PyPENにおける参照の値渡しの実装

中西渉

watayan@meigaku.ac.jp
名古屋高等学校

2025年6月15日

はじめに

高校の必履修教科「情報」（2003〜）
→大学入学共通テストに導入（2025〜）
（ようやく認知度が向上？）

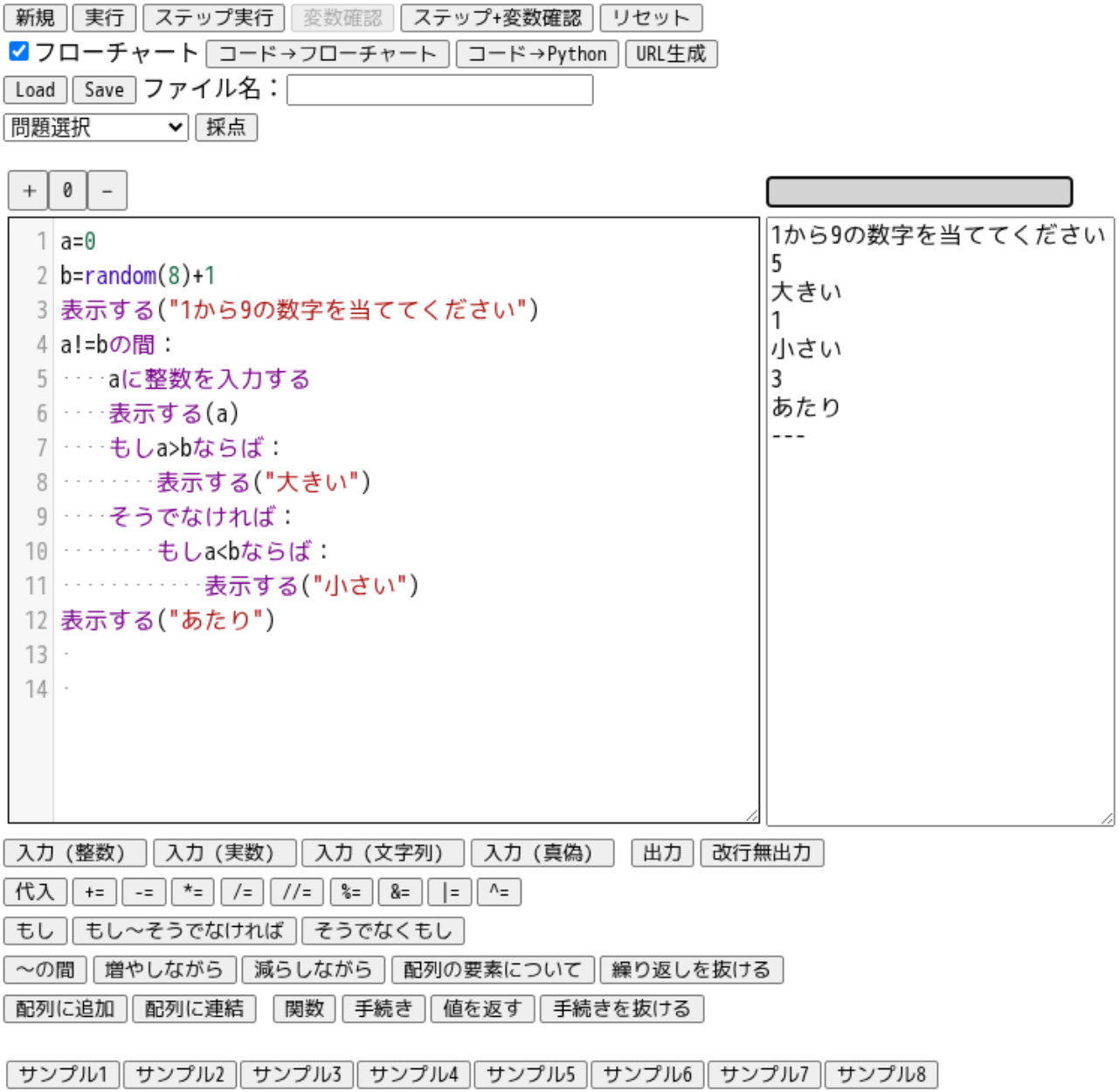
プログラミングの言語
→学習指導要領に定めなし
→教科書では複数の言語が採用
→大学入学共通テストでは日本語ベースの疑似コード
→ 2026 年度からの教科書ではそれを取り入れたものも刊行予定

```
(01)  Akibi = [5, 3, カ]
(02)  buinsu = 3
(03)  tantou = 1
(04)  buin を 2 から buinsu まで 1 ずつ増やしながら繰り返す:
(05)  |  もし キ ならば:
(06)  |  |  tantou = buin
(07)  表示する("次の工芸品の担当は部員", tantou, "です。")
```

