

# SATySF<sub>I</sub> Language Server の 現状と今後

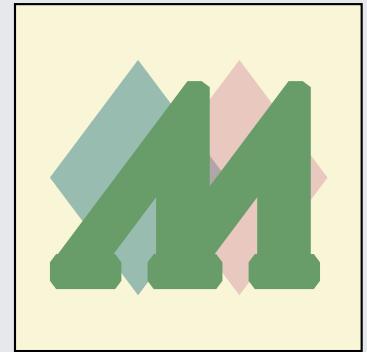
monaqa

2021 年 6 月 26 日

# 自己紹介

---

- ◆ 名前： monaqa
  - ▶ Twitter: @mo\_naqa
  - ▶ GitHub: monaqa
- ◆ 技術系コミュニティ所属
  - ▶ SATySF Slack
  - ▶ vim-jp Slack
  - ▶ Rust-jp Slack
- ◆ SATySF 歴： 2年ぐらい



# 今までに作った SAT<sub>Y</sub>SF<sub>I</sub>関連のプログラム

## ◆ SAT<sub>Y</sub>SF<sub>I</sub> クラスファイル・パッケージ

**SLyDIF<sub>I</sub>** スライド作成用のクラスファイル (本スライドも SLyDIF<sub>I</sub> 製)

**easytble** シンプルな記法で書ける表組版 (SAT<sub>Y</sub>SF<sub>I</sub> Conf 2020で発表)

**figbox** 柔軟な図表の配置

**railway** グラフィックスの作成支援

**enumitem** 豊富なスタイルの箇条書き

**AZMath** アクセント、括弧、amsmathライクな数式環境

## ◆ その他

**satysfi-parser** Rust 製の SAT<sub>Y</sub>SF<sub>I</sub>パーサ

**satysfi-language-server** Rust 製の SAT<sub>Y</sub>SF<sub>I</sub> language server (今日のお話)

# 目次

---

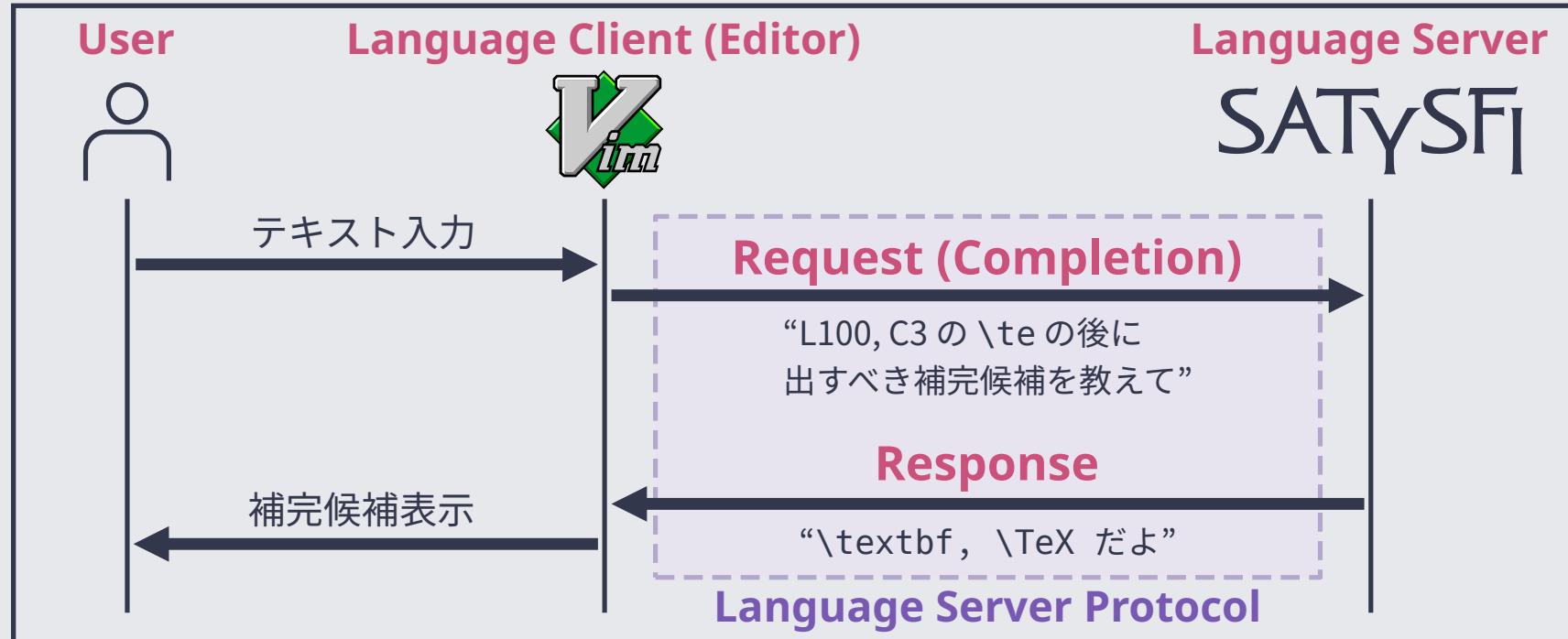
- ◆ Language Server Protocol
- ◆ SATySFI Language Server の現状
- ◆ SATySFI Language Server の実装
- ◆ SATySFI Language Server の今後の課題と発展性

