end" in:#random
in: #general from:@jenna



@zeigenvector
jenna.is/at-jsconfhi

parse for the course

Read my co-odeeee
♪ ♪ ♪

Semisonic - Closing Time
https://github.com/jennazee/sparse/blob/master/parse.js
https://jenna.is/sparse/parse.html

@zeigenvector
jenna.is/at-jsconfhi

the more complicated stuff...

advanced grammar school

```
/in: ?([^ ]+)|from: ?([^ ]+)'|"([^"]+)"|\'([^\']+)\'|([^ ]+)'/
```

@zeigenvector
jenna.is/at-jsconfhi

advanced grammar school

A "regular grammar" is one where all the production rules are one of the following:

$$A \rightarrow a$$

$$A \rightarrow aB$$

https://en.wikipedia.org/wiki/Regular_grammar

advanced grammar school

$$A \rightarrow a$$

$$A \rightarrow aB$$

$$Query \rightarrow Term$$

$$Query \rightarrow Term\ Query$$

$$Query \rightarrow Filter$$

$$Query \rightarrow Query\ Filter$$

@zeigenvector
jenna.is/at-jsconfhi

advanced grammar school

$$A \rightarrow a$$

$$A \rightarrow Ba$$

$$Query \rightarrow Query\ Filter$$

$$Filter \rightarrow Modifier\ Entity$$

@zeigenvector
jenna.is/at-jsconfhi

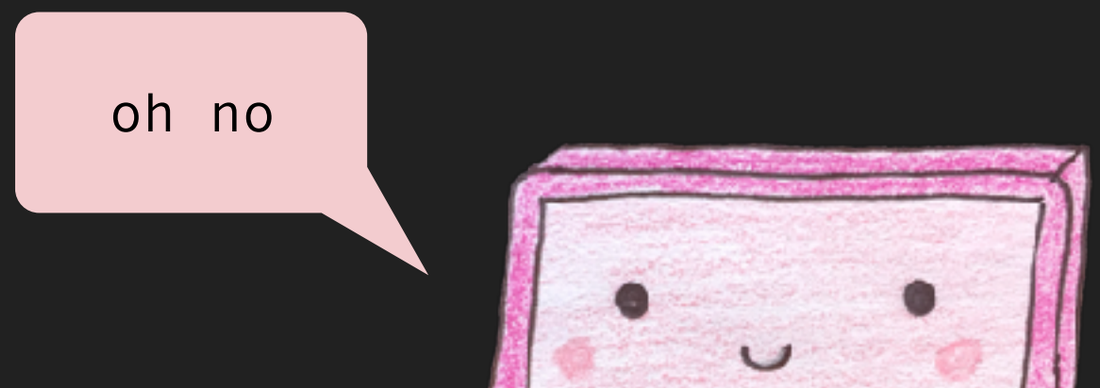
advanced grammar school

$$A \rightarrow a$$

$$A \rightarrow Ba$$

Query → Query Filter

Filter → Modifier Entity



@zeigenvector
jenna.is/at-jsconfhi

advanced grammar school

A "context-free grammar"
has rules that follow

$$A \rightarrow \alpha$$

where A is a non-terminal
and α is a combo of
terminal and non-terminal

https://en.wikipedia.org/wiki/Context-free_grammar

@zeigenvector
jenna.is/at-jsconfhi

advanced grammar school

$$S \rightarrow SS$$

$$S \rightarrow ()$$

$$S \rightarrow (S)$$

$$S \rightarrow []$$

$$S \rightarrow [S]$$

https://en.wikipedia.org/wiki/Context-free_grammar#Well-formed_parentheses

advanced grammar school

```
<div class="Wrapper">
  <input class="Input"/>
  <div class="Visualizer">
    <div class="Token">Grammars!</div>
  </div>
</div>
```

