情報リテラシー実践 2B レポート課題 2

情報科学科 22140026 谷 知拓

目次

Approach

- 「ヒットアンドブロー」のプログラムを書く.
- 言語は, Python を用いる.
- クラスを有用に使うことで, **汎用性(e.g. 桁数の可変)**と**可読性(e.g. 同一性と同値性の使い分け)**を向上させる.

機能

- ユーザの指定に応じて桁数を可変にする(1~9)
- 乱数で答えを生成するときに, 同じ数字を異なるケタで重複して使わないようにする
- 意図しないユーザの入力値によるクラッシュを回避する
- 1 つの返り値で, 正解の場合および不正解の場合の両方に対応する(不正解の場合は, ヒット数とブロー数も返す)

コーディング上の工夫(Solutions)

Line	Features	Solutions
9~14, 42~45	ユーザの指定に応じて桁数を可変にする	hab(hit_and_blow)クラス の初期化時に, ユーザの input した桁数を渡し, それに応じてその桁数の乱数を生成する
17~19	乱数で答えを生成するときに, 同じ数字を異なるケタで重複して使わないようにする	生成した値を, 1 ケタごとのリストにして, **len(numa) == len(set(numa))** の比較を使って, 重複がないか確認する. **(set 型のメンバがユニークになることを利用)** 重複があった場合, 重複のない値になるまで, 乱数生成を繰り返す.
51~61	意図しないユーザの入力値によるクラッシュを回避する	**try-except 文と raise を用いる**ことで, 例外処理であることを明確にし, 可読性を上げた.

