Cooking parsers

with Winnow

Santiago

locations 🇦🇷 🇳🇱

experience 🐍 🦀 TS

kpn

Content

1. Beginnings
2. Parsers
3. Winnow
4. Demo

An Idea 💡

Photo by John Paulsen on Unsplash and edited by me

What idea?

Exchanging and tune recipes

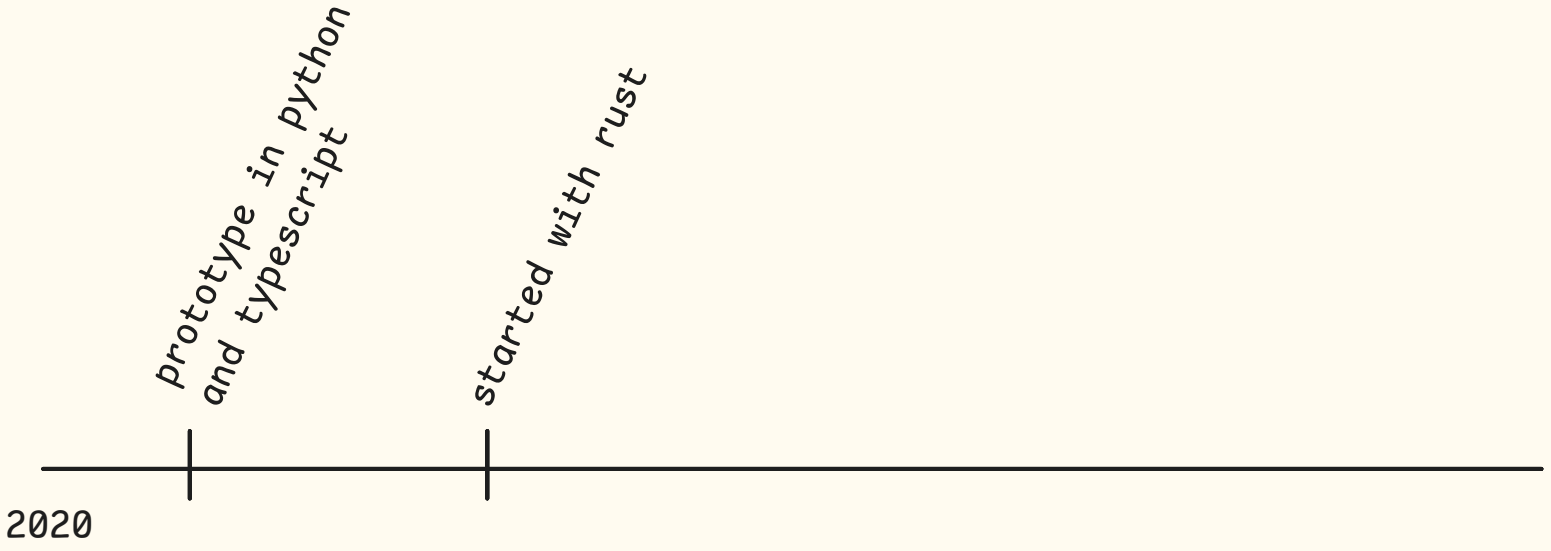




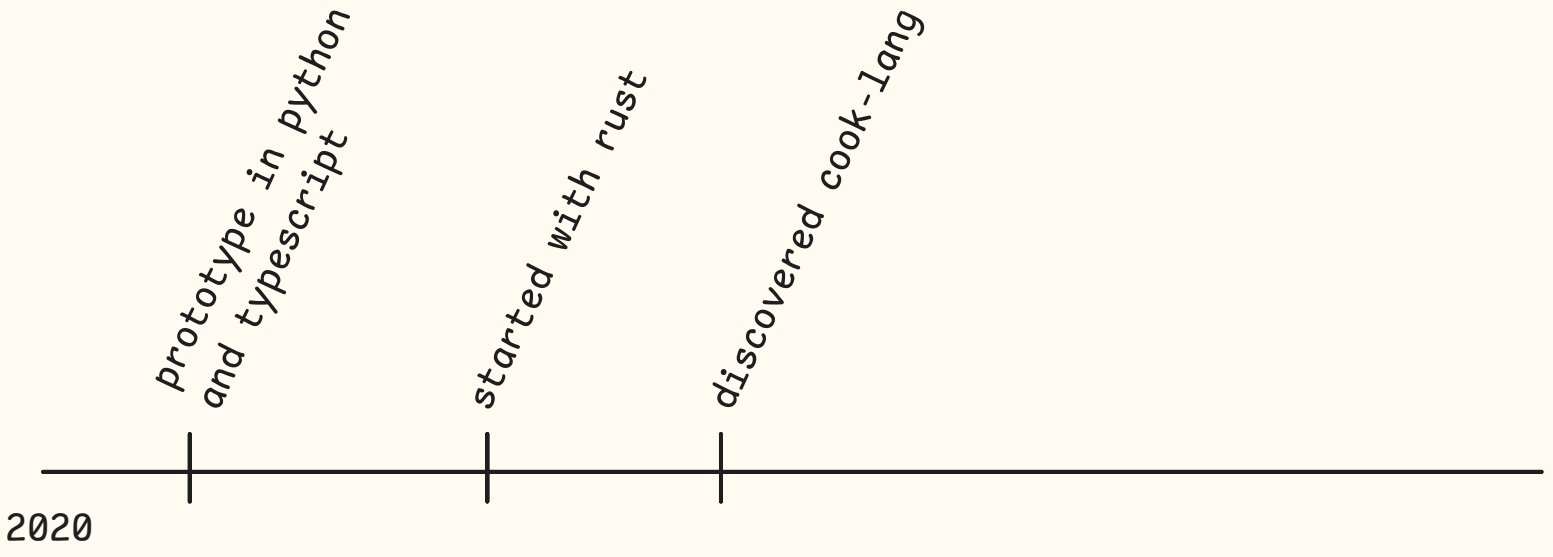



Short history

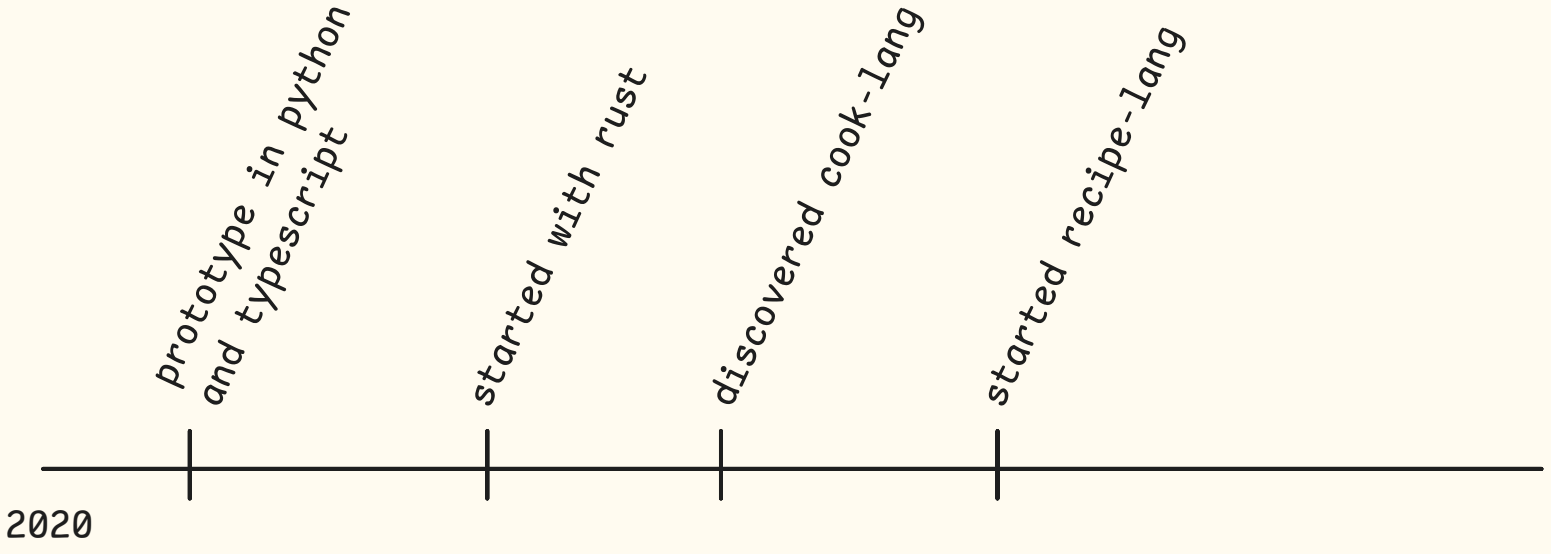
Short history