# Language Server Protocol

Гандыңде сервер үргөсөл

# Language Server Protocol (LSP)

コード補完やエラー情報の提示など、開発 (+ 執筆) に役立つ様々な機能をあらゆるエディタに提供するために考案されたプロトコル。



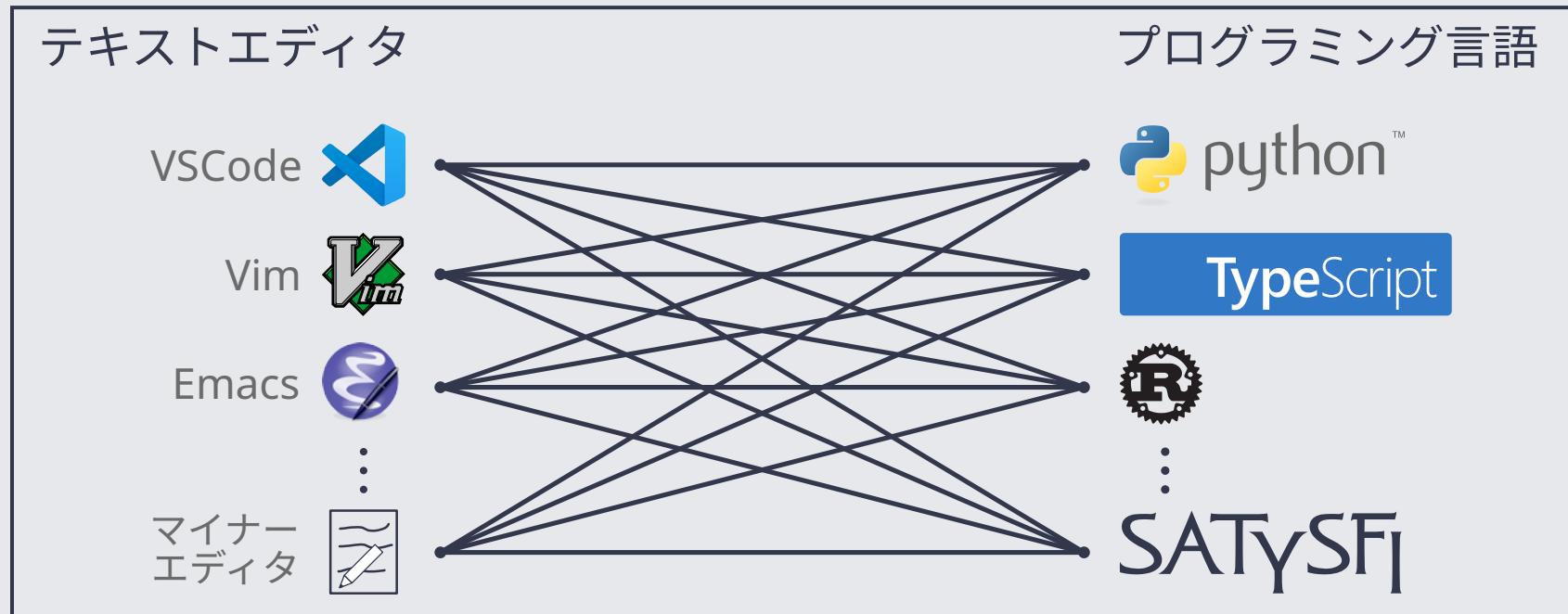
# Language Server (言語サーバ)

