PyPEN

中西渉

watayan@meigaku.ac.jp
名古屋高等学校

2021 年 8 月 15 日

氏名	中西渉
通称	わたやん
生年	1966 年（丙午）
出身地	三重県
勤務校	名古屋高等学校 (1989〜)
教科	情報 (2004〜)，数学 (1989〜?)
Web サイト	https://watayan.net
twitter	@watayan
Facebook	wataru.nakanishi

2025 年度からの共通テストに関して

- 「情報」の導入が決定
- 大学入試センターの試作問題，サンプル問題
  → 新しい DNCL（以下 DNCL2）

従来の DNCL

```
current ← 2021
もし current-1964 ≧ 18 ならば
    | 「おいおい」を表示する
を実行する
```

DNCL2

```
current=2021
もし current-1964>=18 ならば:
 └表示する ("おいおい")
```

Python

```
current=2021
if current-1964>=18:
    print("おいおい")
```

DNCL の実行環境

- PEN
- どんくり
- wPEN
- XTetra
- Objective-DNCL
- PenFlowchart
- WaPEN
- ...

DNCL2 の実行環境

- どんくり
- つちのこ
- Picthon
- PyPEN
- ...

PyPEN を使ってみよう（ただし Internet Explorer は不可）

https://watayan.net/prog/PyPEN/

2.1 変数，代入，四則演算

変数

- 値を入れるための，名前をつけた箱のようなもの
- 英字で始まる英数字列（大文字小文字は区別する）

代入

- 変数に値を入れること
- 《変数》＝《値》
- 気持ち的には《変数》←《値》

値…4 つの型

- 整数・実数・文字列・真偽

四則演算

+	加算	/	除算の商（実数）
−	減算	//	除算の商（整数）
*	乗算	%	除算の余り（整数）

- 乗算・除算は加算・減算より優先
- カッコは（ ）のみ

2.2 入力，出力

入力

- 《変数》に《型》を入力する

出力（値はコンマ区切りで複数指定できる）

- 表示する (《値》)
- 改行無しで表示する (《値》)

2.3 制御構造

選択（分岐）

もし

もし《条件》ならば：
　　《処理》

もし～そうでなければ

もし《条件》ならば：
　　《処理》
そうでなければ：
　　《処理》

《条件》は 2 つの値の比較

- ==, !=, >, <, >=, <=
- 《配列》の中に《値》
- 2 つの条件を and や or で結んだり，not を前置して否定したり

繰り返し（ループ）

～の間

《条件》の間：
    《処理》

増やしながら

《変数》を《値》から《値》まで《値》ずつ増やしながら：
    《処理》

減らしながら

《変数》を《値》から《値》まで《値》ずつ減らしながら：
    《処理》

2.4 関数，手続き

- 複数の処理をまとめたもの
- 関数は必ず値を返す
- 手続きは値を返せない

関数

関数《関数名》(《仮引数》):
    《処理》
    …
    《値》を返す

手続き

手続《手続き名》(《仮引数》):
    《処理》
    …
    手続きを抜ける
    …

3 ガチャのプログラム

ガチャのプログラムを作ってみる

- 景品は 10 種類
- すべての景品は同確率
- すべての景品を集めたら終了

終了するまでに何回クジを引くかをシミュレーションする

3.1 1 回ずつやってみる

例 1

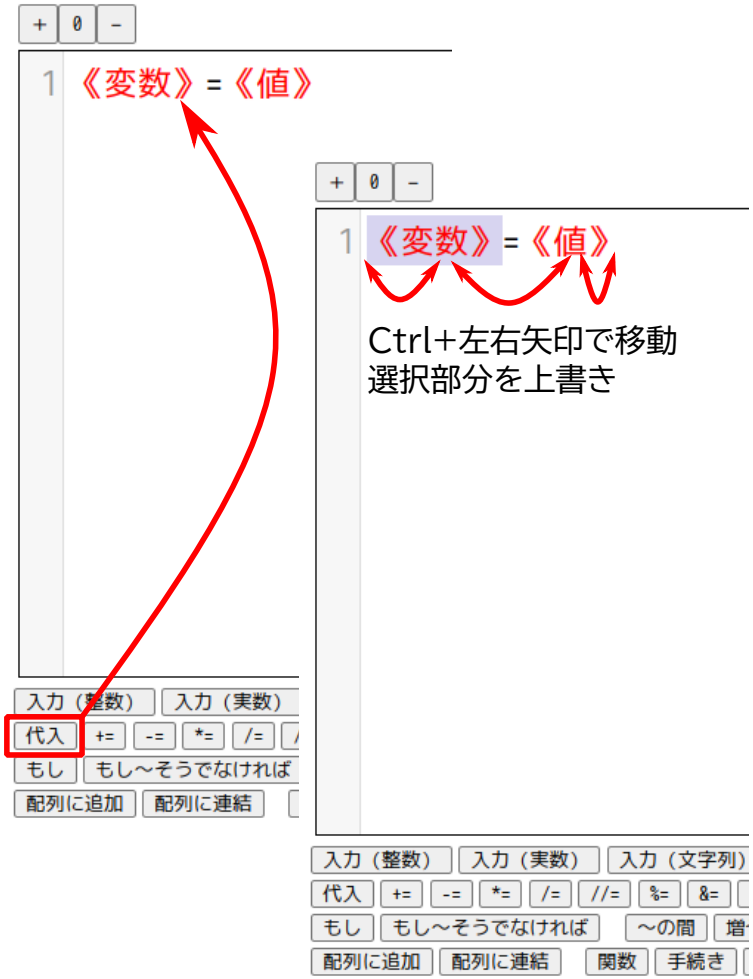
```
n=0
a=[0,0,0,0,0,0,0,0,0,0]
a!=[1,1,1,1,1,1,1,1,1,1] の間：
    k=random(9)
    a[k]=1
    n=n+1
    表示する (n,a)
```

