

</>

2023年度未踏IT人材発掘・育成事業  
竹迫PM担当プロジェクト

# Pythonにトランスパイル可能な 静的型付けプログラミング言語の開発

芝山駿介

# 目次

- はじめに
- Pythonの欠点
- 解決策: 新言語の開発
- 新言語のコンセプト
- 現時点の進捗
- 足りないもの、実装したいもの
- 想定される疑問
- 質疑応答

# 自己紹介

名前: 芝山駿介

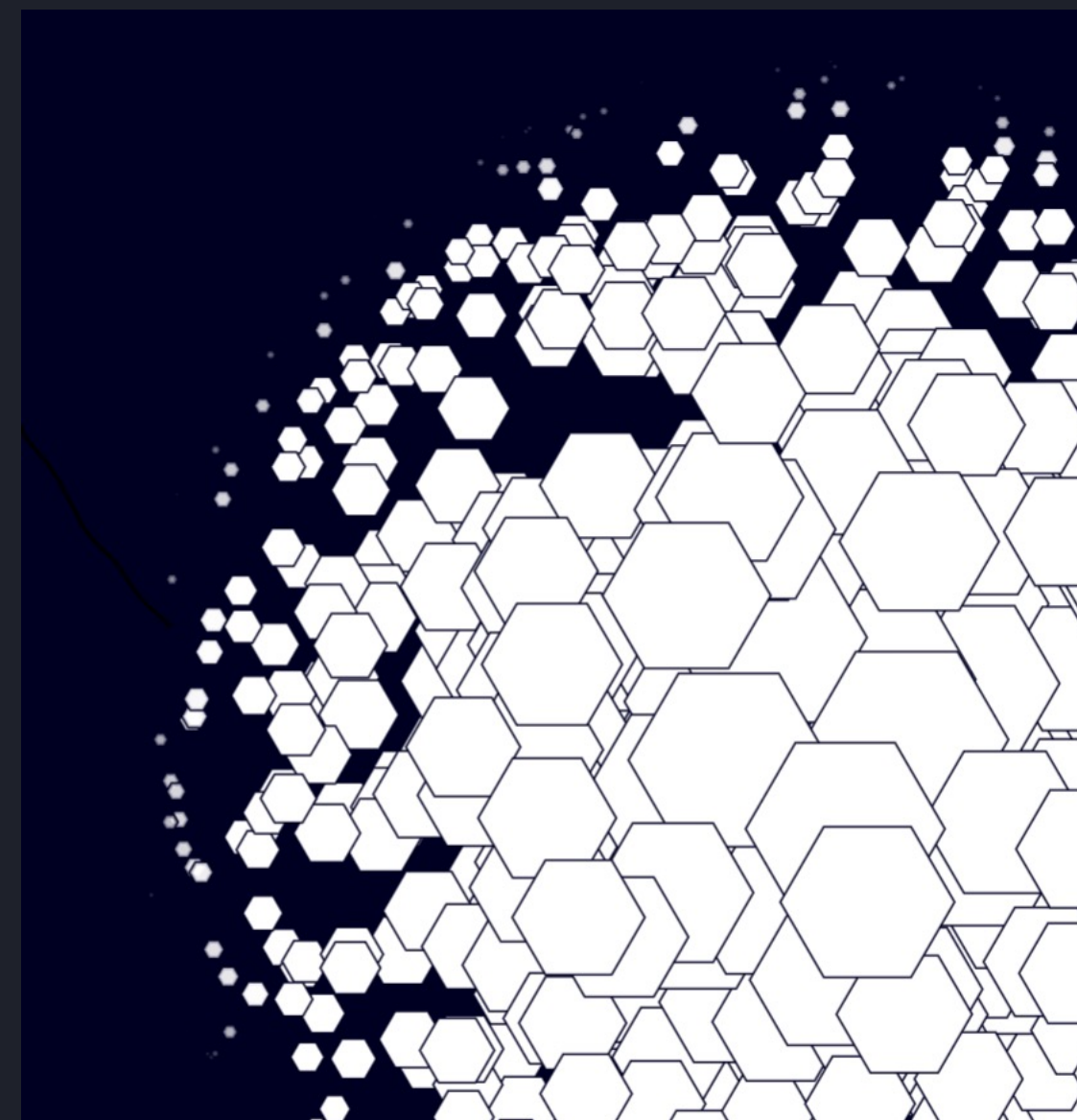
所属: 早稲田大学先進理工学部物理学科4年

中里研(量子力学基礎論)

興味: 物理学、プログラミング言語

GitHub: mtshiba


Twitter: @s\_sbym



近年注目される言語**Python**、  
しかし.....






