Analyse syntaxique avec Tree-sitter

15 nov. 2023

Analyse syntaxique

```
#test
require ["imapflags","fileinto"];

if anyof (
    header :contains "subject" "test"
) {
    stop;
    addflag "\\Seen";
    fileinto "INBOX";
}
```

script sieve

Analyse syntaxique

```
#test
require ["imapflags","fileinto"];

if anyof (
    header :contains "subject" "test"
) {
    stop;
    addflag "\\Seen";
    fileinto "INBOX";
}
```

script sieve

```
{
  actions: [
    { method: "stop" },
    { argument: "seen", method: "addflag" },
    { argument: "INBOX", method: "fileinto" }
  ],
  name: "test",
  rules: [
    {
      field: "subject",
      operator: "contains",
      value: "test"
    }
  ]
}
```

filtre sogo

Analyse syntaxique

```
#test
require ["imapflags","fileinto"];

if anyof (
    header :contains "subject" "test"
) {
    stop;
    addflag "\\Seen";
    fileinto "INBOX";
}
```

```
{
  actions: [
    { method: "stop" },
    { argument: "seen", method: "addflag" },
    { argument: "INBOX", method: "fileinto" }
  ],
  name: "test",
  rules: [
    {
      field: "subject",
      operator: "contains",
      value: "test"
    }
  ]
}
```

script sieve

filtre sogo

Analyse syntaxique

```
#test
require ["imapflags","fileinto"];

if anyof (
    header :contains "subject" "test"
) {
    stop;
    addflag "\\Seen";
    fileinto "INBOX";
}
```

script sieve

```
{
  actions: [
    { method: "stop" },
    { argument: "seen", method: "addflag" },
    { argument: "INBOX", method: "fileinto" }
  ],
  name: "test",
  rules: [
    {
      field: "subject",
      operator: "contains",
      value: "test"
    }
  ]
}
```

filtre sogo

Analyse syntaxique

```
#test
require ["imapflags","fileinto"];

if anyof (
    header :contains "subject" "test"
) {
    stop;
    addflag "\\Seen";
    fileinto "INBOX";
}
```

```
{
  actions: [
    { method: "stop" },
    { argument: "seen", method: "addflag" },
    { argument: "INBOX", method: "fileinto" }
  ],
  name: "test",
  rules: [
    {
      field: "subject",
      operator: "contains",
      value: "test"
    }
  ]
}
```

script sieve

filtre sogo

Analyse syntaxique

```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```

Analyse syntaxique

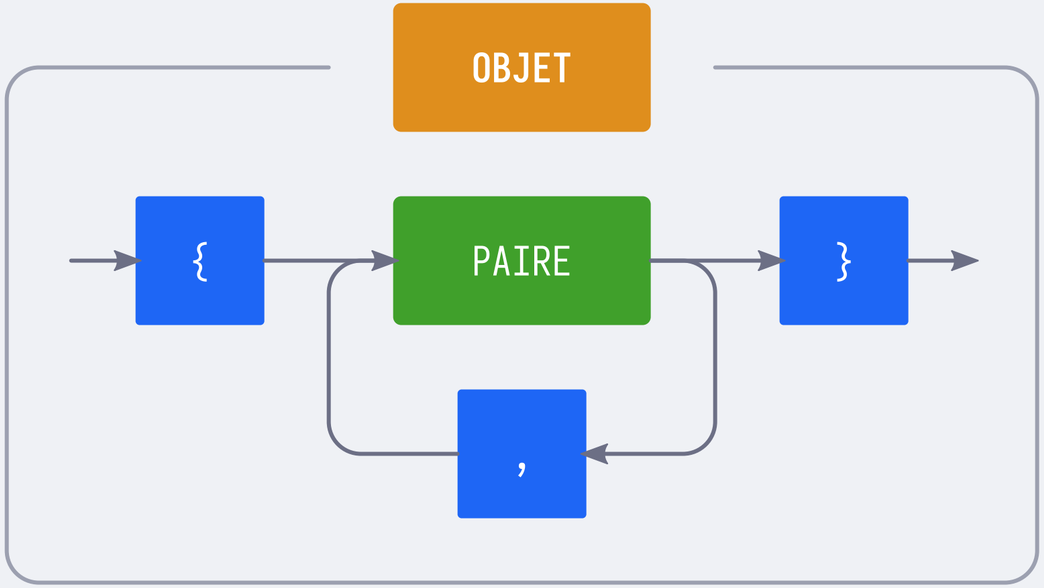
```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```

Analyse syntaxique

```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```

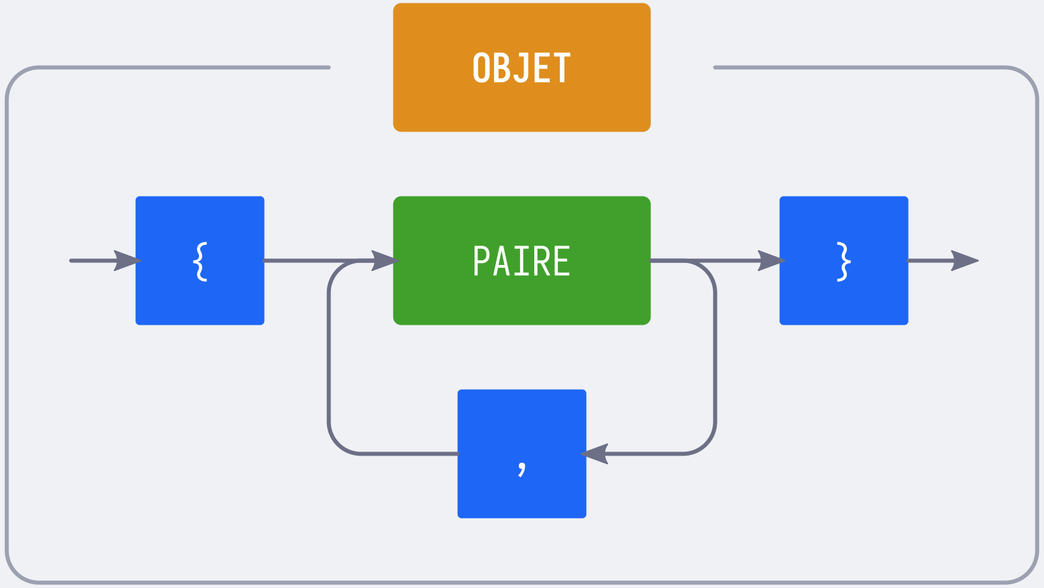
Analyse syntaxique