ある言語の開発に役立つ機能をサービスとして提供したもの。  
言語別に実装され、以下のような機能を提供する。

補完	ユーザが挿入しようとしている変数名・メソッド名などを予測して候補を表示する
診断情報表示	静的解析によるエラーや警告を表示する
詳細表示	カーソル上の変数・関数の型情報やドキュメンテーションなどを表示する
定義ジャンプ	カーソル上の変数が定義されている場所に移動する
リネーム	カーソル上の変数名を変更する
コードアクション	言語に応じて様々な処理を実行する（カーソル下にあるモジュールのインポート、getter/setter の自動作成など）
:	:

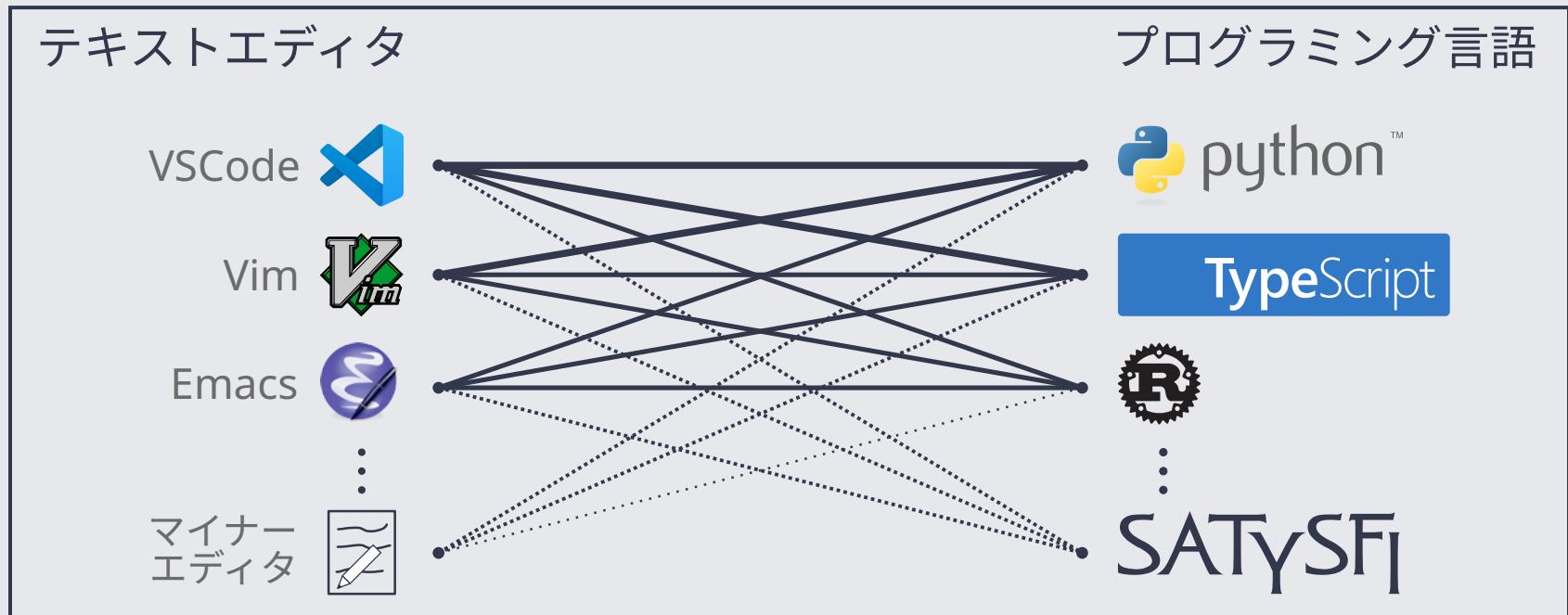
# なぜ LSP が注目されているのか

- ◆ LSP の登場前
  - ▶ エディタ・言語毎に各々プラグインや拡張を作る必要があった



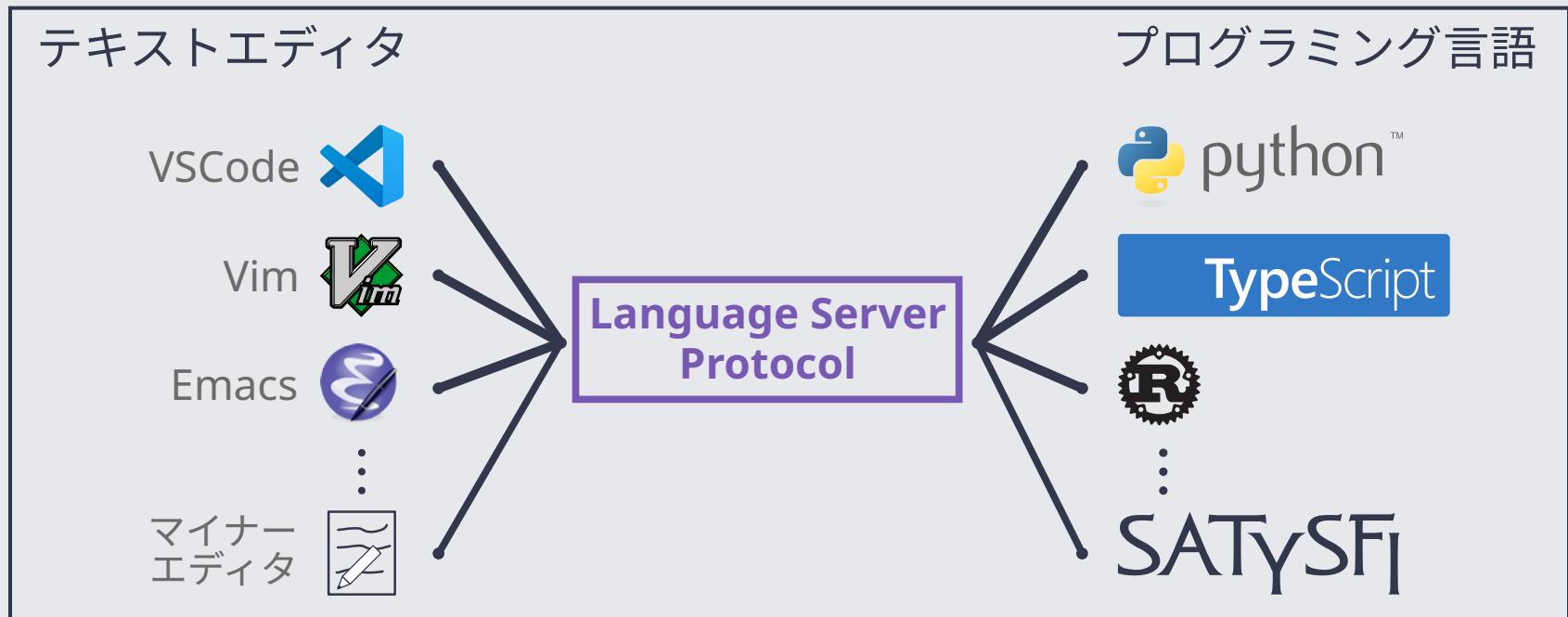
# なぜ LSP が注目されているのか

- ◆ LSP の登場前
  - ▶ マイナーなエディタ・言語では使用者が少なく実装も進まない



# なぜ LSP が注目されているのか

- ◆ LSP の登場後
  - ▶ 各言語・エディタの実装が 1 つで済むため、無駄が少なく開発も楽



# SATySFI Language Server の現状

SATySFI ランゲージサーバーの現状

# SATySFI Language Server とは

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー  
エディタ 

Language Server  
Protocol

プログラミング言語

 python™

TypeScript



SATySFI

# SATySFI Language Server とは

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー  
エディタ 

Language Server  
Protocol

プログラミング言語

 python™

TypeScript



ココ！

SATySFI

# SATySFI Language Server

- ◆ 以下の GitHub リポジトリで開発中

 [monaqa/satysfi-language-server](#)

The SATySFI Language Server

 Rust    ★ 15    ⚡ 1

- ◆ 現在動作を確認しているクライアント

- ▶ [Neovim + coc.nvim](#) (本スライドの作成にも使用)
- ▶ [Visual Studio Code](#) (拡張機能のデバッグモードでのみ)
  - 本スライドでは利用者が多いであろう VSCode でのデモの様子を掲載
  - ちゃんとした VSCode の拡張をどなたか作ってもらえると助かります