@zeigenvector
jenna.is/at-jsconfhi

real world parsing

@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

javascript "front end"

in: #random

in: #general

from: @jenna

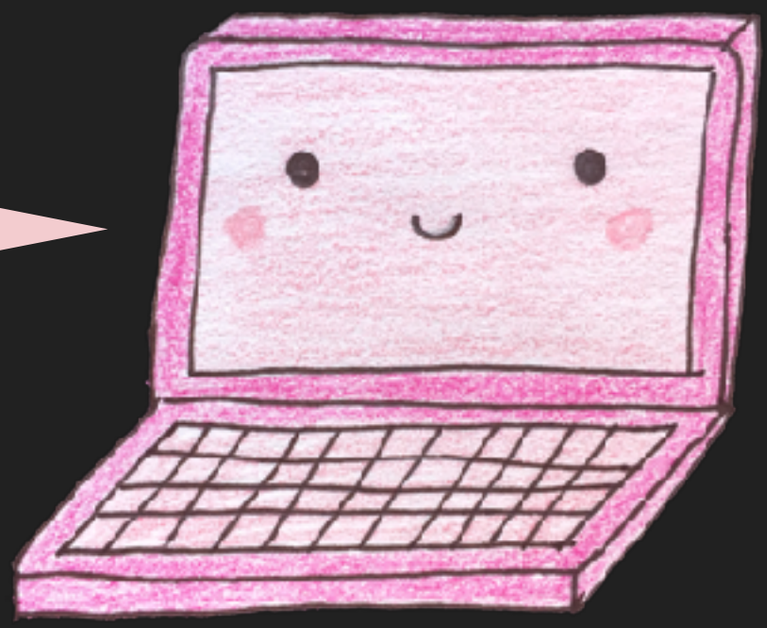
real world: parsers

Term	Term
Modifier	Entity
Modifier	Entity
Modifier	Entity

real world: parsers

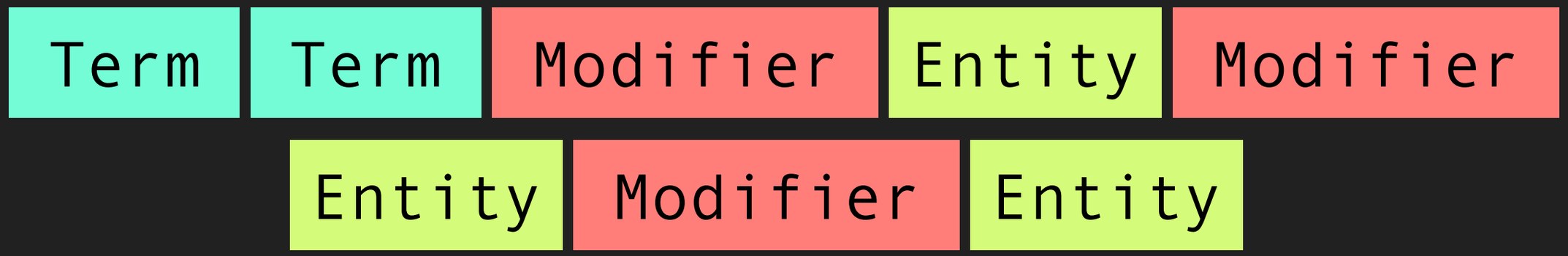
then, the parser goes through and matches the tokens to production rules

It's as if you know me better
Than I ever knew myself
I love how you can tell
All the pieces, pieces, pieces of me
♪ ♪ ♪



Ashlee Simpson - Pieces of Me
https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

real world: parsers



https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Term | Term | Modifier | Entity | Modifier