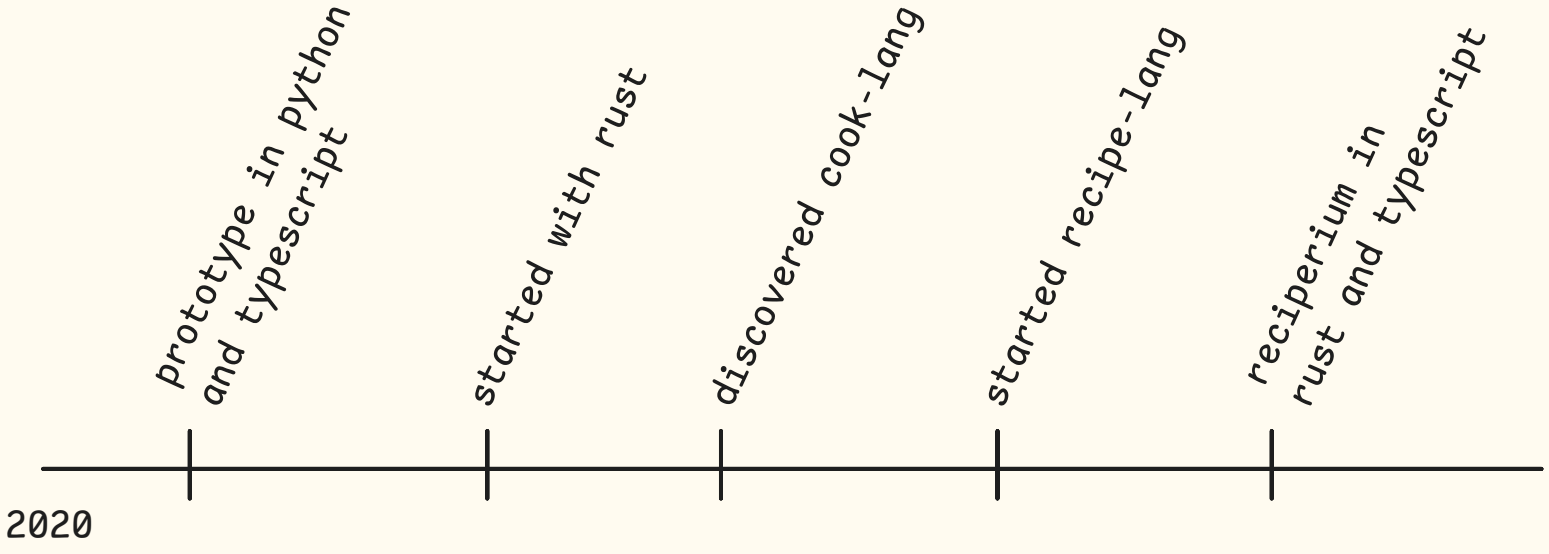
Short history



Short history



Short history



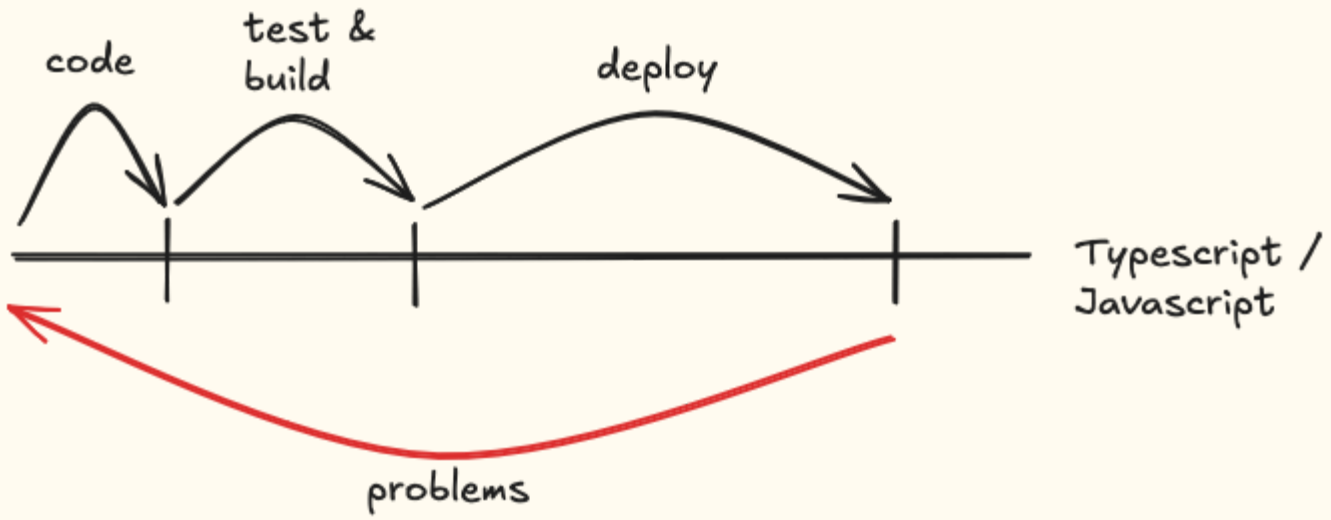
recipe-lang

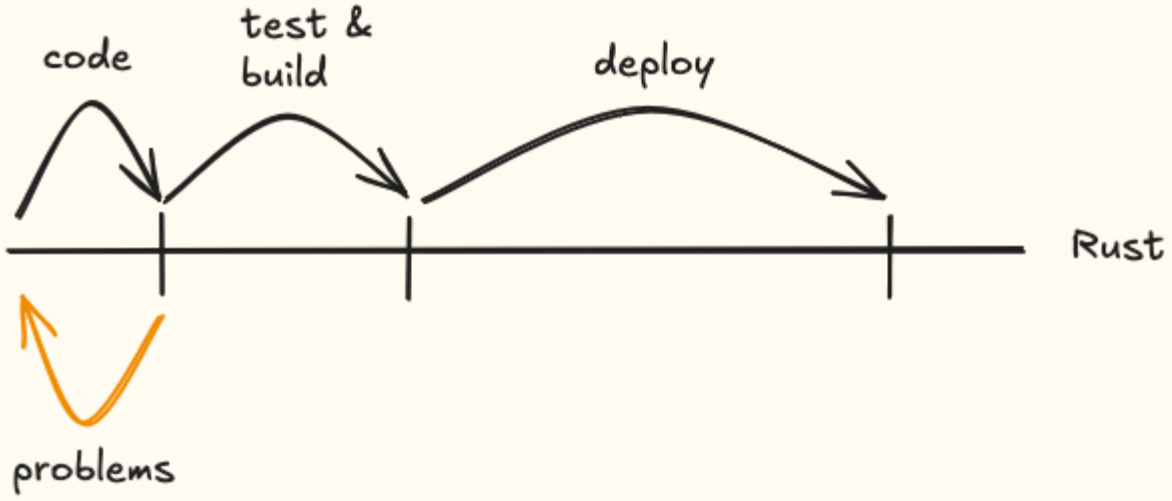
```
Take {potatoes}(3) and wrap them in &{aluminium foil}.
Place them directly into the coals of the grill.
Let cook for t{1 hour} until the potatoes are fork-tender.
```

reciperium/recipe-lang

Why Rust as a solo dev?