```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```



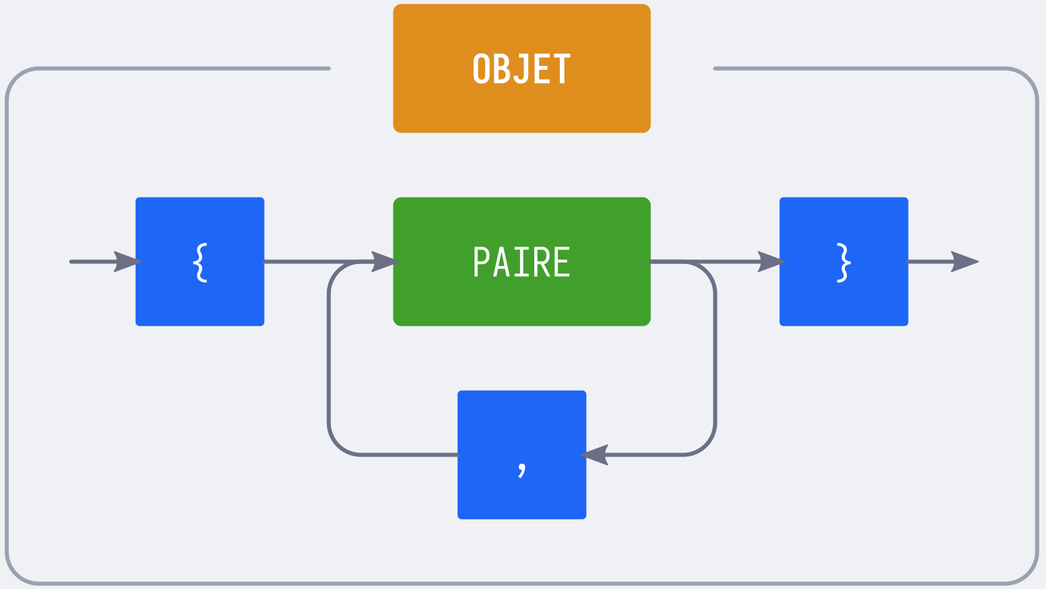
Analyse syntaxique

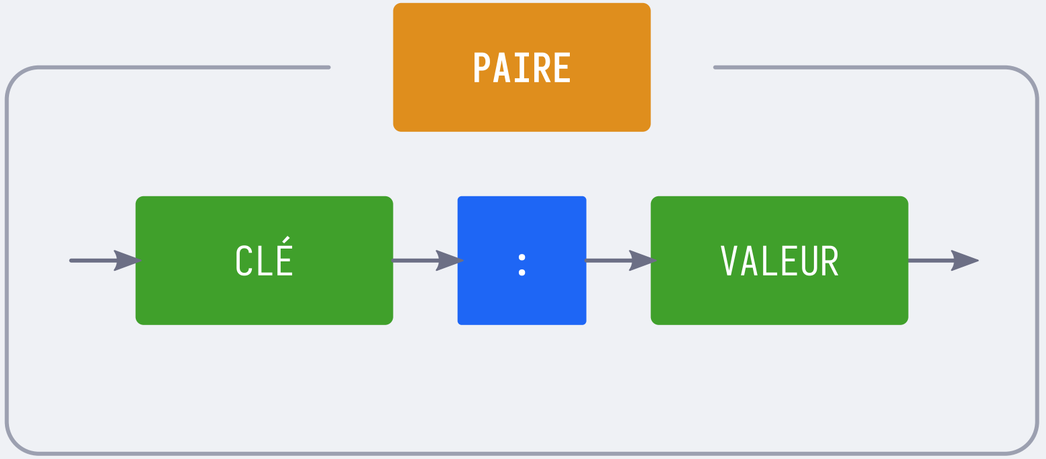
```
{
   Lorem : "Ipsum",
   Dolor : {
      sit : "amet"
   }
}
```



Analyse syntaxique

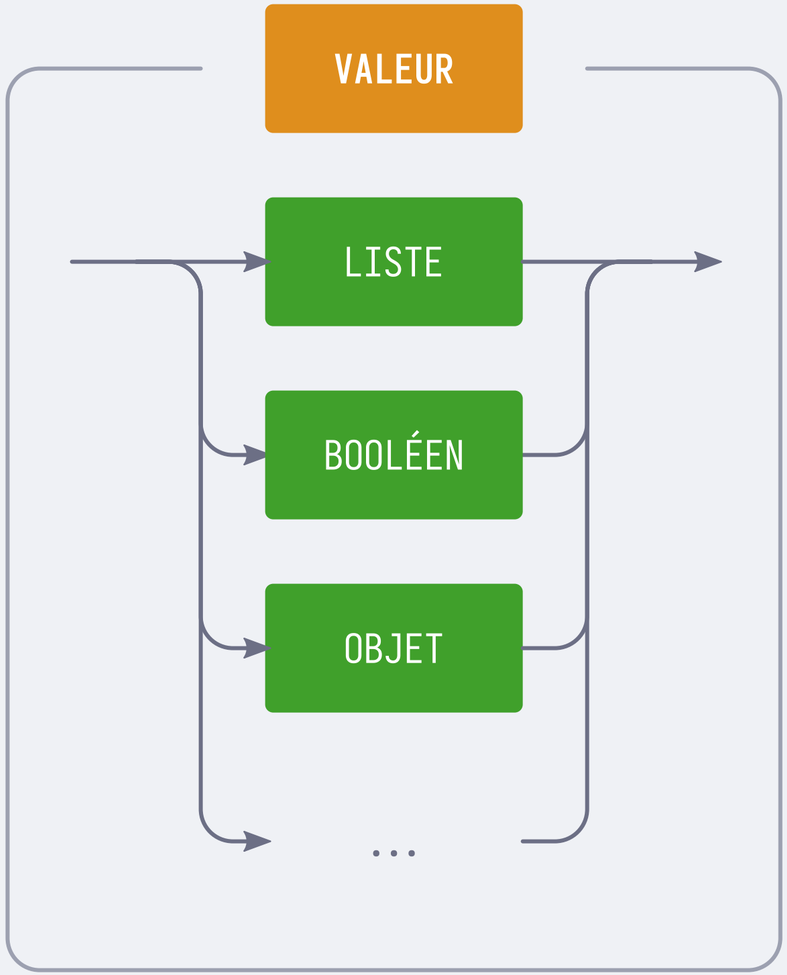
```
1
{
    2     3 4   5   6 7
   Lorem : "Ipsum",
   Dolor : {
     sit : "amet"
   }
}
```





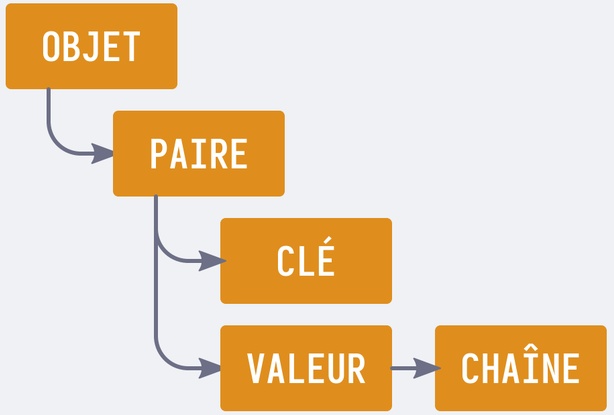
Analyse syntaxique

```
{
   Lorem : "Ipsum",
   Dolor : {
      sit : "amet"
   }
}
```



