プログラミング学習環境 PyPEN は純粋な共通テスト 演習環境であるべきか

中西渉

watayan@meigaku.ac.jp
名古屋高等学校

2026.02.08

（しょっぱなから）付記

LuaLaTeX 用にクラスファイルを修正

- 学会提供クラスファイルは pLaTeX 用

→ LuaLaTeX でコンパイルできるように修正

- 本予稿はこれで書いてみた
- GitHub で公開中
  https://github.com/watayan/ipsj_cls_for_techrep
- ipsjunsrt.bst も微修正
  url パッケージで URL を折返し表示

> [6] watayan: PyPEN, (online), available from ⟨https://github.com/watayan/PyPEN⟩ (accessed 2025.12.22).

- 権利関係は未確認

はじめに

大学入学共通テスト（以下「共通テスト」）

- 2025 年度から「情報」
  プログラミング問題「共通テスト用プログラム表記」[1]
  - 日本語ベースの疑似コード
  - Python に似た構文
- 教科書で用いられているのは Python, JavaScript, VBA, Scratch など
  - 授業はこのような実用的な言語で行うべき [4]
  - 疑似コード設計者もこの言語 (?) での学習は望まない [5]
- しかし演習は必要

  → そのための実行環境はほしい（プログラミングだから）
  - PyPEN はわりと使われてる（らしい）

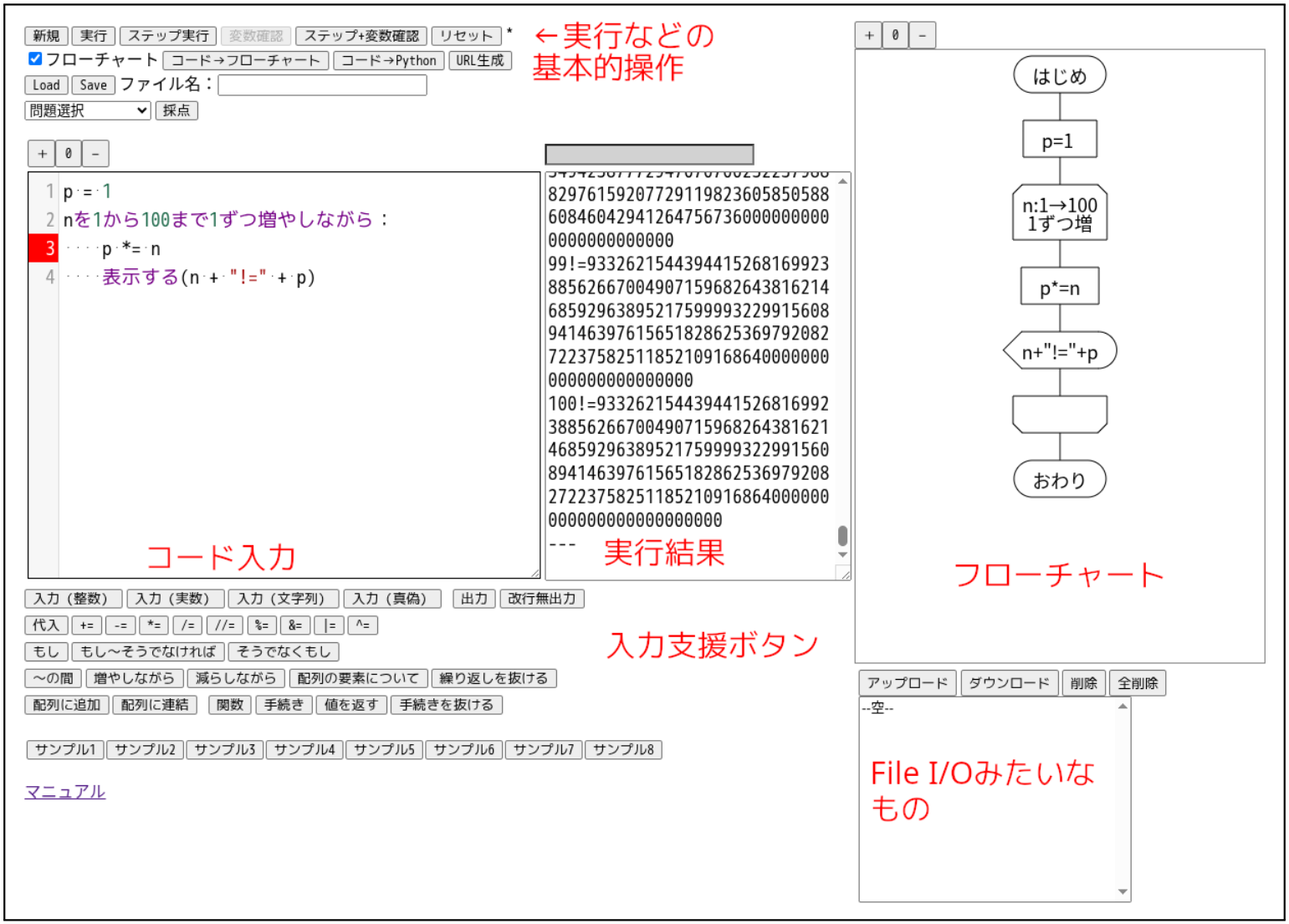
PyPEN の位置づけはどうあるべきなのか
純粋な演習環境？
自由なプログラミング環境？