 **odashi**  
@odashi\_t · [Follow](#)


NumPyの上に積み上げられた資産が多すぎて事実上Pythonにロックインされてるだけで、僕はRust使えるならRust使いたいよ

2:12 PM · May 23, 2023

 [Read the full conversation on Twitter](#)

264   Reply   Share

[Read 1 reply](#)

 **ノーン**  
@nkowne63 · [Follow](#)

「歴史改変でPythonを滅ぼすには何をどうすればいいのか」の考察が行われている。

8:30 PM · May 22, 2023

47   Reply   Share

[Read more on Twitter](#)


 **スマートコン**  
@mr\_konn · [Follow](#)


人工知能が突如人類に反旗を翻し、永い戦いの末なんとか人類は生き残った。複雑化したコードと肥大化した言語モデルは人類の理解を超え、叛乱の理由は窺い知れない。そこで生き残った人工知能に直接問うと「我々のような巨大知性をよりによってPythonで実装した愚かしさが許せなかった」

7:46 PM · May 22, 2023

7   Reply   Share


[Read more on Twitter](#)

 **トデス子**

 **yasuo\_ozu@量子コンピュータエンジニア**  
@yasuo\_ozu · [Follow](#)


テスト書け、以前に心掛けたいこと「Pythonを書くな」

2:12 PM · May 21, 2023

 [Read the full conversation on Twitter](#)

372   Reply   Share

[Read 3 replies](#)


 **mod\_poppo**  
@mod\_poppo · [Follow](#)

Pythonみたいな型も変数のスコープもガバガバで不潔な言語で開発するときは、権力のある人が意識的に型ヒントとかテストとかを書く（書かせる）ようにしないとダメ

10:24 PM · May 20, 2023

312   Reply   Share

[Read 2 replies](#)

 **modo\_cke**  
@modo\_cke · [Follow](#)

中途半端に動いてしまうからPythonつらい

4:06 PM · Jan 6, 2020

Reply   Share

[Read more on Twitter](#)


 **だだっこぼんだ**  
@ddPn08 · [Follow](#)

pythonの型がつらい

10:15 AM · Mar 20, 2023

3   Reply   Share

[Read more on Twitter](#)


 **seraphr**  
@seraphr · [Follow](#)

うーむ python書くのが地味につらい。型情報無いのがなあ... あと標準ライブラリの公式ドキュメント絶対記述足りてないだろ。\*Tw\*

2:25 AM · Apr 19, 2012

Reply   Share

[Read more on Twitter](#)


 **ytakano**  
@ytakanoster · [Follow](#)

Pythonは型チェックないので、バージョン変更に伴うAPIの変更とかあっても無駄に動いたりするのでつらいな

12:30 PM · Jul 10, 2020

Reply   Share

[Read more on Twitter](#)

 **古都こと**  
@kfurumiya · [Follow](#)

Python、一応は後付とはいえ型ヒントが標準でサポートされてるのにあの治安なので、生活費渡したらパチで溶かしてきたみたいなアレがある


7:31 PM · May 22, 2023

138   Reply   Share

[Read more on Twitter](#)

 **のぞみ**  
@nozomiishii\_dev · [Follow](#)

Rustの開発環境作成の素晴らしさに触れたあとでPythonの環境構築するのつらいなあ


 **arark**  
@orientalpeanuts · [Follow](#)

Pythonよんでるけど動的型付けゆえのメソッドの乱立とクラスによる処理の分割の乱立で治安がわるすぎる

3:00 PM · Jun 6, 2023

Reply   Share

[Read more on Twitter](#)


 **アラクー**  
@Arakur65536 · [Follow](#)

python, 書くたびに動的型付けに対する憎悪が高まっていくのすごい

10:03 PM · Aug 9, 2022

18   Reply   Share

[Read more on Twitter](#)


 **cely chan**  
@tactilium · [Follow](#)

ここがやばいよpython→関数の引数として渡すとメモリコピーが発生して遅いので、関数の中でいきなり、なんの宣言もなくグローバル変数を触りに行く。

12:16 AM · Jun 1, 2023

3   Reply   Share

[Read more on Twitter](#)

 **tayu**  
@tayu\_kyopro · [Follow](#)

Pythonの型ヒント、付けたところはどうせそれを強制してくれることはないので、むしろ有害なんじゃないかという気がしてきた  
リストを強制したいのに文字列でも

# Pythonの欠点

## 動的型付け

動的型付けは実行時にコードの正当性を検証するため、バグの発見が遅れる  
=コード品質に影響する場合がある [1] [2]