使っている変数

- n クジを引いた回数
- a どの景品が取得済みか
- k 引いた景品の番号

3.1.1 代入（入力支援ボタンの使い方）

入力支援ボタンの使い方



3.1.2 配列

配列

- 何個かの値に番号をつけてまとめたもの
- [ ] の中にコンマ区切り
- 番号は 0 から始まる

```
a=[0,0,0,0,0,0,0,0,0,0]
```

- a[0]，a[1]，a[2]，…，a[9] がそれぞれ 0 になる
- k 番目の景品を取得済みかどうか→ a[k] が 1 か 0 か
  - 最初は全部 0
  - k 番目の景品を取得→ a[k] を 1 にする

3.1.3 繰り返し（ループ）

景品が全部揃うまでくじ引きを繰り返す→繰り返しのプログラム
「〜の間」を使う

- a が [1,1,1,1,1,1,1,1,1,1] になったら終わり

→ a が [1,1,1,1,1,1,1,1,1,1] でない間続ける

→ 《条件》は a!=[1,1,1,1,1,1,1,1,1,1]

繰り返し部分

a!=[1,1,1,1,1,1,1,1,1,1] の間：
　　《処理》

では繰り返しの中の処理は…

乱数で景品の番号を決める

- random(9) で 0〜9 の乱数

繰り返し部分

a!=[1,1,1,1,1,1,1,1,1,1] の間：

```
    k=random(9)
    a[k]=1
    n=n+1
    表示する (n,a)
```

「表示する」は「出力」ボタン

3.1.4 実行

例 1

```
n=0
a=[0,0,0,0,0,0,0,0,0,0]
a!=[1,1,1,1,1,1,1,1,1,1] の間：
    k=random(9)
    a[k]=1
    n=n+1
    表示する (n,a)
```