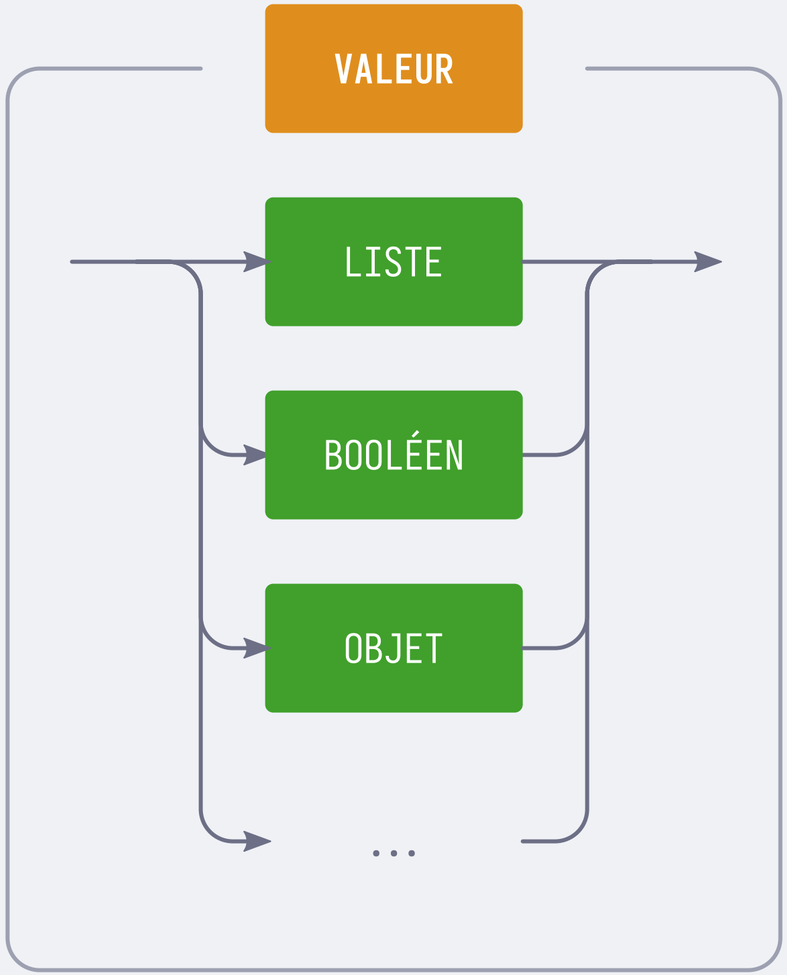
Analyse syntaxique

```
1
{
    2     3 4   5   6 7
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```



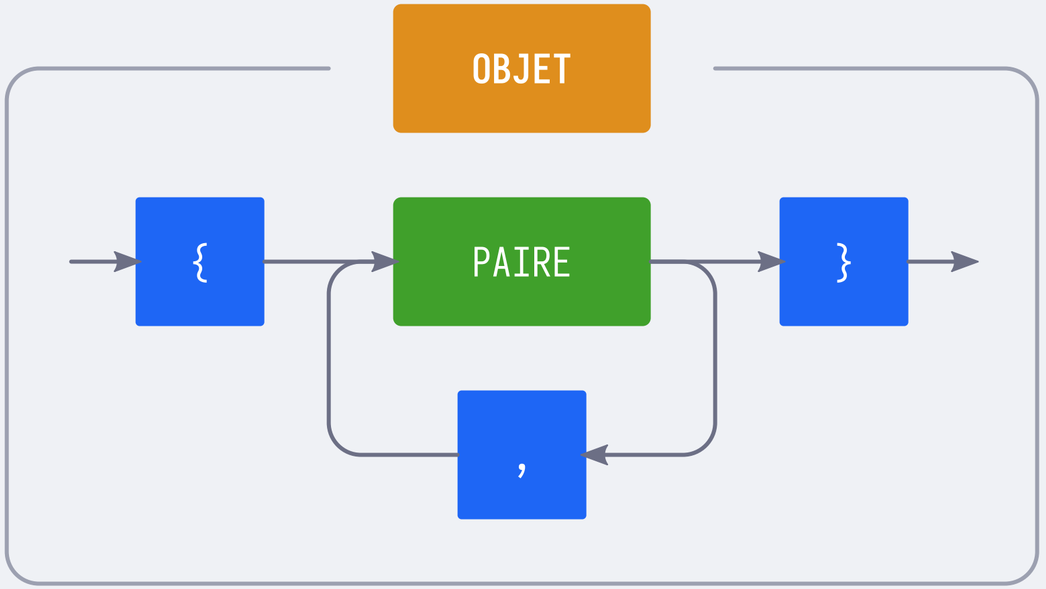
Analyse syntaxique

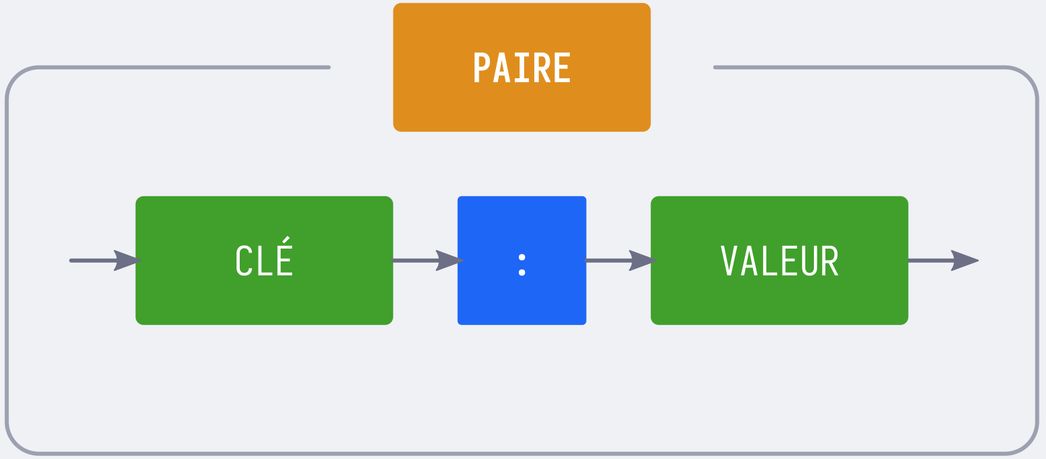
```
{
   Lorem : "Ipsum",
   Dolor : {
      sit : "amet"
   }
}
```



Analyse syntaxique

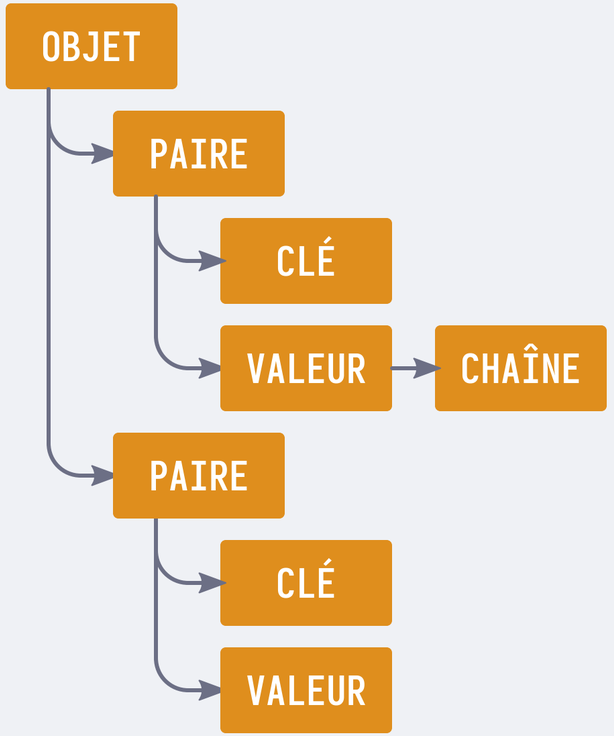
```
1
{
    2     3 4   5   6 7
   Lorem : "Ipsum",
   Dolor : {
     sit : "amet"
   }
}
```





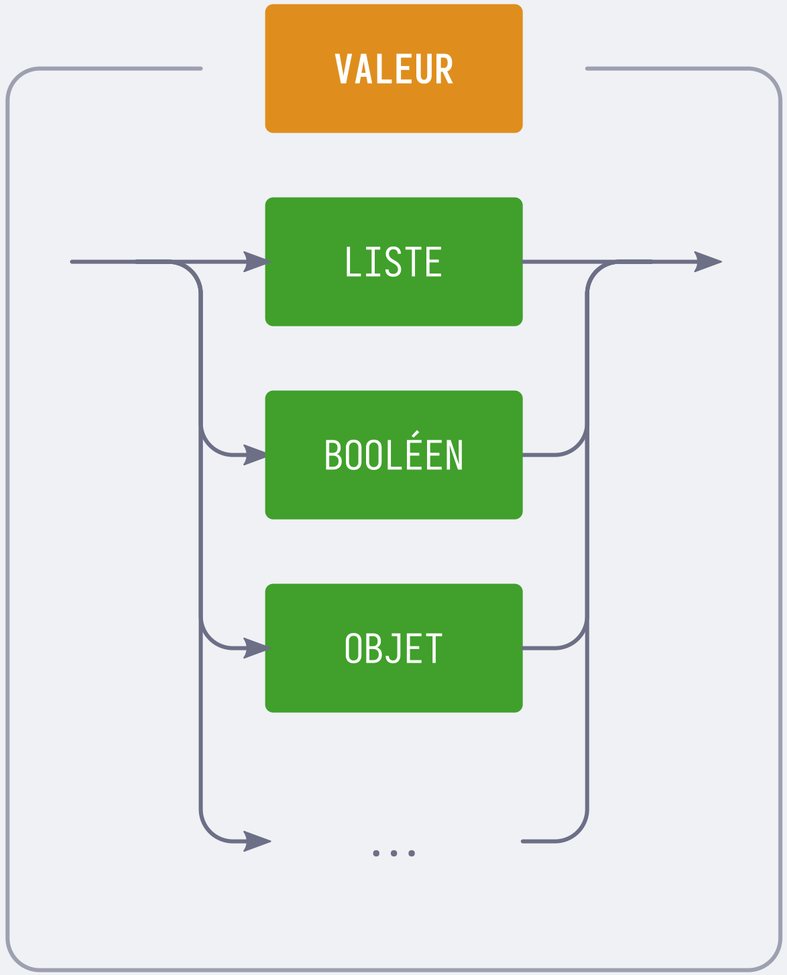
Analyse syntaxique

