

# 関数型プログラミング

## 第1回 HASKELLの概要

---

萩野 達也

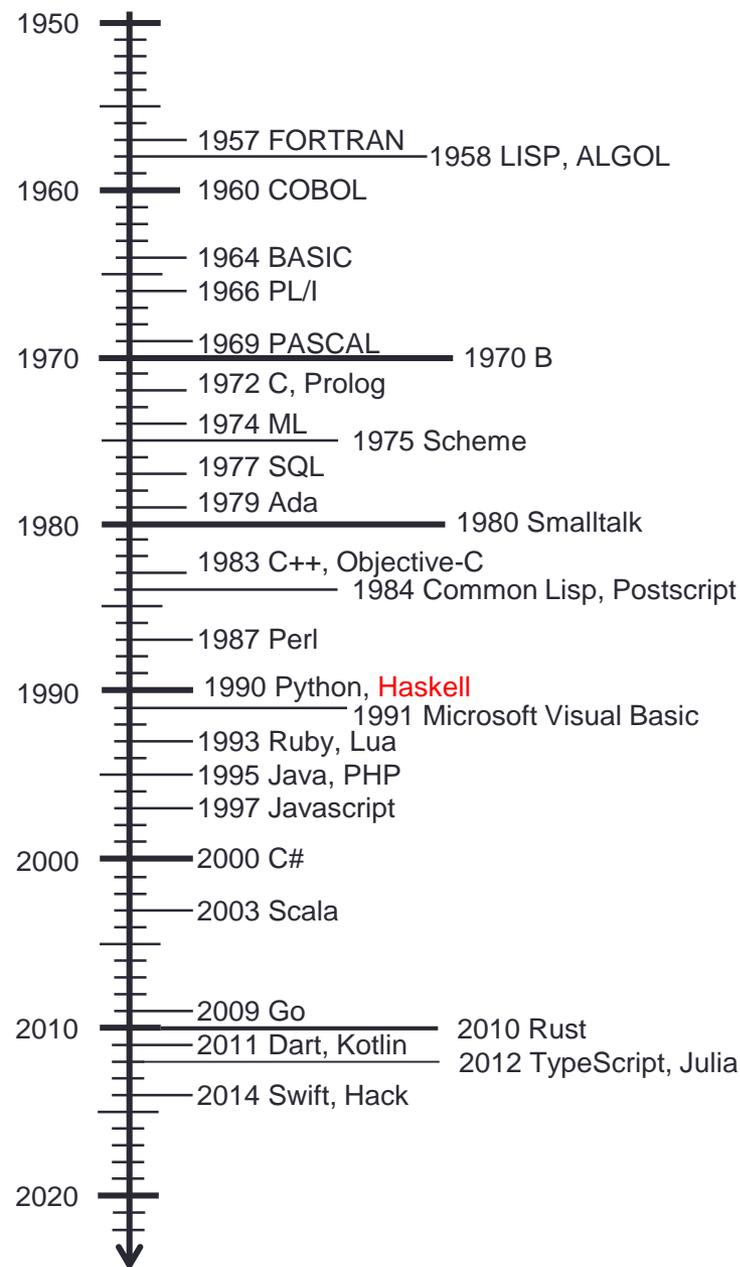
hagino@sfc.keio.ac.jp

Slide URL

<https://vu5.sfc.keio.ac.jp/slide/>

# プログラミングパラダイム

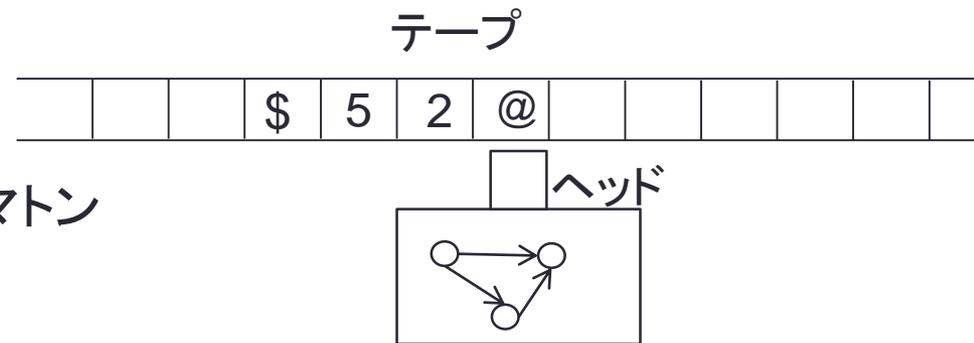
- 手続き型プログラミング
  - 問題を解くための手順を書く
  - プログラムは行ごとに順番に実行される
  - 最も一般的なプログラミング言語
  - 例: FORTRAN, C, Java, Javascript, ...
- 論理型プログラミング
  - 問題を解くための論理式を書く
  - 論理式の書く順序はあまり関係ない
  - 副作用がない
  - 代入文がない
  - 例: PROLOG
