2 学期 プログラミング基礎

前置き

2 学期は, さまざまなプログラムを見て, 構造を考えることをメインにしていきます.
その中で, コードを改善していき, 目的のプログラムを作成してもらいます.
この経験が, 1 学期の内容の理解を深めることに役に立つでしょう.

空白を多く取ってみます. 見やすくなるでしょうか.

2 年　　組　　番 氏名

7　探索アルゴリズム

配列の中から, 探している値が何番目にあるのかを探すアルゴリズムを考える.

7.1 線形探索

純粋に左から順に 1 個ずつ調べていく方法.

手順書

配列 a と探している数字が与えられたとき, 以下の手順で探索する.

(1) a[0] が探している数字と同じか調べる.
　(a) もし同じであれば, 0 を返し, 繰り返しを抜ける.
(2) この操作を残り $1, 2, \cdots$ で配列の長さ分だけ繰り返す.
(3) 繰り返した回数が配列の長さになったときに見つかっていない場合は, 探している値は存在しない.

7.1.1 仕組みを捉える

トランプを用いて, 上の手順通りに探索してみよう.

7.1.2 python で線形探索

[ ] に当てはまる文を考えて入れよう.
(配列の中の要素は各自好きに入れると良い.)

```
a=[1, 3, 2, 5, 12, 4, 7, 8, 21, 10, 9]
search_num=int(input("search number = ", ))
for i in [ ]:
    if a[i]==search_num:
        print("No. ",i+1)
        break
    [ ]:
        print("Not found.")
```

7.2 二分岐探索

事前に昇順に並んだデータに対して使える探索手法. 目的の値が中央よりも右か左かを探索していく. 計算量は $O(\log n)$ であり, かなり効率のいいアルゴリズム.

手順書

昇順に並んだ配列 a と探している数字が与えられたとき, 以下の手順で探索する.

(1) 探索範囲の左端の配列番号を left , 右端の配列番号を right で初期化する.
(2) left の値が right の値以下の場合, 以下を繰り返す.
  (a) (left+right)÷2 の商の値を mid とする. (中央の値を得る)
  (b) mid 番目の配列が調べている数であれば, その配列番号を出力してループ脱出.
  (c) mid 番目の配列 < 調べている数であれば, left を mid+1 にする.
  (d) mid 番目の配列 > 調べている数であれば, right を mid−1 にする.
(3) left>right の場合, Not Found と出力する.

7.2.1 仕組みを捉える

トランプを用いて, 上の手順通りに探索してみよう.

7.2.2 python で二分岐探索

[ ] に当てはまる文を考えて入れよう.
(配列の中の要素は各自好きに入れると良い.)

```
data=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_num=int(input("search number=",))
left=0
right=len(data)-1
while [ ]:
    mid=[ ]
    if data[mid]==search_num:
        print("No.",mid)
        break
    elif [ ]:
        left = mid + 1
    else:
        right = mid - 1
if left>right:
    print("Not found")
```

7.2.3 模範解答例

線形探索

```
a=[1, 3, 2, 5, 12, 4, 7, 8, 21, 10, 9]
search_num=int(input("search number = ", ))
for i in range(len(a)):
    if a[i]==search_num:
        print("No. ",i+1)
        break
    if i==len(a)-1:
        print("Not found.")
```

二分岐探索

```
data=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_num=int(input("search number=",))
left=0
right=len(data)-1
while left<=right:
    mid=(left+right)//2
    if data[mid]==search_num:
        print("No.",mid)
        break
    elif data[mid]<search_num:
        left = mid + 1
    else:
        right = mid - 1
if left>right:
    print("Not found")
```

8　ソートアルゴリズム

昇順・降順に並び替えるアルゴリズムについて学ぶ.

8.1 バブルソート

手順書

配列 a を以下の手順で並べ替えていく.

(1) 以下の操作を $i = 0$ から (配列長 -1) 回繰り返す.
  (a) 以下の操作を $j = 0$ から ((配列長) $- i - 1$) 回繰り返す.
    i. j 番目の数 $> j + 1$ 番目の数であれば数の入れ替え
    ii. そうでなければ何もしない.

8.1.1 仕組みを捉える

トランプを用いて, 上の手順書通りに並べ替えてみよう. 何手で並べ替えることができるか. 入れ替わりの流れを残しておこう.

8.1.2 python でバブルソート

```
l=[2,3,1,5,9,6,7]
for i in range(len(l)-1):
 for j in range(len(l)-i-1):
  if l[j]>l[j+1]:
   tmp=l[j+1]
   l[j+1]=l[j]
   l[j]=tmp
print(l)
```

8.2 選択ソート

手順書

配列 a を以下の手順で並べ替えていく.

(1) 以下を (配列長 − 1) 回繰り返す.
  (a) least=i とする.
  (b) 以下を $j = i + 1$ から (配列長 − 1) 未満で繰り返す.
    i. 配列の j 番目の数と least 番目の数を比較し, j 番目が小さければ least=j とする.
(2) i 番目の配列と least 番目の配列を入れ替える.

