オブジェクト指向プログラミング

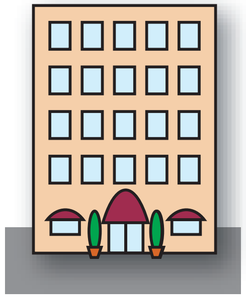
第１０回

箕原辰夫

配列・リスト

- リストには１つの変数の中に複数のデータ値を入れることができる。

一戸建て（普通の変数）には一世帯



マンション（配列変数）には複数の世帯

- 通常の変数をスカラー変数、リストや配列をベクトル変数と呼ぶこともある。

リストの記法

- 基本的には、リストを示す[ ]で値やオブジェクトを囲むだけ
- 例： xlist = [ ] # 空のリストが作成される

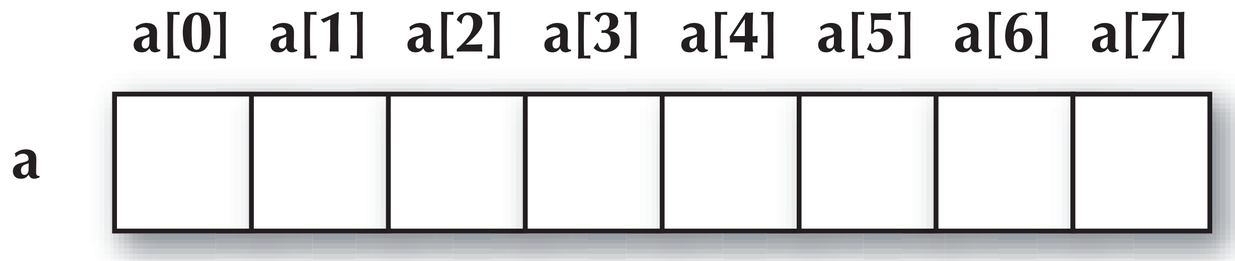
- 組込みのlist( ) 関数を使ってもリストを作成することができる
- 例： xlist = list( )

リストの確保と初期値

- リストについては、各要素の初期値を指定しないと、サイズ分確保されない。
- 例： xlist = [1, 2, 4, 9] # 4つのサイズのリストが作られる
- 0クリアされたリストを作りたい場合は、次のようにリストに掛け算を行なって作る
- 例：
  - a = [0] * 8 # サイズは、8となる
  - b =[0.0] * 20 # サイズは、20となる

リストの要素へのアクセス

- 確保されたリストには、各データを参照するための部屋番号がついている。これをインデックス（指標・添え字）と呼ぶ。



- インデックスは、「0〜サイズ-1」の範囲の整数に限られる。マイナスでアクセスする場合は、「-1〜-サイズ」に限られる。範囲外だと実行時エラー（例外）が発生する。

要素への参照と代入

- リストの要素は、通常の変数のように用いることができる。
- 要素を参照するには、次のような書式を用いる（インデックスは整数の式である）。
  - 変数名[インデックス]
  - 例： c.create_line(x[1], y[1], x[2], y[2]);
  - z = x[4] * 3
- 要素に個々に代入するには次のような書式を用いる
  - 変数名[インデックス] = 式
  - 例： a[3] = 10

インデックスの式

- インデックスは整数の式なので、変数や計算するものであっても良い。
  - x = 3であると仮定する
  - 例： a[x] = 5
  - 例： a[x+3] = 10

- リストの要素を参照する場合は、まずはそのインデックスの式から評価される。
  - 例： a[a[x+2]] = 20

    x = 3; a[5] = 4 → a[a[5]] ⇒ a[4] = 20

マイナスの値のインデックス

- マイナスの値を使うと最後の要素の値から順番に数えることができる（Python, swiftだけ）
- 例：
  - a = [5, 7, 9, 3, 6] # と初期化されているものとする
  - a[-1] ⇒ 6 … 最後の要素
  - a[-len(a)] ⇒ 5 … 最初の要素
  - a[-3] ⇒ 9

リストの要素への代入

- 要素へ代入するとき、代入文の省略形も使える
  - a[4] += 67
  - a[4] = a[4] + 67
  - a[5] += 1
  - a[5] = a[5] + 1

リストと繰返し

- リストは繰返しを用いて操作する。
- 代入をしてみる（アプリケーションで）
  - インデックスの値を使う
  - 漸化式を使う
  - 乱数を使う
    - int (random.random( ) * d) + a
    - a ... a+d-1の間の整数を発生させる

リストへの初期値代入

- 代入時に各要素の初期値が代入できる
  - リスト名 = [初期値, ... , 初期値]
  - 例：a = [10, 7, 4, 6, 3]

    ▸等価：a =[0] * 5

    a[0]=10; a[1]=7; a[2]=4; a[3]=6; a[4]=3
- list関数とrange( )関数を利用してリストを作成することができる
  - list(range(1, 8)) ⇒ [1, 2, 3, 4, 5, 6, 7]