# 現在提供している機能

---

- ◆ 診断情報の表示
- ◆ 詳細情報のホバー表示
- ◆ 定義ジャンプ
- ◆ 補完

# 現在提供している機能

---

- ◆ 診断情報の表示
- ◆ 詳細情報のホバー表示
- ◆ 定義ジャンプ
- ◆ 補完

# 診断情報の表示

## 構文エラーの箇所と原因を表示



The screenshot shows a code editor with two tabs: "slide.saty 4, M" and "arctic.sathy". The "slide.saty" tab is active, displaying the following LaTeX code:

```
90 +frame{自己紹介}<
91
92 +fig-on-right(include-image 130pt `fig/pdf/logo-190727.pdf` |> frame 0.8pt (Co
93
94 +lis Incomplete inline command.
95 * Try adding semicolon or arguments after the command name. Syntax Error
96
97 \textbf
98 ** GitHub: \link(`https://github.com/monaqa`);
99
100 * \SATySFI; 歴: 2年ぐらい
```

A tooltip is displayed over the line starting with "+lis", containing the message "Incomplete inline command." and a note "\* Try adding semicolon or arguments after the command name. Syntax Error". Below the tooltip, there is a link "View Problem (CFB)" and a note "No quick fixes available".

- ◆ いくつかのエラーについては具体的なエラー原因を表示
- ◆ パーサの手に負えないものはエラー箇所と期待される字句のみ表示

# 現在提供している機能 (再掲)

---

- ◆ 診断情報の表示
- ◆ 詳細情報のホバー表示
- ◆ 定義ジャンプ
- ◆ 補完

# 詳細情報のホバー表示

変数やコマンドにカーソルを合わせると、その詳細を表示

```
92 |   date = { | 2021年6月26日 |};  
93 | );  
94 |  
95 +frame{自己紹介}< int?-> length -> string -> figbox  
96 |  
97 +fig-on-right(include-image 130pt `fig/pdf/logo-190727.pdf` |> frame  
98 |  
99 +listing{  
100 |   * 名前: monaqa  
101 |   ** Twitter: \link(`https://twitter.com/mo\_naqa`);  
102 |   ** GitHub: \link(`https://github.com/monaqa`);
```

(現在は変数 / コマンドの種類と、判明しているもののみ型情報を表示)

# 詳細情報のホバー表示

変数やコマンドにカーソルを合わせると、その詳細を表示

```
92 |   date = { | 2021年6月26日 |};  
93 | );  
94  
95 +frame{自己紹介}<  
96  
97   +fig-on-right(include-image 130pt `fig/pdf/logo-190727.pdf` |> frame  
98  
99     +listing{  
100       * 名前: monaqa [inline-text?; string] inline-cmd  
101       ** Twitter: \link(`https://twitter.com/mo\_naqa`);  
102       ** GitHub: \link(`https://github.com/monaqa`);
```

inline command

inline-text?; string

(現在は変数 / コマンドの種類と、判明しているもののみ型情報を表示)

# 現在提供している機能 (再掲)

---

- ◆ 診断情報の表示
- ◆ 詳細情報のホバー表示
- ◆ 定義ジャンプ
- ◆ 補完

# 定義ジャンプ

コマンドや変数の定義にジャンプ (ファイルを跨いでも有効)

```
slide.saty 3, M •
slide.saty
91
92     +fig-on-right(include-image 130pt `fig/pdf/logo-190727.pdf` |> fra
93
94     +listing{ inline command
95         * 名前 : monaqa [inline-text?; string] inline-cmd
96             ** Twitter: \link(`https://twitter.com/mo_naqa`);
97             ** GitHub: \link(`https://github.com/monaqa`);
98
99         * \SATySFI; 歴 : 2年ぐらい
100
```

# 定義ジャンプ

コマンドや変数の定義にジャンプ（ファイルを跨いでも有効）

A screenshot of a code editor showing a context menu. The menu items are:

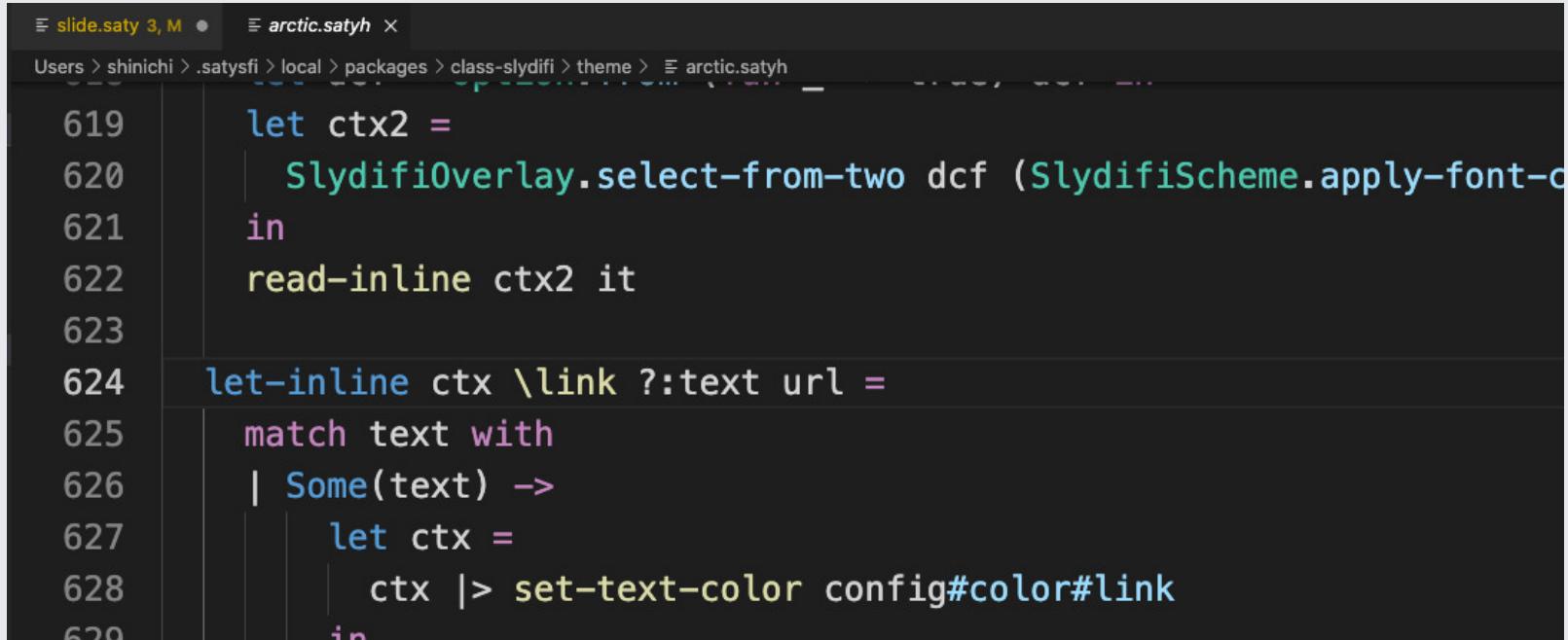
- Go to Definition F12
- Peek ▶
- Change All Occurrences ⌘F2
- Cut ⌘X
- Copy ⌘C
- Paste ⌘V
- Command Palette... ⌘⌘P

The 'Go to Definition' option is highlighted in green.

```
slide.saty
91
92     +fig-on-right(include-image 130pt `fig/pdf/logo-190727.pdf` |> fra
93
94     +listing{
95         * 名前 : monaqa
96         ** Twitter: \l
97         ** GitHub: \li
98
99         * \SATySFi; 歴 : 2
100
101        * 技術系コミュニティ所属
```

# 定義ジャンプ

コマンドや変数の定義にジャンプ（ファイルを跨いでも有効）



```
slide.saty 3, M • arctic.saty ×
Users > shinichi > .satysfi > local > packages > class-slydifi > theme > arctic.saty
619 let ctx2 =
620   | SlydifiOverlay.select-from-two dcf (SlydifiScheme.apply-font-c
621   in
622     read-inline ctx2 it
623
624 let-inline ctx \link ?:text url =
625   match text with
626   | Some(text) ->
627     let ctx =
628       ctx |> set-text-color config#color#link
629     in
```

# 現在提供している機能 (再掲)

---

- ◆ 診断情報の表示
- ◆ 詳細情報のホバー表示
- ◆ 定義ジャンプ
- ◆ **補完**

# コマンドの補完機能

ユーザが示す文字列に続く名前のコマンドを候補に表示

```
159 +p{  
160 | コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能をあらゆるエ  
161 }  
162  
163 |  
164 |  
165 |  
166 |  
167 |  
168 |  
169 >
```

(ソースコードを読んで補完候補を生成)