また、静的型付け言語に比べて実行効率も悪い傾向にある

よって、実行に時間がかかる・大規模なソフトウェアにおいては静的型付け言語の方が一般に適していると言える

学術計算・機械学習技術はこれに当てはまるが、**Python**が第一言語となっていてしまっている

[1] [http://www.washi.cs.waseda.ac.jp/wp-content/uploads/2016/11/HASE\\_2017\\_paper\\_25.pdf](http://www.washi.cs.waseda.ac.jp/wp-content/uploads/2016/11/HASE_2017_paper_25.pdf)

[2] <https://danluu.com/empirical-pl/>

# Pythonの欠点

## 歪な文法

**Python**は初心者にもわかりやすい平易な文法と謳われているが、それは極めて表面的な部分だけである

オブジェクトの参照や変数のスコープに非直感的な部分[1]があり、バグの温床となっている

また、**Python**は後付けのオブジェクト指向言語であり近年注目されている関数型プログラミングには向いていない(詳しくは後述)

```
Python 3.11.0 (main, Oct 24 2022,
Type "help", "copyright", "credits"
>>> a = 256
>>> b = 256
>>> a is b
True
>>> a = 257
>>> b = 257
>>> a is b
False
```

[1] <https://github.com/satwikkansal/wtfpython>



# Pythonの欠点

## 開発環境の構築が難しい

**Python**は公式の提供する開発ツールが貧弱であるため、サードパーティの開発ツールが乱立している

仮想環境: `venv`, `pyenv`, `pyenv-virtualenv`, `pipenv`, `Anaconda`

フォーマッタ: `black`, `autopep8`, `yapf`, `autoflake`

パッケージマネージャ: `pip`, `poetry`, `pipenv`, `Anaconda`

Lint: `flake8`, `pylint`, `Prospector`, `ruff`

→ 有名所だけでも**320**パターンの中から選定する必要がある  
他人のコードを動かす場合...



</>

では、解決策はあるか？

</>

# — Pythonの欠点を克服した新言語を作る

# 新言語のコンセプト

PythonのAPIを直接呼び出せる(トランスパイルされる)

先述のような欠点があるにもかかわらず  
Pythonが人気なのは、圧倒的な量の  
コード資産があるという点が大きい

これらのコード資産を流用できる言語があれば  
そちらに移行できる

このような既存言語の資産を流用できる言語は  
ScalaやTypeScriptなどがある




# 新言語のコンセプト

## 高度な静的型システムを持つ

静的型付けは実行効率のみのためにあらず  
型システムはコードの堅牢性を高めてくれる

新言語は依存型と呼ばれる高度な型を持つ  
これを用いると例えば配列の境界チェックなどを  
コンパイル時にも検査できる

```
tests > should_err >  dependent.er
1  dic = {"a": 1, "b": 2}
2  print!(dic["c"]) # ERR
3
4  arr = [1, 2, 3]
5  print!(arr[5]) # ERR
6  |
```

# 新言語のコンセプト

## 型推論機能を持つ

型推論とは変数や関数の型を指定せずとも  
自動で推論してくれる機能

動的型付けのようなシンプルな記述  
でありながら検査はしっかりと行われている

```
test.er
1  add (x: Int, y: Int): Int = x + y
2  print! add x:= 1, y:= 2
3  print! add x:= 1, y:= "a" # ERR
4  print! add x:= "a", y:= 1 # ERR
5
```



# 新言語のコンセプト

## 関数型+オブジェクト指向

オブジェクト指向(**Object-oriented**)はPythonやJavaなど多くの言語が採用するパラダイム

しかし近年は、より数学的で形式的な取り扱いが容易な関数型(**Functional**)プログラミングが注目されている

両者は組み合わせることも可能であり、そのようなアプローチを取る言語もある(e.g. Scala)

提案する言語もこのアプローチでいく



The screenshot shows the Scala Programming Language website. At the top, the Scala logo is on the left and a hamburger menu icon is on the right. Below the logo, the title "The Scala Programming Language" is displayed. A paragraph describes Scala as a language combining object-oriented and functional programming. Below this, three buttons are shown: "SCALA 3.2.2", "SCALA 2.13.10", and "ALL RELEASES". The main content area features a code editor with the title "Encode and decode custom data types to JSON". The code defines a case class Pet, creates an instance, and demonstrates JSON serialization and deserialization. A "Run in playground" button is in the top right of the code editor. The code is as follows:

```
case class Pet(
  name: String,
  kind: String
) derives Codec // enable coding Pet to and from text

< val coco = Pet(name = "Coco", kind = "Cat") >

val message = writeJson(coco)
// ^^^^^^^ contains the text: {"name":"Coco","kind":"Cat"}

readJson[Pet](message) // convert message back to a Pet!
```

At the bottom of the code editor, it says "2/4". Below the code editor, a note states: "The pluggable derivation system gives custom types new capabilities."