Entity | Modifier | Entity

Query

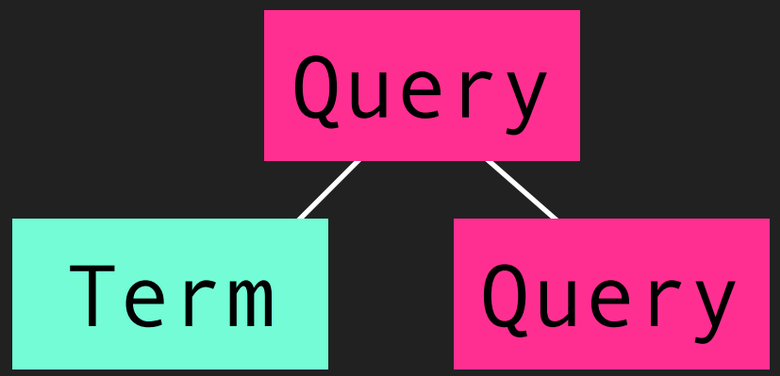
https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Term | Modifier | Entity | Modifier

Entity | Modifier | Entity

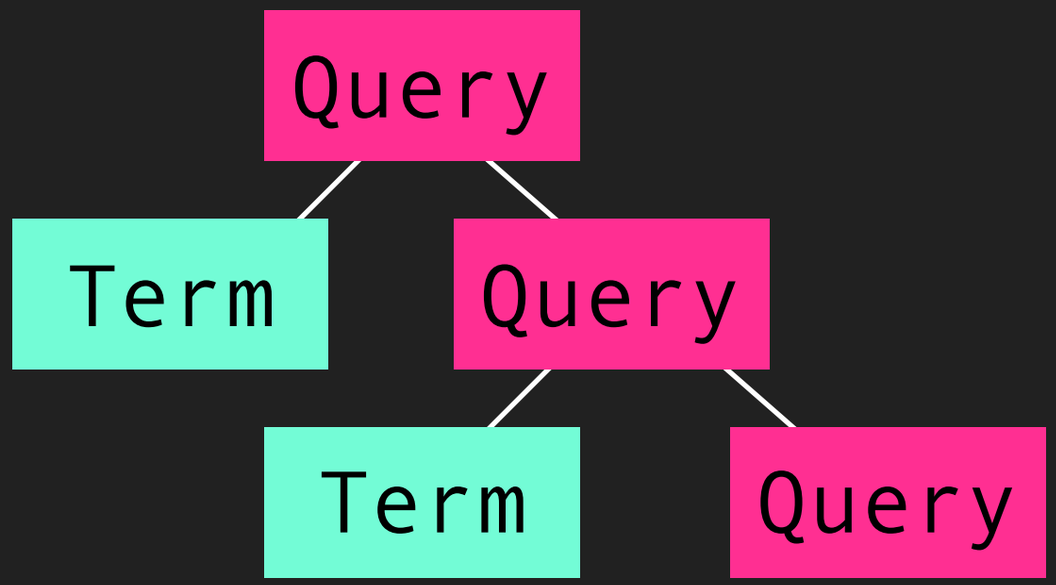