INSANE CONFIDENCE


code
test &
build
deploy
Typescript /
Javascript
problems


code
test &
build
deploy
Rust
problems

What was I needing?

- A way to write recipes that was easy for humans and machines

What's the problem with recipes?

Many sections

- ingredients
- materials
- instructions
- and more

recipe-lang

```
Take {potatoes}(3) and wrap them in &{aluminium foil}.
Place them directly into the coals of the grill.
Let cook for t{1 hour} until the potatoes are fork-tender.
```

playground

Why building a parser in rust?

- for fun
- for performance
- for portability

What is a parser?

First: What is a grammar?

a finite set of rules to generate strings

that are in the grammar

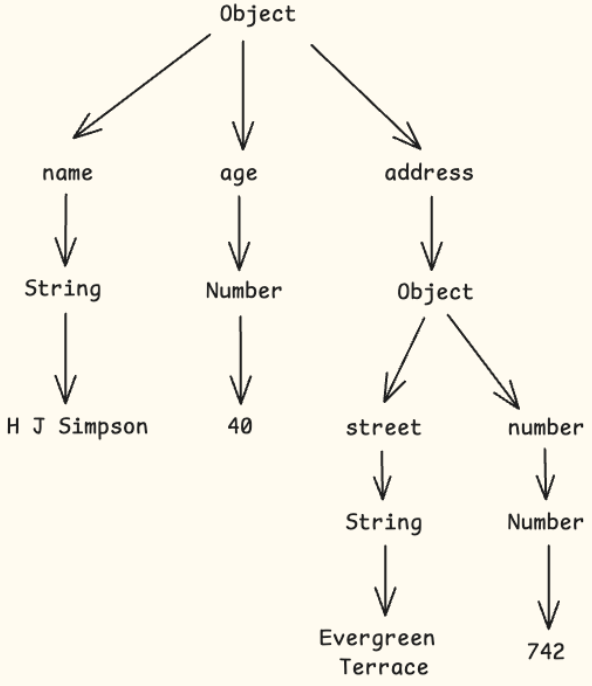
```
SELECT ( ALL | DISTINCT )?
    ( <star> | (
        <select sublist> ( <comma> <select sublist> )* )
    )
```

- SELECT is a terminal
- <select sublist> is a non-terminal

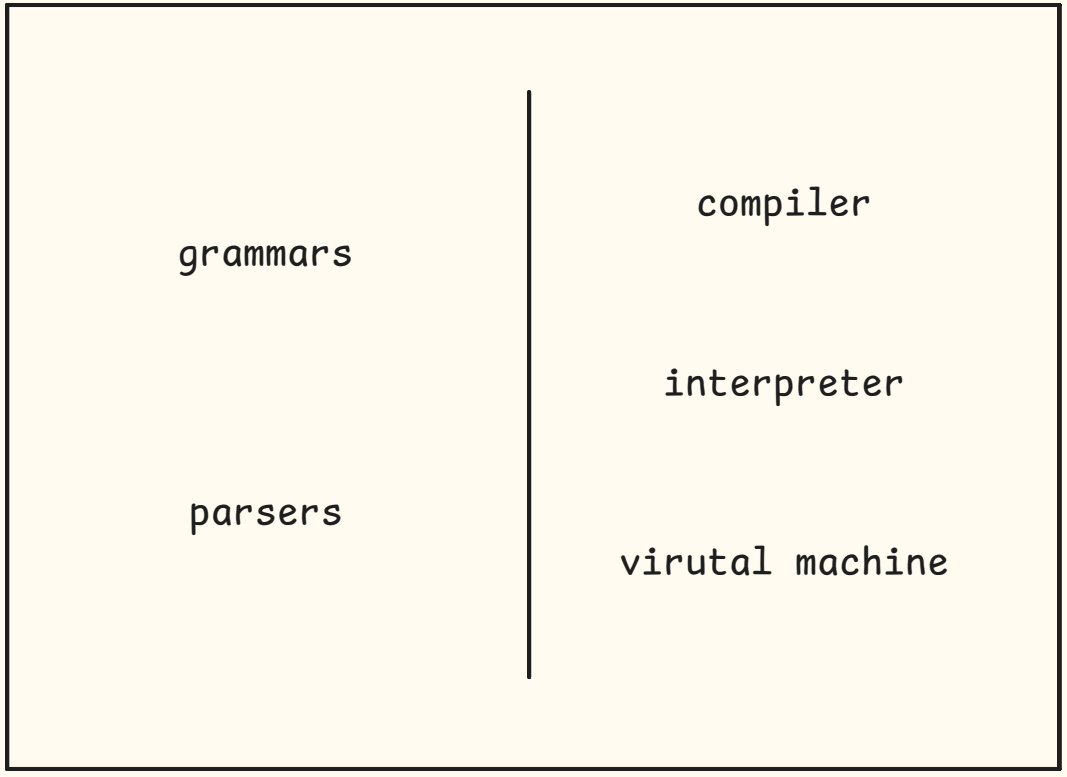
Then: What is a parser?

A tool that maps a series of tokens to grammar rules to create a syntax tree, identifying errors if the token sequence is invalid

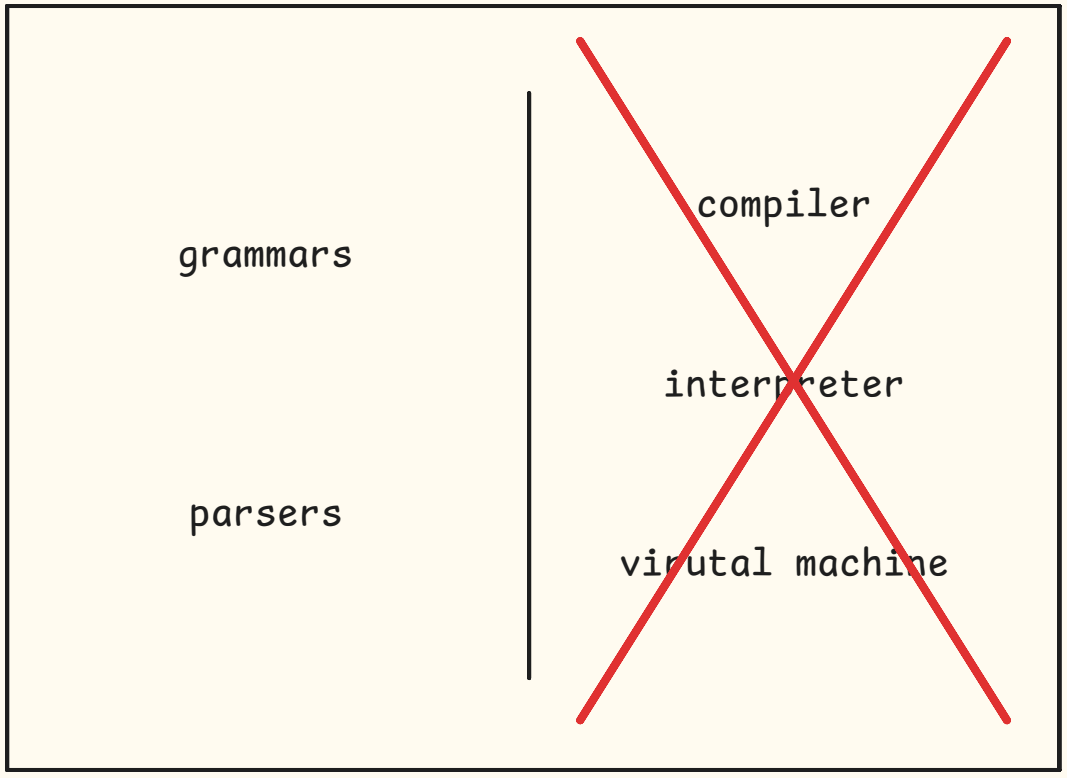
```
{
  "name": "H J Simpson",
  "age": 40,
  "address": {
    "street": "Evrgrn Terace",
    "number": 742
  }
}
```



