Using "peg.js" to make a pizza-based programming language for kids (and anyone else that likes pizza or fractions (or both))

Using "peg.js" to make a pizza-based programming language for kids (and anyone else that likes pizza or fractions (or both))

- What is peg.js
- Syntax rules - easy
- Example
- Syntax rules – harder
- Running a program using a parser
- Audience participation
- Other uses

What is peg.js

- "a simple parser generator for JavaScript …
  …You can use it to process complex data or **computer languages** and build transformers, interpreters, compilers and other tools"

- Online (or npm package or cmd line)

  https://pegjs.org/online
  Easy and fun to experiment with in a browser

- Simple examples
  - cheese or tomato
  - add cheese, add tomato
  - numbers

Example – Fractonio's Pizzeria

```
Make a pizza with cheese and Tomato called p
make a pizza with tomato and mushrooms called q
add cheese to p
Add cheese to q
Remove mushrooms from q
```

Program 1:

```
program = modifypizza*

modifypizza = 'add' WS ingredient WS

ingredient = "cheese"i / "tomato"i / "spinach"i

WS 'whitespace' = [ \t\n]*
```

Program 2

```
number = fraction / wholenumber

wholenumber = [0-9]+

fraction = wholenumber '/' wholenumber
```

Syntax rules

- A bit like regular expressions but a lot of differences

- The easy stuff:

    - Defining rules and using / (ordered choice)
    - Using . + and *
    - Whitespace
    - Case
    - Comments
    - Labelled expressions
    - Adding JavaScript code

See program 1

Syntax rules

- A bit like regular expressions but **a lot of differences**

- The harder stuff:

    - Counts do not exist {3} {3, } - but see later
    - Ordered choice
    - Limited backtracking
    - Greedy and blind
    - & {} and ! {}

Ordered choice & limited backtracking

'/' doesn't really mean 'or'

As long as one of the options matches successfully, then even if the rule is unsuccessful, it will not return to the next option

Example 1 - precedence

```
program = number
number = wholenumber / fraction
wholenumber = [0-9]+
fraction = wholenumber '/' wholenumber
```

Example 2 - cheesey

```
program = (a1 / a2) (b1 / b2)

// a2 b2 matches
// but a1 is chosen and b1 and b2 both fail so the whole thing fails

a1 = "che"
a2 = "chee"
b1 = "ese"
b2 = "sey"
```

```
program = a* b
a = 'X'
b = 'tomato'
```

This does match XXXtomato

```
program = a* b
a = 'tom'
b = 'tomato'
```

This does not match tomtomtomato

It does not backtrack on '*'

```
program = s

s = 'tom' s / 'tomato'
```

This works.

Greedy and blind

It matches as many characters as possible, regardless of other patterns before or after them.

Example "Anything followed by tomato" :

```
program = .* 'tomato'
```

This does not work like a regexp. In fact, it will never match anything.

```
program = s

s = 'tomato' / . s
```

This works

Using !

`! <something>` means:

"try to match 'something'. If you succeed, consider the match failed. Otherwise, do not consume it, just continue"

```
program = (!'tomato'.)* 'tomato'
```

Keep matching '.' (any character) as long as you don't first match 'tomato', and then match 'tomato'

Example 2:

```
program = starttoend

nostart = (!'start'.)*

noend = (!'end'.)*

starttoend = nostart 'start' result:noend 'end' .* {
    return result
}
```

Using ! (continued - keywords)

This does not work (unless you swap the order) - you cannot make a variable called "ifalreadyorderedapizza"

```
program = variable {return "make a variable..."} / keyword {return "this is a keyword..."}

variable  = !keyword [a-z]+

keyword = "else" / "if"
```

This does work:

```
program = variable {return "make a variable..."} / keyword {return "this is a keyword..."}

variable  = !keyword [a-z]+

keyword = ("else" / "if") alone

alone = ![a-z]
```

Using & { < … return bool> }

You can write JavaScript snippets to control the matching better.

Example: Match the regexp a{3} b+

```
program = a:'a'+ &{return a.length === 3} 'b'+
```

Matches aaabbbbbbbbbbb…

Another way to solve the variable/keyword problem

```
{
   var keywords = ["else", "if"];
}

program = variable {return "make a variable..."} / keyword {return "this is a keyword..."}

variable = s: ([a-zA-Z]+) &{ s = s.join("");return !keywords.includes(s) }

keyword = s: ([a-zA-Z]+) &{ s = s.join("");return keywords.includes(s) }
```

Using & to make conditional logic ('if', 'else')

We can use & to conditionally match statements or nothing depending on whether the & function returns true or false.

Example : "if statements"

```
ifstatement = ifstatementpass / ifstatementfail

noendif = (!'endif'.)

ifstatementpass = 'if' WS b:boolean &{return b} WS statement* WS 'endif'

ifstatementfail = 'if' WS b:boolean &{return !b} WS noendif* WS 'endif'
```

Running a program using the parser

```
Make a pizza with cheese and Tomato called p
make a pizza with tomato and mushrooms called q
add cheese to p
Add cheese to q
Remove mushrooms from q
if q's tomato is 1/4:
  print "Great"
endif
```

We need to:

- Add numbers
- Add 'if' statements and booleans like 'is'
- Add print
- Return a syntax tree
- Go through it
- Add some location information

Program 2 –main.js

- Audience participation

- Real-life usage

https://coderwall.com/p/my5xgg/a-simple-peg-js-grammar-to-validate-sql-selects

https://dev.to/barryosull/writing-a-dsl-parser-using-pegjs--4igo

https://www.rapid7.com/blog/post/2016/08/25/implementing-filtering-automated-decisions-with-pegjs-react-mentions/