言語紹介N連発

細合 晋太郎
（株式会社チェンジビジョン）

ChangeVision

自己紹介

- 細合 晋太郎 （株式会社チェンジビジョン）
- モデリング技術・ソフトウェア工学を使って、開発を支援するお仕事。
  プログラマのためのプログラマ

- 略歴
- 都島工業高校（機械電気）→ 近畿大学（情報系）→ JAIST修士→JAIST博士（めちゃ留年）：組込み向けモデリング→ 九大 enpit特任教員 → チェンジビジョン

- 趣味：ダイビング、スノボ、SF読み、ゲーム、水槽

ChangeVision

言語と私

- N88Basic、QuickBasic、TurboC、ポケコン（小中学生
- アセンブラ・C（工業高校
- C、Java、Pascal、Perl、PHP（大学
- C、OCamel、Prolog、Ruby、Groovy（修士
- C#、C++、Fortran、Java、Xtend、Xtext、Scala（博士
- Python、VBA（教員
- Java、Kotlin、Javascript/Typescript（なう

ChangeVision

触ったことのある

プログラミング言語リスト from Wikipedia

https://ja.wikipedia.org/wiki/プログラミング言語一覧

- action script
- awk
- basic
- bash
- brainfuck
- C ★
- C# ★
- C++ ★
- clojure
- COBOL
- Coffee Script
- csh
- D
- Delphi
- ECMAScript
- Elixir ★
- Erlang
- F#
- Fortran
- Go
- Groovy ★
- Haskell
- HSP
- Java ★ ★
- JavaScript ★
- Jruby ★
- Julia
- Kotlin ★ ★
- LISP
- Lua
- MATLAB
- Nim
- Ocaml
- Pascal
- Perl
- PHP
- PostScript
- PowerShell
- Prolog
- Python ★
- R
- Ruby ★
- Rust
- Scala
- Scratch
- sh
- Simulink
- Smalltalk
- sql
- Swift
- Tex ★
- tcsh
- TypeScript ★ ★
- Visual Basic
- Verilog
- VHDL
- Viscuit
- Whitespace
- Xtend ★
- Xtext ★
- zen
- zig
- zsh
- なでしこ
- 各種モデリング言語
- 日本語・英語