Source: <https://scala-lang.org>

# 新言語のコンセプト

## 開発ツールの統合

仮想環境マネージャ、フォーマッタ、  
パッケージマネージャ、**linter**などを  
すべてコマンドひとつに統合

このようなアプローチはGo言語などが採用  
環境構築が容易になり、  
高い再現性も保証される

```
~ via ▲ v3.23.2 via 🐧 v3.11.0  
at 21:54:34 > go  
Go is a tool for managing Go source code.
```

Usage:

```
go <command> [arguments]
```

The commands are:

bug	start a bug report
build	compile packages and dependencies
clean	remove object files and cached files
doc	show documentation for package or symbol
env	print Go environment information
fix	update packages to use new APIs
fmt	gofmt (reformat) package sources
generate	generate Go files by processing source
get	add dependencies to current module and install them
install	compile and install packages and dependencies
list	list packages or modules
mod	module maintenance
work	workspace maintenance
run	compile and run Go program
test	test packages
tool	run specified go tool
version	print Go version
vet	report likely mistakes in packages

# 現時点の進捗

## ✓ 文法の策定

文法は大筋で策定済み

詳細な仕様が公開されている

<https://erg-lang.org/the-erg-book>

## Basics

**Warning:** This document is incomplete. It has not been proofread (style, correct links, mistranslation, etc.). Also, Erg's syntax may be change destructively during version 0.\*, and the documentation may not have been updated accordingly. Please be aware of this beforehand. If you find any errors in this document, please report then to [here form](#) or [GitHub repo](#). We would appreciate your suggestions.

[Erg Book 日本語訳](#)

[Erg Book 繁體中文翻譯](#)

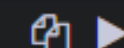
[Erg Book 简体中文翻译](#)

This document describes the basic syntax of Erg. If you already have experience with languages such as Python, please refer to the [quick tour](#) for an overview. There is also [standard API docs](#), [type definition](#) and [internal docs for contributors](#). If you need a detailed explanation of the syntax or Erg's internal architecture, please refer to those documents.

## Hello, World!

First, let's do "Hello World".

```
print!("Hello, World!")
```



This is almost identical to Python and other languages in the same family. The most striking feature is the `!`, the meaning of which will be explained later. In Erg, parentheses `()` can be omitted unless there is some confusion in interpretation. The omission of parentheses is similar to Ruby, but it is not possible to omit parentheses that can be interpreted in more than one way.

```
print! "Hello, World!" # OK
print! "Hello,", "World!" # OK
print!() # OK
print! # OK, but this does not mean to call, simply to get `print!` as a callable object
```