# コマンドの補完機能

ユーザが示す文字列に続く名前のコマンドを候補に表示

```
159 +p{
160   | コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能をあらゆるエ
161 }
162
163 +fig-
164   ↪ +fig-abs-pos
165   ↪ +fig-block
166   ↪ +fig-center [figbox] block-cmd
167   ↪ +fig-on-left
168   ↪ +fig-on-right
169 >
```

(ソースコードを読んで補完候補を生成)

# コマンドの補完機能

ユーザが示す文字列に続く名前のコマンドを候補に表示

```
159 +p{  
160 | コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能をあらゆるエ  
161 }  
162  
163 +fig-center();  
164  
165  
166  
167  
168  
169 >
```

(ソースコードを読んで補完候補を生成)

# 変数名の補完

プログラムモードでは変数や関数名も補完できる

```
159 +p{
160   | コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能をあらゆるエ
161 }
162
163 +fig-center(incl);
164     ⚑ include-image    int?-> length -> string -> figbox
165     ⚑ include-image-with-height
166     ⚑ include-image-with-size
167     ⚑ inline-graphics
168     ⚑ inline-graphics-outer
169 >     ⚑ direct inline-cmd
```

# 変数名の補完

プログラムモードでは変数や関数名も補完できる

```
159 +p{  
160   | コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能をあらゆるエ  
161 }  
162  
163 +fig-center(include-image);  
164  
165  
166  
167  
168  
169 >
```

# 補完機能の特徴

カーソル位置のモードを考慮して補完候補を表示

プログラムモードのとき

```
75     glass-box ?: (align-center, align-center) wid 40pt fb
76         |> bgcolor bg-color
77
78     read-block ctx '<+fig-center(fbf wid);>
79
80     +fig-
81     @ fig-ghost      (int -> bool) -> FigBox.figbox ...
82     in    @ fig-phantom
83
84     document '<
85
86     +make-title()
```

水平モードのとき

```
159 +p[
160     コード補完やエラー情報の提示など、プログラミングに役立つ様々な機能
161
162     +fig-
163     }
164     @ \fig-abs-pos
165     @ \fig-block
166     @ \fig-center [figbox] inline-cmd
167     @ \fig-inline
168     @ \fig-on-left
169     +frame{ @ \fig-on-right
170     @ \fig-widfill
```

(垂直モード以外のところではブロックコマンドの補完が出てこない)

# 補完機能の特徴

コマンド引数の型に応じて異なるスニペットを展開

```
56 |     &gt; Twitter: (link( https://twitter.com/mo_naga ))
57 |     &gt;> GitHub: \link(`https://github.com/monaga`).
58 |
59 | inline command
60 |
61 | [ruleptn list?; (TableBuilder.builder -> TableBuilder.builder) list?;
62 | cellfmt list; inline-text list] inline-cmd
63 |
64 | \easytable[]{}|||
65 |
66 |
```

```
88 | \.
89 | block command
90 | [inline-text?; inline-text list; block-text;] block-cmd
91 | +section{}|||<
92 |
93 | >
94 |
```

# 補完機能の特徴

コマンド引数の型に応じて異なるスニペットを展開

型	スニippet展開結果
inline-text	{}
block-text	< >
inline-text list	{   }
他の list	[];
他の型	();

# SATySF<sub>I</sub> Language Server の使用感

---

- ◆ 基本的に、静的型付け言語と LSP は相性が良い
  - ▶ 静的解析で型が判明すれば賢い補完や診断情報をユーザに提示できるため
- ◆ SATySF<sub>I</sub> も例外ではない
  - ▶ 変数やコマンドに型が付くことで開発体験が向上する
    - モードに応じた補完候補の出し分け
    - 型に応じたスニペットの出し分け
    - etc.
  - ▶ SATySF<sub>I</sub> の型システムに詳しくないライトユーザこそ使ってほしい
  - ▶ 開発者の実力とリソースさえ揃えば、かなり便利な機能が実現できる

# SATySFI Language Server の実装

SATySFI Language Server の実装

# 実装言語：Rust

---

- ◆ Rust を選んだ理由

- ▶ Rust を学びたかった
- ▶ 優れたパフォーマンス・省メモリ性・安全性に惹かれた
- ▶ 静的型付け言語であること由来の書きやすさに惹かれた
- ▶ Rust で書かれた代表的な language server が複数あった

---

Rust    [rust-analyzer](#)

Deno    [Deno Language Server](#)

LATEX    [TexLab](#)

---

# パーサの実装

- ◆ パーサは language server と別リポジトリで開発

 [monaqa/satysfi-parser](#)

SATySFi parser written in Rust.

