

Web ブラウザ上のプログラミング学習環境 PyPEN を用いた授業の提案

中西 渉

watayan@meigaku.ac.jp
名古屋高等学校

2021 年 3 月 19 日

1 はじめに

2 PyPEN について

3 PyPEN を用いた授業の提案

- 提案の動機
- 授業の流れ

1 はじめに

2022 年度から高校で必修修「情報Ⅰ」

→プログラミングが必須

→言語は指定しない…が…

『高等学校情報科教員研修用教材』（文部科学省）では…

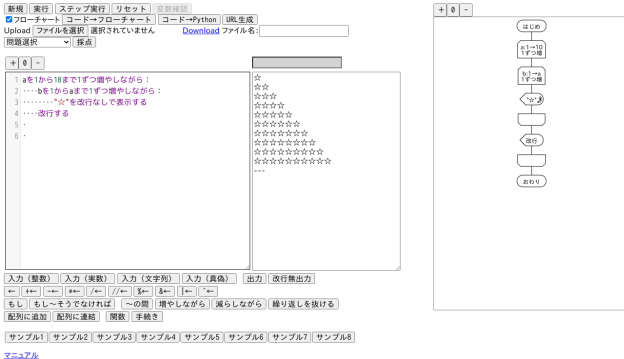
情報Ⅰ 最初に公開されたのが Python 版
他言語版は遅れてリリース

情報Ⅱ Python や R が使われている

→ Python が使われることが増える？

PyPEN を開発

- Web ブラウザ上のプログラミング学習環境
- GitHub, 筆者のサイトで公開
<https://github.com/watayan/PyPEN.git>
<https://watayan.net/prog/pypen.html>



本稿で提案する授業で目指すもの

- PyPEN を用いたプログラミング学習の導入
- Python でのプログラミングへの移行を促す

2 PyPEN について

PyPEN とは

- Web ブラウザ上で稼働するプログラミング学習環境
ただし Internet Explorer を除く
- 使用言語は DNCL を Python に似せたもの
- JavaScript で実装
Web サーバに設定不要（ローカルでも実行可能）

DNCL

a を 1 から 100 まで 1 ずつ増やしながら、
 | もし $a \% 10 = 3$ または $a / 10 = 3$ または $a \% 3 = 0$ ならば、
 | | 「(´ · ω · `)」を表示する
 | | を実行し、そうでなければ
 | | a を表示する
 | | を実行する
 を繰り返す

PyPEN

a を 1 から 100 まで 1 ずつ増やしながら：
 もし $a \% 10 = 3$ または $a // 10 = 3$ または $a \% 3 = 0$ ならば：
 " (´ · ω · `)" を表示する
 そうでなければ：
 a を表示する

PyPEN ができるまで

PEN 大阪学院大学・大阪市立大学の共同プロジェクト
DNCL を元にした xDNCL
Java アプリケーション
筆者は関わっていない

PenFlowchart PEN にフローチャートを付加

WaPEN PenFlowchart の Web アプリ化

PyPEN WaPEN の言語を Python っぽく変更

原稿を書いた後の変更

- コメントが書けるようにした
- エディタに CodeMirror を採用
- Web フォントで空白の数をわかりやすくした

もっと最近の変更

- ループは「繰り返す」がないのを標準に
(あっても動くけど)
- Ctrl+左右で次の《〜》や行頭・行末に移動
- 「一時停止する」の追加 (ブレークポイント代わり)

PyPEN の実行画面

+ 0 -

```

1 aを1から10まで1ずつ増やしながら：
2 ……bを1からaまで1ずつ増やしながら：
3 ………“☆”を改行なしで表示する
4 ……改行する
5 .
6 .