- 関数型プログラミング
  - 関数を組み合わせて問題を解く
  - 実行の順番は関係ない
  - 副作用がない
  - 代入文がない
  - 例: LISP, FP, ML, Haskell



# 計算のモデル

- チューリング機械

- 無限のテープと有限状態オートマトン
- 万能チューリング機械



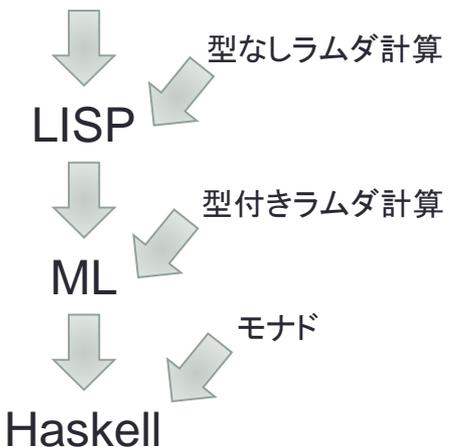
- 帰納的関数

- 原始帰納的関数 + 最小オペレータ
- $f(x+1) = (x+1) \times f(x)$ ,  $f(0) = 1$

- ラムダ計算

- 関数抽象 + 関数適用
- $(\lambda x. (\lambda y. xy)) (\lambda x. x) \rightarrow (\lambda y. (\lambda x. x) y) \rightarrow \lambda y. y$