図 4　次に割り当てる工芸品の担当部員を表示するプログラム

（2025 年度大学入学共通テスト「情報」第 3 問　問 2 より）

PyPEN：この言語に近いプログラミング学習環境



「共通テスト用プログラム表記」

- 細かい仕様は不明

→ PyPEN では筆者が勝手に決定

- 例：変数は型を持たない

→ 入力では型を明示する（a に整数を入力する）

- 例：引数はすべて値渡し

→ 大きいオブジェクトでもそれでいいのか？

PyPEN での代入・関数への値の渡し方を変更：

基本型 値渡し

配列・辞書 参照の値渡し

PyPEN 開発に至る経緯

PEN 2005 頃，大阪市立大・大阪学院大で開発
使用言語は DNCL を拡張した xDNCL

PenFlowchart 2011 頃，PEN にフローチャートを追加

WaPEN 2018 頃，Web アプリケーションとして実装

PyPEN 2018 頃，言語を Python っぽいものに変更

PyPEN の特徴

PyPEN の特徴：

- プログラミング学習環境
- 日本語ベースの言語（Python っぽい構文）
- Web アプリケーション
  サーバに依存しない（ローカルでも動作可能）

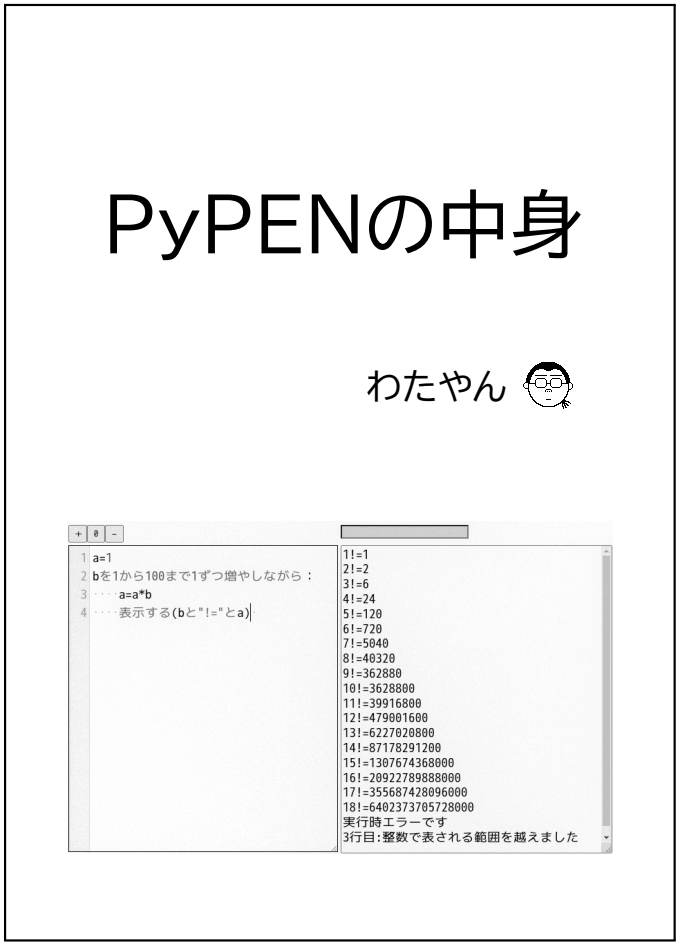
余談：Copilotが思ってるPyPEN

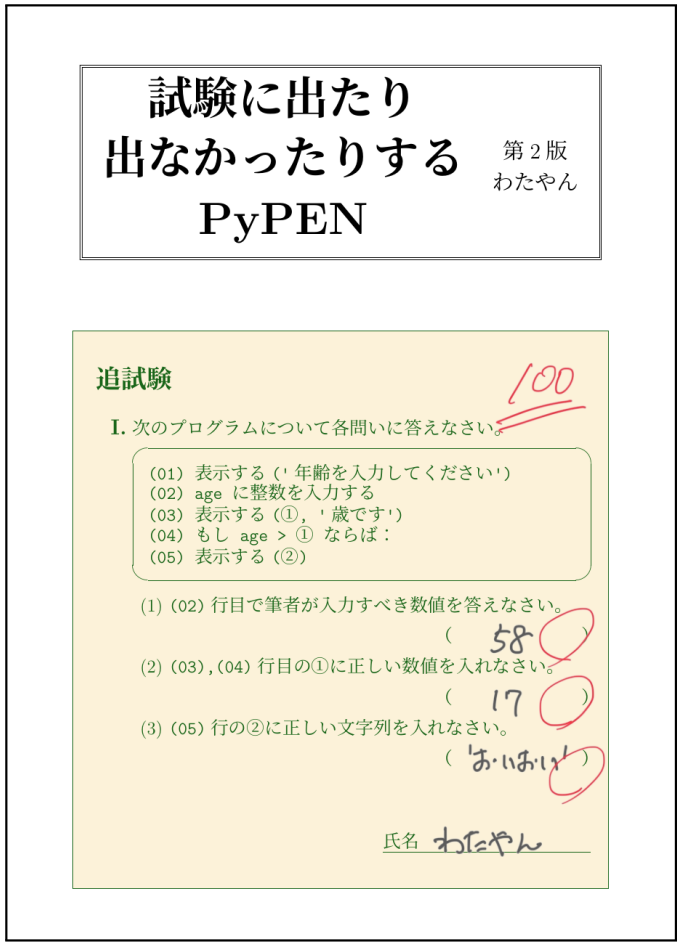
```
\subsection{PyPENの特徴}
\begin{frame}[t]\frametitle{PyPENの特徴}
    \begin{itemize}
        \item Pythonに近い文法
        \item 変数の型はない
        \item 変数は値渡し
        \item 配列や辞書は参照の値渡し
        \item 関数は値を返す
        \item 組み込み関数が豊富
    \end{itemize}
\end{frame}
```