「実行」ボタンで実行

- 構文エラー…エラーの箇所をよく見てみる
- 実行エラー…0 や 1 の個数間違えてない？
- なかなか終わらないとか…「リセット」で止める

3.2 何回もやってみる

運が良かったり悪かったり…
平均はどのくらい?
→100 回くらいやってみよう

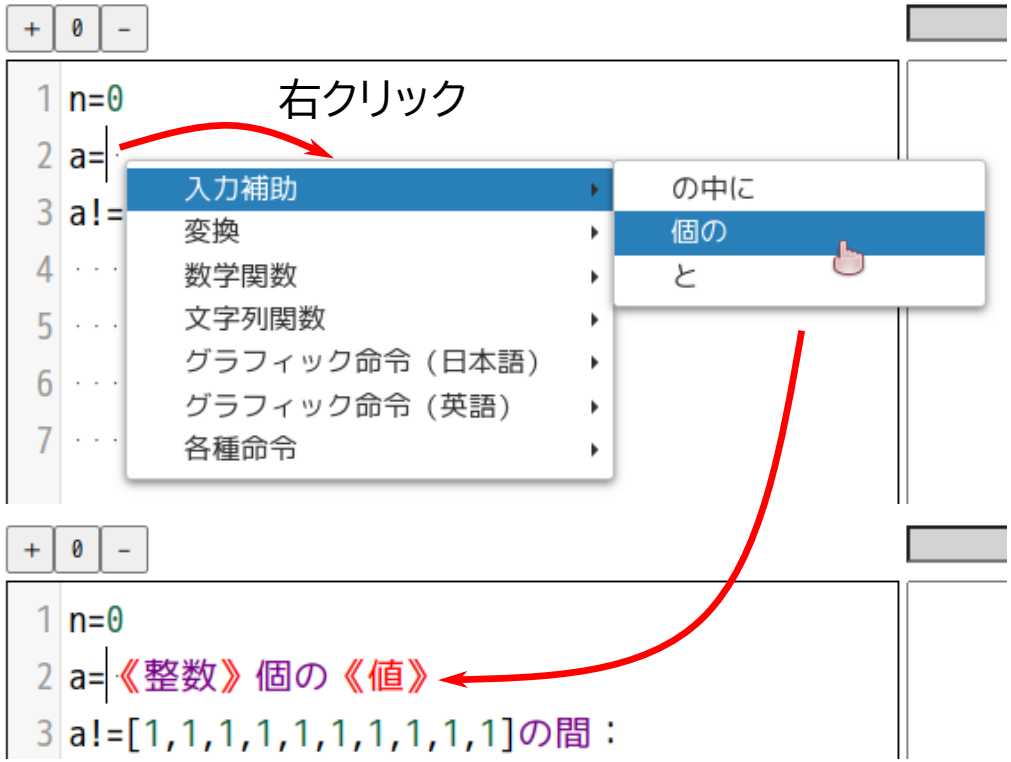
3.2.1 新しい構文と右クリックメニュー

新しい構文

例 2

```
n=0
a=10 個の 0
a の中に 0 の間：
    k=random(9)
    a[k]=1
    n+=1
    表示する (n,a)
```

a=[0,0,0,0,0,0,0,0,0,0] なんてダサい
→「《整数》個の《値》」を使う



「《配列》の中に《値》」も使う
n+=1 は n=n+1 と同じ

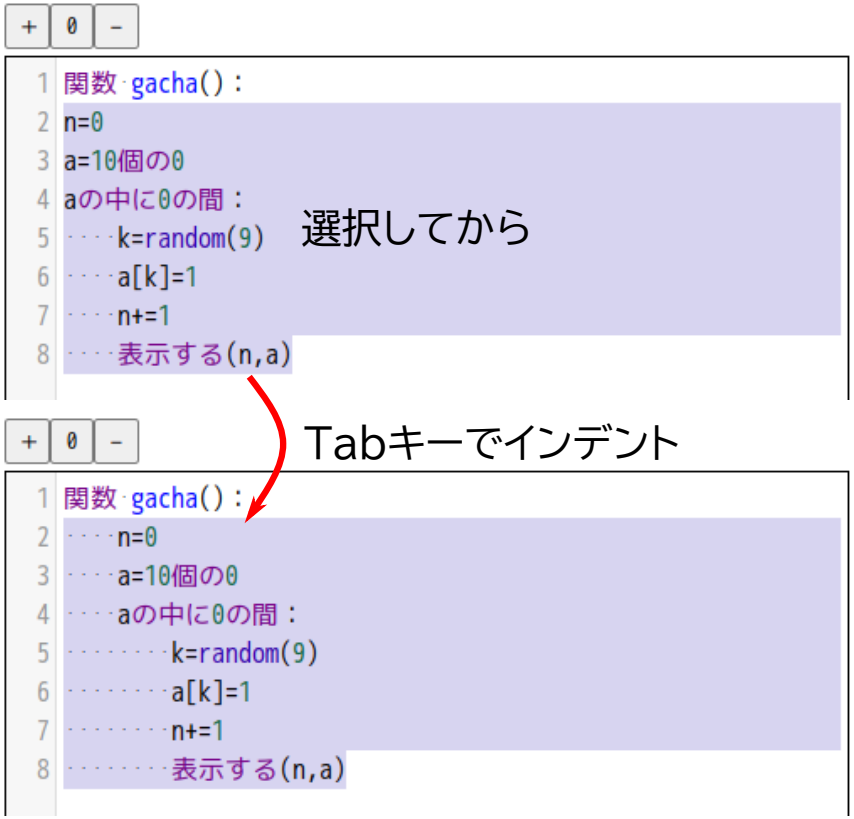
3.2.2 関数

回数を返す関数にする

例 3

```
関数 gacha():
    n=0
    a=10個の0
    aの中に0の間:
        k=random(9)
        a[k]=1
        n+=1
    nを返す
```

「関数」を最初の行に入れて…



最後の行を「n を返す」にする

この関数を 100 回呼び出して平均をとる
$\rightarrow$「増やしながら」を使う

例 4（関数 gacha の外で）

```
s=0
n を 1 から 100 まで 1 ずつ増やしながら：
    s+=gacha()
表示する (s/100)
```