## ラムダ計算

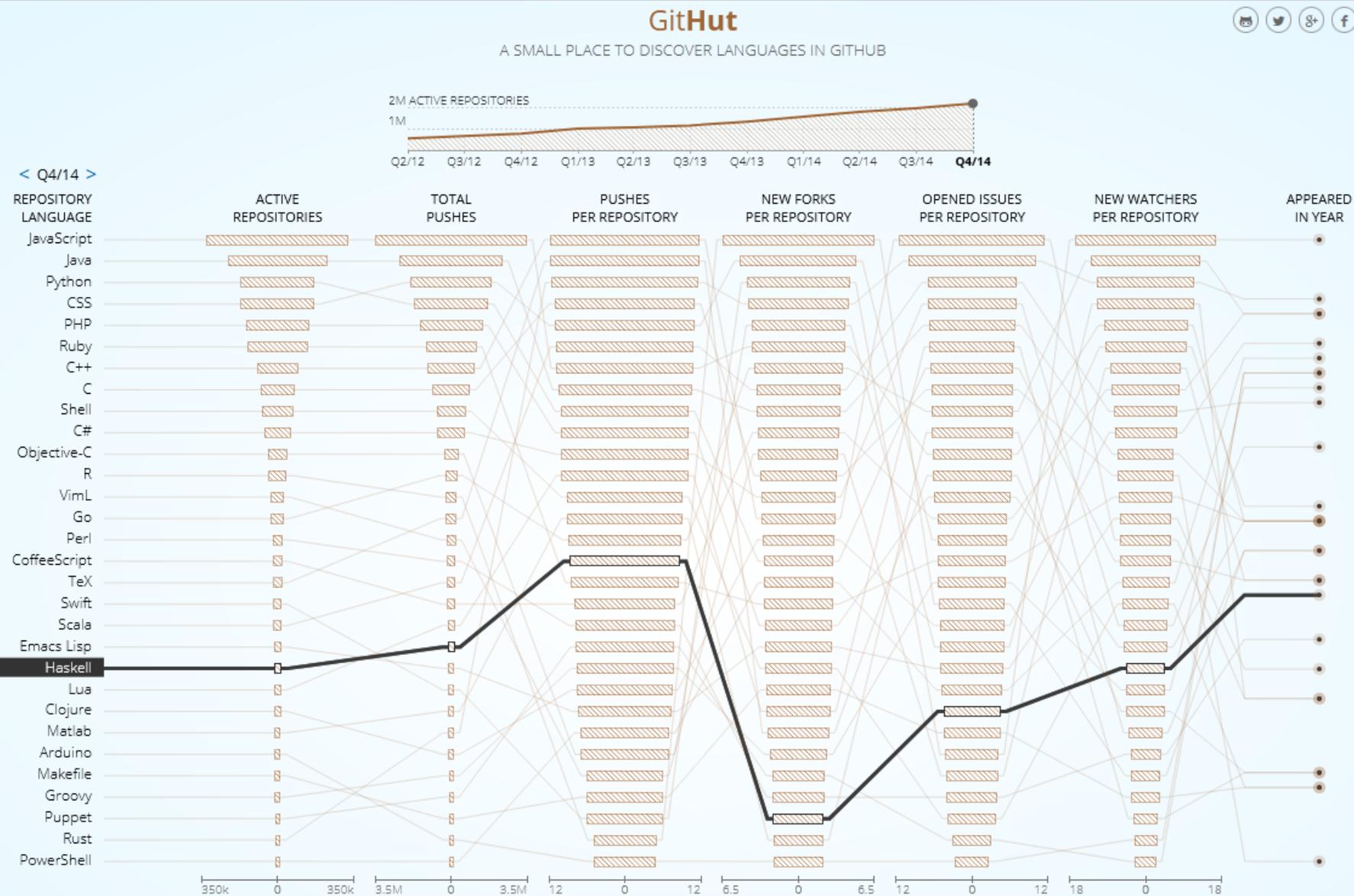


# Haskell

- 純粋な関数型プログラミング言語
  - 副作用がない
  - 参照透過性がある
- 強い型を持つ
  - コンパイル時に型チェックが行われる
- 多相型 (Polymorphism)
  - 関数は複数の型に適用可能
- 正格ではない
  - 遅延評価
- モナド
  - 計算の順序を与える

# Popular Languages

<https://github.info/>



# Haskell Brooks Curry



- アメリカの数学者 (1900/9/12 ~ 1982/9/1)
- 組み合わせ論理
  - S, K, I
  - ラムダ計算と同値
  - 変数は不要
- Curryのパラドックス
  - 「この文が真なら, サンタクロースは実在する. 」
  - 上記の文は偽ではないので真, したがって「サンタクロースは実在する. 」
- Curry-Howard対応
  - 論理学  $\leftrightarrow$  計算
  - 証明はプログラム
- カリー化 (Currying)
  - $(A \times B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C))$
  - 2引数の関数を1引数ごとに分けることができる

# 手続き型 v.s. 関数型

- $n$ 個のものを並べる並び替えの数  $n!$  を計算する.

## 手続き型

```
int fact(int n) {
  int f = 1;
  int i;
  for (i = 1; i <= n; i++) {
    f = f * i;
  }
  return f;
}
```

$$n! = 1 \times 2 \times \dots \times (n - 1) \times n$$

fact(3)

n	3
i	1
f	1

## 関数型

```
fact n = if n == 0 then 1
         else n * fact(n - 1)
```

$$\left\{ \begin{array}{l} n! = n \times (n - 1)! \\ 0! = 1 \end{array} \right.$$

$$\begin{aligned} \text{fact } 3 &= 3 * \text{fact } 2 \\ &= 3 * (2 * \text{fact } 1) \\ &= 3 * (2 * (1 * \text{fact } 0)) \\ &= 3 * (2 * (1 * 1)) \\ &= 3 * (2 * 1) \\ &= 3 * 2 \\ &= 6 \end{aligned}$$

# アルゴリズム

問題:  $n$ 個の整数の配列  $a$  が与えられたとき, 最大値を求めなさい.