https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

real world: parsers

Modifier | Entity | Modifier

Entity | Modifier | Entity



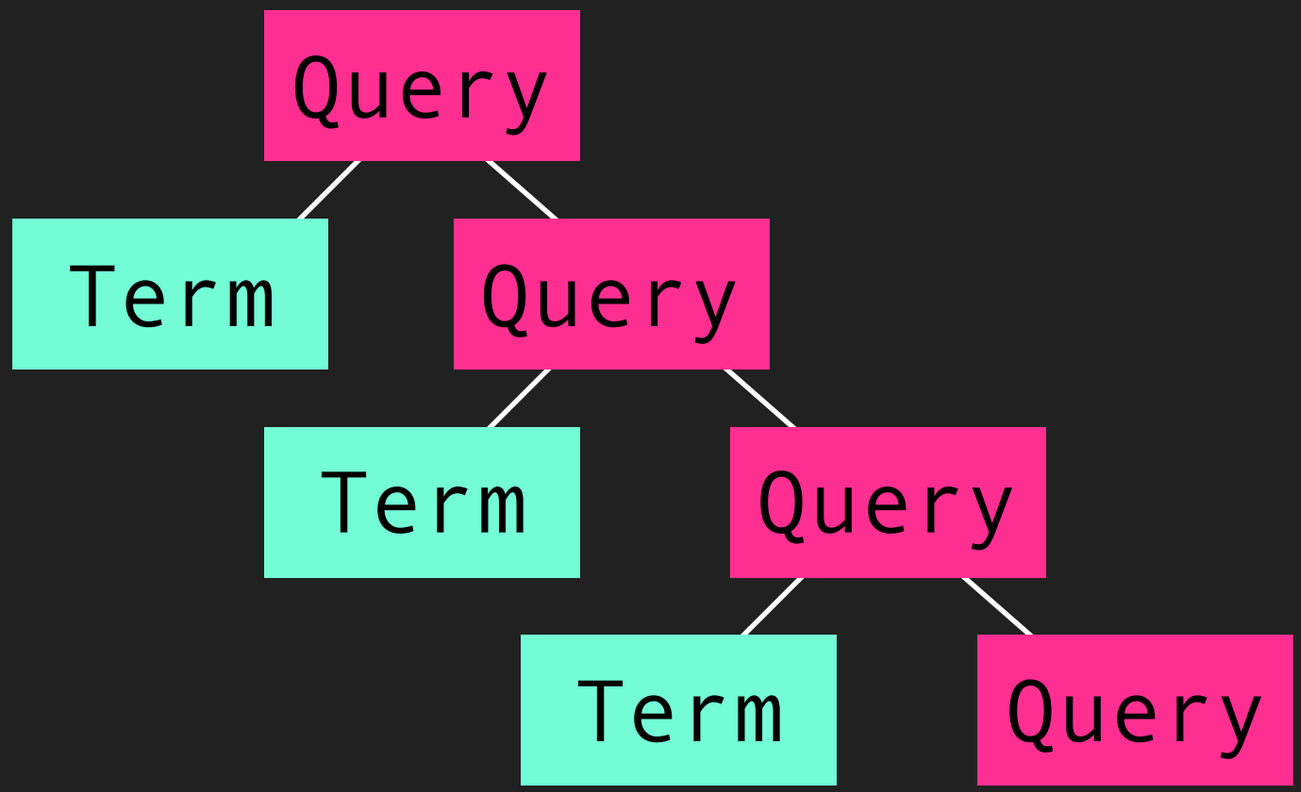
https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Modifier | Entity | Modifier

