PEGs in Janet for fun and Profit

intro

- Thanks to Justin Huffman for the topic and a starting point
- Any errors or poor pacing are my own

intro

- Andrew Owen, pronouns: he/him
- Software Developer: games, languages, web apps
- https://junglecoder.com/talks/PEGs/slides.pdf

What we'll be covering

- What is parsing?
- Parsing Knife-work
- Comprehensive parsing
- PEGs (in detail)

What is parsing?

What is parsing?

- Giving Flat Data Structure

Parsing: Flat data to structured data

- **Two general approaches**
- Knife-like, small target, a lot of noise, slice out interesting bits, permissive of incorrcetly formatted data
- Comprehensive sorting of every character to a place and category, usually error if data formatted incorretcly

Parsing Knifework

- Slicing out the interesting bits

Parsing Knifework: Usual suspects

CLI Tools

- cut
- sed
- grep

Programming techniques

- substring
- split
- explode
- gsub
- gmatch

Parsing knifework: Boxed in

- Fixed width data is easy to deal with

```
Data:
boxbox&box
boxHEADbox
boxBODYbox
boxFEETbox
boxbox&box
```

```
Parsing:
tail -n +2 data.txt |
head -n 3 |
cut -b 4-7
```

```
Result:
HEAD
BODY
FEET
```

Parsing knifework: Boxed in

- But doesn't cope with variance well

```
Data:
boxbox&box
boxHEADbox
boxARMSbox
boxFINGERbox
boxBODYbox
boxFEETbox
boxbox&box
```

```
Parsing:
tail -n +2 data.txt |
head -n 5 |
cut -b 4-7
```

```
Result:
HEAD
ARMS
FING
BODY
FEET
```

Parsing knifework: Boxed in

- Patterns can help pick things out

Data:

```
boxbox&box
boxHEADbox
boxARMSbox
boxFINGERbox
boxBODYbox
boxFEETbox
boxbox&box
```

Parsing:

```
sed -E "s/&|box//g" data.txt
```

Result:

```
HEAD
ARMS
FINGER
BODY
FEET
```

Parsing knifework: CSV

- Sometimes a format is simple

Data:

```
x,text,y
10,Hello,20
10,World!,40
```

Parsing:

```
cut -f 2 -d, data.csv
```

Result:

```
text
Hello
World!
```

Parsing knifework: CSV

- Until it isn't

Data:

```
x,text,y
10,"Hello,
World!",20
10,Enjoy this?,60
```

Parsing code:

```
cut -f 2 -d, data.csv
```

Result:

```
text
"Hello
20
Enjoy this?
```

Comprehensive parsing

- Parsing in Scripts 201

Comprehensive parsing

- Characters don't always mean the same thing

Comprehensive parsing

- String literals, outside `"` means "start a string literal"
- Inside a string literal, `"` means "end a string literal"

Tools for comprehensive parsing

Programs

- yacc/lexx
- ANTLR
- jq
- csvkit
- nushell
- Powershell

In Code

- Big Switch Statements
- Argument Parsing
- Recursive Descent Parsing
- Parser Combinators
- *PEGs*

Parsing Expression Grammars

- **PEGs model Recursive Descent parsers**
- Loosely equivalent to a collection of functions that
  - Call into each other
  - Switch on input tokens
  - Build up a structure

Parsing Expression Grammars

- **PEGs in use**
  - Janet uses PEGs in lieu of regex
    - it even has an `re` library that uses a PEG to compile a subset of regex syntax to another PEG
  - Python's parser got switched to a PEG in 2020 (PEP 617)
  - DuckDB plans on switching to a PEG parser for flexibility (https://duckdb.org/2024/11/22/runtime-extensible-parsers.html)

Parsing Expression Grammars

- **PEG Libraries**
  - Lua, lpeg, has name recognition
  - Janet, has PEGs built in
  - (many other exist for other languages, Raku's grammars are similar)