```
{
   Lorem : "Ipsum",
   Dolor : {
      sit : "amet"
   }
}
```

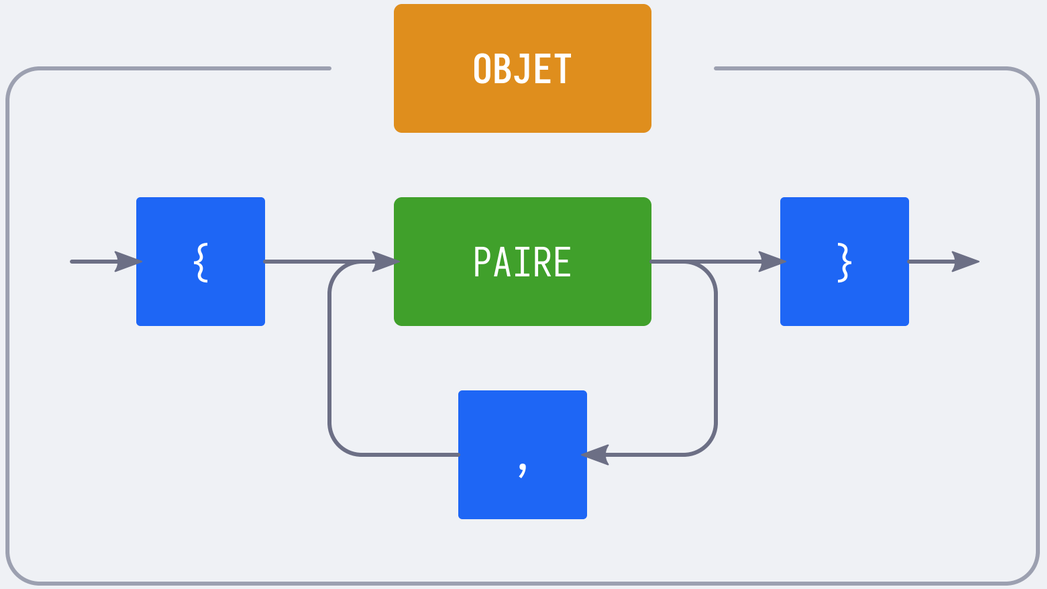
Analyse syntaxique

```
{
   Lorem : "Ipsum",
   Dolor : {
      sit : "amet"
   }
}
```



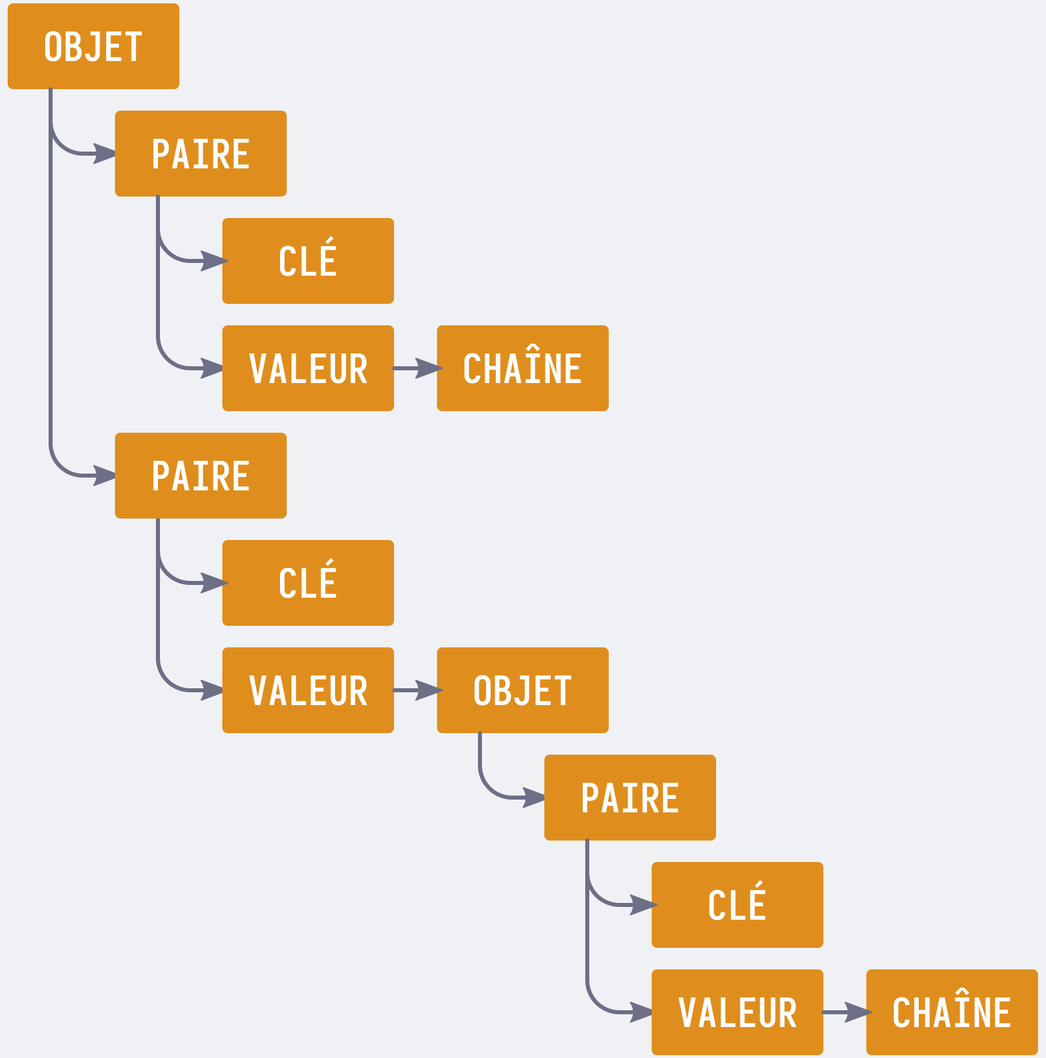
Analyse syntaxique