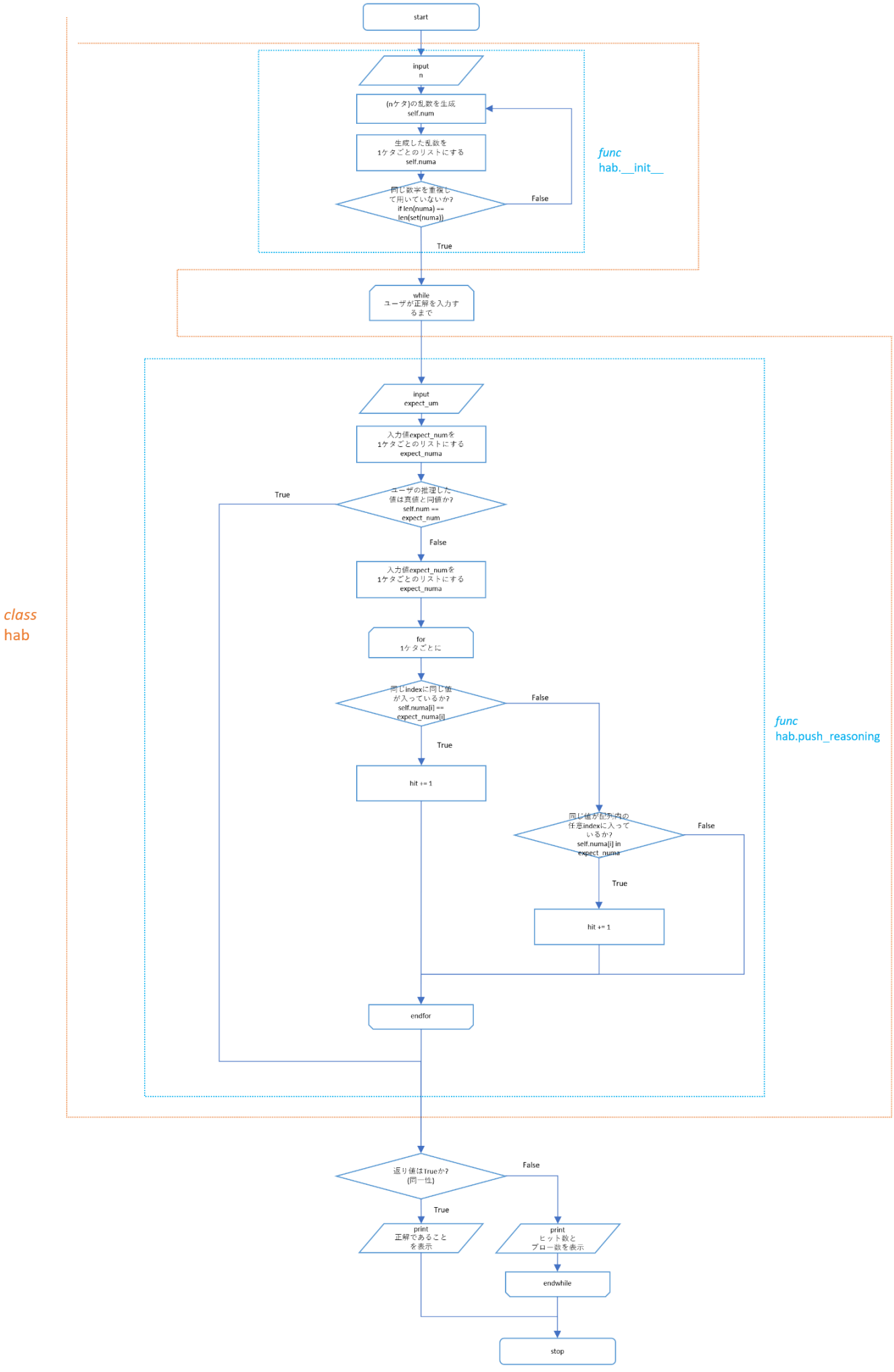
© Tomohiro Tani / Taniii.com

		raise を用いてユーザ定義のエラーとして， 「桁数の入力ミス」と「同じ数字の重複使用入力ミス」のエラーを発生させている． これを try-except で拾い，エラーメッセージを出力した上で，再度 input を呼び出し，適切な値が入力されるまで繰り返されるようになっている．
22~38, 63~71	1 つの返り値で，正解の場合および不正解の場合の両方に対応する(不正解の場合は，ヒット数とブロー数も返す)	クラス上では， 正解の場合 → bool 型の True を返す 不正解の場合 → ヒット数とブロー数を，list 型の連想配列で返す 呼び出し側では，まず先に**厳密評価(同一性の評価 is)**で，**参照オブジェクト自体が同一かどうか**を判定する．次に，同一でなかった場合は，「不正解」と判断し，ヒット数とブロー数が返却されているものとして処理を続ける． *** 「同値性の評価」でなく，「同一性の評価」を用いることに注意する．この 2 つは明確に異なっていて，「同値性の評価」を用いると，意図しない暗黙の型変換により，期待した通りに動かない．**

動作のフローチャート

(次のページへつづく)

© Tomohiro Tani / Taniii.com


start
input
n
(nケタ)の乱数を生成
self.num
生成した乱数を
1ケタごとのリストにする
self.numa
同じ数字を重複して
用いていないか?
if len(numa) ==
len(set(numa))
False
True
func
hab.__init__
while
ユーザが正解を入力す
るまで
input
expect_um
入力値expect_numを
1ケタごとのリストにする
expect_numa
ユーザの推理した
値は真値と同値か?
self.num ==
expect_num
True
False
入力値expect_numを
1ケタごとのリストにする
expect_numa
for
1ケタごとに
同じindexに同じ値
が入っているか?
self.numa[i] ==
expect_numa[i]
False
True
hit += 1
同じ値が配列内の
任意indexに入って
いるか?
self.numa[i] in
expect_numa
False
True
hit += 1
endfor
class
hab
func
hab.push_reasoning
返り値はTrueか?
(同一性)
False
True
print
正解であること
を表示
print
ヒット数と
ブロー数を表示
endwhile
stop