$a$	31	41	59	26	53	58	97	.....	93	23
-----	----	----	----	----	----	----	----	-------	----	----

$n$

- 問題の定数を変数に置き換える. 「 $i$ 個の整数の配列の最大値を求める」
  - $n$ は配列の大きさを与えた定数
- 変数を1から定数まで変化させ, 解を近づけていく.
  - 「1個の整数の配列の最大値を求める」
  - 「 $i$ 個の整数の配列の最大値を求める」
    - ➡ 「 $i + 1$ 個の整数の配列の最大値を求める」

```
m = a[1];
for (i = 2; i <= n; i++) {
    if (a[i] > m) m = a[i];
}
```

# 分割統治

- 問題:  $n$ 個の整数の配列  $a$  が与えられたとき, 最大値を求めなさい.

$a$	31	41	59	26	53	58	97	.....	93	23
-----	----	----	----	----	----	----	----	-------	----	----

$n$

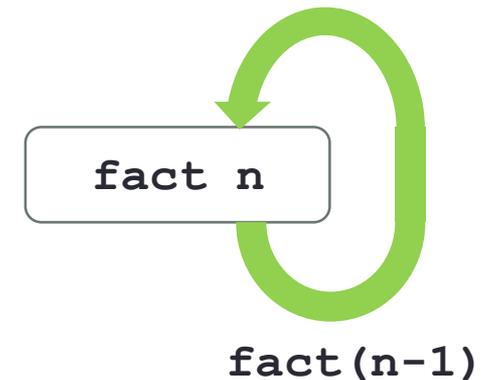
- 分割統治
  - 大きな問題を小さな問題に分割する
  - それぞれの問題を解決し, それらの答えを合わせて全体の解を作る
- 再帰呼び出し
  - 分割した問題が元の問題とサイズ違いなだけの場合がある
  - 分割を繰り返し適用する
  - 解が自明になるまで繰り返し適用する
- $n$ 個の整数の配列の最大値を求める
  - $(n - 1)$ 個の整数の配列の最大値を求める

```
max [x] = x
```

```
max (x:xs) = let m = max xs
              in if x > m then x else m
```

# 繰り返し文がない

- 手続き型言語には繰り返し文がある
  - for 文
  - while 文
  - do ... while 文
- 関数型言語には繰り返し文がない
  - 文自身がない
  - すべては式
- 再帰呼び出しで行う
- 再帰関数 = 自分自身を直接・間接的に呼び出す関数



```
fact n = if n == 0 then 1
         else n * fact(n - 1)
```