Disclaimer



Disclaimer



Choosing a parser

Grammars	Combinators	Regex
New syntax	Familiar lang	New syntax
Maintanable	Maintanable	Depends
Reusable*		Reusable*
Rely on macros		

* In theory

Combinator example

winnow

```
fn hex_primary(input: &mut &str) -> PResult<u8> {
    take_while(2, |c: char| c.is_ascii_hexdigit())
        .try_map(|input| u8::from_str_radix(input, 16))
        .parse_next(input)
}
```

docs.rs/winnow/latest/winnow/#example

Grammar example

pest

```
alpha = { 'a'..'z' | 'A'..'Z' }
digit = { '0'..'9' }

ident = { (alpha | digit)+ }

ident_list = _{ !digit ~ ident ~ (" " ~ ident)+ }
```

pest.rs/

Regex

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?
^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-
\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-
z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?|\[(?:(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-
9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])|
[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-
\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

What's in the rust market?

- grammars: pest, peg
- combinators: winnow, nom, chumsky
- regex: regex

and more

Winnow

- Excellent documentation with tutorials
- Very extendable
- Performant
- A fork of `nom` by epage

Setup

```
cargo add winnow
```

Main usage

```
1  use winnow::combinator::alt;
2  use winnow::{PResult, Parser};
3
4  pub fn parse_foobar_bool(i: &mut &str) -> PResult<bool> {
5      alt((
6          "foo".value(true),
7          "bar".value(false)
8      )).parse_next(i)
9  }
10 fn main() {
11     let input = "foo";
12     let out = parse_foobar_bool.parse(input).unwrap();
13     println!("{}", out);
14 }
```

Demo

JSONOO

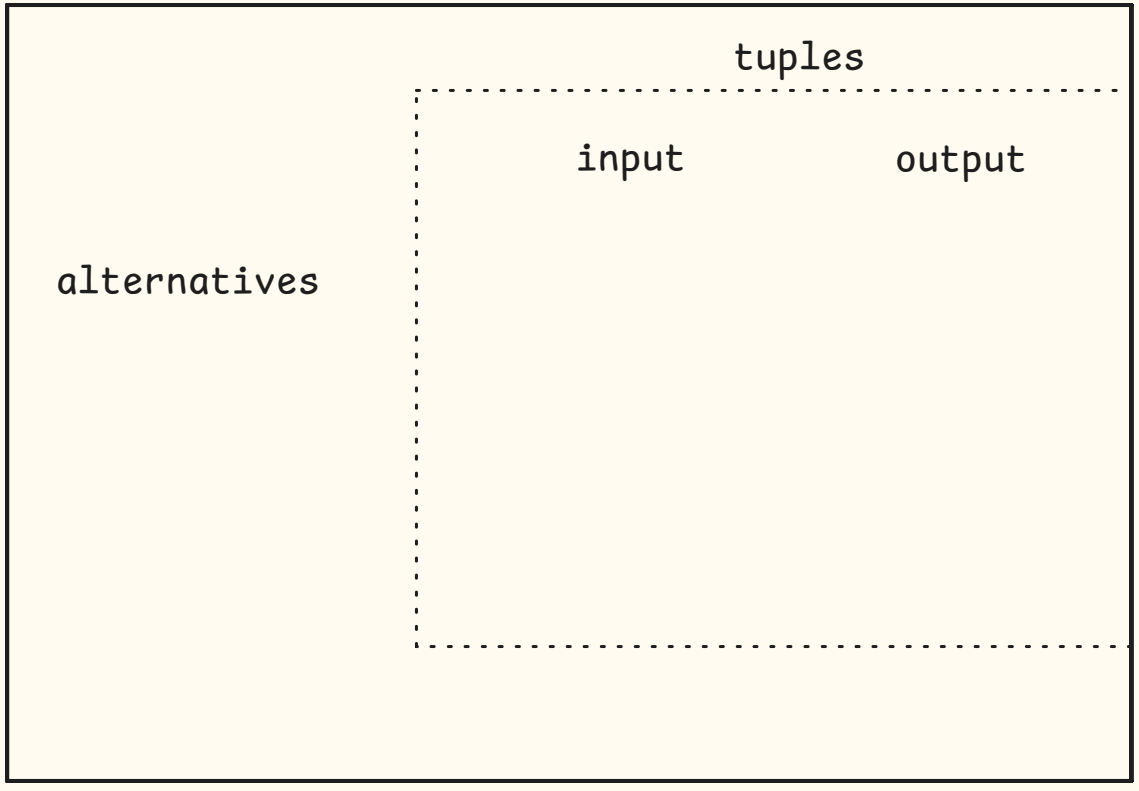
(JSON Object Only)

/Jay • SO • NOOOO/

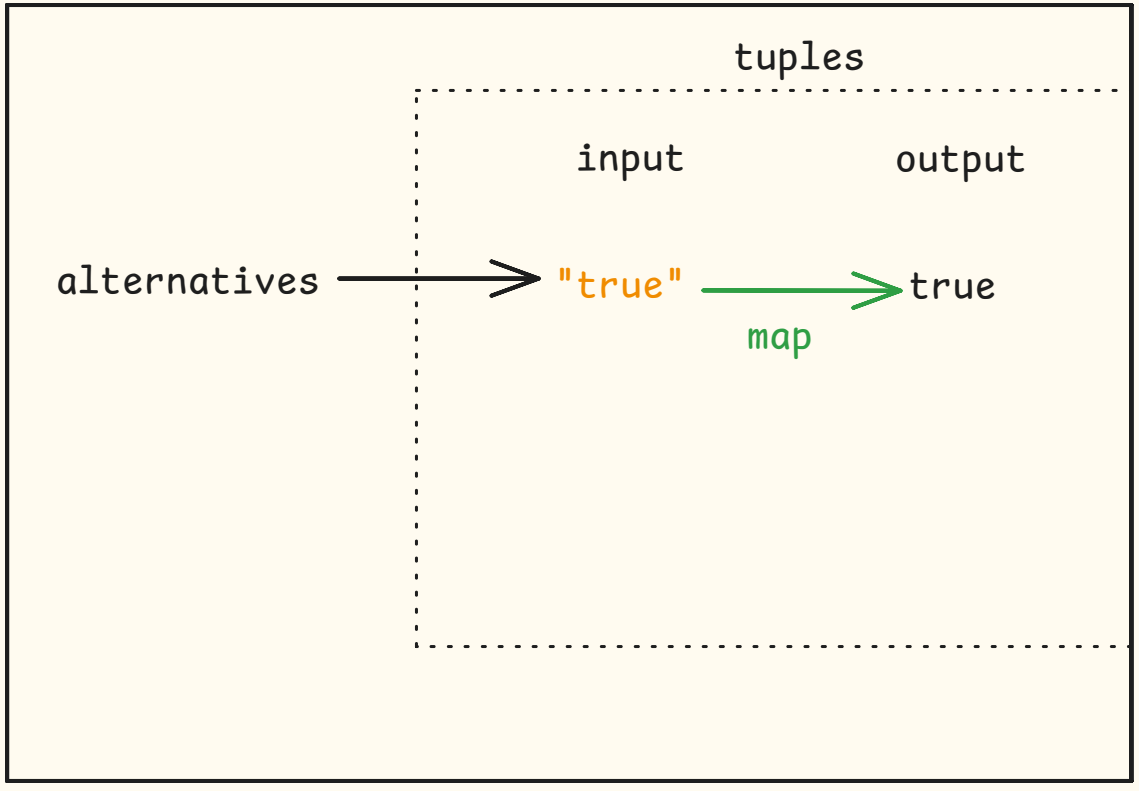
```
{"event": "subscribed", "name": "bruce", "isadmin": false}
{"event": "subscribed", "name": "marta", "isadmin": true}
```