※ちょっと遊んだだけのものも含む

★よく使ってた言語、★ ★普段よく使ってる言語

もくじ

言語の分類

言語機能

ドメイン毎言語紹介

ChangeVision

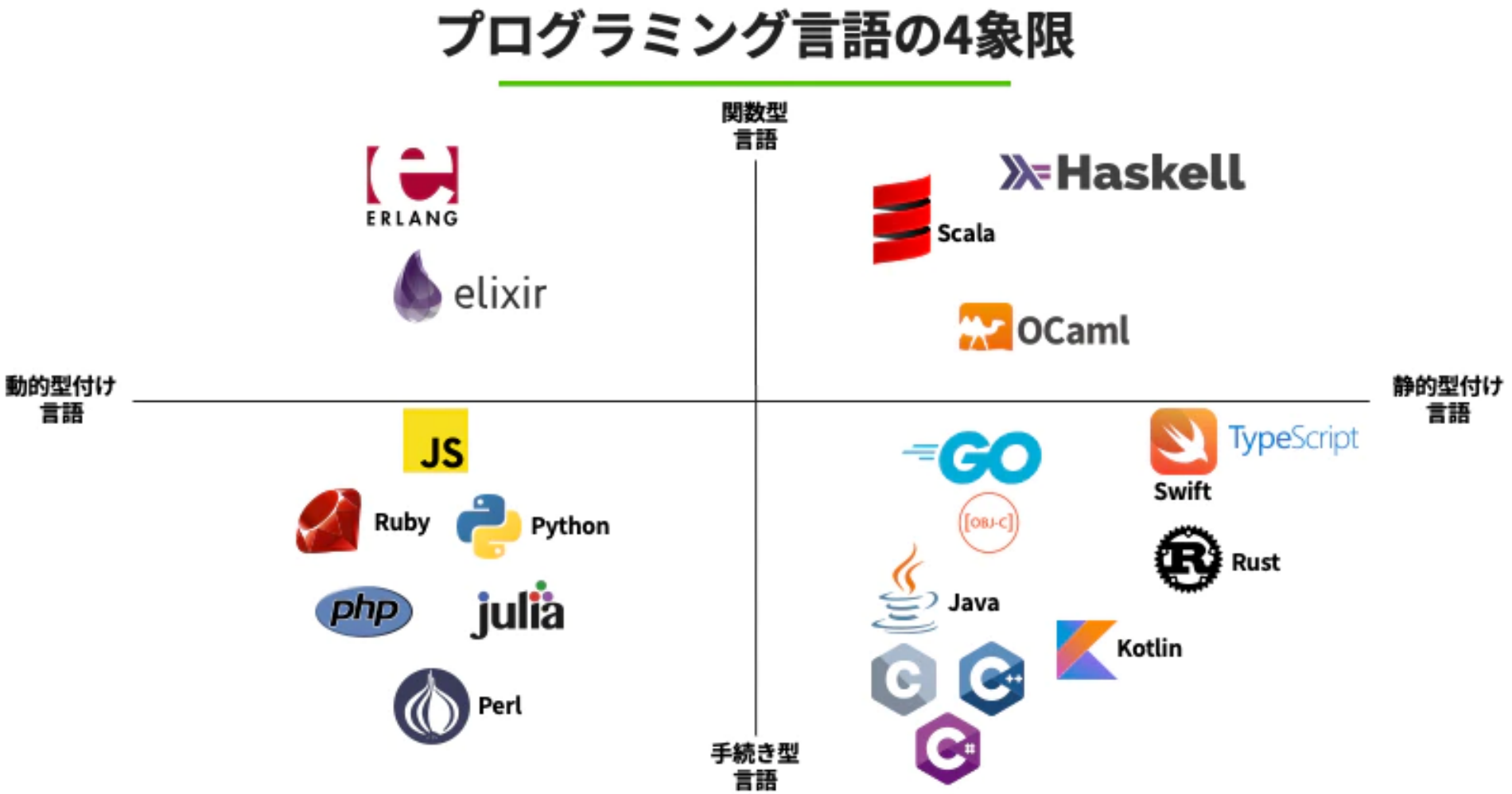
言語とは。

- コンピュータ様にお願いごとを伝えるお手紙
  - お願い事によって伝えやすさがことなる
- ナイフで釘が打てますか？木を切り倒せますか？
  - できるっちゃできる。
  - けど、合わない道具は使わない方がいいよね。
- 特化と汎化
  - 特化するほど、その分野での開発効率は上がるが、汎用性は下がる。汎化はその逆。 Cはかなり汎化寄り

目的に合わない・好みに合う言語がないとき
人は言語を作り出す

言語の分類

https://qiita.com/danwatanabe/items/b2dae4b73f6c18056f3d



ChangeVision

言語の数

https://ja.wikipedia.org/wiki/プログラミング言語一覧

- Wikipediaのプログラミング言語一覧
- 329個ありました。(2021/09/01現在)
- 結構入っていないのもあったので、500くらいはあるはず。
- さらに、独自の自作言語なども含めればさらに数倍に

ChangeVision

選び方（1）

- 課題解決力
  - できないといみない
  - 達成までどれくらい難しいか
- 制約
  - 環境、性能、安全性
- コスト
  - 購入コスト、導入コスト、運用・保守コスト、教育コスト、売り上げ
- 人気
  - 詳しい人が多くなる、採用も増える、解決策・教材が多い
  - ノイズが増える、玉石
  - メンテナが増える

選び方（2）

- プラットフォーム、ランタイム
  - CPU、OS、VM
- ライブラリ、フレームワーク
  - 十分に枯れているか、メンテされているか
- パッケージマネージャ、リポジトリ
  - バージョンコントロール、セキュリティ
- 言語仕様
  - 自身の頭の中を簡潔に記述できる機能・技術を選ぶ
- ドキュメント、利用者数

究極の言語

- 至高の汎用言語
  - アセンブリ
  - 機械語
  - りろんじょうは・・コンピュータのすべてを掌握できるが、まぁ無理っすね。
- 究極の特化言語
  - 書かない
  - 一文字だけ
- プログラム以外の解決策を探す

ChangeVision

言語機能

- 定義・構造
- 制御
- 環境
- メタ

ChangeVision

データ型

- int, doubleなどなど、基本的なデータ構造の扱い方
- 真偽値、整数、浮動小数、文字・文字列
- 複素数、ベクトル
- 構造体、クラス、列挙型
- 配列、リスト、マップ、タプル

変数

- データを入れておくための箱。殆どの言語で、型を基づいている。（動的言語でも内部的には持ってる）

- 静的型付け：予めコード内で明示的に型指定
- ちょっと賢い静的型付け：厳密に指定しなくても型推論でうまく解決してくれる。コンパイル時に解決
- 動的型付け：明示的に型付けせず、入っているものや演算に合わせて、その場で型が判別される。
- Mutable / Immutable
  - 変数に再代入可能か否か。
  - 極力Immutableにすることで、複雑さや状態を減らせる。レビュー時もImmutableになってると見る範囲が減って楽。
- 所有権。スコープの明示。

ChangeVision

関数

- 数学の関数と概ね同じ。何かしらの操作群をまとめたもの。
  - メソッド：クラスに属する関数。
  - サブルーチン：返り値無し。
  - 無名関数・ラムダ・クロージャ：　名前なしの関数。（厳密には全部違うけど、大体似たものって認識で大丈夫かと）
- 関数型言語
  - 厳密には、手続き的な記述でなく、すべて関数で記述する（Haskellとか）
  - 一級オブジェクトとして関数が扱えるもの。くらいの認識が広がってそう。