8.2.1 仕組みを捉える

トランプを用いて, 上の手順書通りに並べ替えてみよう. 何手で並べ替えることができるか. 入れ替わりの流れを残しておこう.

8.2.2 python で選択ソート

```
l=[1,6,3,9,2,7,5]
for i in range(len(l)-1):
 least=i
 for j in range(i+1,len(l)):
  if l[j]<l[least]:
   least=j
 tmp=l[i]
 l[i]=l[least]
 l[least]=tmp
 print(l)
```

8.3 その他のソート

授業では扱わないが, 他にも様々なソートの手法がある.

(1) 挿入ソート
(2) クイックソート
(3) マージソート

などなど...

これらの手法の方法を調べ, 自分なりにまとめてみよう. (手順書 + ソート例)

9　作図

9.1 作図

記号を組み合わせることで, 図を描いてみよう.

9.2 コード

コードを読んで,

1. どのような出力がされるかを予想
2. 余白に書き残す
3. 実際に実行して結果を確認

(1)

```
for i in range(10):
    for j in range(i):
        print("*",end="")
    print("")
```

補足) 「end=””」で, 改行されることを強制的に解除している.

(2)

```
for i in range(10):
    for j in range(9-i):
        print("-",end="")
    for j in range(2*i+1):
        print("*",end="")
    for j in range(9-i):
        print("-",end="")
    print("")
```

(3)

```
N=int(input("number= "))
for i in range(N):
    for j in range(N):
        if (i+j)%2==0:
            print(" ",end="")
        else:
            print("*",end="")
    print("")
```

予想) 実行後に 6 と入力した際の予想をしてください.

9.2.1 練習問題

実行後, 以下のように表示されるプログラムを作成せよ.

(1)

```
*********
 *******
  *****
   ***
    *
```

(2)

```
   ***
   ***
   ***
*********
*********
*********
   ***
   ***
   ***
```

11　ヌメロン

11.1　ゲームのルール

　ヌメロンというゲームを知っているだろうか. 各自 3 桁の数字を作成し, 相手の数字を当てるゲームである. 3 桁といったが, 0 から始めてもいいが, 同じ数字の重複は許されない.
【ルール】

(1) 先攻・後攻を決定.
(2) 先攻は相手の数字を予想し, その数字をコールする.
(3) 後攻は, コールされた数字と自身の数字がどの程度合っているかを発表する.
　(a) 数字と桁が揃っている場合「EAT」.
　(b) 数字は合っているが, 桁が違う場合「BITE」
　　コール (135), 自身の番号 (156) の場合, 「1-EAT, 1-BITE」
(4) (2),(3) を先攻・後攻交互に繰り返し, 先に相手の番号を完全に当てた方が勝ちである.

これを, 授業用に少しルール変更してプログラミングしてみた.

【授業用ヌメロンルール】

(1) コンピュータの数字を 10 回以内に当てることができれば勝ち.
(2) 人間はコンピュータの数字を予想し, その数字をコールする.
(3) 以下の場合はその CALL はスキップ.
　(a) 入力が 0 ～ 999 の間ではない.
　(b) 各桁の中に同じ数字が入っている.
(4) コンピュータはコールされた数字がどの程度合っているかを発表する.

11.2 コード

```
import random
print("Let's play numeron.")
print("You have ten times of chances.")
print(" ")
i=0
cpu_num1=random.randrange(10)
while i==0:
    cpu_num2=random.randrange(10)
    if cpu_num2!=cpu_num1:
        break
while i==0:
    cpu_num3=random.randrange(10)
    if (cpu_num3!=cpu_num1)and(cpu_num3!=cpu_num2):
        break
cpu_num=[cpu_num1,cpu_num2,cpu_num3]
for i in range(10):
    print("No.",i+1)
    n=int(input("CALL = ",))
    call=[0,0,0]
    call[0]=n//100
    call[1]=(n-call[0]*100)//10
    call[2]=(n-call[0]*100-call[1]*10)
    if (call[0]==call[1])or(call[1]==call[2])or(call[2]==call[0]):
        print("Fraud. Please change the number of each figure.")
        print("Pass")
    elif not(9<n<1000):
        print("Fraud. Please input three columns of numbers.")
        print("Pass")
    else:
        bite=0;eat=0
        for j in range(3):
            if call[j]==cpu_num[j]:
                eat=eat+1
        for j in range(3):
            for k in range(3):
                if (j!=k)and(call[j]==cpu_num[k]):
                    bite=bite+1
        print(" ",eat,"- EAT")
        print(" ",bite,"- BITE")
    if eat==3:
        print("You Win")
        break
    if i==9:
        print("Yow Lose")
        print("CPU's number is",cpu_num)
    print(" ")
print("Game End")
```

11.3 改良

新しい欄で, 以下の改良を実施せよ.

(1) 10 回以内という制限を撤廃し, ユーザまたは CPU が正解するまで継続するようにせよ.

(2) 4 桁でヌメロンができるようにせよ.