```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```



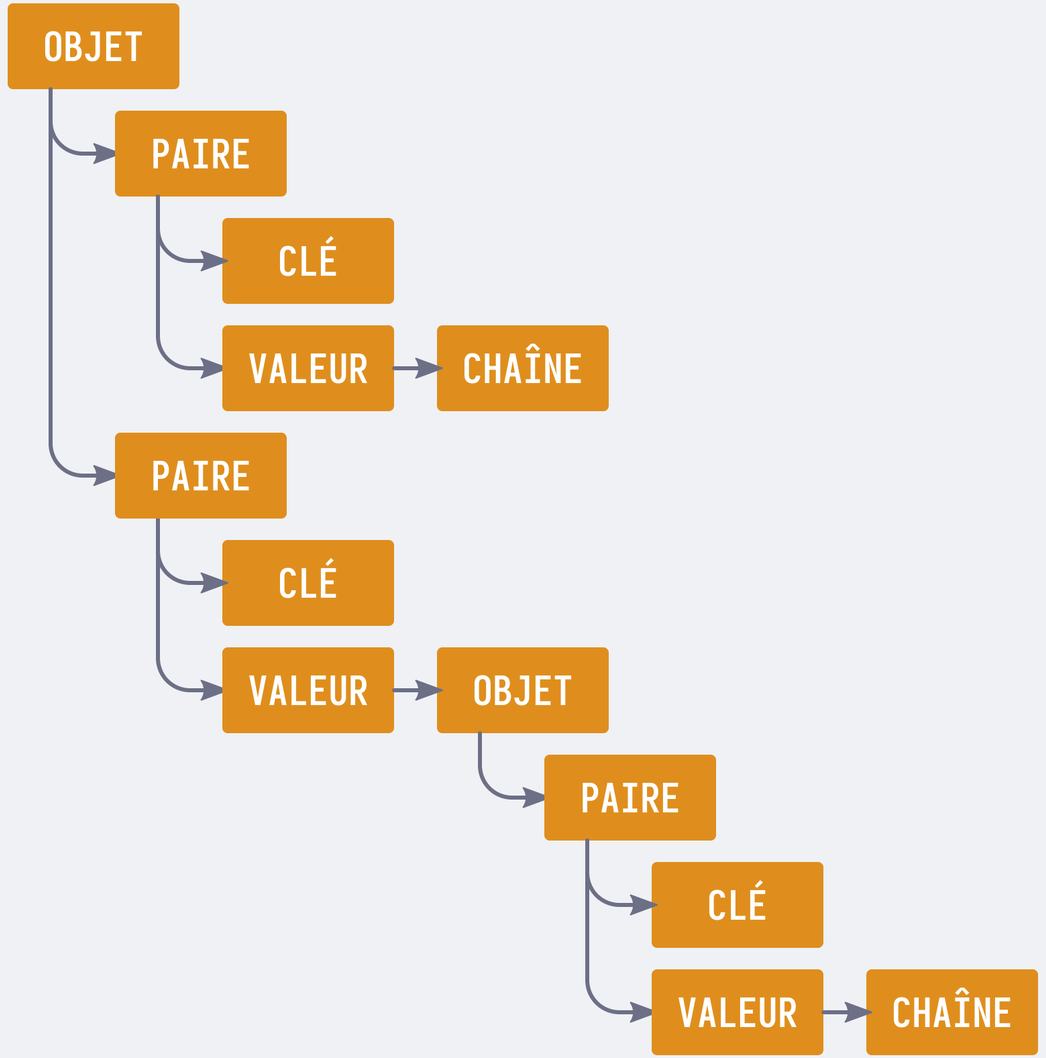
Analyse syntaxique

```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```

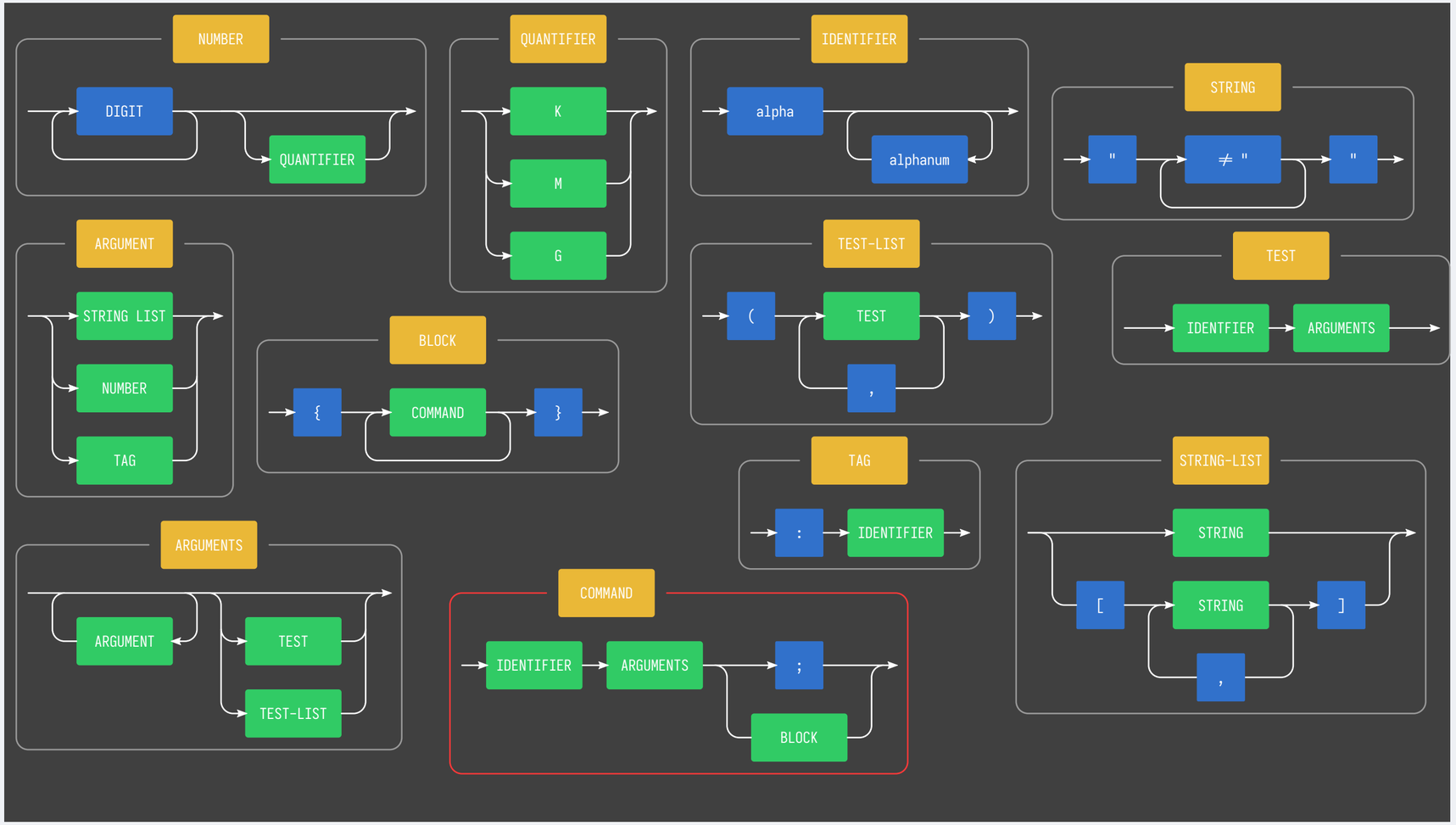


Analyse syntaxique

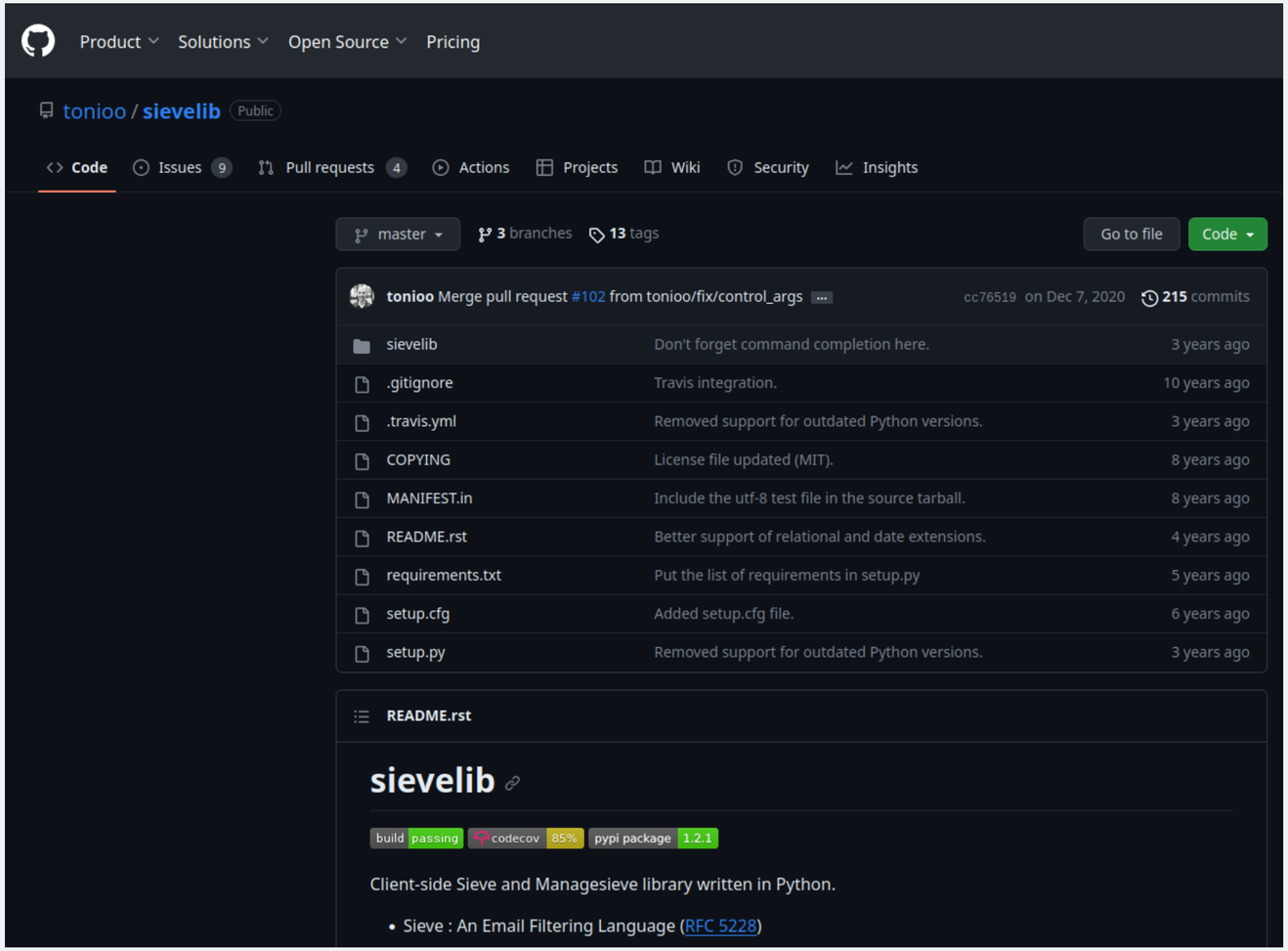
```
{
  Lorem : "Ipsum",
  Dolor : {
    sit : "amet"
  }
}
```



Analyse syntaxique



Analyse syntaxique



https://github.com/tonioo/sievelib

Tree-sitter

GitHub repository

Introduction
- Language Bindings
- Parsers
- Talks on Tree-sitter
- Underlying Research

Using Parsers

Creating Parsers

Syntax Highlighting

Implementation

Contributing

Playground

Code Navigation Systems

Introduction

Tree-sitter is a parser generator tool and an incremental parsing library. It can build a concrete syntax tree for a source file and efficiently update the syntax tree as the source file is edited. Tree-sitter aims to be:

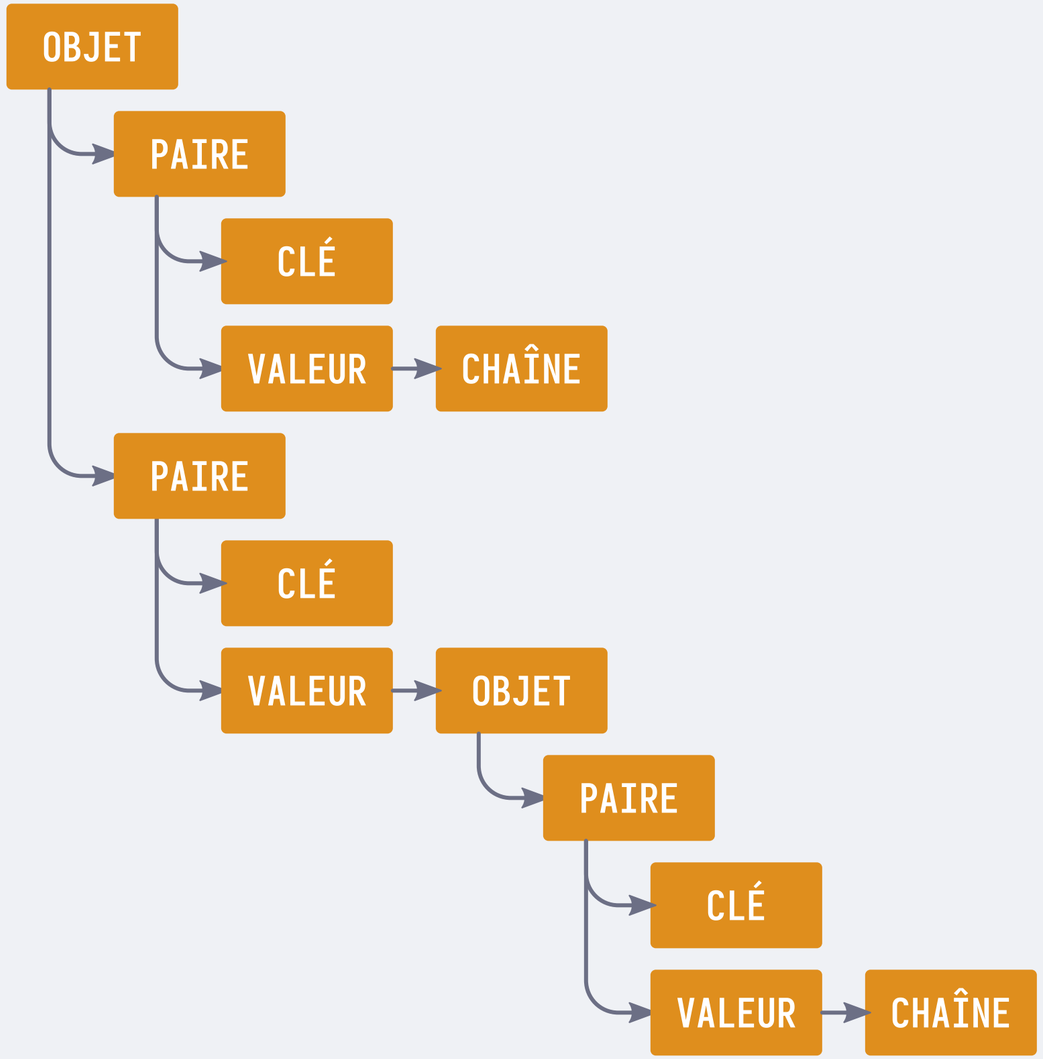
- **General** enough to parse any programming language
- **Fast** enough to parse on every keystroke in a text editor
- **Robust** enough to provide useful results even in the presence of syntax errors
- **Dependency-free** so that the runtime library (which is written in pure C) can be embedded in any application

Language Bindings

There are currently bindings that allow Tree-sitter to be used from the following languages:

- Go
- Haskell
- Java
- JavaScript (Node.js)
- JavaScript (Wasm)
- Kotlin
- Lua
- OCaml
- Perl
- Python
- Ruby

Tree-sitter