（GitHub Copilot による補完）

さらに余談

技術書典で「うすい本」を頒布中（本日まで）





○○渡しの違い

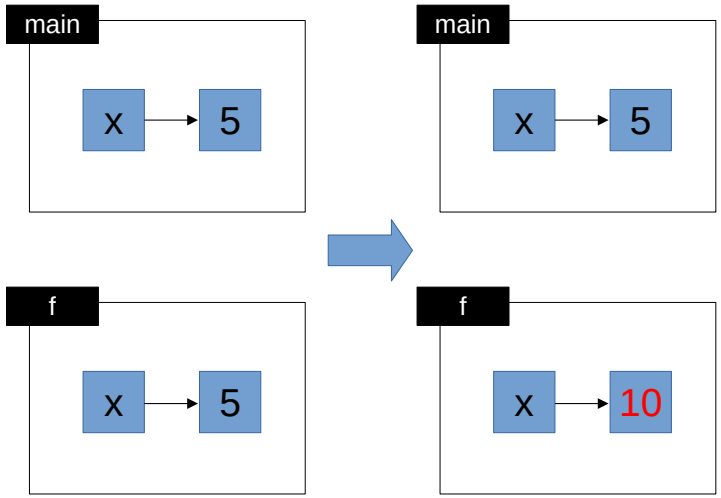
ここでは次の 3 つを考える：

- 値渡し
- 参照渡し
- 参照の値渡し

例：値渡しの例

C での値渡し

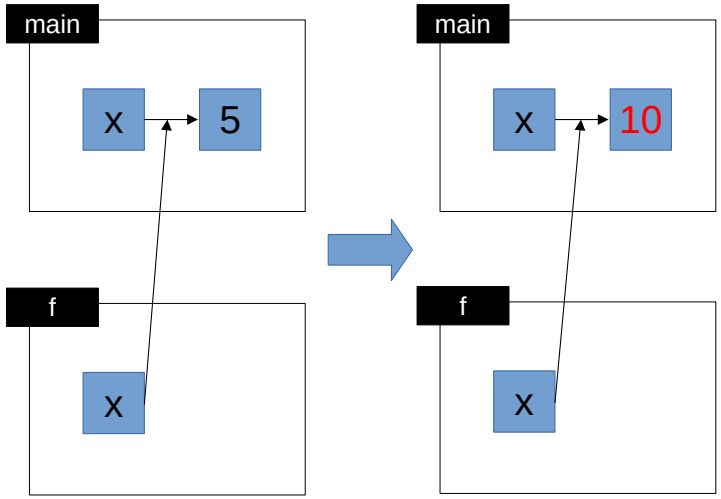
```
int f(int x) {
    x = 10;
    return x;
}
int main() {
    int a = 5;
    printf("%d\n", f(a)); // 10
    printf("%d\n", a);    // 5
    return 0;
}
```



例：参照渡しの例

C++での参照渡し

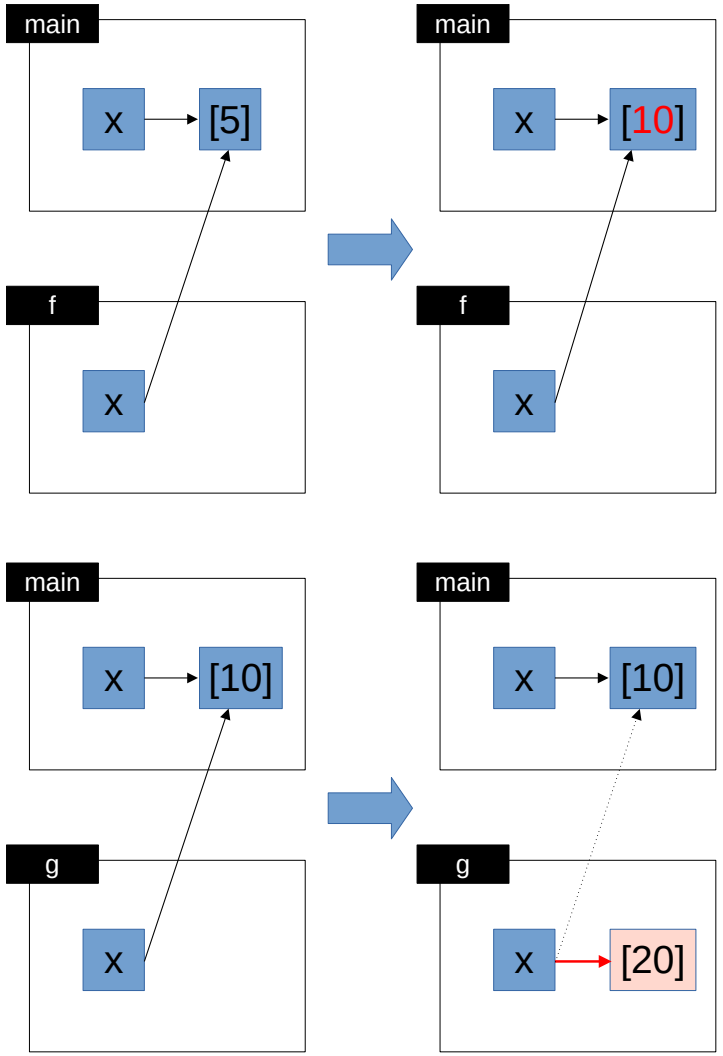
```
int f(int& x) {
    x = 10;
    return x;
}
int main() {
    int a = 5;
    printf("%d\n", f(a)); // 10
    printf("%d\n", a);    // 10
    return 0;
}
```