alt

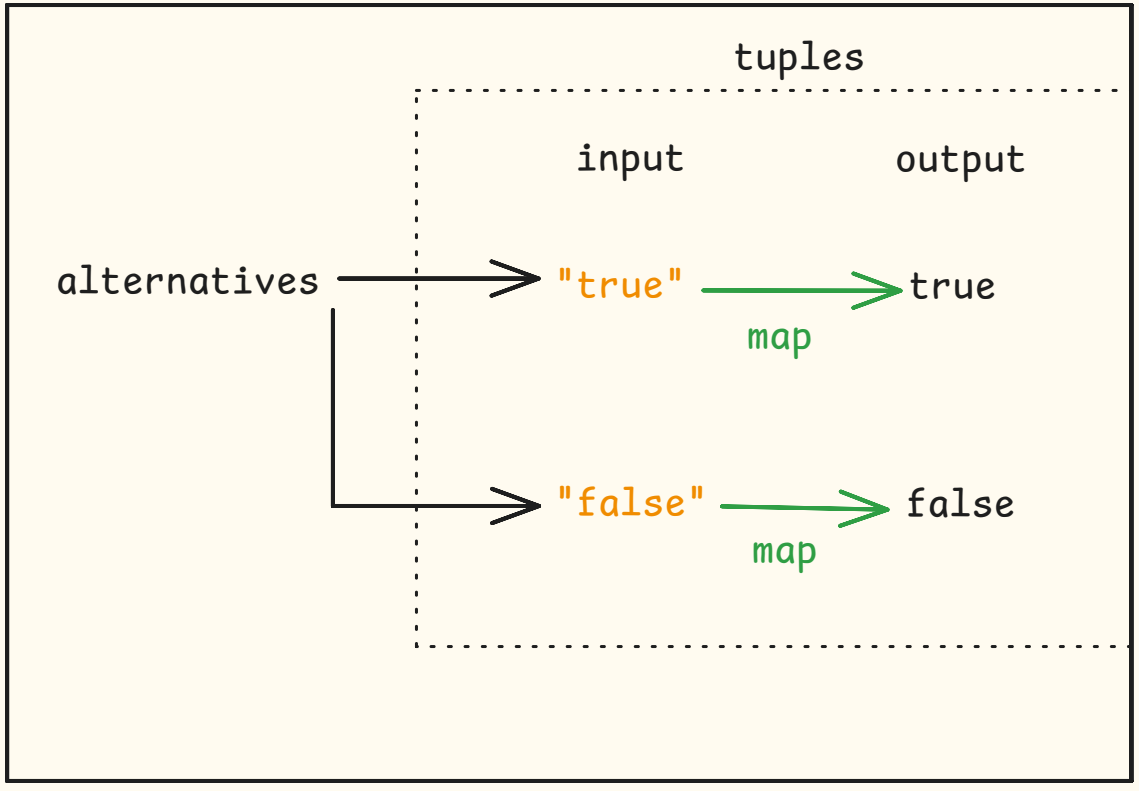
alt



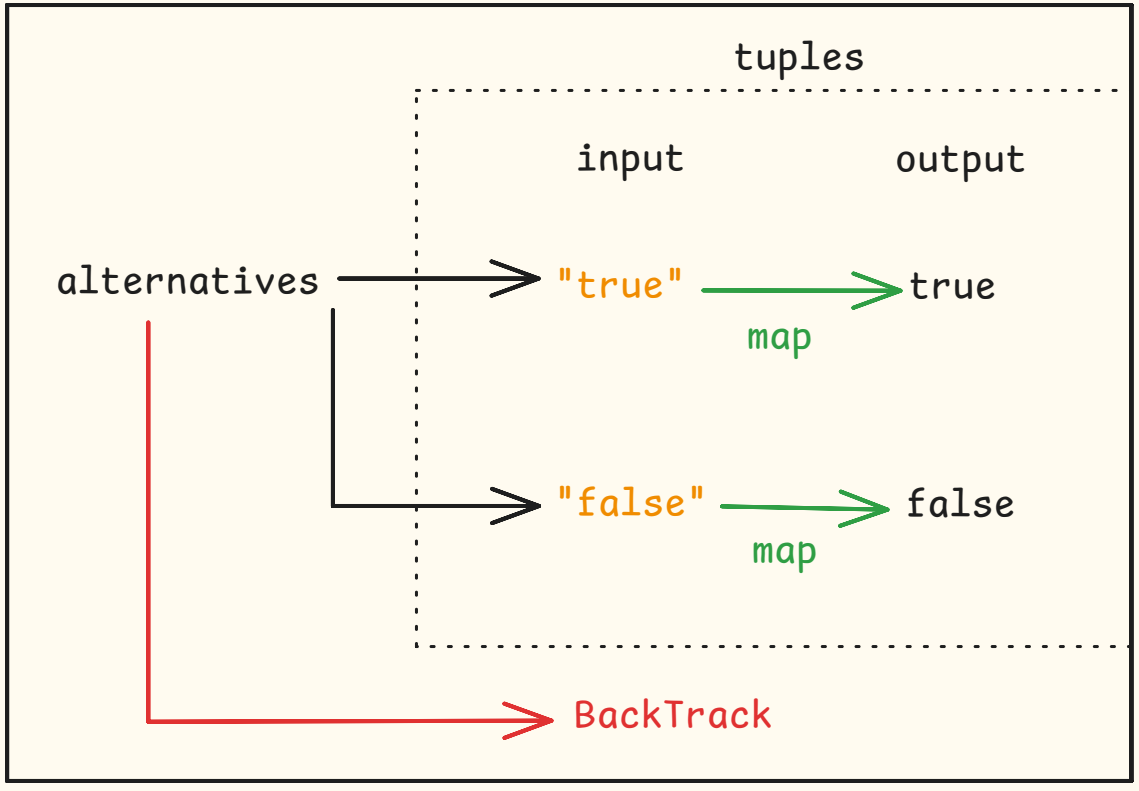
alt



alt


tuples
input
output
alternatives
"true"
true
map
"false"
false
map

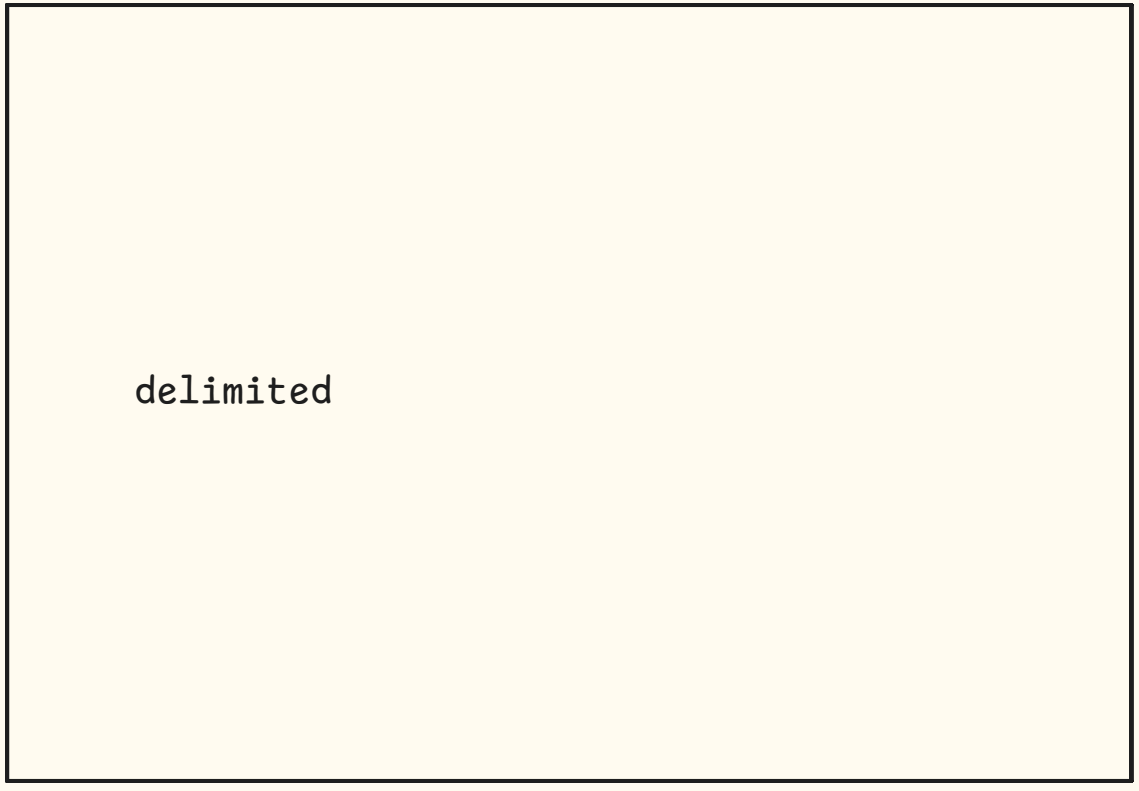
alt


tuples
input
output
alternatives