実は「クーポンコレクター問題」として有名
n 個の景品を集めるまでの回数の期待値は $n(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n})$
($n=10$ のときは約 29.3 回)

3.3 バリエーション

Wikipedia のクーポンコレクター問題のページの

> 全 *50* 種類のクーポンを収集するには，平均で約 *225* 回の試行が必要となる

を検証したい場合

よくある間違い

```
関数 gacha()：
    n=0
    a=50 個の 0
    a の中に 0 の間：
        k=random(9)
        a[k]=1
        n=n+1
    n を返す
```

例 5

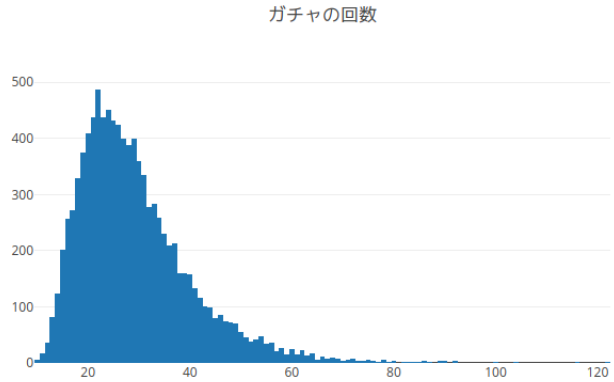
```
関数 gacha(N)：
    n=0
    a=N 個の 0
    a の中に 0 の間：
        k=random(N-1)
        a[k]=1
        n=n+1
    n を返す
```

実は確率分布は難しいっぽい
→PyPEN のグラフ機能でヒストグラムを描いてみる

例 6

```
関数 gacha(N)：
    n=0
    a=N 個の 0
    a の中に 0 の間：
        k=random(N-1)
        a[k]=1
        n+=1
    n を返す
a=[]
n を 1 から 100 まで 1 ずつ増やしながら：
    a に gacha(10) を追加する
layout={"title":"ガチャの回数"}
data={"type":"histogram", "x":a}
グラフ描画(layout,[data])
```

ちなみに 10000 回やってみると…



レアアイテムを作りたい

- 0 は 1%
- それ以外は 11%

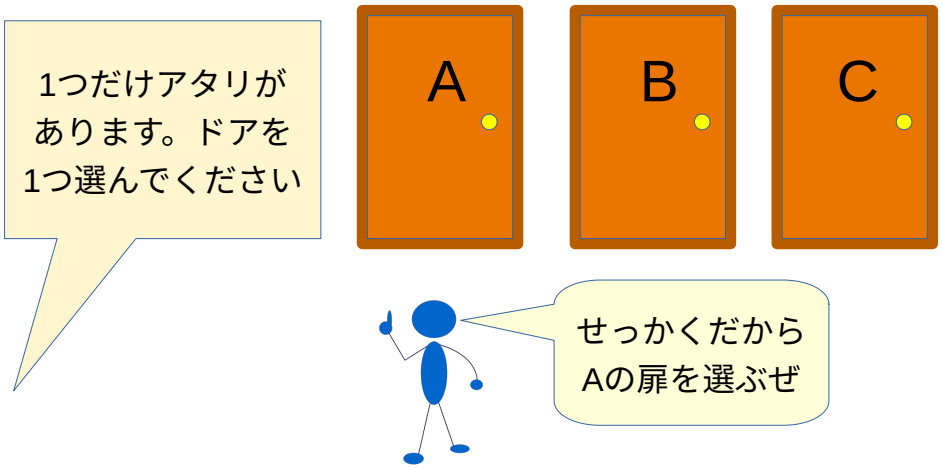
たとえば…

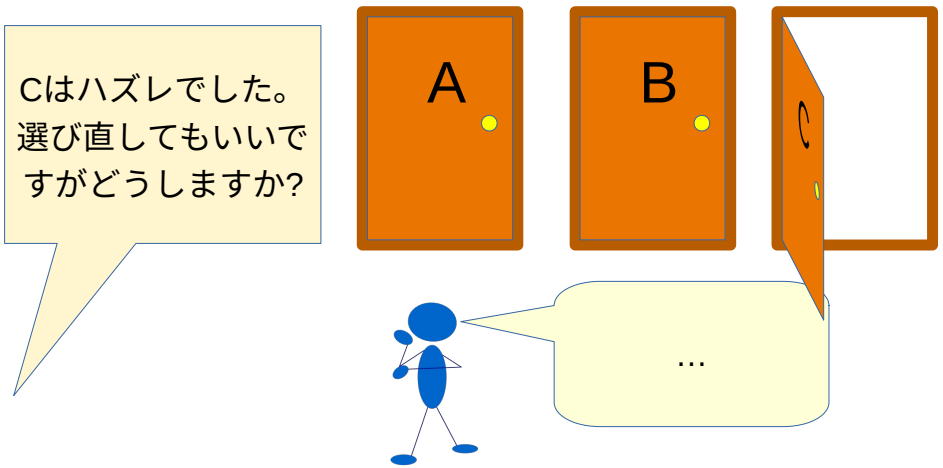
```
k=(random(99)+10)//11
```

たとえば…

```
k=random(9)
```
もし k==0 ならば：
```
    k=random(9)
```

4 モンティ・ホール問題





選んだドアを変えたほうがいい?変えないほうがいい?