```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( OBJET
      ( PAIRE
        ( CLÉ )
         ( VALEUR ( CHAÎNE )))))))
```

Tree-sitter — Query

```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR
      ( OBJET
        ( PAIRE
          ( CLÉ )
          ( VALEUR ( CHAÎNE )))))))
```

```
( VALEUR ( CHAÎNE ))
```

Tree-sitter — Query

```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR
      ( OBJET
        ( PAIRE
          ( CLÉ )
          ( VALEUR ( CHAÎNE ))))))
```

```
( VALEUR ( CHAÎNE )) @MOT
```

Tree-sitter — Query

```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR
      ( OBJET
        ( PAIRE
          ( CLÉ )
          ( VALEUR ( CHAÎNE ))))))
```

```
( VALEUR ( CHAÎNE )) @MOT

( OBJET
  ( PAIRE
    ( VALEUR ( OBJET ))))
```

Tree-sitter — Query

```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR
      ( OBJET
        ( PAIRE
          ( CLÉ )
          ( VALEUR ( CHAÎNE )))))))
```

```
( VALEUR ( CHAÎNE )) @MOT

( OBJET
  ( PAIRE
    ( VALEUR ( OBJET ) @ENFANT2 )))
```

Tree-sitter — Query

```
func add(x int, y int) int {
 return x + y
}
```

code source

Tree-sitter — Query

```
func add(x int, y int) int {
 return x + y
}
```

code source

```
(function_declaration
  name: (identifier)
  parameters: (parameter_list)
    (parameter_declaration
      name: (identifier)
      type: (type_identifier))
    (parameter_declaration
      name: (identifier)
      type: (type_identifier))
  result : (type_identifier)
  ...
```

arbre de syntaxe

Tree-sitter — Query

```
func add(x int, y int) int {
 return x + y
}
```

code source

```
(function_declaration
  name : (identifier) @nom
  parameters : (parameter_list
    (parameter_declaration
      name : (identifier) @parametre )))
```

requête

```
(function_declaration
  name: (identifier)
  parameters: (parameter_list)
    (parameter_declaration
      name: (identifier)
      type: (type_identifier))
    (parameter_declaration
      name: (identifier)
      type: (type_identifier))
  result : (type_identifier)
  ...
```

arbre de syntaxe

Tree-sitter — Query

```
function add1(a, b) { return a+b; }

let add2 = (a, b) => { return a+b; };
```

code source

Tree-sitter — Query

```
function add1(a, b) { return a+b; }

let add2 = (a, b) => { return a+b; };
```

code source

```
(function_declaration
  name: (identifier)
  ...

(variable_declarator
  name : (identifier)
  value : (arrow_function)
    ...
```

arbre de syntaxe

Tree-sitter — Query

```
function add1(a, b) { return a+b; }

let add2 = (a, b) => { return a+b; };
```

code source

```
(function_declaration
  name : (identifier) @nom )

(variable_declarator
  name : (identifier) @nom
  value : (arrow_function))
```

requête

```
(function_declaration
  name: (identifier)
  ...