PyPENの概要

PyPENとは

- Webブラウザで実行できるプログラミング学習環境
- 独自言語
  - 日本語ベース
  - Pythonに似た構文
- JavaScriptで実装（サーバの機能を利用しない）
  置き場所を選ばない
- GitHubで公開（MITライセンス）

- コード →Python コード
- コード →URL
- コード ↔ フローチャート



PENからPyPENに至る経緯

PEN

- 西田らが開発した初学者用プログラミング学習環境 [7]
- 「情報関係基礎」の DNCL を元にした xDNCL
- Java アプリケーション

PenFlowchart

- 筆者が PEN を拡張し，フローチャート 機能を追加 [9]

WaPEN

- PenFlowchart を Web アプリ化 [10]

PyPEN

- WaPEN の言語を Python 風に変更 [11]

PyPENで用いる「言語」の細部

「共通テスト用プログム表記の例示」[2] とは

- 「言語」ではない
- あくまでも「例示」
- Python3，JavaScript，VBA との対比を掲載
  これらの言語での相違点は使われないだろう

i を 0 から 9 まで 1 ずつ増やしながら繰り返す：
    表示する (i)
表示する (i)

Python

```
for i in range(10):
    print(i)
print(i) # 9
```

VBA

```
FOR i=0 TO 9
    Cell(i,1).Value = i
NEXT i
Cell(1,2).Value = i ' 10
```

JavaScript

```
for(i = 0; i < 10; i++){
    console.log(i);
}
console.log(i) // 10
```

「共通テスト用プログム表記の例示」[2] とは

- 「言語」ではない
- あくまでも「例示」
- Python3，JavaScript，VBA との対比を掲載
  これらの言語での相違点は使われないだろう

しかし PyPEN はプログラム実行環境
∴「言語」を「定義」する必要がある

- 共通テスト演習で使われる以上，出題されるプログラムが動作する必要がある
- しかし，出題されない部分についてはどうするか？
  使われない部分は独自に定義していいのでは？

きっと使われないと判断して，独自に定義した機能

1. 参照の値渡し
2. BigInt への対応
3. 関数・手続きの一本化
4. 組み込み関数の充実
5. 正規表現
6. range もどき
7. Python のキーワード 導入

参照の値渡し

旧版

```
x = [1, 2, 3]
y = x
y[0] = 9
表示する (x)  # [1, 2, 3]
表示する (y)  # [9, 2, 3]
```

新版

```
x = [1, 2, 3]
y = x
z = copy(x)
y[0] = 9
表示する (x)  # [9, 2, 3]
表示する (y)  # [9, 2, 3]
表示する (z)  # [1, 2, 3]
```

- 関数の引数，代入の右辺は参照の値渡し
  （従来はすべて値渡し）
- 「この変更で動かなくなるコードがあるのでは？」
  共通テストの範囲なら大丈夫
  それ以外ではありえる→告知の方法を検討したい

BigInt への対応

```
p = 1
n を 1 から 100 まで増やしながら：
    p = p * n
    表示する (n + '!=' + p)
```