Janet

- **Janet, a quick review**
  - https://janet-lang.org
  - Built as a lisp-alike language, macros, parens
  - Inspired by Lua & Clojure, about 2x-3x bigger than Lua
  - Packs a lot into a small package (cross platform async runtime, PEGs, etc)

```
# FizzBuzz in Janet
(defn divides [by n] (zero? (mod n by)))
(each n (range 101)
      (cond (divides 15 n) (print "FizzBuzz")
            (divides 5 n)  (print "Buzz")
            (divides 3 n)  (print "Fizz")
            true           (print n)))
```

PEGs in Janet

- I'll be using Janet's PEG syntax from here on out, a small example looks like this:

```
(def pattern '(* "abc" "def"))
```

Janet: Macros

- PEGs in Janet use macros and quotation
- Mostly to keep default Janet logic from applying,
  so that the PEG engine can turn the AST into a PEG directly

```
'(this is a quoted list)
'{:and this :is a quoted table}
```

PEGs in Janet: Constants

- Numbers match that number of characters

```
'12 # Matches any 12 characters
```

- Strings match themselves

```
"foo" # Matches "foo" literally
```

PEGs in Janet: Sequence

```
'(* "a" "b" "c")
# or
'(sequence "a" "b" "c")
```

Matches the string **"abc"**

PEGs in Janet: Choice

```
'(+ "a" "b" "c")
# or
'(choice "a" "b" "c")
```

- Matches "a", "b" or "c"

PEGs in Janet: Range

- Range matches any character in the given range(s)

```
'(range "04" "59")
# same as
'(range "09")
# or
'(+ "0" "1" "2" "3" "4"
          "5" "6" "7" "8" "9")
```

- Matches `"0"`, `"9"` and so on

PEGs in Janet: Repeating a pattern

```
# 0 or more repetitions
'(any (range "09"))

# 1 or more repetitions
'(some (range "09"))

# 2 to 4 repetitions
'(between 2 4 (range "09"))

# A specific number of repetitions
'(repeat 4 (range "09"))
```

PEGs in Janet: Naming Patterns

```
'{ :main (some :digit)
   :digit (range "09")}
```

Using a table allows you to name patterns, and then use them in other parts of the grammar.

PEGs that used named patterns start at **`:main`**

PEGs in Janet: Capturing output

```
'{
 :digit (range "09")
 :part (capture (between 1 3 :digit))
 :main (* :part "." :part "." :part "." :part)
}
```

- Given "192.168.0.2", matches and returns `@["192" "168" "0" "2"]`

PEGs in Janet: Capturing output

- Captures in Janet PEGs
  - Manipulate a "capture stack"
  - **`(capture PATT)`**
    - captures all of the characters matched in PATT, pushes them to the capture stack
  - **`(drop PATT)`** drops any captures from PATT

PEGs in Janet: Capturing output

- **`(constant "foo")`**
  - puts "foo" on the capture stack, does not advance the match

`'(* (constant "foo") (constant "bar"))`

- gives: `@["foo", "bar"]` as a result, no matter the input string

PEGs in Janet: Capturing output

- **`(accumulate PATT)`**
  - joins all of the captures in PATT into one string
- ```
  '{
  :foo (constant "foo")
  :bar (constant "bar")
  :baz (constant "baz")
  :main (accumulate (* :foo :bar :baz))
  }
  ```
- Results in **`"foobarbaz"`**
  ```

PEGs in Janet: Compound matches

- **`(if COND PATT)`**
  - attempt PATT if COND matches
- **`(if "A" 1)`**
  - Will match one character as long as it's "A"
- **`(if "/" 3)`**
  - Will match "/" and any two characters

PEGs in Janet: Compound matches

- **`(if-not COND PATT)`**
  - attempt PATT if COND does not match
- **`(if-not "\n" 1)`**
  - Will match one character as long as it is not newline
- **`(if-not (set "\t\r\n ") 1)`**
  - Will match any character not in the set of tab/return/newline/space

PEGs in Action

- How all of this comes together
  - CSV
  - JSON

PEGs in Action: CSV All at once