```

```

☆
☆☆
☆☆☆
☆☆☆☆
☆☆☆☆☆
☆☆☆☆☆☆
☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆☆☆
☆☆

```


[マニュアル](#)

+ 0 -



入力支援ボタン

新規 実行 ステップ実行 リセット 変数確認
☒フローチャート コード→フローチャート コード→Python URL生成
 Upload ファイルを選択 選択されていません Download ファイル名:
 問題選択 採点

+ 0 -

```

1 aを1から10まで1ずつ増やしながら：
2 ……bを1からaまで1ずつ増やしながら：
3 ………“☆”を改行なしで表示する
4 ……改行する
5 .
6 .
  
```

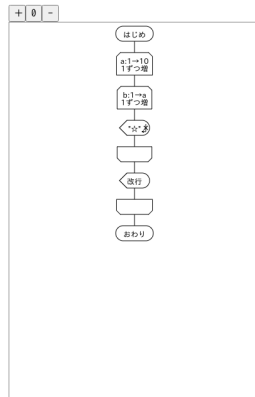
```

☆
☆☆
☆☆☆
☆☆☆☆
☆☆☆☆☆
☆☆☆☆☆☆
☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆☆
☆☆☆☆☆☆☆☆☆☆☆☆
☆☆
  
```

入力（整数） 入力（実数） 入力（文字列） 入力（真偽） 出力 改行無出力
 ← + ← - ← * ← / ← % ← & ← | ← ^ ←
 もし もし～そうでなければ ～の間 増やしながら 減らしながら 繰り返しを抜ける
 配列に追加 配列に連結 関数 手続き

サンプル1 サンプル2 サンプル3 サンプル4 サンプル5 サンプル6 サンプル7 サンプル8

[マニュアル](#)



入力ミスによるエラーの減少を期待

フローチャート

[マニュアル](#)

コード→フローチャートは手動，逆は自動

Python のコード生成

新規 実行 ステップ実行 リセット 変数確認
☒フローチャート ☒コード→フローチャート ☒コード→Python URL生成
 Upload ファイルを選択 選択されていません Download ファイル名:
 問題選択 採点

+ 0 -

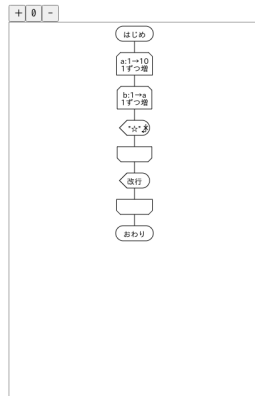
```

1 aを1から10まで1ずつ増やしながら：
2 ……bを1からaまで1ずつ増やしながら：
3 ……… "☆"を改行なしで表示する
4 ……改行する
5 .
6 .
  
```

for a in range(1,10+1,1):
 for b in range(1,a+1,1):
 print('☆',end='')
 print()

入力（整数） 入力（実数） 入力（文字列） 入力（真偽） 出力 改行無出力
 ← + ← - ← * ← / ← % ← & ← | ← ^ ←
 もし もし～そうでなければ ～の間 増やしながら 減らしながら 繰り返しを抜ける
 配列に追加 配列に連結 関数 手続き
 サンプル1 サンプル2 サンプル3 サンプル4 サンプル5 サンプル6 サンプル7 サンプル8

[マニュアル](#)



あとはPython の環境に…

URL 生成

新規 実行 ステップ実行 リセット 変数確認
☒フローチャート ☐コード→フローチャート ☐コード→Python ☐URL生成
 Upload ファイルを選択 選択されていません Download ファイル名:
 問題選択 採点

```
+ 0 -
1 aを1から10まで1ずつ増やしながら：
2   ...bを1からaまで1ずつ増やしながら：
3   ...    ... "☆"を改行なしで表示する
4   ...改行する
5   .
6   .
```

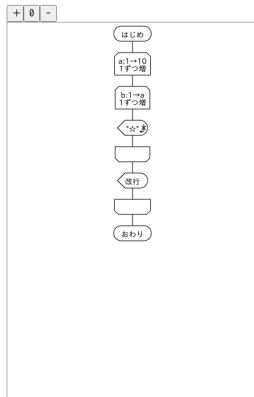
入力 (整数) 入力 (実数) 入力 (文字列) 入力 (真偽) 出力 改行無出力