制御

- if, forなどは大体同じ。（関数型は除く）
- パターンマッチ
  - （1, “a”, 0.1）みたいなタプルを、以下のようなSwitch式のパターンに嵌めると

    (x, “a”, _) =>　← “a”がマッチ、1はxに拘束、0.1はスキップ

    かなり緩い条件で当て嵌められる。Elixir, Scalaなど。

- 文と式
  - 式として評価される。値を返すか否か。式超便利
  - ```
    a = when {
        x > 1 -> hoge(x)
        x == 0 -> foo(x)
        else -> bar() }
    ```

- 非同期記法。async/await, promise。プロセス・スレッド・コルーチン

```
async Promise<T> asyncronousFunc(a){
    result = await heavyProcess()     // ここで呼び出し元に戻る。
    return result }
```

  - Promiseはまだ結果が出ていないものを保持できる型。完了時のコールバックを持つ。
```

型

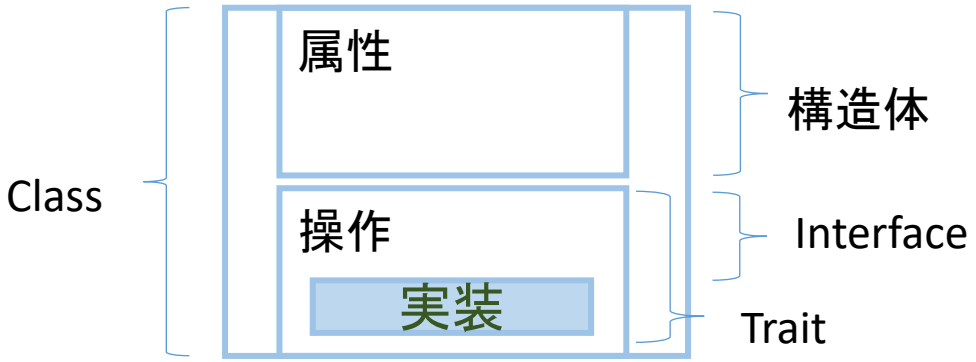
- データ型、クラスと同じ概念だがもう少し広い。

- 構造?に名前を付けたもの。データ構造に限らず、関数の型（Int → String：intを取ってStringを返す、という型）もある。という関数を返す関数型みたいなのも。

- ジェネリクス・テンプレート
  - List<T> みたいなやつ。型を後から差し込める仕組み
- Nullable, Optional, Either

構造体・クラス・インタフェース

- 構造体：任意のデータ型の組合せ
- クラス：データと操作・実装を纏めたもの
- インタフェース：操作のシグネチャのみ。責務の名付け
- Trait、Mixin：　実装込みの関数群。
  継承より安全・柔軟に機能を埋め込める



ChangeVision

module, package, namespace

- クラス群、ファイル群を一纏めにする仕組み、階層化も可能。ライブラリもパッケージの塊で提供されることが多い

- パッケージシステム
  - パッケージのバージョニング、依存関係、ビルド方法など、ライブラリを使う上で必要なあれこれ纏めたもの。大事。

- パッケージリポジトリ
  - ライブラリを取ってきて導入する仕組み
  - 最重要に近い。欲しいライブラリが見つけ易いか、整備されているか、依存関係

マクロ・リフレクション

- マクロ
  - コンパイルタイムにいろいろ埋め込む・書き換える仕組み
- リフレクション
  - ランタイムにあれこれ自身を書き換える仕組み
- 動的ロード、実行時ビルド・ロード、Eval
- DI
- 黒魔術的に扱えるものも多いが、
  凝るのはほどほどに。

ChangeVision

言語機能の裏側を知る

- トレードオフ
  - 便利な機能の長所短所
  - 簡潔に意図を残せる。抽象度が高い
  - 学習コストが高い。知らない人には分からない
  - 使い方を間違えるとパフォーマンス劣化や落とし穴
- 機能の裏側を知る
- 大まかな特性をつかむだけでいい

技術

- 最適化問題
  - ヒューリスティック
  - ソルバ
- 機械学習
  - 強化学習
- 画像解析
- 形態素解析
- パーサ、コンパイラ
- 数値解析、統計
- VR/AR/3Dモデリング

ChangeVision

言語紹介30連発くらい

- ドメイン別にザクザク紹介。
- 気になった言語があったら触ってみてください

ChangeVision

コンパイル　静的

C

http://www.open-std.org/jtc1/sc22/wg14/

とくちょう

信頼と実績のC言語。Unixと共に最初期の言語として長く使われている。低レイヤを扱う組込みでは、最重要言語と言っても過言ではない。

他の言語のように、あれこれシュガーが掛かっていない分、全部自分で面倒を見る必要があるが、細かい挙動まですべて制御できるメリットでもある。半面、下手な組み方をすると、自身の足を打ち抜きかねない。