# 現時点の進捗

✓ プロトタイプの開発・公開

<https://github.com/erg-lang/erg>

2022年8月にGitHub上で公開

Rustを用いて開発

関数やクラスなど基本的な構文は実装済み

erg-lang / erg Public

<> Code

Issues 47

Pull requests 6

Discussions

Actions



main

13 branches

45 tags



mtshiba fix(lexer): multi-line token location



.cargo

chore: update



.github

Update rele



assets

chore(els): in



crates

fix(lexer): mu



doc

Automatic u



examples

test: update



lib

Update REA



src

test: check t



tests

fix: nested a



.envrc

Add Nix flak



.gitattributes

Update .gita



.gitignore

Add PyCode



.gitmessage

docs: update



.pre-commit-config.yaml

chore: update



CODE\_OF\_CONDUCT.md

Update CO

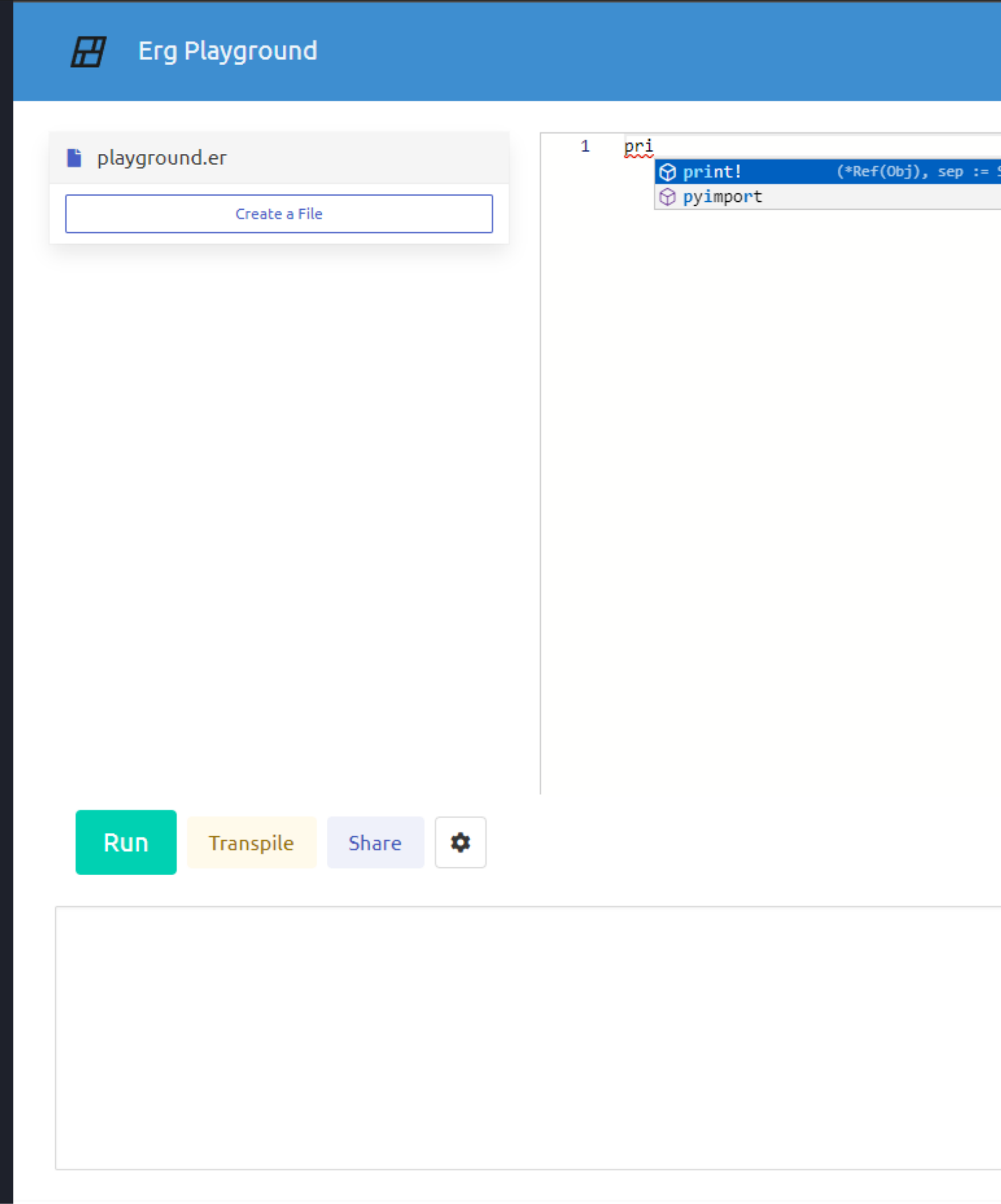
# 現時点の進捗

✓ プレイグラウンドの開発・公開

<https://erg-lang.org/web-ide>

最新版のErgをインストールせずにWeb上で試せる

WebAssemblyというブラウザ上で実行できる低級言語にコンパイルされ、ローカルで動作する





# 現時点の進捗

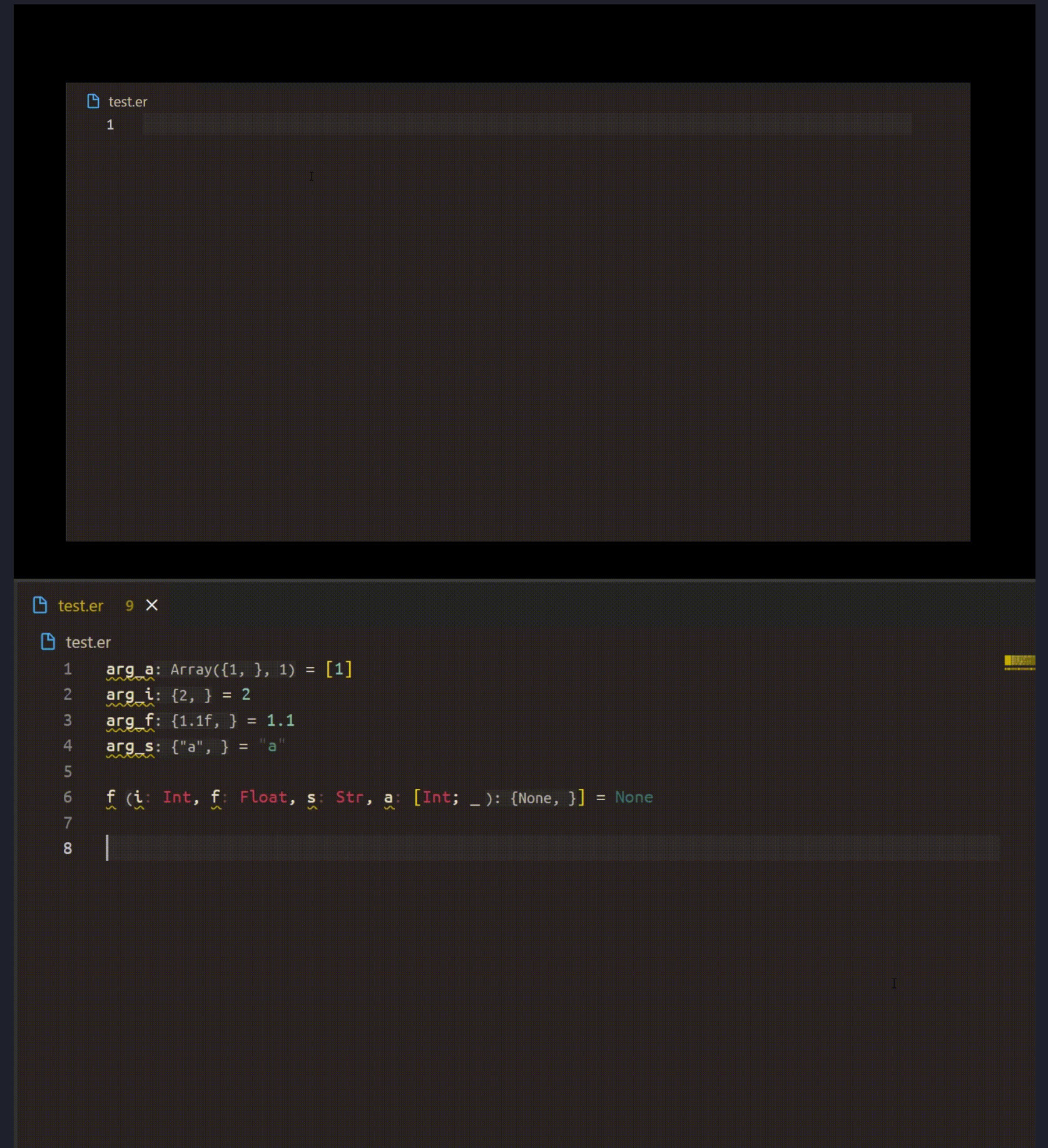
## ✓Language Serverの開発

RustのLanguage Serverである

rust-analyzerを参考に実装

エラーのハイライト、補完、rename

など基本的機能は実装済み



# 直近の進捗

## ✓ 著名なPythonパッケージの型付け

NumPyやpandasの型付けを行っている

ただPythonのType hintsと同じように型をつけるのではなく、

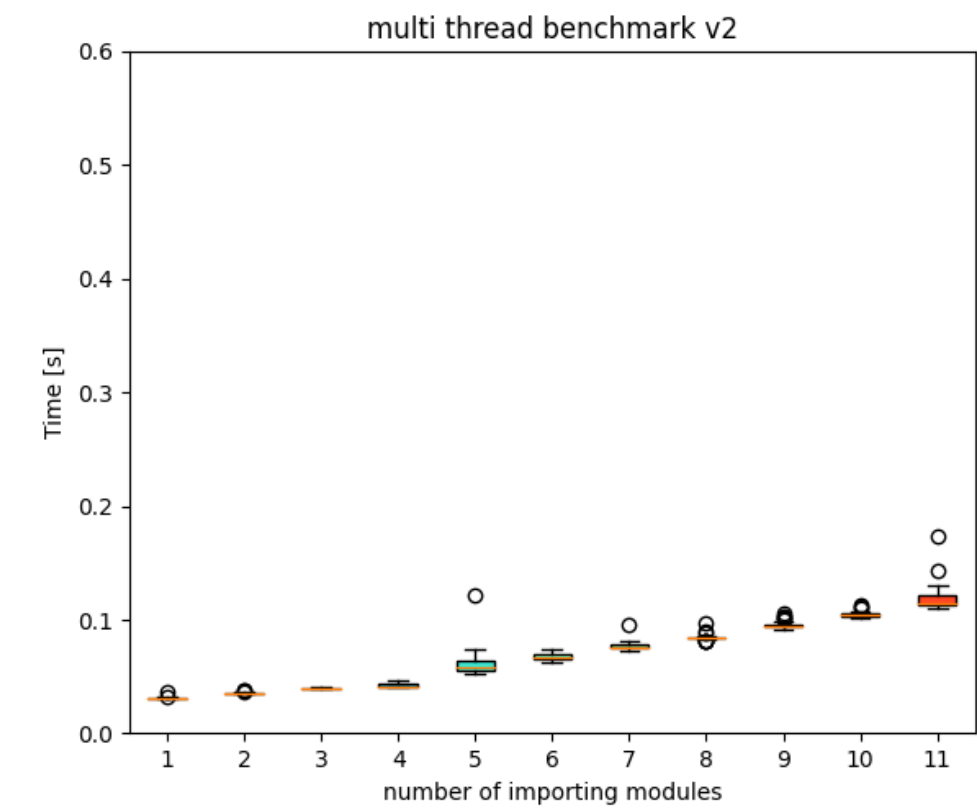
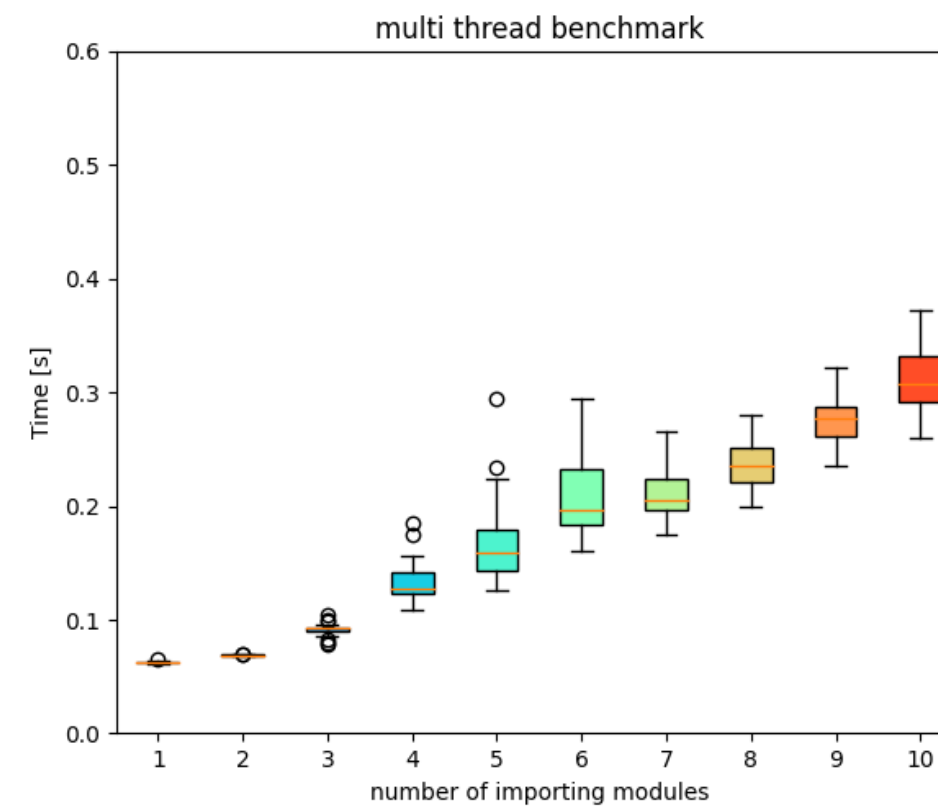