← +← ←← ←←← /← //← %← &← |← ^←

もし もし〜そうでなければ 〜の間 増やしながら 減らしながら 繰り返しを抜ける

配列に追加 配列に連結 関数 手続き

サンプル1 サンプル2 サンプル3 サンプル4 サンプル5 サンプル6 サンプル7 サンプル8

[マニュアル](#)

このアドレスをブラウザに読み込ませると…

自動採点（正解）

新規 実行 ステップ実行 リセット 変数確認 *

☒フローチャート ☐コード→フローチャート ☐コード→Python ☐URL生成

Upload 選択されていません [Download](#) ファイル名:

Q6: 曜日の計算

3つの整数(西暦年, 月, 日)を受け取って,
その日の曜日を番号(0:日曜, 1:月曜, ..., 6:土曜)で表示しなさい。

+ 0 -

```

1 yに整数を入力する
2 mに整数を入力する
3 dに整数を入力する
4 もしm<3ならば:
5     ...y←y-1
6     ...m←m+12
7 j←y//100
8 k←y%100
9 h←d+(13*m+8)//5+k+k//4+j//4+5*j
10 h%7を表示する
11 .

```

```

*** 採点開始 ***
ケース1...成功
ケース2...成功
ケース3...成功
ケース4...成功
ケース5...成功
ケース6...成功
ケース7...成功
ケース8...成功
ケース9...成功
*** 合格 ***

```

入力（整数） 入力（実数） 入力（文字列） 入力（真偽） 出力 改行無出力

← +← -← *← /← %← &← |← ^←

もし ~ そうでなければ ~の間 増やしながら 減らしながら 繰り返しを抜ける

配列に追加 配列に連結 関数 手続き

サンプル1 サンプル2 サンプル3 サンプル4 サンプル5 サンプル6 サンプル7 サンプル8

[マニュアル](#)

+ 0 -

はじめ

おわり

自動採点（不正解）

新規 実行 ステップ実行 リセット 変数確認 *

☒ フローチャート ☐ コード→フローチャート ☐ コード→Python ☐ URL生成

Upload 選択されていません [Download](#) ファイル名:

Q6: 曜日の計算

3つの整数(西暦年, 月, 日)を受け取って,
その日の曜日を番号(0:日曜, 1:月曜, ..., 6:土曜)で表示しなさい。

+ 0 -

```

1 yに整数を入力する
2 mに整数を入力する
3 dに整数を入力する
4 j←y//100
5 k←y%100
6 h←d+(13*m+8)//5+k//4+j//4+5*j
7 h%7を表示する
8 .

```

```

*** 採点開始 ***
ケース1...成功
ケース2...成功
ケース3...失敗
結果が違います。
ケース4...失敗
結果が違います。
ケース5...成功
ケース6...失敗
結果が違います。
ケース7...成功
ケース8...成功
ケース9...失敗
結果が違います。
--- 不合格 ---

```

入力（整数） 入力（実数） 入力（文字列） 入力（真偽） 出力 改行無出力

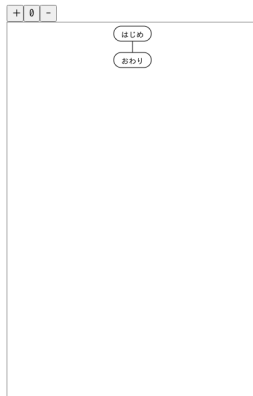
← +← -← *← /← %← &← |← ^←

もし もし～そうでなければ ～の間 増やしながら 減らしながら 繰り返しを抜ける

配列に追加 配列に連結 関数 手続き

サンプル1 サンプル2 サンプル3 サンプル4 サンプル5 サンプル6 サンプル7 サンプル8

[マニュアル](#)