"true"
true
map
"false"
false
map
BackTrack

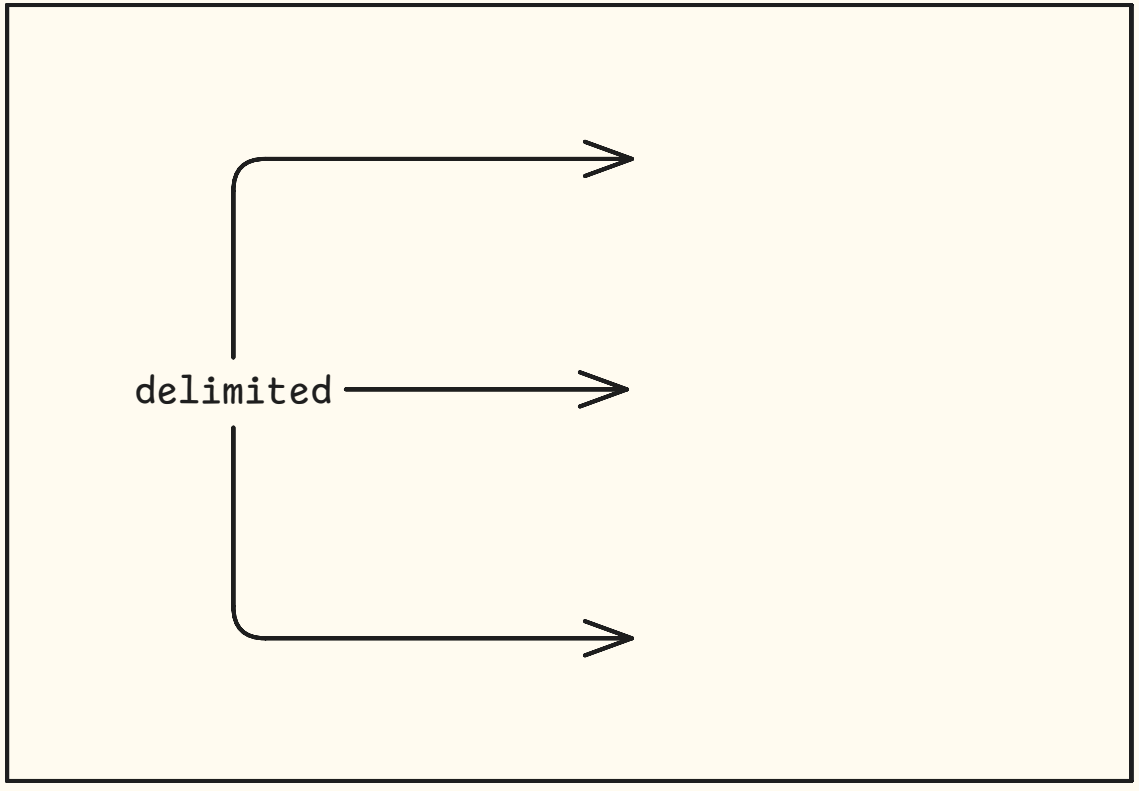
```rust
use winnow::ascii::{alpha1, digit1};
use winnow::combinator::alt;

fn parser(input: &str) -> IResult<&str, &str> {
  alt((alpha1, digit1)).parse_peek(input)
}
```