いいところ

凡そなんでもできる。

まずはCでポーティングされることが多いので、ほぼすべての石で使える、はず。

むずかしいところ

他の言語と比べ、言語仕様の改定が少なく、ちょっと凝ったことをしようとすると、えらく大変

組込みシステム

コンパイル　静的

C++

https://isocpp.org/

とくちょう

こちらも長い歴史のある言語だが、Cに比べ相当はっちゃけてる。多くのOSやシステム周りのライブラリはC++で書かれており、性能の求められるものにはとりあえずC++といった感がある。

Boostが掛かったことで、言語仕様の外側で型周りや関数型、その他もろもろ継ぎ足されている。先進的な機能は他の言語にも大きく影響を与えている。っぽい。

Cが便利なサバイバルナイフなら、チェーンソーといったところ。

いいところ

Cと同等の自由と、強化されまくったパワフルさ。大体欲しいものは誰かが作ってくれてる安心感。

むずかしいところ

通常の仕様ならともかく、ちょっと凝ったことをしてるC++は、ほんと魔術に見える。ほんとむずい。

ChangeVision

コンパイル・LLVM　静的

Rust

https://www.rust-lang.org/

とくちょう

C, C++に代わるシステム言語を目指して作られた、比較的新しい言語（2010～）。特徴的なのがメモリ安全周りの機能で、リソースの所有者を指示することで、静的に生存期間を解決できる。また変数が基本的にイミュータブルなことも一躍買っている。

欲しい言語仕様は揃いつつ、危険な所を削ぎ落し・閉じ込め、安全なシステムプログラミングを提供する。

C++がチェーンソーならCNCくらいの安心感がある（ただしGコード手打ち）

いいところ

コンパイラがちゃんと叱ってくれて、下手に書けない・壊せない安心感がすごい。

むずかしいところ

思想に慣れるまで大変で、難しい言語と感じる。

llvm前提なので、ポーティングされてない石に持っていけないのが残念。

ChangeVision

コンパイル・LLVM　静的

Go

https://golang.org/

とくちょう

こちらも比較的新しい言語（２００９～）。静的型付けでインタフェースと構造体でモジュールを構築していく。

軽量スレッドのGoルーチン（コルーチン）が特徴的で、言語機能の中に軽量スレッド生成、チャネル通信、同期などが統合されている。

メモリ管理はガベージコレクションで行われ、deferによる遅延呼び出しも合わせて、リソースの取りこぼしにくい。

いいところ

言語機能で軽量スレッドがあるので、じゃんじゃか並行処理させられ、扱いもしやすい。

マスコットキャラがむかつく顔してて可愛い。

むずかしいところ

Rustに比べると、自由度が高くちょっと怖い。頭の中が並列な人じゃないと少し難しい気がする。（しっくり来なくてあまり触れていない）。こちらもLLVM前提。

組込みシステム

スクリプト?、コンパイル・mrb　動的

mRuby

https://mruby.org/

とくちょう

Rubyを組込み向けにポーティングしたもの。ではなく、組込みでも動くようかなり削ぎ落して軽量化したもの。PC上ではスクリプト的にもREPLにも動かせ、実機へはバイトコード化し、軽量VM上で動く。

Cとも組み合わせやすく、やわらかい部分をmRubyで深いところはCでといった使い分けると良さそう。

いいところ

書きやすいRubyで組込みを動かせる。手軽に試せて軽く動かせる。

むずかしいところ

Rubyで書いたときに、リソースや性能どうなるの？が見えにくいのが怖い。

ChangeVision

スクリプト? コンパイル・Beam　静的

Nerves/Elixir

https://github.com/nerves-project/nerves

とくちょう

SWESTでもファンの多いElixirを組込みでも動くようにしたもの。ホットデプロイとか、割とヤベェ機能が入ってて楽しい。ある程度のスペックが必要となるが、ネットワークに繋がってない石でElixir動かしても多分詰まらんので、これぐらいがちょうどいい。

詳細はElixirの項で改めて行うが、並行性がサクサクかけるので、リアクティブな構成にしやすい気がする。

いいところ

Elixir楽しい。IoT周りはかなり向いているかもしれない。

むずかしいところ

低レイヤはCやライブラリ任せなので、リアルタイムとかはちょっと・・。使い分け大事。

スクリプト？ 動的

Node-redとか

https://nodered.org/

とくちょう

センサとアクチュエータだけなら、マイコン上で制御しなくてもいいんじゃね？ネットワーク越しにIOだけ叩きゃいいじゃん的な代物。

ブロックプログラミングで主制御はNodejs上で動いて、マイコン側はネットワーク＋薄いデバドラ層になってそう。これで事足りるなら、これでいいじゃんという割り切りが素敵。IoT/ライトウェイト組込みの裾野を広げてくれている。

