プログラミング言語論 #04

Part 2: JavaScript

2018-05-07

OUTLINE

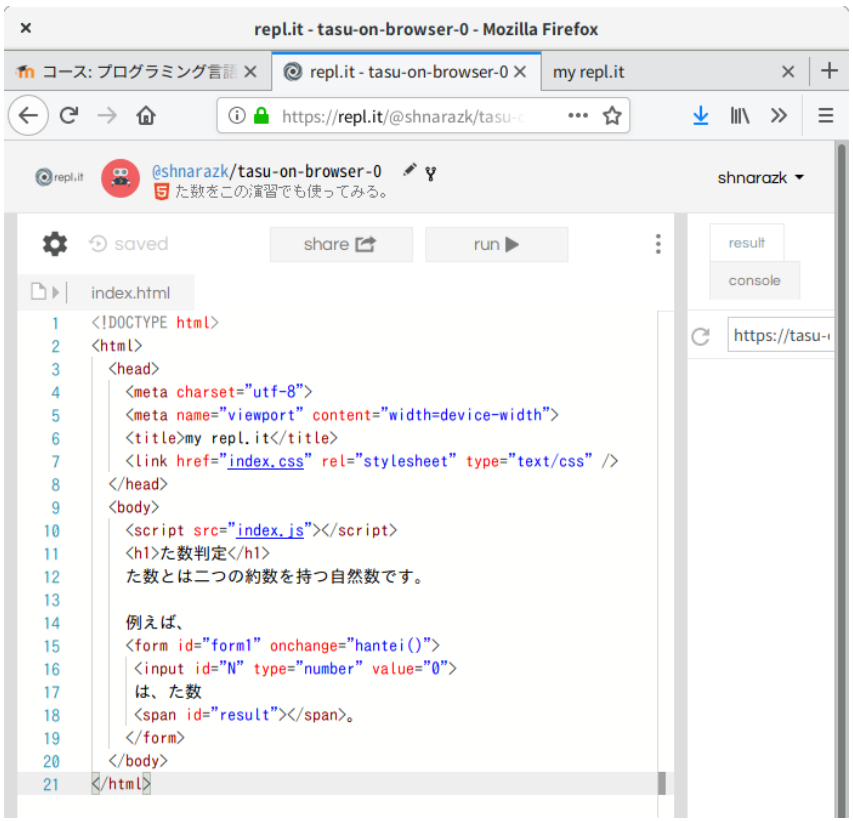
型の重要性を知る上で，異なる立場に立つ言語を知ることも有用であろう．
近年広く使われている言語にもそのようなものがある．

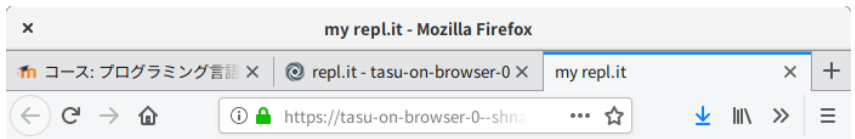
今日は JavaScript 言語について触れる．

サンプル

▸ version 0

- index.html – html ファイル
- index.css – スタイルファイル
- index.js – JavaScript 言語によるスクリプト





- ▸ version 0
- ▸ version 1: 不完全 tasu 関数付き

index.html

1. header 部 – XML 設定，スタイル，外部リンク
2. body 部 – スクリプト指定，本文

```
1  <!DOCTYPE html>
2  <html>
3   <head>
4    <meta charset="utf-8">
5    <title>my repl.it</title>
6    <link href="index.css" rel="stylesheet" type="text/css"/>
7   </head>
8   <body>
9    <script src="index.js"></script>
10   <h1>た数判定</h1>
11   た数とは二つの約数を持つ自然数です.
12
13   例えば,
14   <form id="form1" onchange="hantei()">
15    <input id="N" type="number" value="0">
16    は，た数
17    <span id="result"></span>.
18   </form>
19  </body>
20 </html>
```