delimited

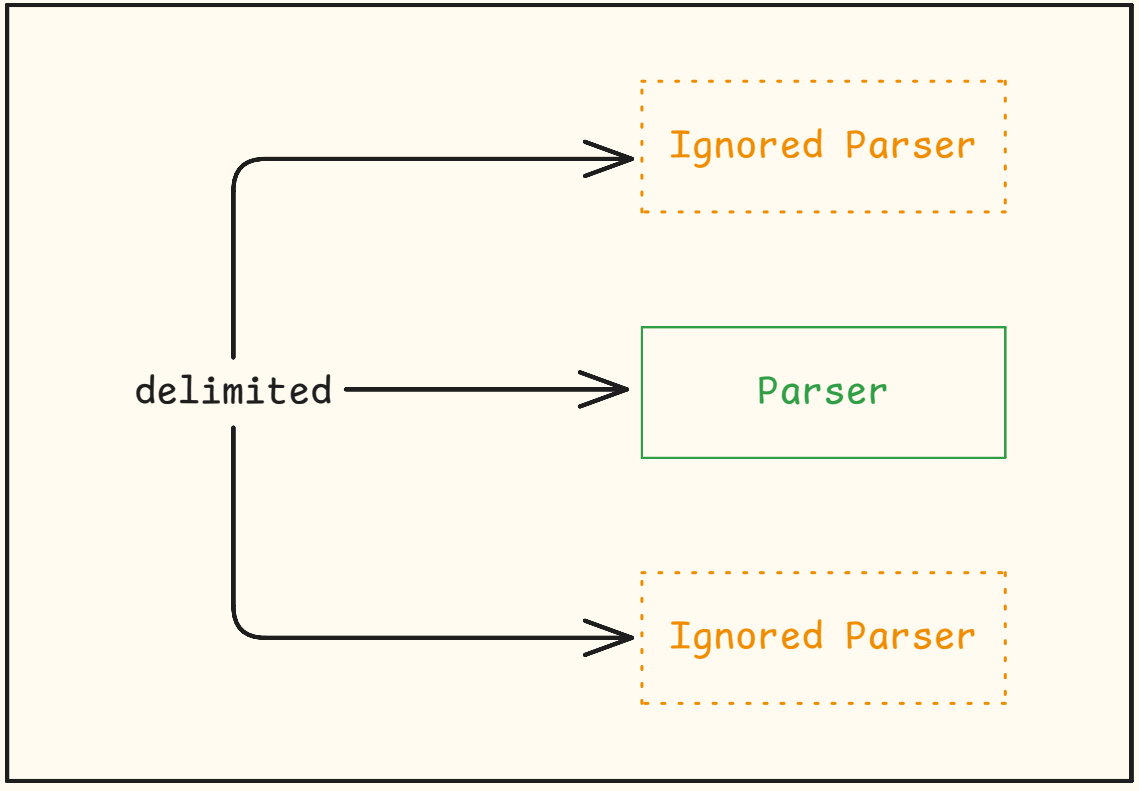
delimited



delimited



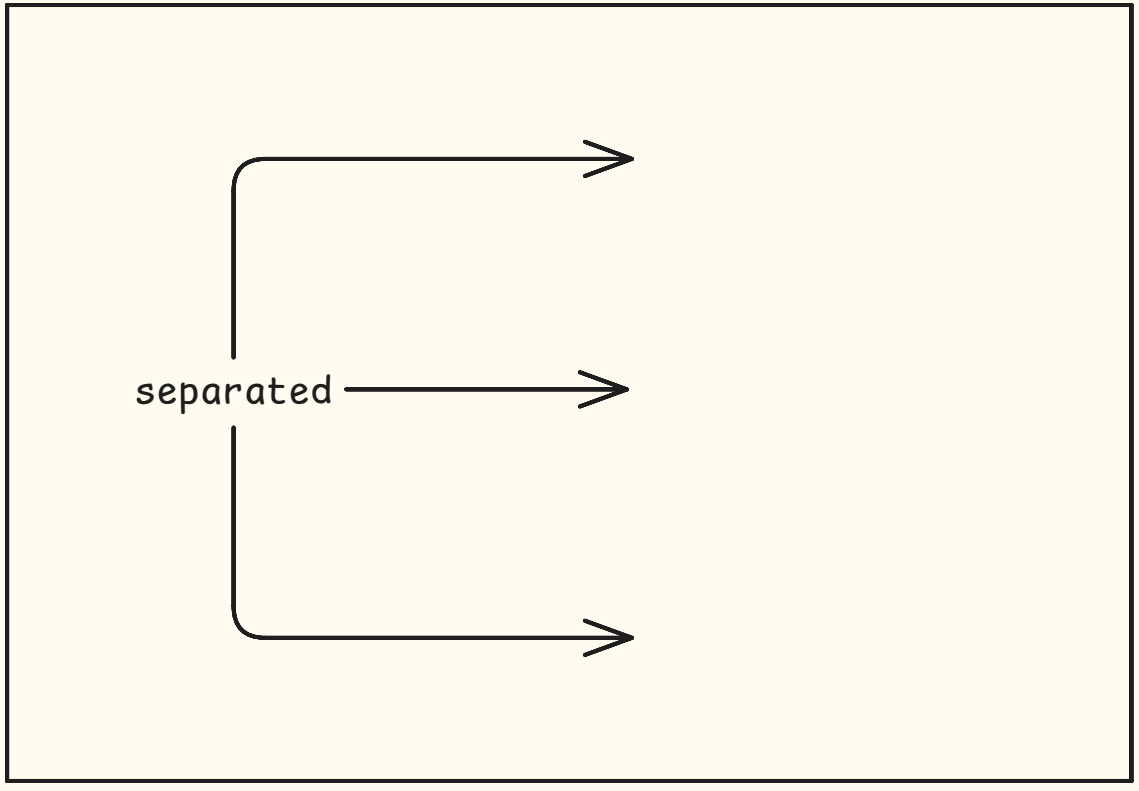
delimited



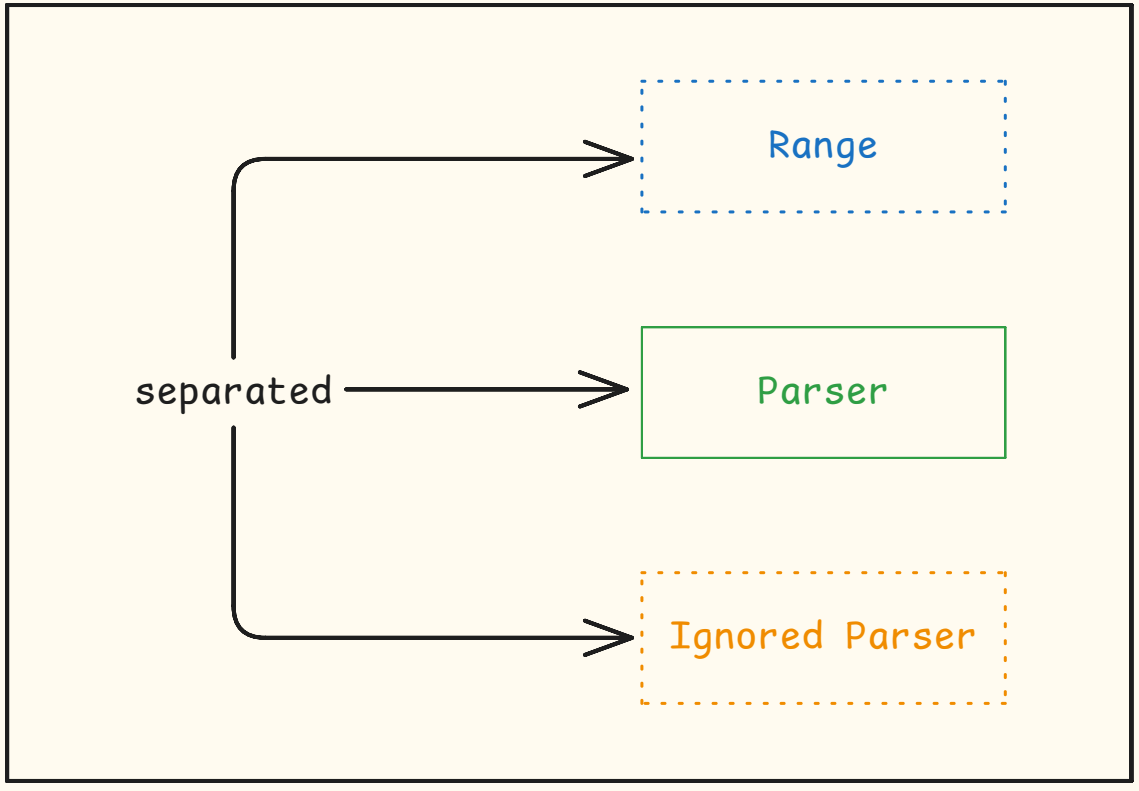
```rust
use winnow::combinator::delimited;

let mut parser = delimited("(", "abc", ")");

assert_eq!(parser.parse_peek("(abc)"), Ok(("", "abc")));
```

separated

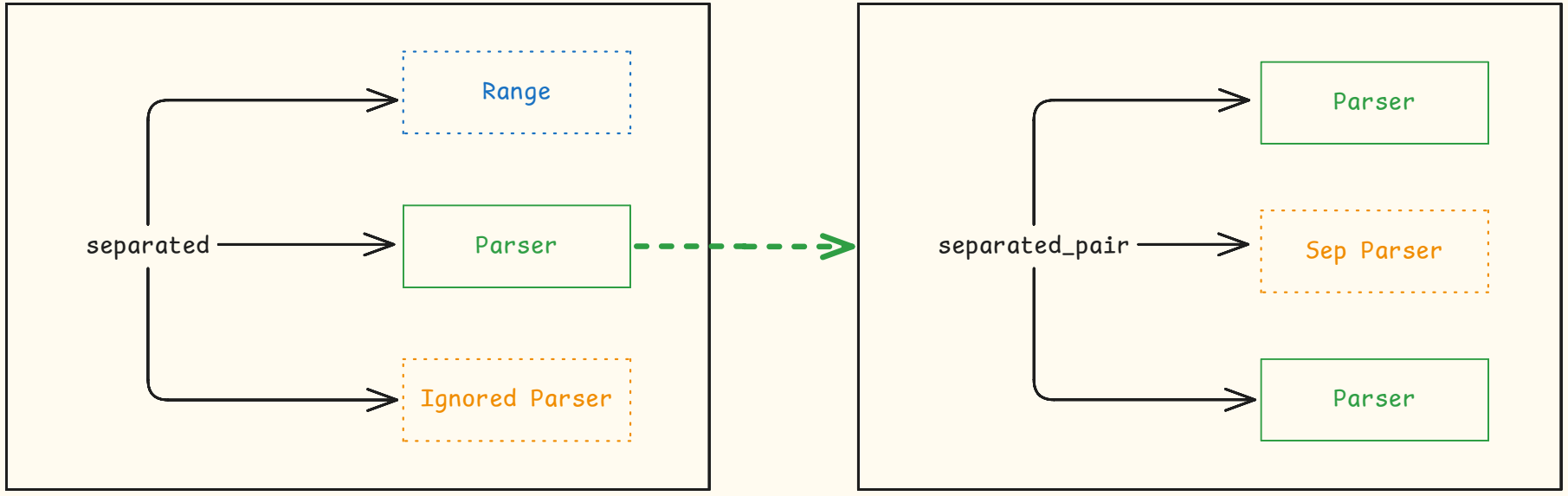
separated



separated



separated


Range
separated
Parser
Ignored Parser
separated_pair
Parser
Sep Parser
Parser

```
// { "key1": "value1", "key2": "value2" }
delimited(
    ("{", space0),
    separated(
        0..,
        separated_pair(parse_string, ":", parse_token),
        (",", space0),
    ),
    (space0, "}"),
)
```