いいところ

ブロックプログラミングで、いろいろ動かせてしまう。初等向けかと思いきや、ラピッドプロトタイピングにも使われているそうな。

むずかしいところ

ネットワークの無い世界では生きてはいけぬ。

ChangeVision

トランスパイル？ 静的

Arduino

https://www.arduino.cc/

とくちょう

メイカー系の裾野を大きく広げたArduinoボード用の言語。C++やん？と思いきやほんのりトランスパイルしてくれてる。言語はともかくエディタが~~クソ使いにくい~~シンプル。

ちょいと書いてサクッと焼いて使える手軽さが素敵。対応ボードも多く、ライブラリも豊富。焼くときだけIDEで、コードはVSCodeであたりが楽。

いいところ

こんなん組込みじゃねぇ、と言うなかれ。お手軽さはやっぱり大事で、組込みの裾野も大きく広げてくれている。

むずかしいところ

中の方が見えづらい事もあり、はっ？勝手にポートやタイマー使ってんじゃねぇみたいな事もしばしば。

トランスパイル　静的

Nim

https://nim-lang.org/

とくちょう

結構あれこれ新しい言語仕様が入った“Cトランスパイラ”言語。個人的には、結構本命にできるんじゃね？と期待を寄せている。トランスパイラ（＝Cに変換されてビルドされる）ので、既存コードとも親和性が高く、コンパイラも選択できる。

低レイヤ周りの記法や、マクロによるあれやこれやで、使い方次第では結構なことができてしまいそう。

いいところ

欲しい言語機能は大体揃いつつ、Cになる安心感。組込みとも相性は良さそう。

むずかしいところ

場合によっちゃ生成されたCを自前でデバッグみたいなツラミもあるかも。

ハードウェア記述

- VHDL, Verilogの話をしようと思ってたけど、殆ど覚えてないのでスキップ

システム、CUI

- C,C++は前途したので省略。
- Perl、Shell、Awk、Sed、Dos batch、Powershellもざっと触れようと思ったけど、スキップ。

バックエンド

- Java
- Ruby
- Elixir
- Erlang
- PHP

コンパイル・JVM　静的

Java

https://www.java.com/

とくちょう

多分、Java利用の大半がWebバックエンド。普通のウェブサイトや、業務系のところでも広く使われている。オブジェクト指向の典型とみなされる言語で、大人数でごついものを作るのが得意。良くも悪くもオブジェクト指向を広めた立役者ともいえる。フレームワークも多く、スケールアップもしやすい。一方で、ゴリゴリ積み上げられてしまうため、設計がまずいと九龍城となってしまうことも多い。

いいところ

程よく硬くて堅実。VMのおかげで資産の使いまわしも行いやすい。言語的にも比較的書きやすい。

むずかしいところ

言語仕様は改定されつつも、ちょっと遅れ気味。Sun、Oracleのあれやこれやがまだつらい。

スクリプト　動的

Ruby

https://www.ruby-lang.org/

とくちょう

元々は汎用スクリプティング言語で、Perl後継？ってポジだったのが、Ruby on RailsのヒットでWebバックエンド界に一気に広がった。割と純オブジェクト指向寄りで動的型付けなので、かなり柔軟な記述にサクサク書ける。

RailsのscaffoldやActive Recordなどは、後に続くフレームワークにも大きな影響を与えてる模様。

いいところ

さくさく思ったように書ける＋フレームワークである程度縛った枠が与えられるということで、根強いファンも多い

むずかしいところ

自由度が高い反面、動的型付け・実行時評価なのでしっかりルールやテストで縛らないとごちゃりやすい。

スクリプト　動的

Elixir

https://elixir-lang.jp/

とくちょう

Erlangのランタイムを引き継ぎつつ、Rubyっぽい皮を被った動的型付け言語。こちらもサクサク書ける上にErlangパワーで並行性もかなり強い。

RoRの思想も言語環境レベルで引き継いでおり、さくっと強いWebアプリが書け、Rubyキラーのスペックは十分備える言語。また、パターンマッチやパイプラインなど、いい感じの言語仕様も素敵。

いいところ

割と難しい並行処理を直観的に書け、パッケージも豊富でいろいろ行いやすい。国内コミュニティが活発で心強い。

むずかしいところ

自由度が高く動的なため、しっかりテストで縛らないと、多人数開発は少し大変かも。パラダイムには比較的慣れやすい。

スクリプト　動的

Erlang

https://www.erlang.org/

とくちょう

交換機の裏側で生まれたらしい並行処理特化言語？軽量プロセスをサクサク作れ、エージェント的に包んで程よく管理できる仕組みが素晴らしい。

一時期Twitterのバックエンドだったり、大量に捌くシステムのバックエンドに割と生息している。

一方で、言語仕様が結構難解で、操るにはかなりの功夫が必要。Elixirがあるんで、もうそっちでいいかなって気がしてる。一方で、ElixirのライブラリにはErlangのものも多く、たまに読み解くスキルも必要となる。