(variable_declarator
  name : (identifier)
  value : (arrow_function)
    ...
```

arbre de syntaxe

Tree-sitter — Coloration syntaxique

```
type Node struct {
  data string
  left * Node
  right * Node
}

func (self * Node) GetData() string {
  return self.data
}

func (self * Node) GetLeft() * Node {
  return self.left
}
```

Regex

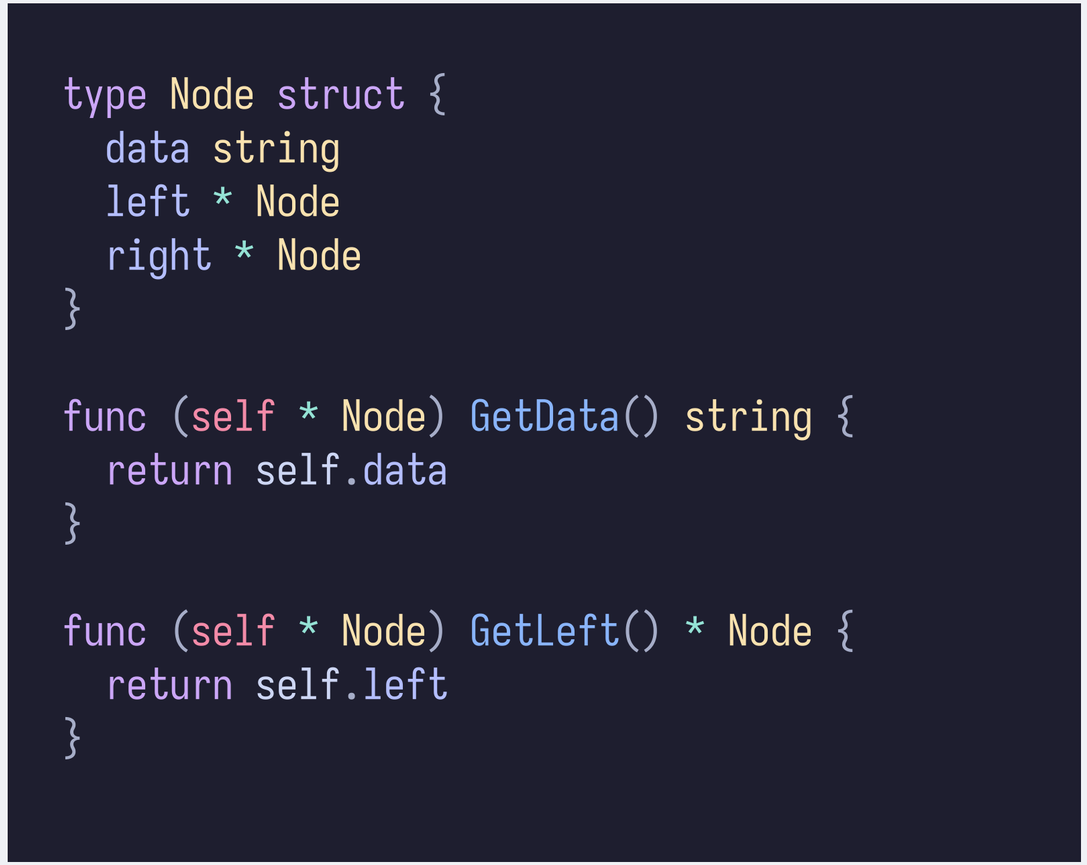
Tree-sitter — Coloration syntaxique

```
type Node struct {
  data string
  left * Node
  right * Node
}

func (self * Node) GetData() string {
  return self.data
}

func (self * Node) GetLeft() * Node {
  return self.left
}
```

Regex



Tree-sitter

Tree-sitter — Coloration syntaxique

```
function test(t)
  for k, v in pairs(t) do
    print(k, v)
  end
end
```

Tree-sitter — Coloration syntaxique

```
for k, v in pairs(t) do
  for i, d in ipairs(v) do
    for j, e in ipairs(d) do
      print(d)
    end
  end
end
```

Tree-sitter — Coloration syntaxique

```
\begin{document}
  \begin{center}
  \begin{align}
  \end{align}
  \end{center}
\end{document}
```

Tree-sitter — Query

```
( OBJET
  ( PAIRE
    ( CLÉ )
    ( VALEUR ( CHAÎNE )))
  ( PAIRE
    ( CLÉ )
    ( VALEUR
      ( OBJET
        ( PAIRE
          ( CLÉ )
          ( VALEUR ( CHAÎNE )))))))
```

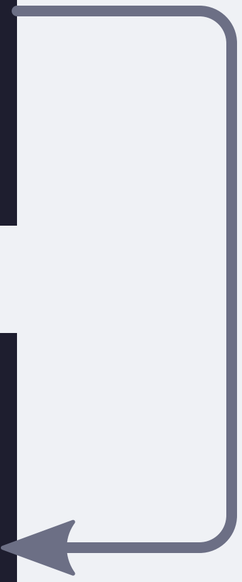
Tree-sitter — Coloration syntaxique

Tree-sitter — Coloration syntaxique