Code

```
"""
    レポート課題 2: ヒットアンドブロー(数あてゲーム)のプログラム
"""

import numpy as np

# hab: hit and blow クラス 初期化時に桁数を指定できる
class hab:
    def __init__(self, digits):
        # 初期化
        # ユーザ指定桁数をもとに，乱数を生成
        # 同じ数字が異なるケタに重複して使われないようにする
        while True:
            num = np.random.randint(10**(digits - 1), 10**digits) # 答えとなる数字
            # 生成した数字をさらに，1 ケタずつのリストにする
            numa = [int(s) for s in str(num)]
            if len(numa) == len(set(numa)):
                self.num, self.numa = num, numa
                break

    def push_reasoning(self, expect_num):
        # ユーザの推理した値を，真値と比較し，正解なら True，不正解なら「ヒット数」「ブロー数」を返す
        if self.num == expect_num:
            return True
        else:
            hit, blow = 0, 0 # init 変数の初期化
            # ユーザの予想値を，1 ケタずつのリストにする
            expect_numa = [int(s) for s in str(expect_num)]
            for i in range(len(self.numa)):
                if self.numa[i] == expect_numa[i]:
                    # 同じ index に同じ数字のとき，hit +1
                    hit += 1
                elif self.numa[i] in expect_numa:
                    # 上でなくて，かつ，配列の中に同じ数字が存在するとき
                    blow += 1
                else:
                    pass
            return {"hit": hit, "blow": blow}


# 実行段階
# ユーザがケタ数を指定
digits = int(input("ケタ数を指定(1~9)(e.g. 3):"))
# ユーザ指定のケタ数で，インスタンスを生成
inst = hab(digits)
count = 0

while True:
    count += 1
    # ユーザに，予想値を入力してもらう
    while True:
        try:
            expect_num = int(input("値を予想({}ケタ): ".format(str(digits))))
            if len(list(str(expect_num))) != digits:
                raise ValueError("invalid input - 入力が正しくありません．再入力してください.")
            if len(list(str(expect_num))) != len(set(list(str(expect_num)))):
                raise ValueError("invalid input - 同じ数字を繰り返し使うことはできません．再入力してください.")
        except ValueError as e:
            print(e)
        else:
            break
    res = inst.push_reasoning(expect_num)
    if res is True: # 厳密評価で先に「True」の場合を処理する
        print("{}回目で正解!!!".format(str(count)))
        break
    else:
        print("-",str(count) ,"試行目 -")
        print("「", str(expect_num), "」")
        print("ヒット:", res["hit"])
        print("ブロー:", res["blow"])
        print("----------¥n")

```

© Tomohiro Tani / Tanili.com

Jupyter での実行結果例

```
ケタ数を指定(1~9)(e.g. 3):3
値を予想(3 ケタ): 1 // 桁数の入力ミス
invalid input - 入力が正しくありません．再入力してください．
値を予想(3 ケタ): 111 // 同じ数字の繰り返しミス
invalid input - 同じ数字を繰り返し使うことはできません．再入力してください．
値を予想(3 ケタ): 123
- 1 試行目 -
「 123 」
ヒット: 1
ブロー: 1
-----------

値を予想(3 ケタ): 142
- 2 試行目 -
「 142 」
ヒット: 1
ブロー: 0
-----------

値を予想(3 ケタ): 135
- 3 試行目 -
「 135 」
ヒット: 2
ブロー: 0
-----------

値を予想(3 ケタ): 678
- 4 試行目 -
「 678 」
ヒット: 0
ブロー: 1
-----------

値を予想(3 ケタ): 136
- 5 試行目 -
「 136 」
ヒット: 2
ブロー: 0
-----------

値を予想(3 ケタ): 137
6 回目で正解!!!
```

考察と感想

以前, Python のコーディング規約 PEP8 に『True との比較には, "=="ではなく，常に"is"を使うこと』とあるのを見たときに，その時は，「**より厳密で，動作のゆらぎを避け，明示的なコーディングを目指す思想**」によるものだろうとしか思わなかったが，今回，返り値に, (True) or (False の時は何らかの別の値)という構造で，データの受け渡しをしてみて, "is"を使った，同一性の評価を先にすることで，ものすごく綺麗にコードを書けることが実感でき，「**同値性」と「同一性」の使い分け**が，ただ単に Readability の問題だけでなく，コードを書く上での Hack としても役立つことが身をもって実感できた.

© Tomohiro Tani / Tanii.com