いいところ

元より並行処理を前提として作られており歴史も長いので、とてもつよい。

むずかしいところ

でもやっぱ難しいからElixirでいいかなぁ。

スクリプト　動的

PHP

https://www.php.net/

とくちょう

フロントもバックも一緒に書けば楽じゃね？的な言語。サーバが対応してれば、ちょちょいと書いてアップロードするだけでWebアプリが作れるお手軽さが売り。Wordpressなど結構多くのCMSがPHPなので、まだまだお目に掛かることも多い。

言語仕様的にも比較的素直で、単体で書く分にはそれほど苦にはならない。歴史も長くDBなどとの運用も行いやすい。

いいところ

お手軽にWebアプリを仕上げるには手っ取り早い。フレームワークも多く、枠内では過ごしやすい

むずかしいところ

仕組み上HTMLとPHPが混在し、さらにJavascriptまで混じってくるとメンテ性が急激に悪化する。弄り倒したWordpressテーマなどは、地獄が広がってる。

サーバレス

- 言語じゃないけど。AWS Lambda

言語による

AWS Lambda

https://aws.amazon.com/jp/lambda/

とくちょう

ちょろっとしたコードを、必要な時に必要なだけ動かしてくれるサーバレスサービス。対応言語：。またDockerも動かせるので、実質なんでもOK。他のAWSサービスとも親和性が高い。

ちょっとしたものだけど、どこで動かそう・・？って問題は大体これで解決できる。うちのエアコンのOFF信号を埋め込んだめっちゃアバウトなコードが毎晩1時に起動して、さっさと寝ろを促してくれてます。

いいところ

サーバ建てたり、マイコン動かし続けたりってことから解放される。

むずかしいところ

微かに課金される程度。

たまに何動かしてたか分からなくなることも。

ChangeVision

インフラ構築

- Ansible, Chef, Vagrant
- Docker
- Terraform

スクリプト？

Ansible, Chef, Vagrant

https://www.ansible.com/ https://docs.chef.io/ https://www.vagrantup.com/

とくちょう

VMが流行りだした頃、続々と出てきたインフラ構築言語。くっそ面倒なインフラ構築・設定をスクリプトにして冪等性を保てるようにしたのがとても嬉しい。ちびちびインストールコマンドを叩いてた日々を塗り替えてくれた。

もっぱら、各種言語の内部DSLとして作られてて、比較的覚えやすい代物だったはず（うろ覚え）

最近はDockerにお株を奪われて、あんまり聞かなくなってしまった。

いいところ

インフラスクリプトを切り開いてくれた開拓者。ある種のパラダイムシフトだったのかもしれない。

むずかしいところ

あれこれ弄り回しているうちに整合性が崩れちゃったり、不要なものが入ったり。結構大変だった気もする。

ChangeVision

スクリプト？

Dockerfile, Docker-compose

https://www.docker.com/

とくちょう

VMを使い捨てにできる日が来るとは思ってなかったですよ。めっちゃ便利。CDからインストールして恐る恐るコマンドを叩いていた日々はなんだったの。

ベースとなるイメージにちょちょいとレシピを加えておけば、いつでもどこでも何度でも同じコンテナを作って捨てられる。

Composeの方では、複数のコンテナを組み合わせてサクサクとシステムを構築できる。

いいところ

とりあえずDocker。

むずかしいところ

いつのまにかゴリゴリとHDDを食い散らかしてくれます。

スクリプト？

Terraform

https://www.terraform.io/

とくちょう

実はまだ殆ど使った事がないんだけども。

AWSやAzure, GCPにあれこれサービス建てるの面倒だよね？スクリプトでゴリゴリやっちゃうよ、的なコンセプトの代物だと把握してる。

クラウドサービスはちょこちょこ使ってるものの、オーケストレーションが必要になる規模のものって作る機会は殆どない。

いいところ

なんかとっても便利らしい

むずかしいところ

課金されるサービスをゴリゴリ叩くのって怖くないっすか・・？

フロントエンド

- Javascript, ECMAScript
- Typescript

スクリプト　動的

JavaScript, ECMAScript

https://developer.mozilla.org/ja/docs/Web/JavaScript https://www.ecma-international.org/

とくちょう

元々ブラウザ上でスクリプトが動く仕組みがあると便利じゃね？みたいなところから始まったとか、そうじゃないとか。Javaと付いてるけどJavaらしさは殆どないし、下手に引き摺ると痛い目を見る、thisとか。

言語仕様としてはシンプルながら、プロトタイプベースのオブジェクト指向というのが割と曲者な気がする。需要に応じてエンジンもゴリゴリチューニングされており結構速い。V8を崇めよ。