例：参照の値渡しの例

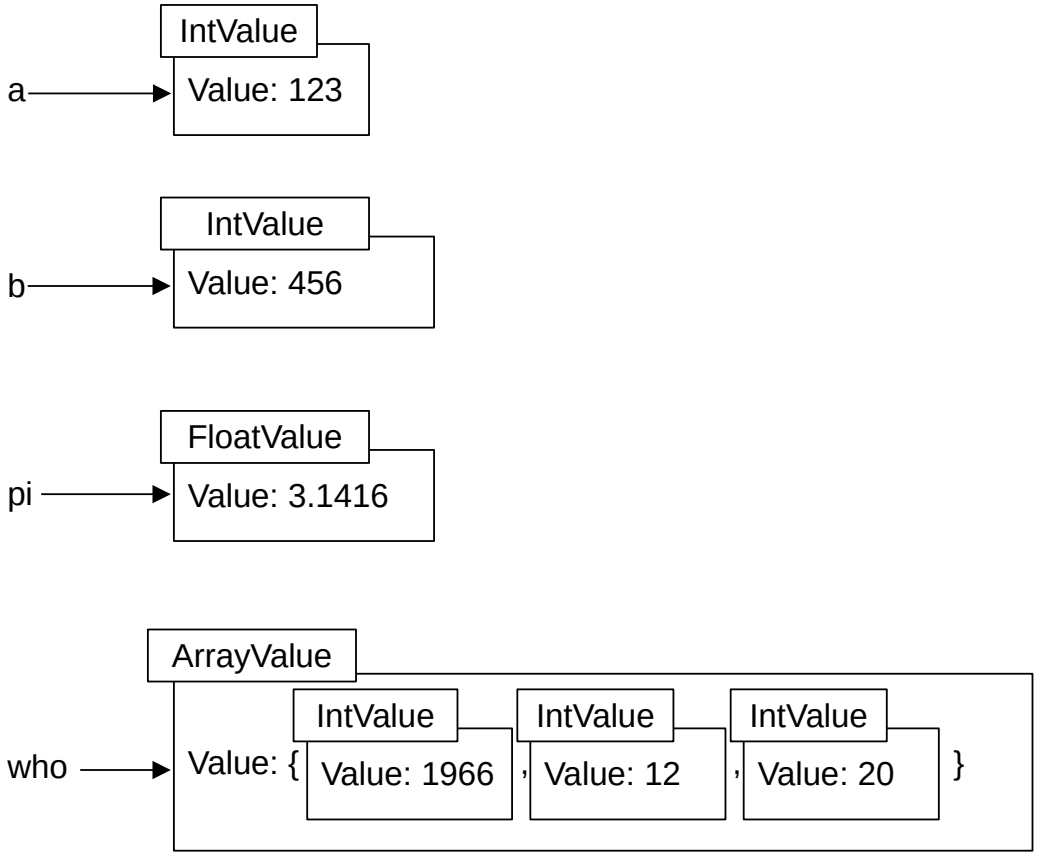
Python での参照の値渡し

```
def f(x: list[int]) -> list[int]:
    x[0] = 10
    return x
def g(x: list[int]) -> list[int]:
    x = [20]
    return x
a = [5]
print(f(a))  # [10]
print(a)     # [10]
print(g(a))  # [20]
print(a)     # [10]
```