4.1 メイン部分

アタリかハズレかを判定する関数 play を後で作る

- アタリなら 1，ハズレなら 0 を返す

これを 1000 回呼び出して，アタリの確率を調べる

例 7

```
N=1000
w=0
n を 1 から N まで 1 ずつ増やしながら：
    w+=play()
表示する (w/N)
```

4.2 変えない場合

変えない場合の play 関数を作る

- ドアの番号は 0〜2
- 0〜2 の乱数は random(2)
- アタリの番号を x，プレイヤーが選ぶドアの番号を y

例 8

```
関数 play() :
    x=random(2)
    y=random(2)
    もし x==y ならば :
        1 を返す
    そうでなければ :
        0 を返す
```

4.3 変える場合

必ず選ぶドアを変える場合の play 関数を作る
→ 選び直すドアの番号を返す関数 reselect を作る

例 9

```
関数 play():
    x=random(2)
    y=random(2)
    z=reselect(x,y)
    もし x==y ならば:
        1 を返す
    そうでなければ:
        0 を返す
関数 reselect(x,y):
    ???
```

ヒント

- 司会者が開けるドアの番号を z とする
  - x==y のとき…z は残り 2 つのどちらか
  - x!=y のとき…z は 1 つに決まる

    z=3−x−y
- 選び直すドアは y でも z でもないのだから…3−y−z

たとえば...

例 10

```
関数 reselect(x,y):
    もし x==y ならば:
        z=x
        z==x の間:
            z=random(2)
    そうでなければ:
        z=3-x-y
    3-y-z を返す
```