JavaScript と html の関係

▸ version 1: 不完全 tasu 関数付き

index.js(version 1.0)

```
1  function hantei () {
2    var s = document.getElementById("N").value;
3    var x = parseInt(s);
4    if (...) { document.getElementById("result").innerHTML = "です";
5    } else   { document.getElementById("result").innerHTML = "ではありません"; }
6  ...
```

index.html

```
8  <body>
9   <script src="index.js"></script>
10  <h1>た数判定</h1>
11  た数とは二つの約数を持つ自然数です.
12
13  例えば,
14  <form id="form1" onchange="hantei()">
15   <input id="N" type="number" value="0">
16   は, た数
17   <span id="result"></span>.
18  </form>
19 </body>
```

1. (L15) N に入力するとイベント発生
2. (L14:onchange) コールバック関数 hantei を呼出し
3. (L2:getElementById) N の値取得
4. (L3) 文字列から数値へ型変換
5. (L5:getElementById) result に出力

Definitions in JavaScript

型の宣言がない

index.js 再掲

```
1  function hantei () {
2    var s = ...
3    var x = ...
4
5  function tasu (n) {
```

var は変数の宣言かつ定義である．ただし型情報はない．

何型？ 自分で考えろ ＞ コンピュータ，他人

function は関数の宣言かつ定義である．ただし型情報はない．

何型？ 自分で考えろ ＞ コンピュータ，他人

Definitions in JavaScript

型の宣言がないのはなぜか

型エラーのない index.js の抽象化

```
1  function hantei () {
2    var s = 1;
3    var x = "です";
4    ...
```

- 型を**プログラマ**が書かない → 書かなくてもよい
- なぜなら，コンパイラが決めているから
- コンパイラは間違えない．なぜならプログラムをよく見ればわかるから
  - 定義時の右辺を見れば何型かわかる
  - inedex.js では文字列を (+) しているのにエラーではない
  - hantei は return していないから返値は void 型で決定

プログラミング言語論専門用語

4

type inference, 型推論

直接与えられない型情報（宣言）を与えられた他の型情報（宣言）を基に推論する．

JavaScript がしていること． 実は Haskell もしていること

type check, 型検査

与えられた型の情報を基に，全ての箇所で型が妥当かどうかを検査する．妥当でなければ型エラー．

C， Haskell, JavaScipt がしていること

C のコンパイラは型推論はしてないが型検査はしている

JavaScript におけるもう一つの不思議な挙動

正しく動く JavaScript のプログラム例

```
1  function f() {
2      var i = 2;    // この時点ではiは整数型
3      ...
4      i = "AAAA";    // この時点でiは文字列型？
5      ...
```

- 型エラーなく動く（型検査でひっかからない）
- 型はコンパイラが決めたはず
- JavaScript では型は時間と共に変化してよい

これは型を人間が宣言するのか，コンパイラが推論するのかとは別の話

決めたものは変えられないのか変えていいのか

プログラミング言語論専門用語

5

strongly typed, 強い型付け言語： C, C++, Haskell, Java ···

変数や式の型が途中で変わることを許さない (時間変化しない)．

かっちり決めたい，なぜなら：

- それはプログラムとしておかしい

weakly typed, 弱い型付け言語： JavaScript, Python ···

強い型付けでない言語

かっちり決めたくない，なぜなら：

プログラミング言語論専門用語

6

static typed, 静的型付け言語： C, C++, Haskell, Java ···

全ての変数の型がコンパイル時に決まっている.

実行前に決めたい，なぜなら：

- 実行前に全てのエラーが検出できるかも
- 高速なプログラムにコンパイルできるかも

dynamic typed, 動的型付け言語： JavaScript, Python ···

静的型付けでない言語

実行しながら型を決めたい．それまではしたくない．なぜなら：

多様性の理解

- 人間がコンパイル時に型を決め，コンピュータはそれを実行前に確認する
- コンピュータがコンパイル中に型を決め，人間はそれに従う
- コンピュータが実行中に型を決め，人間は幸運を祈る

それぞれの立場を代弁せよ