 Rust     7     2

- ◆ **Parsing expression grammar (PEG) を用いたパーサ**
  - ▶ [rust-peg](#) というパーサジェネレータを使用
- ◆ 現在は字句解析と構文解析を同時に行う
  - ▶ エラー回復やカーソル位置のモードの把握に難あり

# サーバ機能の実装 (概観)

---

- ◆ DocumentData 構造体にファイルの内容を保持
  - ▶ ファイルの文字列
  - ▶ 構文木
  - ▶ 環境 (Environment)
    - モジュール (名前、定義の位置)
    - 変数 (名前、型情報、定義の位置)
    - コマンド (名前、型情報、定義の位置)

# 各リクエストに対する処理の流れ（抜粋）

---

- ◆ ファイルが開かれたとき / 変更されたとき (didOpen/didChange)
  - ▶ ファイルを構文解析し、成功したらそのファイルの `Environment` を作成
  - ▶ 未解析の依存パッケージ (`@require: xxxx` / `@import: xxxx`) を構文解析し、成功すれば先程同様 `Environment` を作成
- ◆ 補完リクエストがあったとき (completion)
  - ▶ カーソル位置のモード（水平モード、数式モードなど）を構文木から取得
  - ▶ そのモードに出現しうる変数 / コマンドのうち、カーソル位置をスコープに含むものを `Environment` から検索
  - ▶ 得られた変数 / コマンドを返却
    - コマンドの場合、型情報があればスニペットを作成

# SATySFI Language Server の 今後の課題と発展性

今後の課題と発展性  
SATySFI Language Server の

# エラー回復機構が強力なパーサ

- ◆ エラー回復：文法の誤りを含むコードを適切に処理すること
  - ▶ Language server が処理するテキストには文法エラーが高確率で含まれる
    - 例：補完リクエスト時に送られる、入力途中のテキスト
  - ▶ ちゃんとやるのはかなり難しい
    - 文法には規則があるが、誤りには一般に規則がない
  - ▶ 最低限必要なエラー回復処理はすでに実装
    - 不完全なコマンド文法への対処など
- ◆ 現時点では、1つでも回復不能なエラーがあると情報を全く取れなくなる
  - ▶ カーソル位置のモードや依存パッケージぐらいは取りたい

# 型検査・型推論機能の追加

---

- ◆ まだ型検査・型推論器は実装されていない
- ◆ 変数・コマンドの型推論・型検査によって実現できそうなこと
  - ▶ 型エラーの診断情報表示
  - ▶ シグニチャが明記されていないコマンド・変数の補完、詳細情報表示
  - ▶ レコードのフィールド名での補完候補の絞り込み
  - ▶ パイプライン演算子直後の補完候補の絞り込み
  - ▶ etc.

# その他の発展性

---

- ◆ **Doc comment** のホバー表示

- ▶ ソースコードに直接書かれたドキュメントを参照する機能
  - ▶ 現在 **doc comment** 機能が言語本体の機能として構想されている

- ◆ **Auto import**

- ▶ コマンド補完の際やコードアクションによって必要な依存を自動解決
  - ▶ 予め `~/.satysfi/dist/packages` などの全ファイルを見ておけば可能

- ◆ **Code action** (言語ならではの特別な編集操作を実現する機能)

- ▶ 未定義のコマンドがあるとき、その定義の雛形をプリアンブルに追加
  - ▶ モジュール内で定義した関数・コマンドのシグニチャを `sig` に追加

# まとめ

- ◆ SATySFi language server を開発中
  - ▶ 補完機能、定義ジャンプ、ホバー、エラーの表示などの機能を実装
  - ▶ Neovim + coc.nvim や Visual Studio Code 上で動く
- ◆ 今後も様々な発展性が考えられる
  - ▶ エラー回復機構の強力なパーサ
  - ▶ 型推論・型検査器
  - ▶ その他、Doc comment や Auto import など
- ◆ 是非使ってみてください！



[monaqa/satysfi-language-server](https://github.com/monaqa/satysfi-language-server)

The SATySFi Language Server



Rust



★ 15



฿ 1