余談

大学入学共通テスト『情報』試作問題（検討用イメージ）第5問

```

(01) Angoubun = ["p", "y", "e", "b", ..., (省略) ..., "k", "b", "d", "r", "."]
(02) 配列変数 Hirabun を初期化する
(03) hukugousuu = 26 - 
(04) i を 0 から 要素数(Angoubun)-1 まで1ずつ増やしながら:
(05) |   bangou = 差分()
(06) |   もし bangou != -1 ならば:
(07) |   |   もし  <= 25 ならば:
(08) |   |   |   Hirabun[i] = 文字()
(09) |   |   |   そうでなければ:
(10) |   |   |   |   Hirabun[i] = 文字()
(11) |   |   |   |   そうでなければ:
(12) |   |   |   |   |   Hirabun[i] = 
(13) 表示する(Hirabun)

```

ス ①bangou+hukugousuu
※
セ ③bangou+hukugousuu-26
ソ ⑥Angoubun[i]

3 PyPEN を用いた授業の提案

理想：PyPEN で短時間の学習をすれば、すぐに Python が使える

3.1 提案の動機

日本語プログラミング言語といえは「なでしこ」
v3 でもインデント構文が使える（ようになった）

なでしこ3 Web簡易エディタ

```
!ここまでだるい
aを1から100まで繰り返す
  もしa%10=3またはFLOOR(a/10)=3またはa%3=0ならば
    「(´・ω・`)」を表示。
  違えば
    aを表示。
```

▶ 実行

クリア

保存

v3.1.16

```
(´・ω・`)
28
29
(´・ω・`)
(´・ω・`)
```

「なでしこ」は高機能

- タートルグラフィックス
- DOM 操作
- AJAX, WebSocket
- グラフ描画
- オーディオ
- プラグインによる拡張
- ダウンロード版

「PyPEN」は低機能

- ブラウザ内の閉じた世界
グラフくらいは描けるけど

PyPEN も「高機能」を目指すのか？

- Python がクローズアップされている理由は？
 - シンプルな構文
 - 豊富なライブラリ
- 同じようなことができるのか？

中途半端な「真似事」よりも、学習用に徹したい
早く他の言語に行けるための足がかりでありたい
ドリトルの「1時間で学ぶソフトウェアの仕組み」を目指す
いや、ドリトルは高機能なのだけど

ドリトルの「1時間で学ぶソフトウェアの仕組み」



プログラミング言語「ドリトル」
大阪電気通信大学 兼宗研究室

[最近の変更](#)
[メディアマネージャー](#)
[サイトマップ](#)

トレース: [d1h](#)

1時間で学ぶソフトウェアの仕組み

兼宗 進 (大阪電気通信大学) 2009年11月6日

はじめに

プログラミングは、うまく使うと生徒の集中力や考える力を伸ばします。そして、身の回りの多くのソフトウェアがどのような仕組みで動いているのかを理解できるようになります。

ソフトウェアの仕組みの体験的な学習は、ドリトルを使うと以下に説明する1時間の授業(小中高では45～50分、大学等では90分)が可能です。オンライン版のドリトルを使うと、あらかじめ教室のパソコンで動いてを確認しておくだけで、インストールの手間もありません。ただ実施してみたいかがでしょう。生徒の反応を見ることが可能です。

宝物拾いゲーム

題材は「宝物拾いゲーム」です。プログラムは書籍「ドリトル」では教えやすいように若干変えてあります。進め方は入力しながら、生徒に入力させて、1行ずつ実行させていきます。