```
<div class="cards">
<% cards.forEach( card => { %>
  <%- include("card", card); %>
<% }); %>
</div>
```

Tree-sitter — Coloration syntaxique

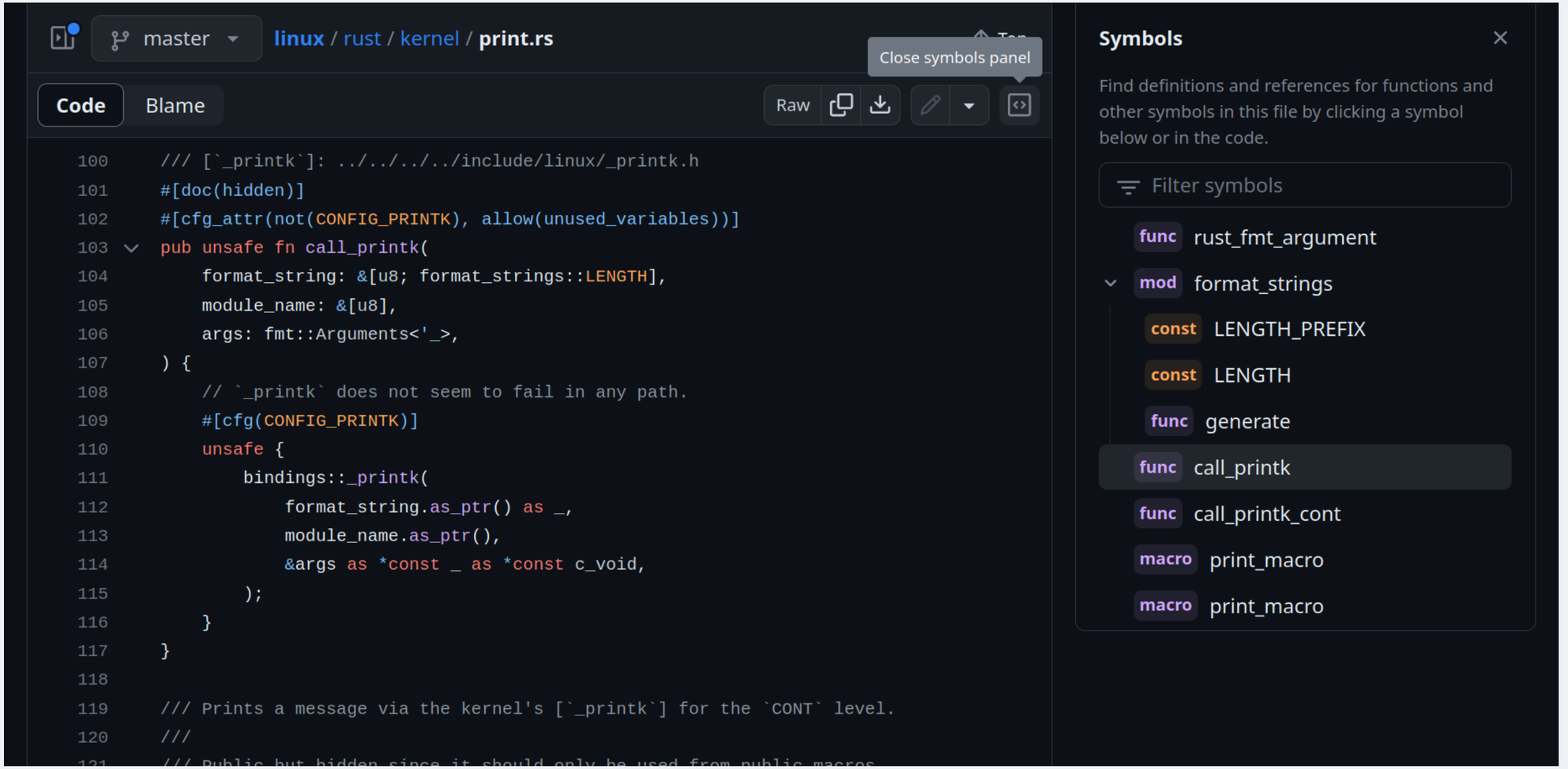
```
<div class="cards">
<% cards.forEach( card => { %>
  <%- include("card", card); %>
<% }); %>
</div>
```



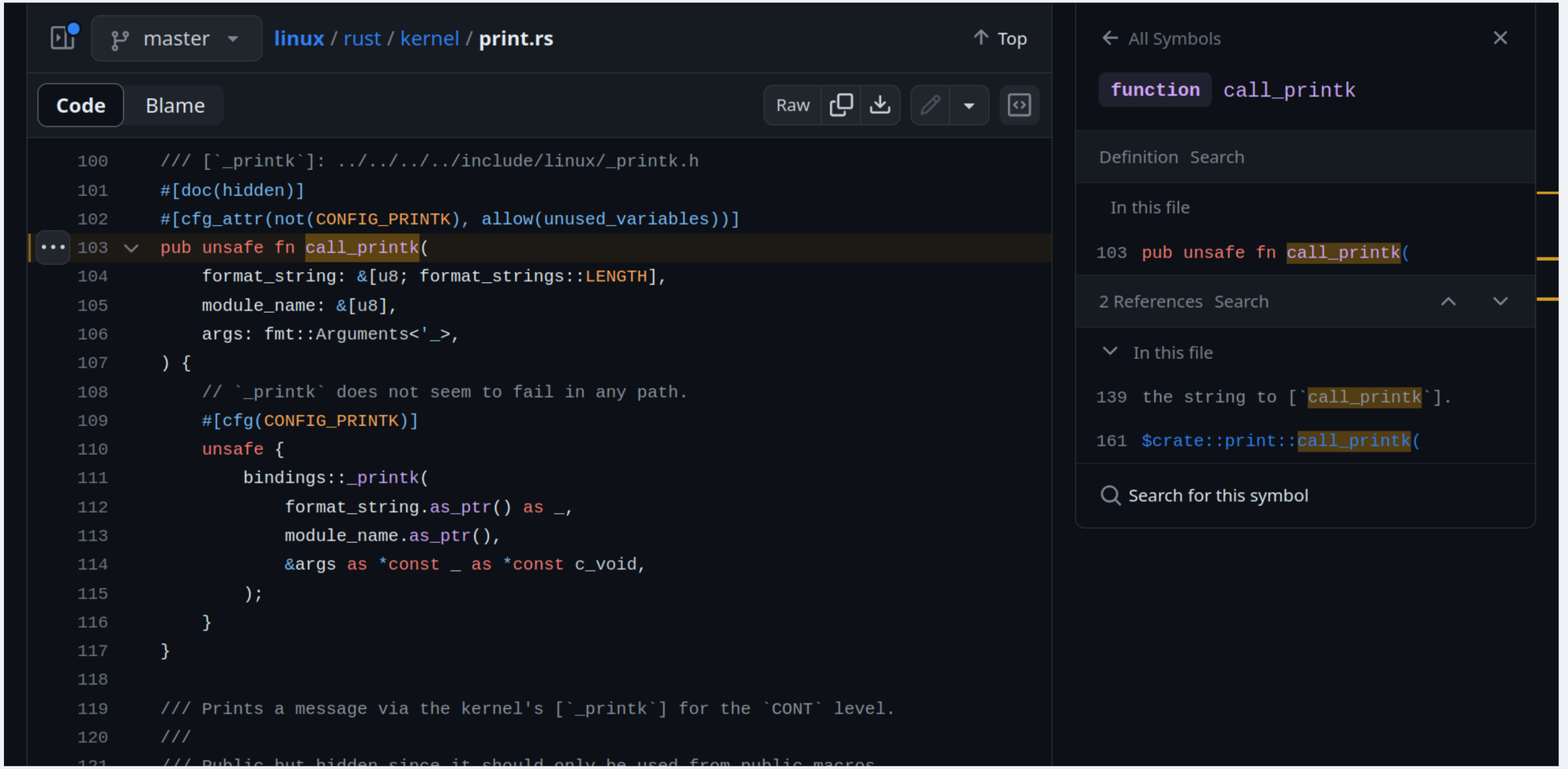
```
<div class="cards">
<% cards.forEach( card => { %>
  <%- include("card", card); %>
<% }); %>
</div>
```

Injection de langage entre
<% et %>

Github


master
linux / rust / kernel / print.rs
Close symbols panel
Code
Blame
Raw
100 /// [`_printk`]: ../../../../include/linux/_printk.h
101 #[doc(hidden)]
102 #[cfg_attr(not(CONFIG_PRINTK), allow(unused_variables))]
103 pub unsafe fn call_printk(
104 format_string: &[u8; format_strings::LENGTH],
105 module_name: &[u8],
106 args: fmt::Arguments<'_>,
107) {
108 // `_printk` does not seem to fail in any path.
109 #[cfg(CONFIG_PRINTK)]
110 unsafe {
111 bindings::_printk(
112 format_string.as_ptr() as _,
113 module_name.as_ptr(),
114 &args as *const _ as *const c_void,
115);
116 }
117 }
118
119 /// Prints a message via the kernel's [`_printk`] for the `CONT` level.
120 ///
Symbols
Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.
Filter symbols
func rust_fmt_argument
mod format_strings
const LENGTH_PREFIX
const LENGTH
func generate
func call_printk
func call_printk_cont
macro print_macro
macro print_macro

Github


master
linux / rust / kernel / print.rs
Top
Code
Blame
Raw
100 /// [`_printk`]: ../../../../include/linux/_printk.h
101 #[doc(hidden)]
102 #[cfg_attr(not(CONFIG_PRINTK), allow(unused_variables))]
103 pub unsafe fn call_printk(
104 format_string: &[u8; format_strings::LENGTH],
105 module_name: &[u8],
106 args: fmt::Arguments<'_>,
107) {
108 // `_printk` does not seem to fail in any path.
109 #[cfg(CONFIG_PRINTK)]
110 unsafe {
111 bindings::_printk(
112 format_string.as_ptr() as _,
113 module_name.as_ptr(),
114 &args as *const _ as *const c_void,
115);
116 }
117 }
118
119 /// Prints a message via the kernel's [`_printk`] for the `CONT` level.
120 ///
All Symbols
function call_printk
Definition Search
In this file
103 pub unsafe fn call_printk(
2 References Search
In this file
139 the string to [`call_printk`].
161 $crate::print::call_printk(
Search for this symbol

Tree-sitter — Nvim

Coloration syntaxique

Folding

Navigation

Text-objects

...

Tree-sitter — Nvim

```
def fib(n) :
    a, b = 0, 1
    while a < n :
        print(a, end=' ')
        a, b = b, a+b
    print()

fib(1000)
```

Tree-sitter — Nvim

```python
def fib(n) :
    a, b = 0, 1
    while a < n :
        print(a, end=' ')
        a, b = b, a+b
    print()

fib(1000)
```

@function.inner

```
(function_definition
  body : (block) @function.inner
) @function.outer
```

Tree-sitter — Nvim

```python
def fib(n) :
    a, b = 0, 1
    while a < n :
        print(a, end=' ')
        a, b = b, a+b
    print()

fib(1000)
```

@function.outer

```
(function_definition
  body : (block) @function.inner
) @function.outer
```

Tree-sitter

GitHub repository

- Introduction
- Using Parsers
- Creating Parsers
- Syntax Highlighting
- Implementation
- Contributing
- Playground
- Code Navigation Systems

Code C ☐ Log ☑ Query

```
1 int main() {
2     return 0;
3 }
```

Query

```
1 (function_definition
2   (function_declarator
3     declarator : (identifier) @function))
4
5 (number_literal) @number
6
7 "return" @keyword
```

Tree 1.0 ms

```
translation_unit [0, 0] - [3, 0]
  function_definition [0, 0] - [2, 1]
    type: primitive_type [0, 0] - [0, 3]
    declarator: function_declarator [0, 4] - [0, 10]
      declarator: identifier [0, 4] - [0, 8]
      parameters: parameter_list [0, 8] - [0, 10]
    body: compound_statement [0, 11] - [2, 1]
      return_statement [1, 1] - [1, 10]
        number_literal [1, 8] - [1, 9]
```