5.1 使わなかった機能

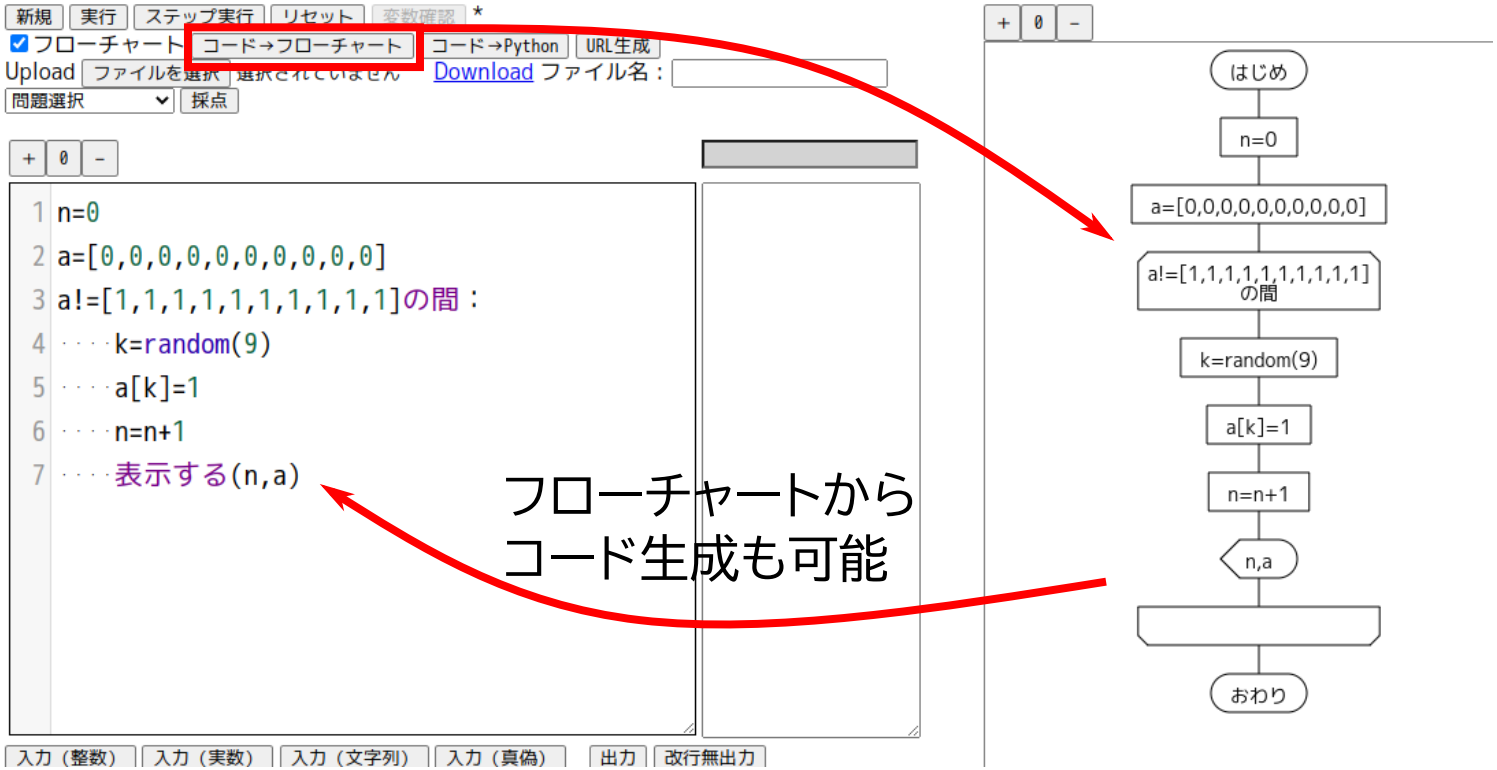
詳細はマニュアルを参照のこと

5.1.1 セーブ・ロード

セーブ・ロードは Download・Upload で

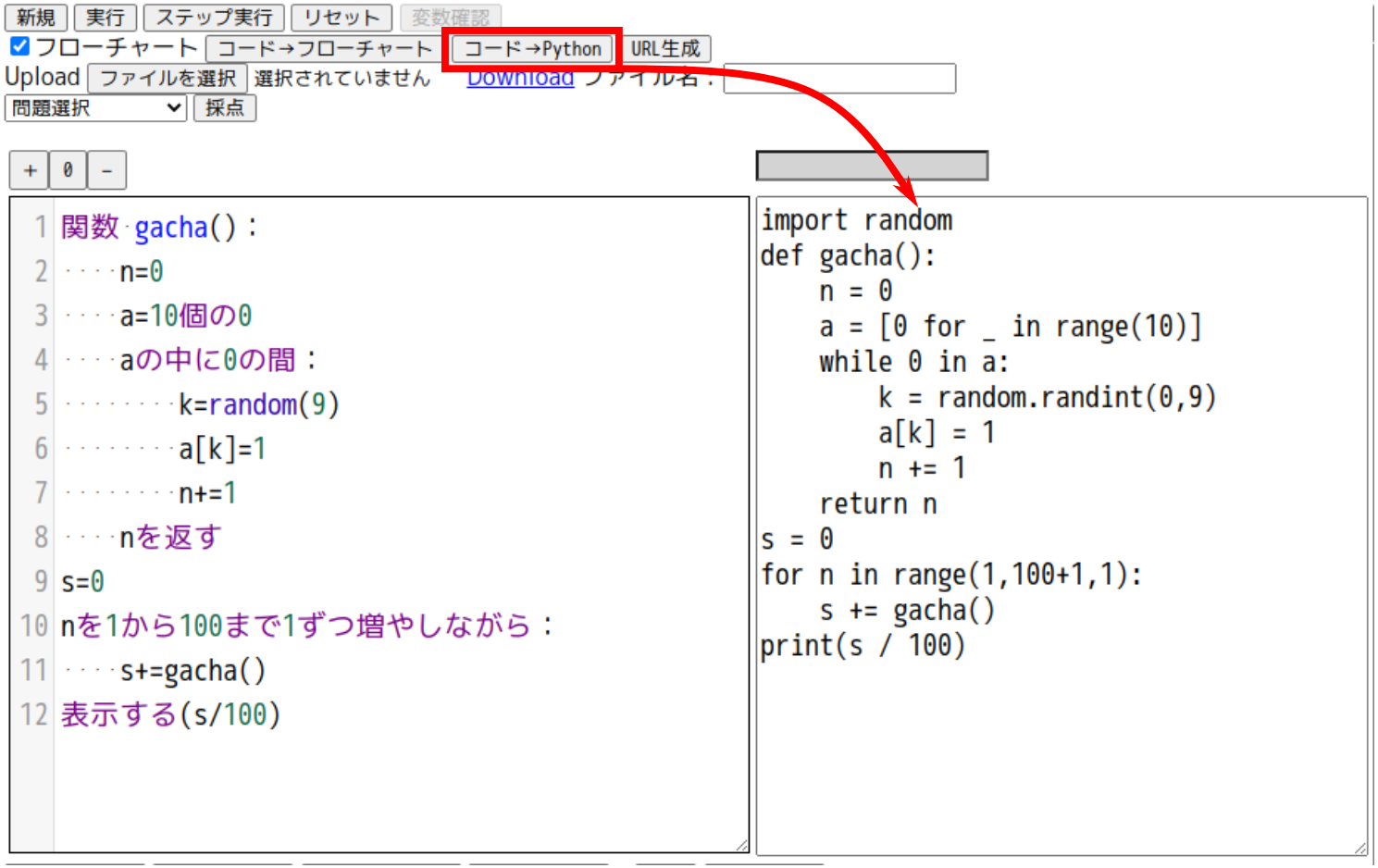
5.1.2 コード→フローチャート

コード→フローチャート生成
（フローチャートからコード生成も可能）