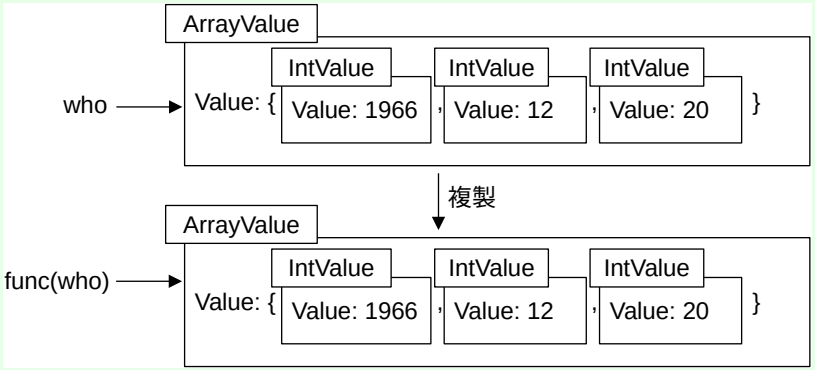
実装の方法

値：Value のサブクラス
変数管理：変数名と値のペアを辞書で管理

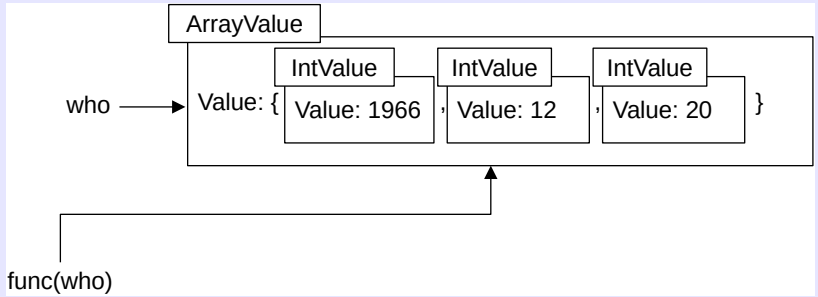


関数ごとに変数テーブルを作成

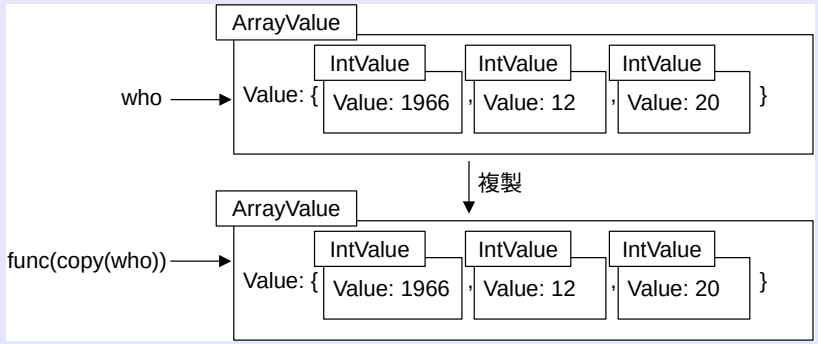
値渡し



参照の値渡し



copy を使った値渡し



実装に伴う変更点

- 変数・代入の引数は参照の値渡し
- 値渡ししたいときは copy 関数を使う
- キュー・スタック操作用の組み込み関数・メソッド
  - push：配列の末尾に要素を追加
  - pop：配列の末尾から要素を取り出す
  - unshift：配列の先頭に要素を追加
  - shift：配列の先頭から要素を取り出す

考察

- 電気通信大学 2025 年度入試問題
- Brainf**k の実装

電気通信大学2025年度入試問題

- 逆ポーランド記法の数式を計算させるプログラム
- 次の命令が使える（M は記憶領域）：

命令	意味	
入れる(x)	M に x を追加	M.push(x)
取り出す1()	M の最後の要素を取り除いてその値を返す	M.pop()
取り出す2()	M の最初の要素を取り除いてその値を返す	M.shift()

問 3　プログラム P1

```
(04) i を 0 から n - 1  まで 1 ずつ増やしながら：
(05)     もし Z[i] が数値ならば：
(06)         入れる(Z[i])
(07)     そうでなくもし Z[i] が +, -, * のいずれかならば：
(08)         x = 取り出す1()
(09)         y = 取り出す1()
(10)         もし Z[i] == "+" ならば：
(11)             入れる(y + x)
(12)         そうでなくもし Z[i] == "-" ならば：
(13)             入れる(y - x)
(14)         そうでなければ：
(15)             入れる(y * x)
```

Python なら…

```
(04) for i in range(n):
(05)     if Z[i].isnumeric()