リストをfor文で作る

- for文を使って初期化されたリストを作成することができる
  - 例：[x **for** x **in** range(1, 10)] ⇒

    [1, 2, 3, 4, 5, 6, 7, 8, 9]
- このときにif文やif式も利用することが可能
  - 例：[x **for** x **in** range(1, 10) **if** x % 2 == 0] ⇒

    [2, 4, 6, 8]
  - 例：a = [n **if** n % 2 == 0 **else** n*10 **for** n **in** range(1, 11)]

    [10, 2, 30, 4, 50, 6, 70, 8, 90, 10]
- for 文のネストも利用可能
  - 例： [n + m **for** n **in** range(10, 30, 10) **for** m **in** range(2, 6)] ⇒

    [12, 13, 14, 15, 22, 23, 24, 25]

リストのサイズ

- リストはlen関数を使えば、リストのサイズが計算される。
  - len(リスト名)
  - 例：len(a)→リストaのサイズが整数で求まる

- Java/JavaScriptの配列では、配列aの属性として、a.lengthがあるので、この書式を使えば、リストのサイズがどのようなものでも対応できる。
- C/C++にはないので不便だが、以下の書式で計算できる
  - **sizeof**(配列名) / 4 を使う(整数が 4 バイトのとき)

リストへの演算

- リスト + リスト
  - リストの追加になる
  - 例： [1, 2] + [3, 4] ⇒ [1, 2, 3, 4]
- リスト * 整数
  - そのリストが、整数回分繋がったものになる
  - 整数が0以下だと空のリストが作成される
  - 例：["a","b"] * 4 ⇒ ["a","b", "a","b", "a","b", "a","b"]
- 値 in リスト
  - その値が、リストの要素として含まれていればTrue
- 値 not in リスト
  - その値が、リストの要素として含まれていなければTrue

要素の値をグラフで表示

- tkinterモジュールのcreate_rectangleを使う
  - create_rectangle(x1, y1, x2, y2, fill="blue")

リストの走査（総和と平均）

- 総和と平均

```
summ = 0
for n in alist:
    summ += n

summ = sum( alist )  # 組込み関数で用意されている
average = summ / len( alist ) # 平均は、総和を要素の個数で割る
```

最大値・最小値のあるインデックス

- 最大値・最小値のインデックスを変数として持つ

```
maxi, mini = 0, 0
for i, ele  in enumerate( alist ):
   if  alist[ maxi ] < ele : maxi = i
   if  alist[ mini ] > ele : mini = i # miniは不等号が逆
```

maxi →最大値のある要素のインデックス

alist[maxi] →最大値

最大値・最小値そのもの

- 最大値・最小値そのものを変数として持つ

```
maxv, minv = alist[ 0 ], alist[ 0 ]
for ele  in alist:
   if  maxv < ele : maxv = ele
   if minv > ele : minv = ele
```

maxv, minv →最大値・最小値を持つ

max(alist), min(alist)と同じ（組込み関数で用意されている）

検索とカウント

- 特定の値の場所を検索、何個あるか

```
target = int( input( "見つけ出す値: " ) )
alist  = [ 5, 2, 565, 222, 111, 344, 22, 99, 348, 22, 2 ]

index, count = 0, 0
for i, n in enumerate( alist ):
    if  n == target : count+=1; index = i

if  count > 0 :
    print( target  +  " は最後に " + index + \
    " で見つかりました " + count + " 個あります" )
else:
    print( target  +  " はありません。 " )
```

頻度表

- 誕生月の頻度を求める

```
birth  = [ 1, 3, 5, 2, 8, 11, 4, … , 6, 7, 12, 4, 2 ]
month = [ 0 ] * 13

for index in range( len( birth ) ) :
    month[  birth[ index ]  ] += 1

for m in range( 1, len( month ) ):
    print( m  +  "月の誕生日の人は" + month[ m ] +"人いました" )
```

リストの並べ替え

- ランダムにシャッフルする

```
a  =[ index for index in range( 1, 101 ) ]

for n in range( len( a ) * 2 ) :
    index1 = int( random.random( ) * len( a ) )
    index2 = int( radom.random( ) * len( a ) )
    a[  index1 ], a[ index2 ] = a[ index2 ], a[  index1 ]
```

randomモジュールの関数

- **import** randomが必要
- random.shuffle(リスト)…そのリストそのものをシャッフルする
- random.sample(リスト, k=len(リスト))…そのリストを元にシャッフルしたリストを新たに返す

関数への引数としてのリスト

- 関数への引数としてのリストは、コピー渡しではなく、参照渡しで渡される
- 関数内で、引数のリストの内容を書き換えると、本体のリストも変更される
- 例：

```
def clear( nlist ):
    for i in range( len( nlist ) ):  nlist[ i ] = 0
numlist = [ 1, 2, 3, 4 ]
clear( numlist )
print( numlist )  # [0, 0, 0, 0]
```

- 関数で本体のリストを書き換えたくない場合、関数冒頭でcopyをする必要がある
- 例：

```
def noclear( nlist ):
    nlist = nlist.copy( )  # 新しいコピーされたリストで再設定

    for i in range( len( nlist ) ):  nlist[ i ] = 0
```