いいところ

ほんと、JavaScriptでここまで何でもできるとは思ってなかったよ。

むずかしいところ

何回か仕様改定が入りつつ、未だにいくつかのJavaScriptが入り混じって、今書いてるこれはどのJavaScriptだっけ・・みたいなこともしばしば。

トランスパイル、スクリプト　静的

Typescript

https://www.typescriptlang.org/

とくちょう

名前の通りJavascriptにがっつり型が入ったトランスパイラ言語。AltJS勢を勝ち抜いてきた勝者。言語仕様としてもモダンなあれこれが入ってて扱いやすい。

型が堅過ぎて設計が固まらないこともしばしば。とはいえ、型の安心感はさすがのもの。コンパイル時におおよその問題は取っ払ってくれる。

最近は殆どのJavascriptライブラリでも型を付与してくれてるので、もうこれでいいんじゃね？感がある。

いいところ

程よく型々してて、好き。

むずかしいところ

型々しすぎてたまにうざい。

ChangeVision

フレームワーク 動的

React, Vue, Angular

https://reactjs.org/ https://jp.vuejs.org/ https://angular.jp/

とくちょう

言語じゃないんだけども。お作法を覚えないと結構つらいフレームワーク達。三大勢力図も割と書き換わり、とりあえずReact？くらいになってきた。

主に仮想DOMの扱いの違いがメインだけども、Vue, Angularが割とサクサク書ける反面、しっかり管理しないと大変。Reactは堅くて堅牢なイメージ。組むの大変だけど、ごちゃりにくい。

これに加えて、イベントや状態管理のフレームワークが入り組んでいて、なかなかに答えが見つけずらい

いいところ

ある程度大きいもの作るなら、React＋Typescriptかなぁ。サクッと作るならVueがいいかも。

むずかしいところ

覚えるのが大変なうえ、まだまだ流行が変わるのが速いんで、久々に触ると浦島る。

スマホ

- Kotlin（Android）
- Swift, Objective-C（iOS)

ChangeVision

スマホ/Android

コンパイル・JVM, スクリプト　静的

Kotlin

https://kotlinlang.org/

とくちょう

Javaに山ほどシュガーを振ったあまあま言語で、Scalaほど厳しくなく、Javaの痒いところは全部掻いてくれる、現時点で最愛の言語。

スマホカテゴリに入れてるけど、主にデスクトップアプリに使ってます。Javaの資産が全部使え、Maven/Gradleとも親和性が高い。IntelliJのサポートが凄まじく、ライブラリドキュメントとか見ずにザクザクかける。きもちいい。

いいところ

新しい機能はどんどん入り、書きたいものをサクサクかける気軽さ、膨大なJava資産。非の打ちどころのない言語！

むずかしいところ

凝るといくらでも難しく書けてしまうため、チーム内の制約が必要。

ChangeVision

コンパイル　静的

Swift、Objective C

https://developer.apple.com/jp/swift/

とくちょう

すみません、宗教上Macを触れないもので。詳しく知りません。

いいところ

むずかしいところ

ゲーム

- Unity
- Unreal Engine

コンパイル？ 静的

Unity

https://unity.com/

とくちょう

言語っぽくないけど、3Dビジュアルプログラミングとしてかなり完成度の高い代物。全部コードで頑張る世界から、ラフにオブジェクトの挙動と、細かい所はC#で補う仕組みはとても素晴らしく、ゲームプログラミングが一気に広がった。

某箱庭もUnityで作られているそうな。

いいところ

入門も揃っていて、サクッとゲームや3Dを弄りたい時は、とりあえずUnityで良さそう。

むずかしいところ

ちょっと凝ったものを作ろうとすると、ゲーム作成のノウハウが襲い掛かってくるイメージ。

コンパイル？ 静的

Unreal Engine

https://www.unrealengine.com/

とくちょう

コンセプト的にはUnityと似た方向性だけど、割と裏側でゴリゴリとC++を書くイメージ（ほんのりとしか触ってない）。Unityよりも表現力が高いような気がする、最近のゲームのタイトルロゴでよく見るよね。

いいところ

超綺麗なゲームとか作れる気がする。

むずかしいところ

むずかしそう。

機械学習、統計解析

- Python
- R
- Jupternote
- Julia

スクリプト　動的

Python

https://www.python.org/

とくちょう

機械学習周りの盛り上がりもあり、とりあえずPythonやっときゃ十分感。

古い言語なのでいろいろしがらみもありつつ、世界的に学校で取り入れてる事も多く、ユーザ数は膨大。おかげで優秀なライブラリも揃いつつ、何にでも使われる。上記の通り、機械学習ならやるならとりあえずPython。

あんまり好きじゃないんだけどなぁ。

いいところ

ライブラリが相当豊富。ユーザ数は力というのを実感する。

むずかしいところ

2とか3とか、PipとかCondaとか。あんまりいい印象なかったけど、改善された？

逐次評価 動的

Jupyter Notebook

https://jupyter.org/

とくちょう

言語としてはPythonだけど。Notebookの名の通り、ノートを取るように、ドキュメント＋コードが混在し評価実行される、文芸的プログラミング環境。環境の導入も行いやすく、さっと機械学習・統計の環境を整えられる。