(06)         M.push(Z[i])
(07)     elif Z[i] in '+-*':
(08)         x = M.pop()
(09)         y = M.pop()
(10)         if Z[i] == "+":
(11)             M.push(y + x)
(12)         elif Z[i] == "-":
(13)             M.push(y - x)
(14)         else:
(15)             M.push(y * x)
```

改変前の PyPEN なら…

```
(04) i を 0 から n - 1 まで 1 ずつ増やしながら：
(05)     もし isNum(Z[i]) ならば：
(06)         M = push(M, Z[i])
(07)     そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)         temp = pop(M)
             M = temp[1]
             x = 整数(temp[0])
...
```

(そのための push，pop の実装)

```
関数 push(a, x):
    a に x を追加する
    a を返す                          # x を追加したあとの配列
関数 pop(a, x):
    [a[-1], a[:-1]] を返す  # 取り出した値と配列からなる配列
```

こう書いたほうが楽ではあるが…

```
(04) i を 0 から n - 1 まで 1 ずつ増やしながら：
(05)     もし isNum(Z[i]) ならば：
(06)       # M = push(M, Z[i])
             M に Z[i] を追加する
(07)     そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)       # temp = pop(M)
           # M = temp[1]
           # x = 整数(temp[0])
             x = M[-1]
             M = M[:-1]
...
```