Entity | Modifier | Entity



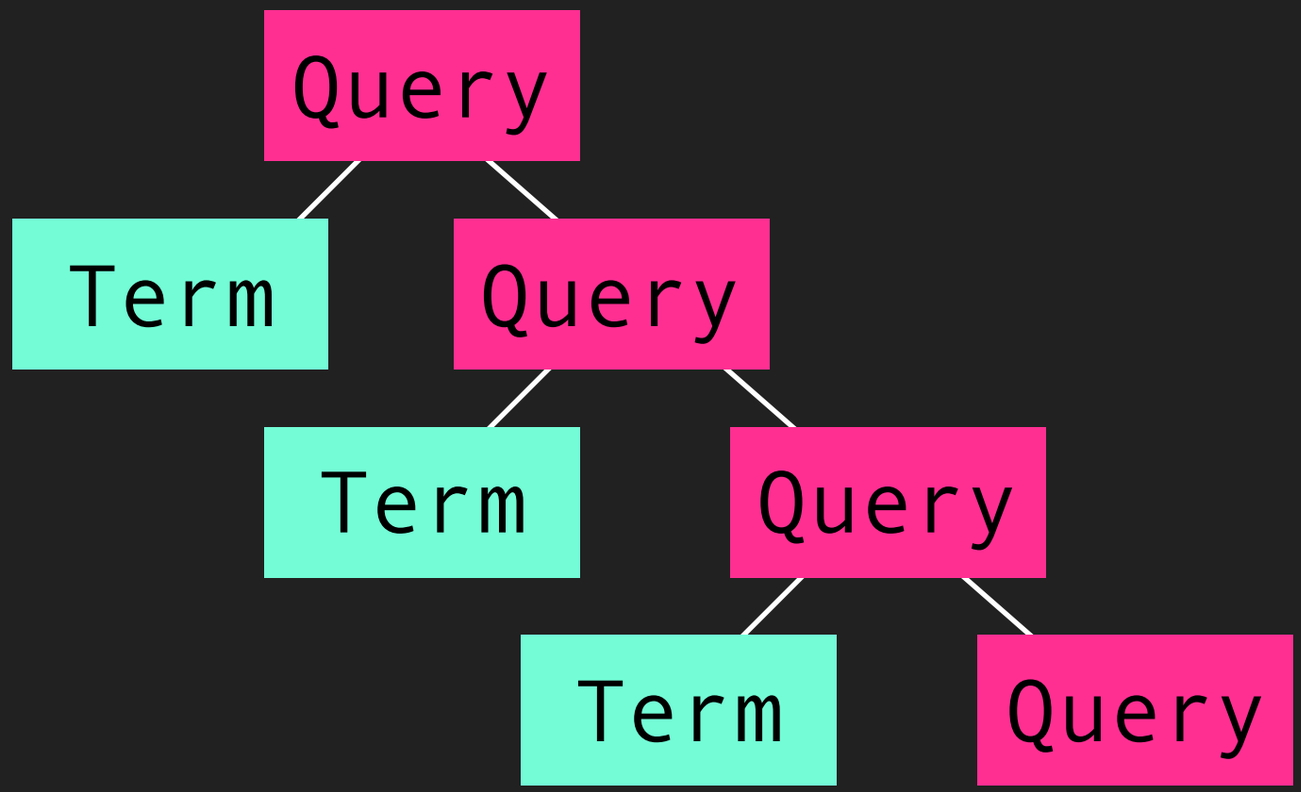
https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

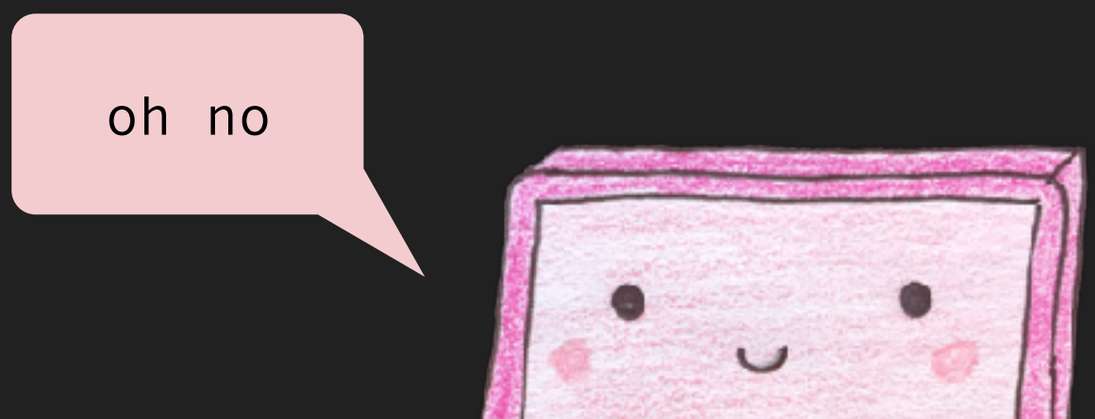
@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Modifier | Entity | Modifier