# 代入文がない

- 手続き型言語には代入文がある

$x = 3$

- 代入により変数の値は変化する

$x = x + 1$

- 関数型言語には代入文がない

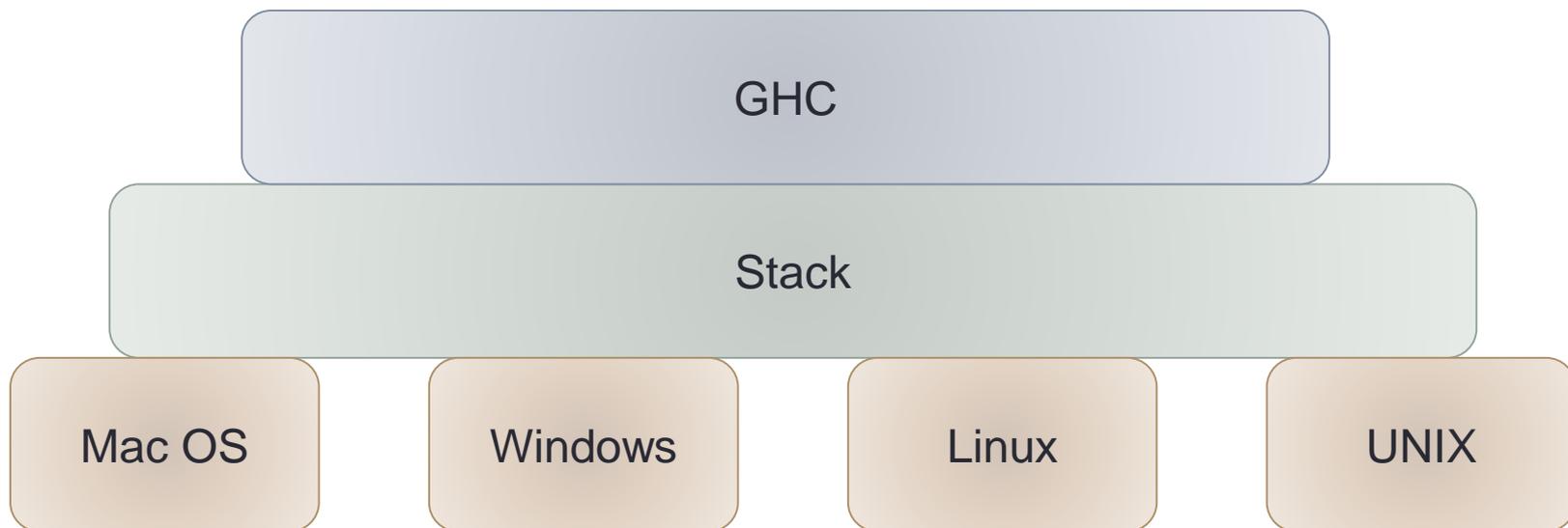
- 変数に値を結びつけると、その値は変化しない

$x = 3$

- 変数の値は変わらない.  $x$  は 3 のまま.
- 参照透過性がある
- 変数を右辺の値と置き換えても意味は変わらない

# Haskellのインストール

- Glasgow Haskell Compiler
  - Haskell Platform
  - 実行環境: Windows, Mac OS X, Linux, other UNIX
- Stack
  - Haskell用CLIビルドツール
  - <https://www.haskellstack.org/>



# Haskellのインストール (Intel Mac)

## Mac OS

### 1. stackのインストール

- Homebrewを利用する場合

```
$ brew install stack
```

- Homebrewを利用しない場合

```
$ curl -sSL https://get.haskellstack.org/ | sh
```

### 2. PATHの設定

```
bash $ echo 'export PATH=~/.local/bin:$PATH' >> ~/.bashrc
```

```
csh % echo 'set path=(~/.local/bin $path)' >> ~/.cshrc
```

### 3. Xcode Command Line Toolsのインストール

```
$ xcode-select --install
```

# Haskellのインストール (Arm Mac)

## Mac OS

### 1. stackのインストール

- Homebrewを利用する場合

```
$ brew install stack
```

- Homebrewを利用しない場合

```
$ curl -sSL https://get.haskellstack.org/ | sh
```

### 2. Rosettaのインストール

```
$ softwareupdate --install-rosetta
```

### 3. PATHの設定

```
bash $ echo 'export PATH=~/.local/bin:$PATH' >> ~/.bashrc
```

```
csh % echo 'set path=(~/.local/bin $path)' >> ~/.cshrc
```