対話的に進められつつ、それがそのままドキュメントとしても残る。一手一手が大きく、重く、意図が飲まれやすい機械学習と特に相性がよさそう。

なかなか使う機会を見いだせていないが、いずれじっくり触ってみたい。

いいところ

文芸的プログラミングってやっぱかっこいい。教育などにもとても強いと思う。

むずかしいところ

ドキュメント＋コード＋結果まで含むため、冗長にもなりがち。な気がする。用途次第。

スクリプト　動的

R

https://www.r-project.org/

とくちょう

こちらも結構古いながら、統計ドメイン特化で強い言語。複素数を含む様々なデータ型をサポートしつつ、統計に必要な演算も大抵揃ってる。

言語機能はシンプルながら、数学苦手勢にはかなりハードルの高い言語。

統計を扱う機会が少ないのもあるが、何度か挑戦して、負けた。

いいところ

統計つよい。

むずかしいところ

統計むずい。

機械学習、統計解析

コンパイル　動的

Julia

https://julialang.org/

とくちょう

数値解析・機械学習に強い言語らしい。動的言語ながらかなりパフォーマンスがよく、ごりごり計算を回しても速いとのこと。また、Cなど外部言語との親和性も高い模様。言語仕様はざっと眺めた感じ、結構盛り盛り。覚えるのは大変そうだけど、なんでも出来そうな感じが伝わってくる。

噂はよく聞くものの、実は触ったことがない。

いいところ

なんかつよそう。

むずかしいところ

かなりいいとこ突きそうな言語だけど、まだ情報少ない？

ChangeVision

言語をつくる言語

- BNF,eBNF
- Xtext

ChangeVision

言語を作る

ジェネレート？　静的

eBNF

https://ja.wikipedia.org/wiki/EBNF　（本家見つからず）

とくちょう

言語自体の定義を記述する言語。

ゴリゴリに自前パーサを書いてる言語も多いけど、言語定義を提供してくれているものもそこそこある。

一見複雑に見えるけど、ルールは単純なので覚えやすい。いっぽうで、読めはするけど、書く（言語を作る）のは大変。

いいところ

書けなくてもいいけど、読めるようになっておくと、言語解析とかが捗る。

むずかしいところ

シンプルながら、全構造を追おうとすると、しんどい。

ChangeVision

ジェネレート？ 静的

Xtext

https://www.eclipse.org/Xtext/

とくちょう

同じく言語自体の定義を記述する言語。なんだけど、メタモデル、AST、パーサ、エディタ、ジェネレータまで一気通貫で作ってくれる優れもの。

ＬＳＰ（Language Server Protocol）にも対応したそうで、自前言語のコンテンツアシストをVSCodeとかに追加したりできるそうな（あんまり追えていない）

博論でデバドラ向け言語を作ろうとして戦った因縁の言語。

いいところ

そこそこ難解ながら、自分だけのプログラミング言語を生み出して、エディタでサクサク書けるのはやっぱり楽しい。

むずかしいところ

結構な量のドキュメントと、概念や仕組みの理解が大変。

紹介したいけど、書ききれなかった言語供養

- ActionScript
- Assmbla(Pic, AVR, Z80, x86)
- basic
- Bytecode
- C#
- COBOL
- Coq/HOL
- F#
- Fortran
- GCode
- Groovy
- HSP
- Haskell
- HTML
- Lua
- OCamel
- PostScriptPostScript
- Prolog
- Scala
- Scratch
- Simulink
- Smalltalk
- Spin/Promera
- StandardML
- Tex
- VB
- Viscuit
- Xtend
- sql
- Zen
- Zig
- なでしこ

ChangeVision

未来のはなし

- プログラミングはAIに奪われる！？
  - 半分Yes。定型化できるもの、前例があるものはヤバい
  - 機械学習も、答えを出してくれるブラックボックスを半自動で作ってくれる
- AIツールとの対話が重要
  - 意図通りに一発で正確に生成するのは当面は無理（要電脳化？）
  - IntelliCode、Tabnine
  - jupyter notebook

ドメイン知識＋システム知識を持ち、AIと楽しくお話できる人は割と安泰

ChangeVision

プログラマがこの先生きのこるには

- “プログラマ”として生き残りたいですか？
  - “今の仕事”は突然の嵐で奪われるかもしれません
  - 一方、相対的にプログラミングが出来る人が減ってきます
  - プログラミングのスペシャリストになる
  - プログラミングを奪うプログラマになる
- “エンジニア”として生き残りたいですか？
  - 言語に拘らず、手段を問わず仕事が楽になる方法を模索し続けましょう
- とはいえ、やっぱりプログラミングは楽しいので、気になる言語があったらどんどん試してみてください！
  Enjoy Programming！

ChangeVision