```
'{
  :nl (+ "\r\n" "\r" "\n")
  :dquote "\""
  :empty 0
  :space? (any " ")
  :capture-ddquote
     (if (* :dquote :dquote) (* (drop 2) (constant `"`)))
  :char-in-quotes (capture (if-not :dquote 1))
  :separators (+ :dquote "," :nl)
  :textdata (+ (capture (some (if-not :separators 1)))
               (* :dquote
                  (accumulate
                     (any (+ :capture-ddquote :char-in-quotes)))
                  :dquote))
  :field (accumulate (+ (* :space? :textdata :space?) :empty))
  :row (* :field (any (* "," :field)) (+ :nl 0))
  :main (some (group :row))}
```

PEGs in action: CSV

- This is a modified version of the CSV grammar at https://github.com/zenlor/janet-csv

PEGs in Action: CSV 1/4

This is a slightly longer grammar, the code in the following slides are inside the '{ below

```
'{
:nl (+ "\r\n" "\r" "\n")
:dquote "\""
:space? (any " ")
:empty 0
```

PEGs in Action: CSV 2/4

```
:capture-ddquote (if
                   (* :dquote :dquote)
                   (* (drop 2) (constant `"`)))

:char-in-quotes (capture (if-not :dquote 1))
```

PEGs in Action: CSV 3/4

```
:separators (+ :dquote "," :nl)
:textdata
  (+
    (capture (some (if-not :separators 1)))
    (* :dquote
       (accumulate
         (any (+ :capture-ddquote :char-in-quotes)))
       :dquote))
```

PEGs in Action: CSV 4/4

```
:field
  (accumulate (+ (* :space? :textdata :space?) :empty))

:row (* :field (any (* "," :field)) (+ :nl 0))

:main (some (group :row))
}
```

PEGs in Action: CSV All at once

```
'{
  :nl (+ "\r\n" "\r" "\n")
  :dquote "\""
  :empty 0
  :space? (any " ")
  :capture-ddquote
     (if (* :dquote :dquote) (* (drop 2) (constant `"`)))
  :char-in-quotes (capture (if-not :dquote 1))
  :separators (+ :dquote "," :nl)
  :textdata (+ (capture (some (if-not :separators 1)))
               (* :dquote
                  (accumulate
                     (any (+ :capture-ddquote :char-in-quotes)))
                  :dquote))
  :field (accumulate (+ (* :space? :textdata :space?) :empty))
  :row (* :field (any (* "," :field)) (+ :nl 0))
  :main (some (group :row))
 }
```

PEGs in Action: JSON

```
 (def json-parser
~{:null (/ (<- "null") ,|[$ :null])
  :bool (/ (<- (+ "true" "false")) ,|[$ :bool])
  :number (/ (<- (* (? "-") :d+ (? (* "." :d+)))) ,|[$ :number])
  :string (/ (* "\"" (<- (to (* (> -1 (not "\\")) "\""))))
                (* (> -1 (not "\\")) "\"")) ,|[$ :string])
  :array (/ (* "[" :value (any (* :s* "," :value)) "]") ,|[$& :array])
  :object (/ (* "{" :s* :string :s* ":" :value
                (any (* :s* "," :s* :string :s* ":" :value))
                "}") ,|[(from-pairs (partition 2 $&)) :object])
  :value (* :s* (+ :null :bool :number :string :array :object) :s*)
  :unmatched (/ (<- (some 1)) ,|[$ :unmatched])
  :main (some (+ :value "\n" :unmatched))})
```

Example of a json parser in a PEG (using a lot of shorthands)
Source: https://calebfiggers.com/blog/parsing-json-in-13-lines-of-janet/

Tips for working with PEGs in Janet

- Have a number of test cases on hand
- Build up the PEG incrementally
- Remember that you have to account for -every- character

Tips for working with Janet

- If this is your first paren-based language, I found rainbow parens to be very useful
- https://janet-lang.org/api/index.html
  ^ this is where I spend a lot of my "look up a thing" time
- `(doc function)` surfaces much of the same information

Thanks!

- https://janet-lang.org/
- https://junglecoder.com/playgrounds/PEGs/
- https://junglecoder.com/talks/PEGs/slides.pdf
- https://junglecoder.com/zines/pegs/PEG_zine.pdf
- https://junglecoder.com/contact/ if you want to find me