旧版

```
…
18!=6402373705728000
実行時エラーです
3 行目: 整数で表される範囲を越えました
```

新版

```
…
100!=93326215443944 15…
```

- SafeInteger の制約 $\pm(2^{53}-1)$ に縛られない
- オーバーフローの体験はできないが…

関数・手続きの一本化

旧版

```
関数 f(x):
    x + 1 を返す
手続き g(x):
    表示する (x + 1)
```

新版

```
関数 f(x):
    x + 1 を返す
関数 g(x):
    表示する (x + 1)
```

- 構文解析で式文を追加
- 「値を返す→関数，値を返さない→手続き」の区別を廃止

→ 「関数」に統一

- 副次的に「a = b = 3」のような連鎖代入が可能に
- 副次的に「x + 3」のような意味のない文も可能に

組み込み関数の充実

旧版

```
関数 sum(a):
    s = 0
    a の要素 i について：
        s += i
    s を返す
表示する (sum([1, 2, 3, 4, 5]))
```

新版

```
表示する (sum([1, 2, 3, 4, 5])
```

- 学習初期→ sum などを作ること自体が学習要素
- 中期以降→組み込み関数で済ませたい

→ setting.js で選択。関数本体は more_function.js

```
var setting =
{
    zenkaku_mode: 0,
    ...
    more_function: 1,
    // 追加関数を有効にするかどうか
    // 0 無効
    // 1 有効
```

正規表現

旧版

```
week = '日月火水木金土'
flag = false
i を 0 から length(week) - 1 まで増やしながら：
    flag |= week[i] == '土' or week[i] == '日'
もし not flag ならば：
    表示する ('終末が楽しみ')
```

新版

```
week = '日月火水木金土'
もし not match('土|日',week) ならば：
    表示する ('終末が楽しみ')
```

- ほしいよね（共通テストでは不要）

rangeもどき

旧版

```
a = []
i を 0 から 9 まで増やしながら：
    a に i を追加する
a の要素 i について：
    表示する (i)
```

新版

```
range(10) の要素 i について：
    表示する (i)
```

- 共通テスト用表記には range はない
  「～の要素～について」もない
- Python で学習した生徒が誤答する傾向 [14]

→ range が使えると違いがわかる

- 実際には生成する数列のリストを生成

Pythonのキーワード導入

旧版

```
n = 27
a = [n]
n > 1 の間：
    もし n % 2 == 0 ならば：
        n = n // 2
    そうでなければ：
        n = 3 * n + 1
    a に n を追加する
表示する (a)
```

新版

```
n = 27
a = [n]
while n > 1：
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3 * n + 1
    a.append(n)
print(a)
```

- 共通テスト用表記と Python はキーワードが違うだけ

→ だったら Python のキーワードを使ってもいいのでは？

- a.append(n) は append(a,n) と同義

試験対策教材としてのPyPEN

試験用に仮想言語を使うことはよくある
仮想言語の実行環境が作られることもよくある
例：情報技術者試験のアセンブリ言語 CASL

余談：技術書同人誌即売会会場にて：
「PyPEN を改造すれば〇〇試験の演習環境になるのでは？」

PEN は DNCL のための環境ではない [7]

- 学習環境の言語のベースが DNCL であるだけ

PyPEN は共通テスト 用表記のための環境ではない

- PEN 由来の環境の言語を Python っぽくしただけ
- 演習環境として「ちょうどいい」のは事実

→ 演習環境として使えることは最低ライン

日本語ベースのプログラミング環境としての PyPEN

多くのプログラミング言語は英語ベース
基本的な英単語だがハードルになることもある
(「length を len にするのがイヤ」との声も)

DNCL や共通テスト用表記は

- ✕ 言語
- ◎ アルゴリズムの表現
- → アルゴリズム伝達には「言語」より適しているのではないか

アルゴリズムの表記に慣れたら，実用的な言語に進めばいい

Web ブラウザ上の Python 実行環境との比較

PyPEN は Jison でパーサを生成

- 共通テスト 表記→（パーサ）→構文木→実行
- 簡単な Python コード →（パーサ）→構文木→実行

Web ブラウザ上の Python 実行環境はいろいろある
→ それを共通テスト 用表記用に改造できないか？

```
for i in range(10):
    print('0' * (i + 1))
```

Run

```
0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
```

Skulpt なら数十行の HTML でこれができる

おわりに

「共通テスト 用表記」と「プログラマの常識」の衝突はありえるか
例：DNCL は「論理演算は左から結合する」
　　「false かつ false でない」
　　→ DNCL：「(false かつ false) でない」→ true
　　→プログラマの常識：「false かつ (false でない)」→ false

生徒は「共通テスト 用表記」に熟練する必要はない
せめて「衝突」だけはしないでほしい
→PyPEN は

- 共通テスト 演習環境
- プログラミング環境

の両方であることができる