前半

まず最初に、「ゲームを作るために、画面に主役を作ろう」と。また、必要に応じて「=」「|」「。」などの記号の入力を行います。

生徒には先生の画面を見ながら同じように入力させ、自分の時点でも「何か出た!」「カメだ!」と歓声が上がります。

かめた＝タートル！作る。

続いて、「次にかめたを操作するボタンを作ろう」と言い

目次

- 1時間で学ぶソフトウェアの仕組み
- はじめに
- 宝物拾いゲーム
 - 前半
 - 後半
 - まとめ
 - おわりに
- (付録)体験から学ぶことの例

かめた＝タートル！作る。

左ボタン＝ボタン！「左」作る。

左ボタン：動作＝「かめた！30 左回り」。

右ボタン＝ボタン！「右」作る。

右ボタン：動作＝「かめた！30 右回り」。

時計＝タイマー！作る。

時計！「かめた！10 歩く」実行。

タートル！作る "tulip.png" 変身する ペンなし 100 100 位置。

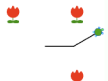
タートル！作る "tulip.png" 変身する ペンなし 100 -100 位置。

タートル！作る "tulip.png" 変身する ペンなし -100 100 位置。

かめた：衝突＝「|相手| 相手！消える」。

左

右



3.2 授業の流れ

- ① 基本的な演算, 命令, 制御
- ② 配列・関数
- ③ シミュレーション

基本的な演算, 命令, 制御

+ 0 -

```
1 n ← 1
2 a ← -1
3 n ≤ 5の間:
4   ... n, aを表示する
5   ... n ← n+1
6   ... a ← 2*a+3
7   .
```

1 -1
2 1
3 5
4 13
5 29

基本的な演算, 命令, 制御

+ 0 -

```
1 n ← 1
2 a ← -1
3 n ≤ 5の間:
4   ... n, aを表示する
5   ... n ← n + 1
6   ... a ← 2 * a + 3
7   .
```

```
n = 1
a = -1
while n <= 5:
    print(n,a)
    n = n + 1
    a = 2 * a + 3
```

Python のコードと一対一対応

配列・関数

+ 0 -

```
1 a←[0,0,0,0,0,0,0,0,0,0]
2 l←length(a)
3 nを0からl-1まで1ずつ増やしながら：
4   ...a[n]に整数を入力する
5 s←0
6 l←length(a)
7 nを0からl-1まで1ずつ増やしながら：
8   ...s←s+a[n]
9 a,sを表示する
10 .
```

```
a = [0,0,0,0,0,0,0,0,0,0]
l = len(a)
for n in range(0, l - 1+1, 1):
    a[n] = int(input())
s = 0
l = len(a)
for n in range(0, l - 1+1, 1):
    s = s + a[n]
print(a,s)
```

配列を使う

配列・関数

+ 0 -

```

1 a←[0,0,0,0,0,0,0,0,0,0]
2 l←length(a)
3 nを0からl-1まで1ずつ増やしなが：
4   ...a[n]に整数を入力する
5 a,sum(a)を表示する
6 関数・sum(a):
7   ...s←0
8   ...l←length(a)
9   ...nを0からl-1まで1ずつ増やしなが：
10    ...s←s+a[n]
11   ...sを返す
12 .

```

```

def sum(a):
    s = 0
    l = len(a)
    for n in range(0, l - 1 + 1, 1):
        s = s + a[n]
    return s
a = [0,0,0,0,0,0,0,0,0,0]
l = len(a)
for n in range(0, l - 1 + 1, 1):
    a[n] = int(input())
print(a, sum(a))

```

関数を使う

日本語ベースであることのメリット (?)

ループ以前

```
s ← 0  
s ← s+a[0]  
s ← s+a[1]  
...  
s ← s+a[9]
```

「 $s \leftarrow s+a[n]$ 」の n を、0 から 9 まで増やしながら繰り返せばいい

ループで表現

```
s ← 0  
n を 0 から 9 まで 1 ずつ増やしながら :  
    s ← s+a[n]
```

そのままだ！

シミュレーション（ガチャ）

+ 0 -

```
1 a←[0,0,0,0,0,0,0,0,0,0]
2 n←0
3 a!= [1,1,1,1,1,1,1,1,1,1]の間:
4   ...n←n+1
5   ...a[random(9)]←1
6   ...n,aを表示する
7 .
```

```
import random
a = [0,0,0,0,0,0,0,0,0,0]
n = 0
while a != [1,1,1,1,1,1,1,1,1,1]:
    n = n + 1
    a[random.randint(0,9)] = 1
print(n,a)
```