関数やメソッドとして書きたい

本当はこうしたい

```
(04) i を 0 から n - 1  まで 1 ずつ増やしながら：
(05)     もし isNum(Z[i]) ならば：
(06)         push(M, Z[i])
(07)     そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)         x = pop(M)
...
```

→ M を参照の値渡しにすればいいのでは？
（今回の本題に至る）

参照の値渡しにしてから

```
(04) i を 0 から n - 1 まで 1 ずつ増やしながら：
(05)     もし isNum(Z[i]) ならば：
(06)         push(M, Z[i])
(07)     そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)         x = pop(M)
...
手続き push(M, x):
    M に x を追加する
関数 pop(M):
    x = M[-1]
    M = M[:-1]
    return x
```

これで M を返す必要がなくなる…と思っていたが…

うまくいかなかった

```
(04) i を 0 から n - 1 まで 1 ずつ増やしながら：
(05)     もし isNum(Z[i]) ならば：
(06)         push(M, Z[i])  # これは機能する
(07)     そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)         x = pop(M)     # M が変化しない！
...
手続き push(M, x):
    M に x を追加する          # これは機能する
関数 pop(M):
    x = M[-1]
    M = M[:-1]             # 代入すると pop 内の M の参照先が変わる
                           # 元の M は変わらない
    return x
```

本当に必要なもの

```
(04) i を 0 から n - 1  まで 1 ずつ増やしながら：
(05)      もし isNum(Z[i]) ならば：
(06)          push(M, Z[i])
(07)      そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)          x = pop(M)
...
手続き push(M, x):
    M に x を追加する
関数 pop(M):
    x = M[-1]
  # M = M[:-1]
    remove(M, -1)          # M に作用する破壊的メソッドが必要
    return x
```

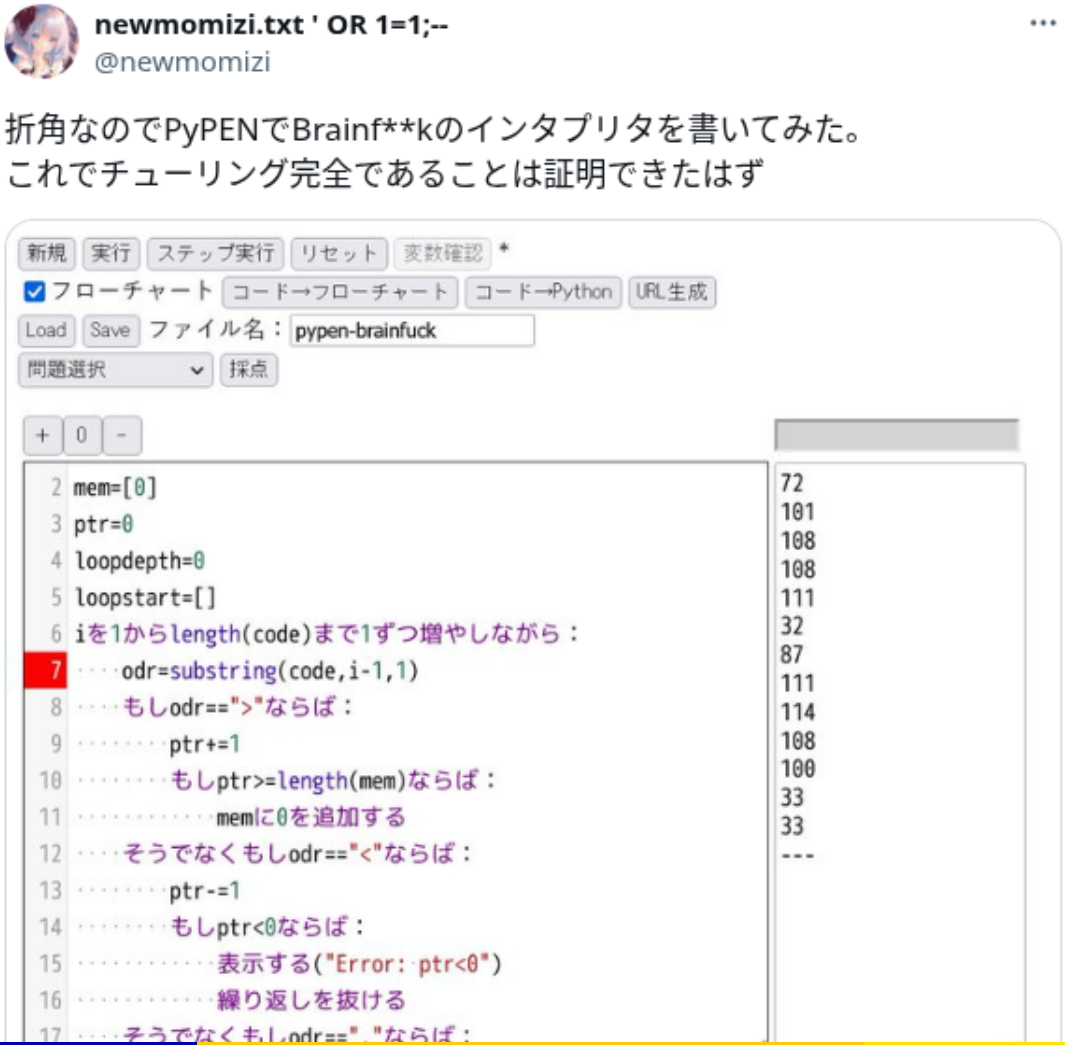
最終的に…

```
(04) i を 0 から n - 1  まで 1 ずつ増やしながら：
(05)      もし isNum(Z[i]) ならば：
(06)          push(M, Z[i])          # 組み込みメソッドとして実装
(07)      そうでなくもし ['+', '-', '*'] の中に Z[i] ならば：
(08)          x = pop(M)             # 組み込み関数として実装
...
```