Entity | Modifier | Entity



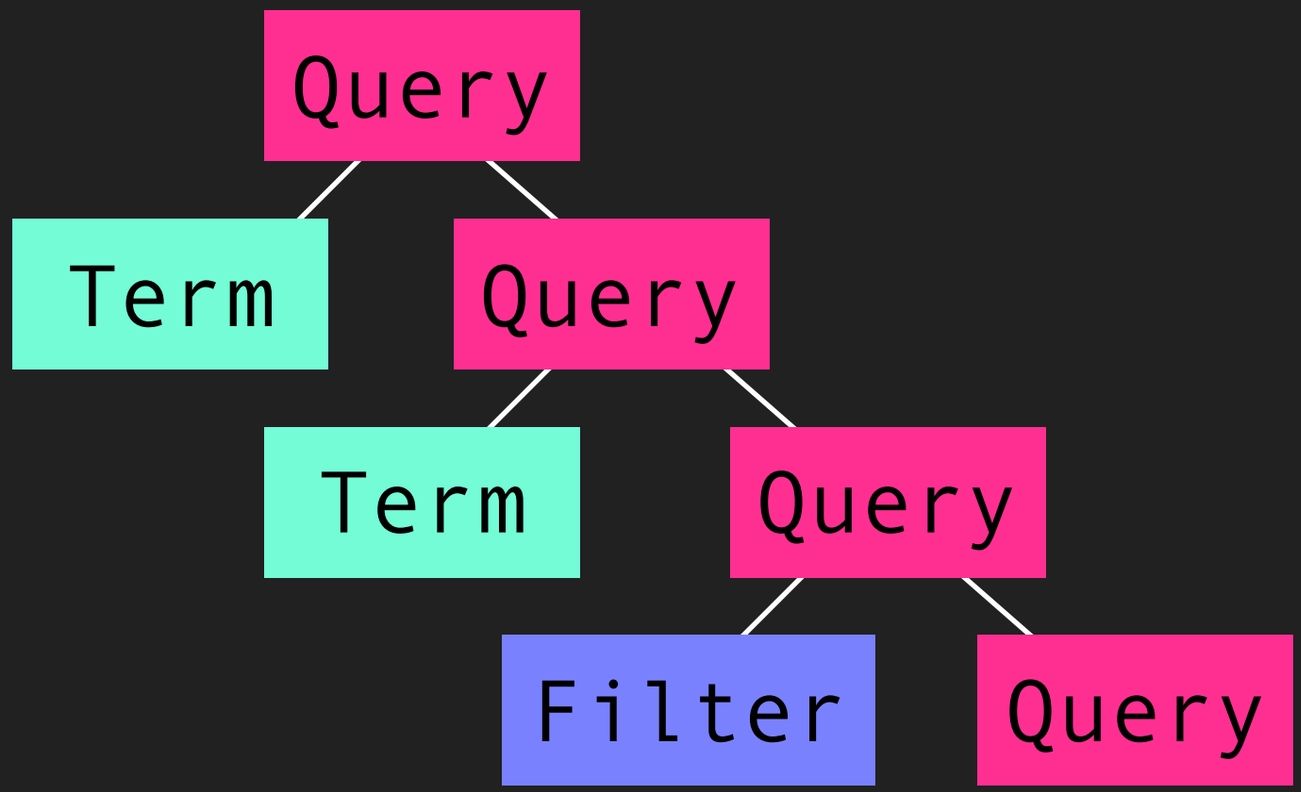


@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Modifier | Entity | Modifier