シミュレーション（ガチャ）

+ 0 -

```
1 a←[0,0,0,0,0,0,0,0,0,0]
2 n←0
3 a!=[1,1,1,1,1,1,1,1,1,1]の間：
4   ...n←n+1
5   ...a[random(9)]←1
6   ...n,aを表示する
7 .
```

```
15 [0,1,1,1,0,0,1,1,0,1]
16 [0,1,1,1,0,0,1,1,0,1]
17 [0,1,1,1,0,0,1,1,0,1]
18 [0,1,1,1,0,0,1,1,0,1]
19 [0,1,1,1,0,0,1,1,0,1]
20 [0,1,1,1,0,0,1,1,1,1]
21 [0,1,1,1,0,0,1,1,1,1]
22 [1,1,1,1,0,0,1,1,1,1]
23 [1,1,1,1,0,0,1,1,1,1]
24 [1,1,1,1,0,0,1,1,1,1]
25 [1,1,1,1,0,0,1,1,1,1]
26 [1,1,1,1,0,0,1,1,1,1]
27 [1,1,1,1,0,0,1,1,1,1]
28 [1,1,1,1,0,0,1,1,1,1]
29 [1,1,1,1,0,1,1,1,1,1]
30 [1,1,1,1,0,1,1,1,1,1]
31 [1,1,1,1,1,0,1,1,1,1]
32 [1,1,1,1,1,0,1,1,1,1]
33 [1,1,1,1,1,1,1,1,1,1]
---
```

シミュレーション（ガチャ）

+ 0 -

```

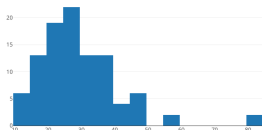
1 関数・gacha():
2  ...a←[0,0,0,0,0,0,0,0,0,0]
3  ...n←0
4  ...a!=[1,1,1,1,1,1,1,1,1,1]の間：
5  ...n←n+1
6  ...a[random(9)]←1
7  ...nを返す
8  a←[]
9  nを1から100まで1ずつ増やしながら：
10 ...aにgacha()を追加する
11 aを表示する
12 グラフ描画({}, [{"type": "histogram", "x": a}])

```

```

[29,27,19,20,26,15,39,12,45,56,20,40,
29,26,23,19,46,32,20,26,24,22,25,43,2
5,21,26,20,39,56,15,37,26,18,35,37,37
,17,27,32,44,39,38,31,33,26,34,16,84,
25,45,30,81,34,19,35,14,35,34,25,47,4
5,15,22,22,20,36,27,29,30,24,23,13,32
,18,27,26,14,15,25,24,31,16,36,25,23,
38,19,25,25,41,22,23,34,13,34,14,22,2
0,48]
---

```



シミュレーション（ガチャ）

+ 0 -

```

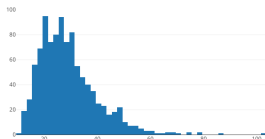
1 関数・gacha():
2  ...a←[0,0,0,0,0,0,0,0,0]
3  ...n←0
4  ...a!=[1,1,1,1,1,1,1,1,1]の間:
5  ...n←n+1
6  ...a[random(9)]←1
7  ...nを返す
8 a←[]
9 nを1から1000まで1ずつ増やしながら:
10 ...aにgacha()を追加する
11 aを表示する
12 グラフ描画({},[{"type":"histrogram","x":a}])