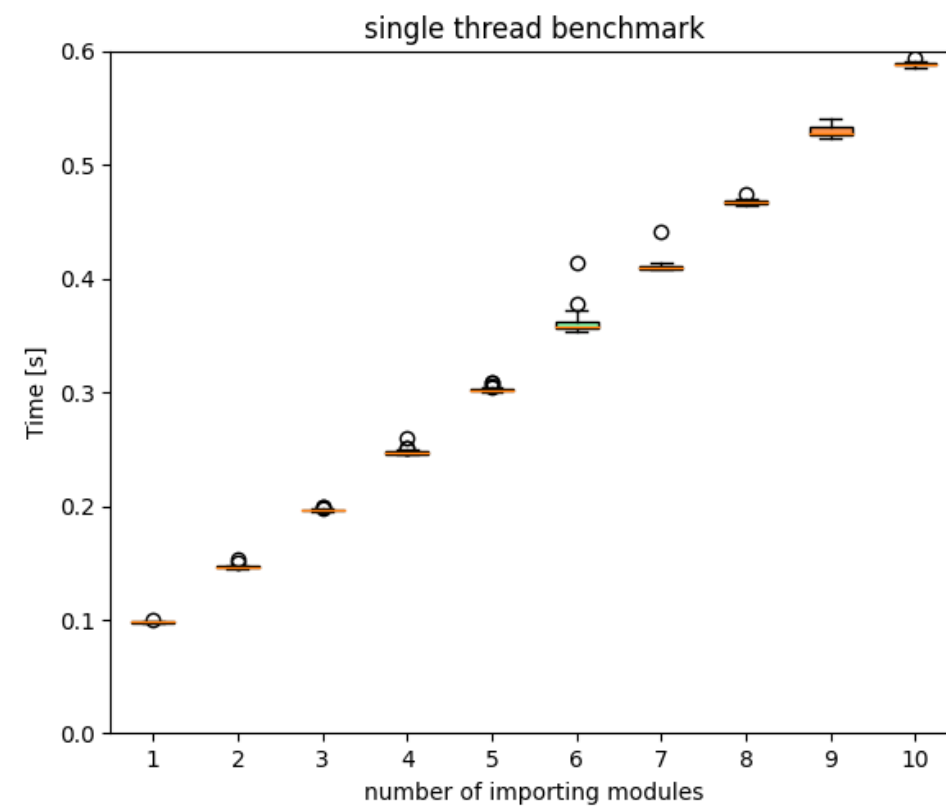
Ergの強力な型システムを活用する型付け

```
plt: PyModule("matplotlib.pyplot") = pyimport "matplotlib.pyplot"

_: matplotlib.figure.Figure, ax: matplotlib.axes._axes.Axes! = plt.subplots!()
_: matplotlib.figure.Figure, ax2: matplotlib.axes._axes.Axes! = plt.subplots!(nrows:= 1, ncols:= 1)
_: matplotlib.figure.Figure, axes: Array(matplotlib.axes._axes.Axes!, _: Nat) = plt.subplots!(nrows:= 1, ncols:= 2)
_: matplotlib.figure.Figure, axes2: Array(Array(matplotlib.axes._axes.Axes!, _: Nat), _: Nat) = plt.subplots!(nrows:= 2, ncols:= 2)
```