- 組み込み関数（メソッド）なら何でもできる
  item[→] 破壊的メソッドを追加で実装

Brainf**k の実装

X の投稿より (https://x.com/newmomizi/status/1731586614579847559)



Brainf**k とは

- プログラミング言語の 1 つ
- チューリング完全な言語
- 以下の 8 つの命令を持つ（p はデータ領域を指すポインタ）

命令	C での表現
>	++p;
<	--p;
+	++*p;
-	--*p;
.	putchar(*p);
,	*p = getchar();
[	while(*p){
]	}

https://www.muppetlabs.com/~breadbox/bf/

構文解析部分（エラー処理は省略）

```
token=code[pc]
もし token==">"ならば：
    ptr+=1
そうでなくもし token=="<"ならば：
    ptr-=1
そうでなくもし token=="+"ならば：
    mem[ptr]+=1
そうでなくもし token=="-"ならば：
    mem[ptr]-=1
そうでなくもし token=="."ならば：
    改行なしで表示する(ascii[mem[ptr]])
そうでなくもし token==","ならば：
    input に文字列を入力する
    i を 0 から length(ascii)-1 まで 1 ずつ増やしながら：
        もし ascii[i]==input ならば：
```

構文解析部分の続き（エラー処理は省略）

```
そうでなくもし token=="["ならば：
    もし mem[ptr]==0 ならば：
        (対応する] まで飛ばす処理)
    そうでなければ：
        stack[sp]=pc
        sp+=1
そうでなくもし token=="]"ならば：
    pc=stack[sp-1]
```

つまり"["の位置をスタック stack に積んでいる。

構文解析部分の続き（エラー処理は省略）

```
そうでなくもし token=="["ならば：
    もし mem[ptr]==0 ならば：
        (対応する] まで飛ばす処理)
    そうでなければ：
        stack[sp]=pc                    # push(stack, pc)
        sp+=1                           #
そうでなくもし token=="]"ならば：
    pc=stack[sp-1]                      # pc = pop(stack)
```

いっそループ自体をスタックに積んでしまったら？
↑それは PyPEN でのループの実装手段

おわりに

疑似コードの「仕様」が細かく定められることはないだろう
→ 細部は実装者が勝手に決める
→ 実装ごとにばらばら　しかしそれが問題になることはない

せっかく言語があるなら，いろいろ作りたいのが人情

- WaPEN でコンウェイのライフゲームを作った生徒
- PyPEN で Brainf**k の実装をした@newmomizi 氏
- …

「教材」から解放されてもいいよね？ →