```

```

19,17,38,28,28,22,25,39,35,26,39,32
16,18,26,29,49,28,13,17,21,39,24,2
7,25,20,40,28,23,41,17,34,43,27,24,
39,21,24,35,33,38,44,13,26,26,28,11
30,22,23,20,45,21,26,25,25,26,17,4
2,18,21,41,17,50,19,46,32,21,30,41,
31,46,28,30,17,12,19,20,39,79,28,27
13,38,29,44,28,13,26,35,40,12,26,3
5,33,20,28,17,21,18,20,19,30,27,18,
23,15,23,23,28,20,33,16,22,28,33,25
42,31,25,27,25,19,25,29,37,46,36,4
1,24,26,25,41,13,56,15,32,39,27,41,
16,27,33,20,49,29,31,31,39,37,48,49
22,17,31,48,74,30,35,19,21,47,37,2
2,34,32,32,23,48,48,29,28,44,21,43,
17,21,14,17,30,28,37,25,48,43,20,29
22,24,21,32,25,14,17,17,30,15,13,2
1,22,31,40,26,25,47,33,47,19,26,40,
47,63,21,30,69,21,29,18,17]
---

```



1000 回だとこんな感じ

シミュレーション（サイコロ）

+ 0 -

```
1 関数・dice():  
2 ・・・a←[0,0,0,0,0,0]  
3 ・・・kを1から60まで1ずつ増やしながら：  
4 ・・・・・・a[random(5)]+←1  
5 ・・・aを返す  
6 dice()を表示する
```

[8,11,9,9,15,8]

シミュレーション (サイコロ)

+ 0 -

```
1 関数・dice():  
2   ...a←[0,0,0,0,0,0]  
3   ...kを1から60まで1ずつ増やしなが  
4   ...a[random(5)]+←1  
5   ...n,aを表示する  
6   ...aを返す  
7   n←1  
8   dice()!=[10,10,10,10,10,10]の間:  
9   ...n←n+1  
10  .
```

```
176 [9,14,6,13,10,8]  
177 [6,12,15,13,7,7]  
178 [10,9,11,8,11,11]  
179 [13,8,13,9,2,15]  
180 [15,8,11,11,8,7]  
181 [7,4,12,7,17,13]  
182 [4,13,11,7,14,11]  
183 [11,12,13,5,11,8]  
184 [10,8,13,14,4,11]  
185 [9,10,14,11,10,6]  
186 [10,6,9,6,12,17]  
187 [8,11,10,12,9,10]  
188 [10,8,10,11,10,11]  
189 [10,11,11,12,11,5]  
190 [6,9,15,17,4,9]  
191 [10,8,19,7,8,8]  
192 [9,7,7,5,17,15]  
193 [8,13,6,12,7,14]  
194 [10,10,8,11,9,12]  
195 [10,11,8,11,11,9]
```

シミュレーション (サイコロ)

+ 0 -

```
1 関数・dice():  
2   ...a←[0,0,0,0,0,0]  
3   ...kを1から60まで1ずつ増やしながら:  
4   ...a[random(5)]+←1  
5   ...n,aを表示する  
6   ...aを返す  
7   n←1  
8   dice()!= [10,10,10,10,10,10]の間:  
9   ...n←n+1  
10  .
```

2949 [14,6,14,12,7,7]
2950 [11,12,5,9,10,13]
2951 [8,12,12,11,12,5]
2952 [10,10,17,5,11,7]
2953 [9,11,15,5,14,6]
2954 [12,10,8,13,7,10]
2955 [6,16,7,9,10,12]
2956 [8,12,5,10,12,13]
2957 [10,7,8,11,10,14]
2958 [7,17,8,8,14,6]
2959 [10,9,11,10,13,7]
2960 [10,13,6,9,10,12]
2961 [12,10,10,11,8,9]
2962 [12,6,9,8,15,10]
2963 [11,8,7,13,14,7]
2964 [14,13,13,7,4,9]
2965 [10,10,12,10,9,9]
2966 [8,13,15,4,8,12]
2967 [5,11,14,7,9,14]
2968 [10,5,13,9,12,11]

なかなか終わらない…PyPEN じゃ遅すぎる

シミュレーション (サイコロ)