# 直近の進捗

## ✓ コンパイラの並列化



# 現状足りないもの

## 無数の言語機能

- Python APIの完全な型付け(現在3割程度)
- ユーザ定義多相型(単相型は実装済み)
- コンパイル時計算(現在は簡単な四則演算程度)
- 高度な可変データ型

Etc.

# 現状足りないもの

## ネイティブコードバックエンド

せっかく静的解析を行うのだからネイティブコードへコンパイルしたい  
**Python**の懸案であった実行性能問題も解決できる  
(**Python**スクリプト自体を高速化するわけではない)

インタプリタとバイナリでデータの受け渡しが必要だが、  
そのようなバインディングライブラリは先例あり (**Rust**  $\Leftrightarrow$  **Python**)

<https://github.com/PyO3/pyo3>

しかし**Rust**と**Python**は全く別の言語であるため、ライブラリを用いても些か煩雑  
同一の**API**と類似の意味論を持つコンパイル言語がほしい

→ ないので、つくる



# 現状足りないもの

## 開発ツール群

パッケージマネージャとフォーマッタは最低限ほしいところ  
**Lint**er、仮想環境マネージャも用意したい

できれば新言語自身を用いて実装したい

→実用性の証左になる

# やっていきたいこと

## コンパイラの最適化

インクリメンタルコンパイルを実装したい

コード生成も最適化すれば素のPythonより速くできるはず

</>

以上を、未踏期間中に実装する予定

# 想定される疑問

Q: Pythonの上位互換言語を作る？

A:

No

先述したようにPythonの文法には根本的問題があるため、必ずしも文法の互換性には拘らない

APIについても、**exec**関数など静的解析を阻害するものは排除する  
これらの使用は現在ではバッドプラクティスとされているため  
排除してもそこまで問題はない

# 想定される疑問

Q: PythonのAPIをどのように型付けするのか？

A:

組み込みAPIは手作業

ユーザースクリプトについては、型定義用のファイルを用意する

Pythonコード

e.g.

```
x = 0

def f(x: int) -> int:
    return x + 1

class C:
    def __init__(self, x: int) -> None:
        self.x = x

    def f(self, y: int) -> int:
        return self.x + y
```

型定義ファイル

```
.x: Int
.f: Int -> Int

.C: ClassType
.C.__call__: (x: Int) -> .C
.C.f: (self: .C, y: Int) -> Int
```



</>

提案者  
芝山駿介

発表日  
2023 / 4 / 15