Entity | Modifier | Entity



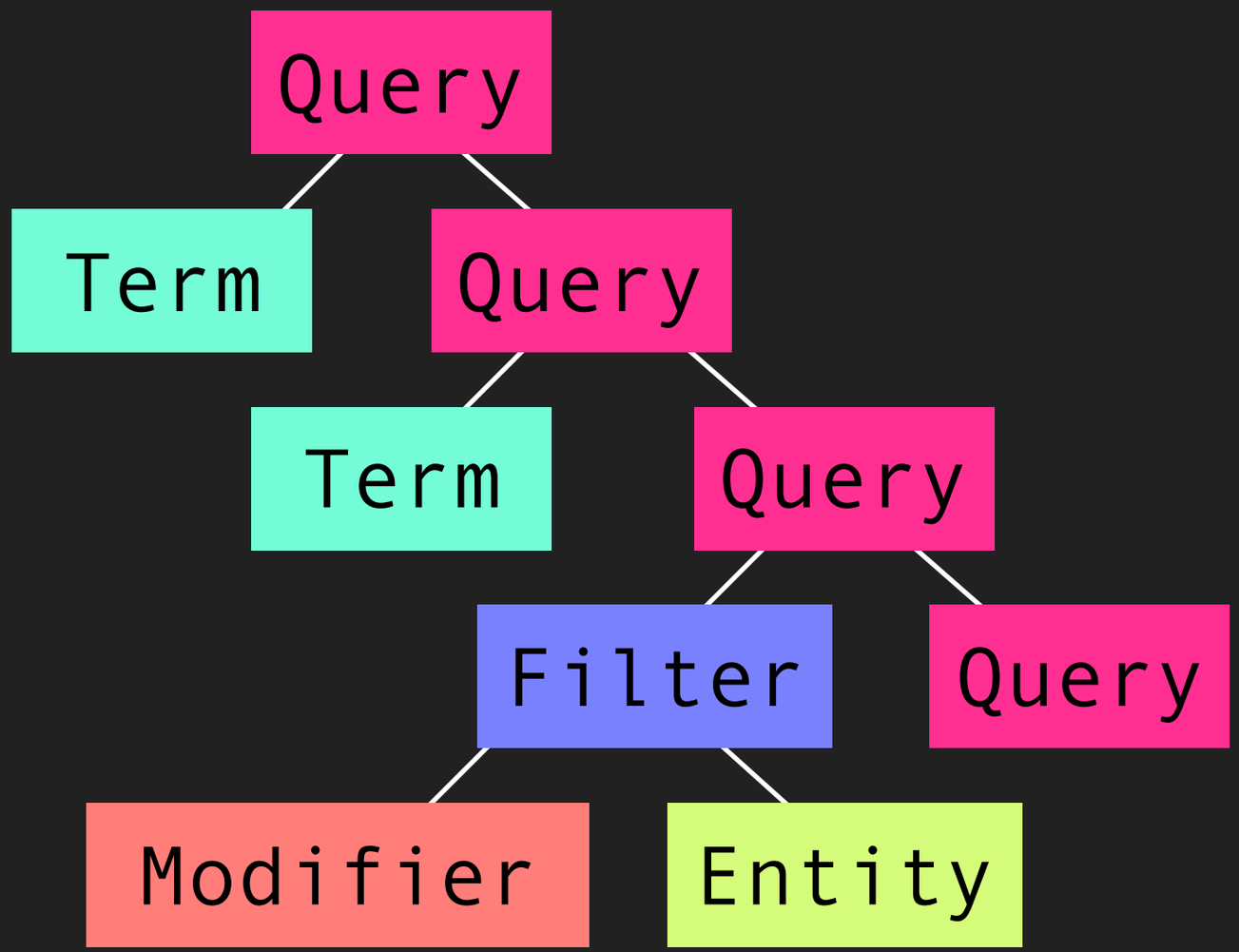
https://www.cs.umd.edu/~mvz/cmsc430-s07/M08topdown.pdf

@zeigenvector
jenna.is/at-jsconfhi

real world: parsers

Modifier

Entity | Modifier | Entity



Grammars!

Lexers!

Tokens!

Parsers!

Trees!

@zeigenvector
jenna.is/at-jsconfhi

"thank you"

"JSConf Hawaii"

from: @zeigenvector

jenna.is/at-jsconfhi

extra credit

the "Chomsky hierarchy" describes different classes of formal grammars

Stop! Grammar time!
♪ ♪ ♪

MC Hammer - U Can't Touch This
https://en.wikipedia.org/wiki/Noam_Chomsky
https://en.wikipedia.org/wiki/Chomsky_hierarchy

extra credit

Type 0: Recursively Enumerable

Type 1: Context-Sensitive

Type 2: Context-Free

Type 3: Regular

https://en.wikipedia.org/wiki/Chomsky_hierarchy

@zeigenvector
jenna.is/at-jsconfhi