Error handling

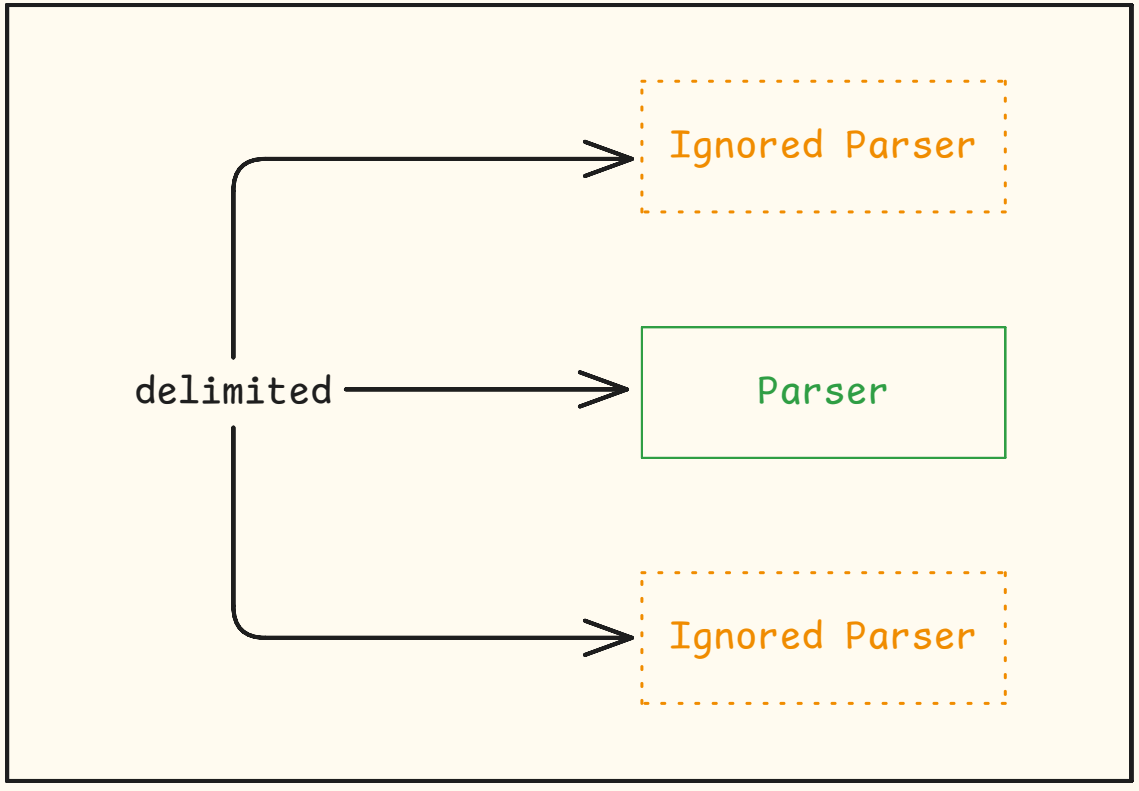
ErrMode

Backtrack: recoverable (default)

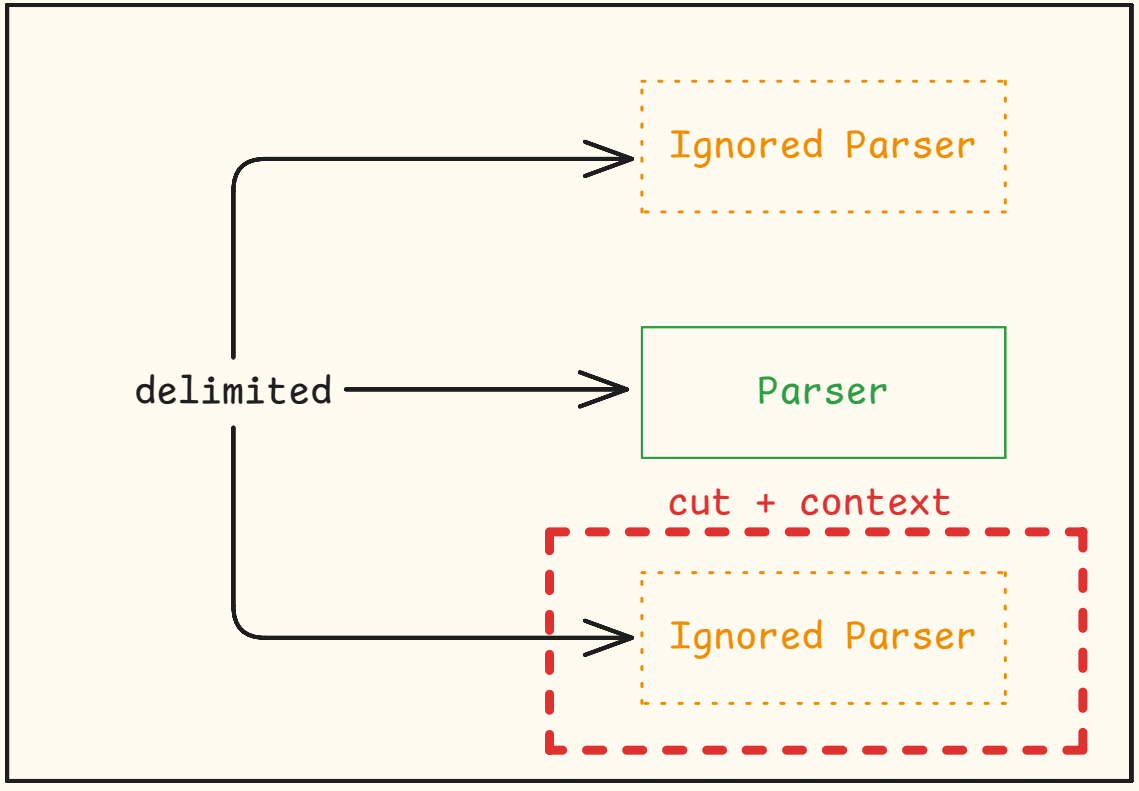
Cut: unrecoverable

~~`Incomplete`~~

delimited



delimited


Ignored Parser
delimited
Parser
cut + context
Ignored Parser

```rust
use winnow::combinator::cut_err;
use winnow::combinator::delimited;

let mut parser = delimited(
    "(",
    "abc",
    cut_err(")").context(
        StrContext::Expected(
            StrContextValue::CharLiteral('}')
        )
    )
);

assert_eq!(parser.parse_peek("(abc)"), Ok(("", "abc")));
```

Context

```
StrContext::Label  // what is currently being parsed
StrContext::Expected  // expected grammar item
StrContextValue::CharLiteral
StrContextValue::StringLiteral
StrContextValue::Description
```

Wrap up

- winnow is a well-documented powerful library
- check out crafting interpreters book
- let me know if you write a parser!

Thanks

@woile@hachyderm.io

woile

santiwilly@gmail.com