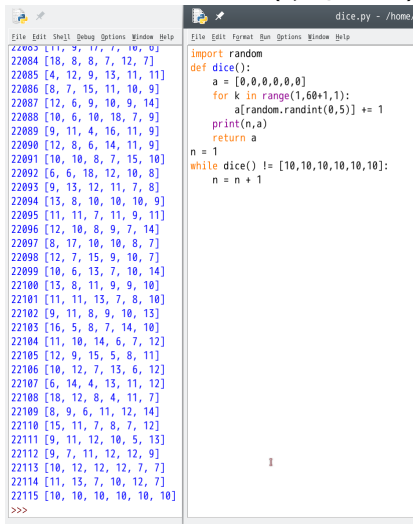
+ 0 -

```
1 関数・dice():
2  ...a←[0,0,0,0,0,0]
3  ...kを1から60まで1ずつ増やしながら:
4  .....a[random(5)]+←1
5  ...n,aを表示する
6  ...aを返す
7  n←1
8  dice()!=[10,10,10,10,10,10]の間:
9  ...n←n+1
10 .
```

```
import random
def dice():
    a = [0,0,0,0,0,0]
    for k in range(1,60+1,1):
        a[random.randint(0,5)] += 1
    print(n,a)
    return a
n = 1
while dice() != [10,10,10,10,10,10]:
    n = n + 1
```

Python のコードを生成して…

シミュレーション（サイコロ）



The image shows a Python IDE with two windows. The left window displays the output of a dice simulation script, showing 22 random rolls of a 6-sided die. The right window displays the source code of the script, which defines a `dice()` function and uses a `while` loop to simulate rolls until a specific sequence is achieved.

```
File Edit Shell Debug Options Window Help
22003 [11, 9, 17, 7, 10, 0]
22084 [18, 8, 8, 7, 12, 7]
22085 [4, 12, 9, 13, 11, 11]
22086 [8, 7, 15, 11, 10, 9]
22087 [12, 6, 9, 10, 9, 14]
22088 [10, 6, 10, 18, 7, 9]
22089 [9, 11, 4, 16, 11, 9]
22090 [12, 8, 6, 14, 11, 9]
22091 [10, 10, 8, 7, 15, 10]
22092 [6, 6, 18, 12, 10, 8]
22093 [9, 13, 12, 11, 7, 8]
22094 [13, 8, 10, 10, 10, 9]
22095 [11, 11, 7, 11, 9, 11]
22096 [12, 10, 8, 9, 7, 14]
22097 [8, 17, 10, 10, 8, 7]
22098 [12, 7, 15, 9, 10, 7]
22099 [10, 6, 13, 7, 10, 14]
22100 [13, 8, 11, 9, 9, 10]
22101 [11, 11, 13, 7, 8, 10]
22102 [9, 11, 8, 9, 10, 13]
22103 [16, 5, 8, 7, 14, 10]
22104 [11, 10, 14, 6, 7, 12]
22105 [12, 9, 15, 5, 8, 11]
22106 [10, 12, 7, 13, 6, 12]
22107 [6, 14, 4, 13, 11, 12]
22108 [18, 12, 8, 4, 11, 7]
22109 [8, 9, 6, 11, 12, 14]
22110 [15, 11, 7, 8, 7, 12]
22111 [9, 11, 12, 10, 5, 13]
22112 [9, 7, 11, 12, 12, 9]
22113 [10, 12, 12, 12, 7, 7]
22114 [11, 13, 7, 10, 12, 7]
22115 [10, 10, 10, 10, 10, 10]
>>>
```

```
File Edit Format Run Options Window Help
dice.py - /home/

import random
def dice():
    a = [0,0,0,0,0,0]
    for k in range(1,60+1,1):
        a[random.randint(0,5)] += 1
    print(n,a)
    return a
n = 1
while dice() != [10,10,10,10,10,10]:
    n = n + 1
```

速い！！

おわりに

「いきなり Python」でいい生徒も多いが
 段差を感じる生徒もいる
 それを越えるための足場を作りたい