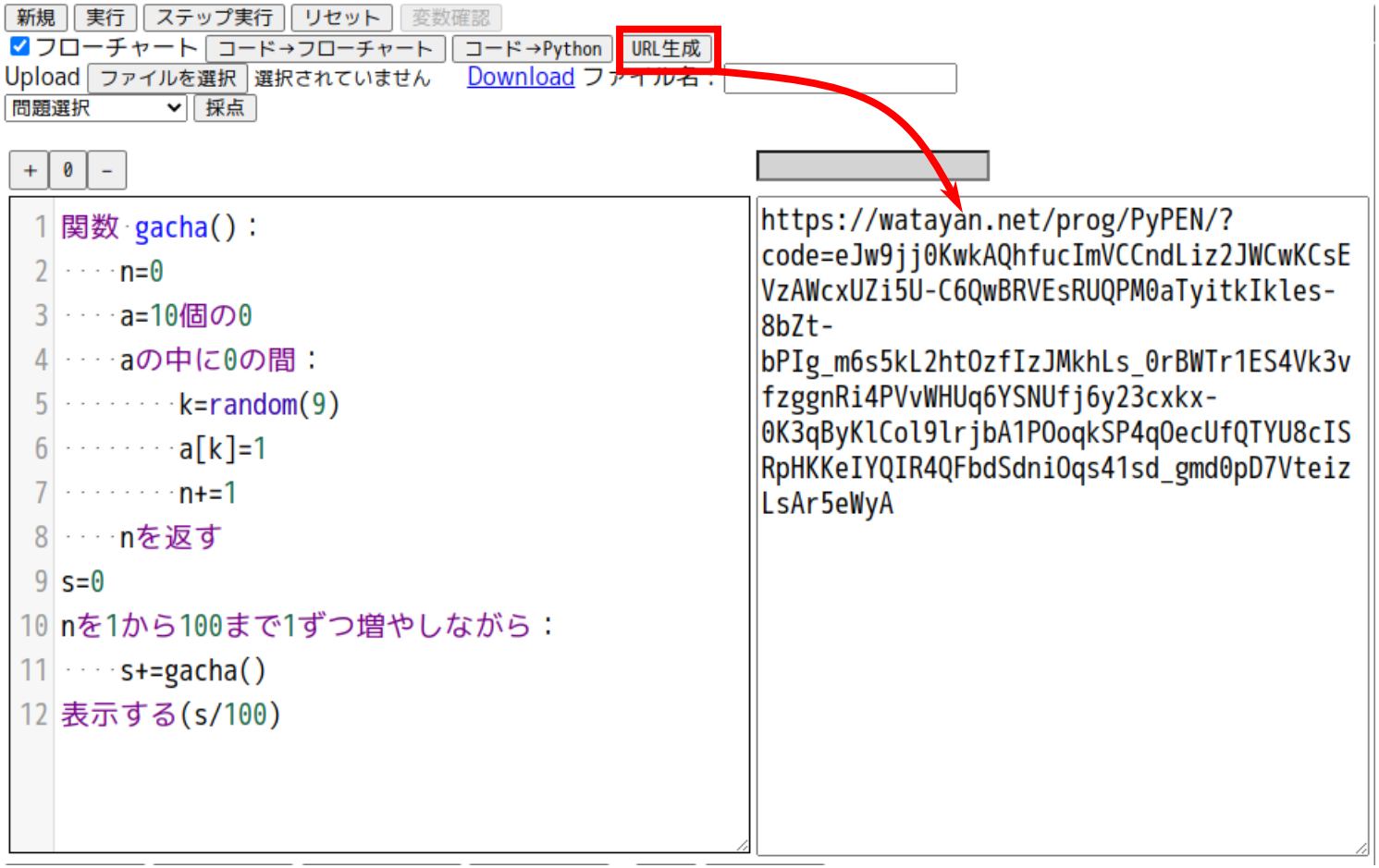
5.1.3 コード→Python

Python に「翻訳」したコードを出力



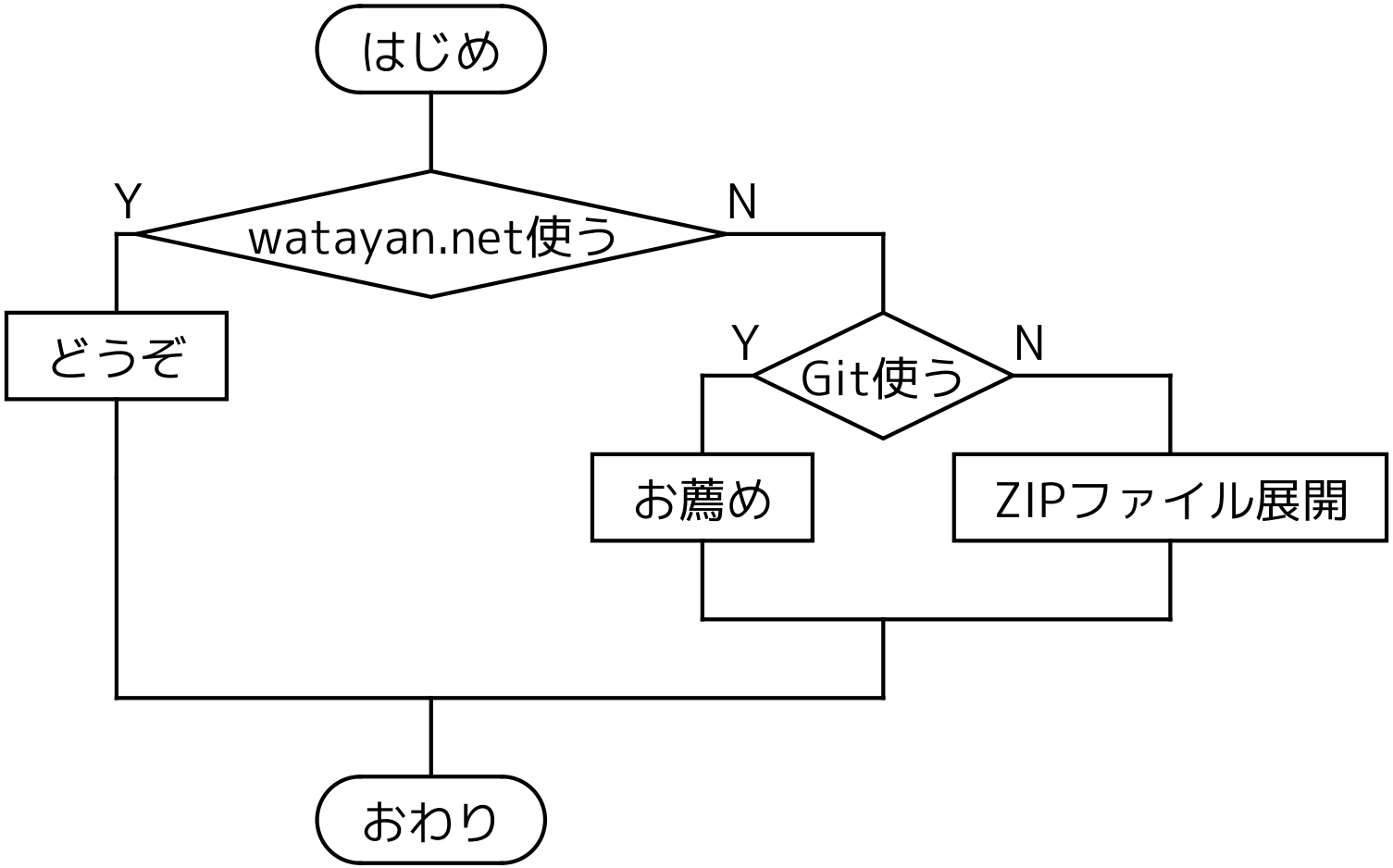
5.1.4 URL 生成

コードを含む URL を生成



5.2 授業等で使う場合

自由にお使いください（MIT ライセンス）



筆者のサイトのを使う場合

- 何の準備もいらない
- サンプル等が変えられない
- 勝手にバージョンアップしてるかも

自分の環境で使う場合

- 用意するのが手間
- サンプルとかを自由に変えられる
- バージョンを固定できる

サーバの機能は使わないので,
ローカルファイルでも OK

git を使う場合

- git の準備が必要
- バージョンアップが簡単

具体的な手順はプリントを参照のこと

ZIP ファイルを展開する場合

- git が不要
- バージョンアップが上書きになる

おわりに

不具合等あればお知らせください