### 4. Xcode Command Line Toolsのインストール

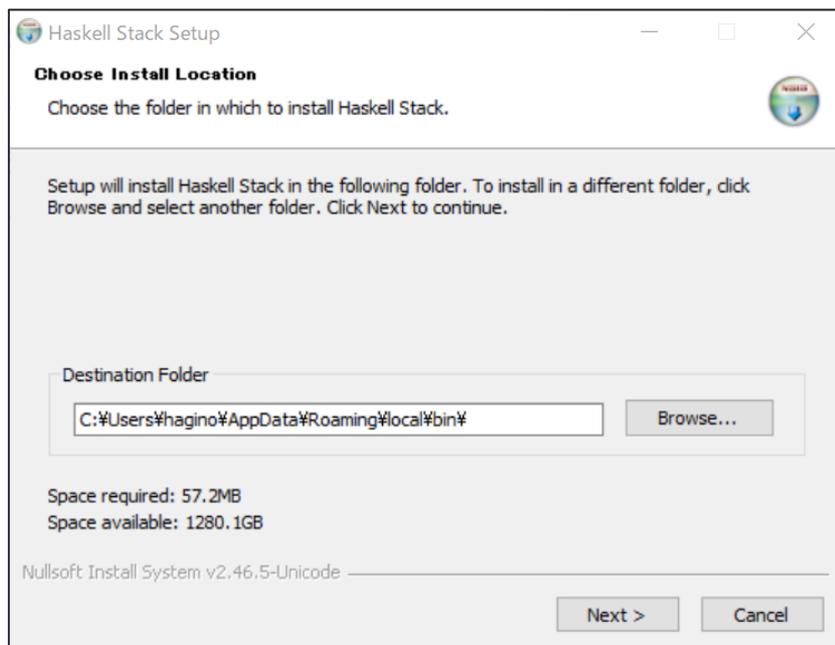
```
$ xcode-select --install
```

# Haskellのインストール (Windows)

## Windows

- 全角文字の含まれないユーザで行う。
- Windows 64-bit Installer をダウンロードして実行

[https://get.haskellstack.org/stable/windows-x86\\_64-installer.exe](https://get.haskellstack.org/stable/windows-x86_64-installer.exe)



# Haskellのインストール(GHC)

- stackをすでにインストール済みの場合には最新のものにする

```
$ stack upgrade
```

- GHCをインストール

```
$ stack setup
```

- インストールの確認

```
$ stack ghci
Configuring GHCi with the following packages:
GHCi, version 8.8.4: https://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from C:\cygwin\tmp\haskell-stack-ghci\2a3bbd58\gh
ci-script
Prelude> 1+2
3
Prelude> :q
%
```

# Hello, World!

- Haskellのプログラムを実行してみる.

1. 次のプログラムをhello.hsに準備する(テキストエディタを利用).

```
main = putStrLn "Hello, World!"
```

2. ghcコマンドを使ってコンパイルする(コマンドプロンプト, ターミナル).

```
% stack ghc hello.hs  
[1 of 1] Compiling Main ( hello.hs, hello.o )  
Linking hello ...
```

3. コンパイルされたプログラムを実行する(コマンドプロンプト, ターミナル).

```
% hello  
Hello, World!
```

Windows

```
C:¥> hello.exe  
Hello, World!
```

# 直接実行と対話的実行

- プログラムを開発中などは、コンパイルする時間が面倒になる時があります。
- `runghc`コマンドを使って直接実行することも可能です。

```
% stack runghc hello.hs
Hello, World!
```

- 対話的に`ghc`を起動することも可能です。

```
% stack ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Prelude> putStrLn "Hello, World!"
Hello, World!
Prelude> 1 + 1
2
Prelude> :quit
```

# mainアクション

```
main = putStrLn "Hello, World!"
```

- これは変数mainの定義です.
- mainの値は関数ではなく、アクションです.
- putStrLnは関数です.
  - putStrLnの引数は文字列リテラル"Hello, World!"です.
  - putStrLnは文字列を出力するアクションを返します.
- Haskellのプログラムが起動されるとmainのアクションを評価します.
- 関数の適用に括弧は必要ありません.

<pre>main = putStrLn("Hello, World!")</pre>	$f(x)$
<pre>main = (putStrLn "Hello, World!")</pre>	$(f\ x)$
<pre>main = putStrLn "Hello, World!"</pre>	$f\ x$

- 上記のどれも同じです.

## 2つ以上のアクションを行う

- "Hello, World!" と "Hello, SFC!" を2行に書いてみよう！

```
main = putStrLn "Hello, World!" "Hello, SFC!"
```

↓ の意味は

```
main = ((putStrLn "Hello, World!") "Hello, SFC!")
```

- 動かない

```
main = putStrLn "Hello, World!"  
      putStrLn "Hello, SFC!"
```

- 動かない

```
main = (putStrLn "Hello, World!") >> (putStrLn "Hello, SFC!")
```

- 動く

# 練習問題1-1

- 3行以上の自己紹介を出力する `intro.hs` を作成しなさい。

```
% stack runghc intro.hs
My name is Tatsuya Hagino.
I am from Hyogo prefecture.
I like cooking.
```

- 文章は日本語でもかまいません。
- 文字列の中にUTF-8で書いてください。
- 宿題はK-LMSを使って提出してください。
- 期限は授業のあった週の土曜日までです。
- 授業で説明した内容のみで回答してください。
  - 未説明の内容を使った場合には0点となります。
  - ネットのものをコピーしたり, 人のものをコピーした場合には, Dが確定します。