リストのソート

- ソート(sort)・ソーティング(sorting)・整列…順番に並び替えること
- 直接選択法（straight selection sort）
  - 小さいものを選び、入れ替えをする

```
for i int range( len(a)-1 ) :
    mini = i
    for j in range( i+1, len( a ) ):
        if  a[ mini ] > a[ j ] : mini = j

    a[ i ], a[ mini ] =  a[ mini ], a[ i ]
```

直接選択法でのソーティング

直接選択法のソーティングの様子：

```
19  64   3  30  10  39  20  53
↑i       ↑min
3   64  19  30  10  39  20  53
    ↑i           ↑min
3   10  19  30  64  39  20  53
        ↑i, min
3   10  19  30  64  39  20  53
            ↑i              ↑min
3   10  19  20  64  39  30  53
                ↑i          ↑min
3   10  19  20  30  39  64  53
                    ↑i, min
3   10  19  20  30  39  64  53
                        ↑i  ↑min
```

２分探索法

- ソートしたリストを探索することができる
  - 最初に中間地点の要素と、検索したい値を較べる
  - 値と同じであれば、その位置を返す
  - 値より要素の値が大きければ、前半の部分に対して同じことを行なう
  - 値より要素の値が小さければ、後半の部分に対して同じことを行なう
- ２分探索法では、以下のオーダーで探索を終了することができる。データのサイズをnとすると

$$\log_2 n$$

- たとえば、データの件数が1000件あった場合は、平均でlog2 1000≒10回の探索で目的の値に辿り着く（なければないという判定をする）ことが可能である。

リストの最大値を再帰で

- リストのサイズが１個だったら、
  - その要素が最大値であるとして返す
- リストのサイズが２個以上だったら、
  - サイズを１つ減らして、最大値を求める
  - 求められた最大値と、最後の要素の値を比較して、大きい方を返す

リストのソートを再帰で

- リストのサイズが１だったら、
  - ソートされているものとして終わる
- リストのサイズが２以上だったら、
  - 最後の１つ前までをソートする
  - 最後の要素を、順番的に良い場所に挿入する（この部分は、直接挿入ソートと同じ）

リストを直接挿入法でソートする関数

- 再帰を使わない形での実装

```
def straightInsertion( a ):
    for i in range( 1, len( a ) ):
        comer = a[ i ]
        j = i - 1
        while j >= 0:
            if a[ j ] < comer: break
            a[ j+1 ] = a[ j ]
            j = j - 1
        a[ j+1 ] = comer
```

直接挿入ソートの例

直接挿入法のソーティングの様子：

```
19  64   3   30   10   39   20   53
     ↑target
19→64→3   30   10   39   20   53
          ↑target
3   19   64→30   10   39   20   53
               ↑target
3   19→30→64→10   39   20   53
                    ↑target
3   10   19   30   64→38   20   53
                         ↑target
3   10   19   30→38→64→20   53
                              ↑target
3   10   19   20   30   38   64→53
                                   ↑target
```

スライス

- リスト[最初 : 終わりの次]
  - 部分的なリストが生成される
  - 例：alist[4: 7]
- リスト[: 終わりの次]
  - 最初の要素の位置は、0と仮定される
  - 例：alist[: 7]
- リスト[最初:]
  - 指定された最初の位置から、最後までになる
  - 例：alist[4:]
- リスト[-インデックス]
  - 最後の要素の次から、順次インデックスの値が引かれていく
  - 例：alist[-5], alist[-5:]
- リスト[-インデックス: -インデックス]
  - 例：alist[-5:-2]

飛び飛びのスライス

- リスト[最初: 最後の次: 間隔]
  - 最初から最後までの間で、間隔が空いたサブリストが作成される

- リスト[: : 間隔]

スライスによる代入

- スライスで一定の部分に代入ができる
  - a = [9, 11, 4, 5, 8, 7, 6]
    a[1:3] = [2, 3] # [9, 2, 3, 5, 8, 7, 6]
- 同じサイズでなくても構わない、自動的に拡張・縮小される
  - a = [9, 11, 4, 5, 8, 7, 6]
    a[5:6] = [2, 3] # [9, 11, 4, 5, 8, 2, 3, 6]
  - a = [9, 11, 4, 5, 8, 7, 6]
    a[1:4] = [2, 3] # [9, 2, 3, 8, 7, 6]

リストに適用できる組込み関数

- all(論理値のリスト)…全ての要素がTrueならTrue
- any(論理値のリスト)…Trueの要素が 1 つでもあればTrue
- enumerate(リスト)…インデックスと要素の対を作る
- len(リスト)…リストの長さを整数で返す
- list(リスト)…リストに変換する
- max(リスト)…リストの要素の中の最大値を返す
- min(リスト)…リストの要素の中の最小値を返す
- reversed(リスト)…逆順になった新しいオブジェクトを返す（list関数で変換する必要あり）
- sorted(リスト)…要素が整列された新しいリストを返す
- sum(リスト)…要素の総和を求める

リストの追加・削除・検索

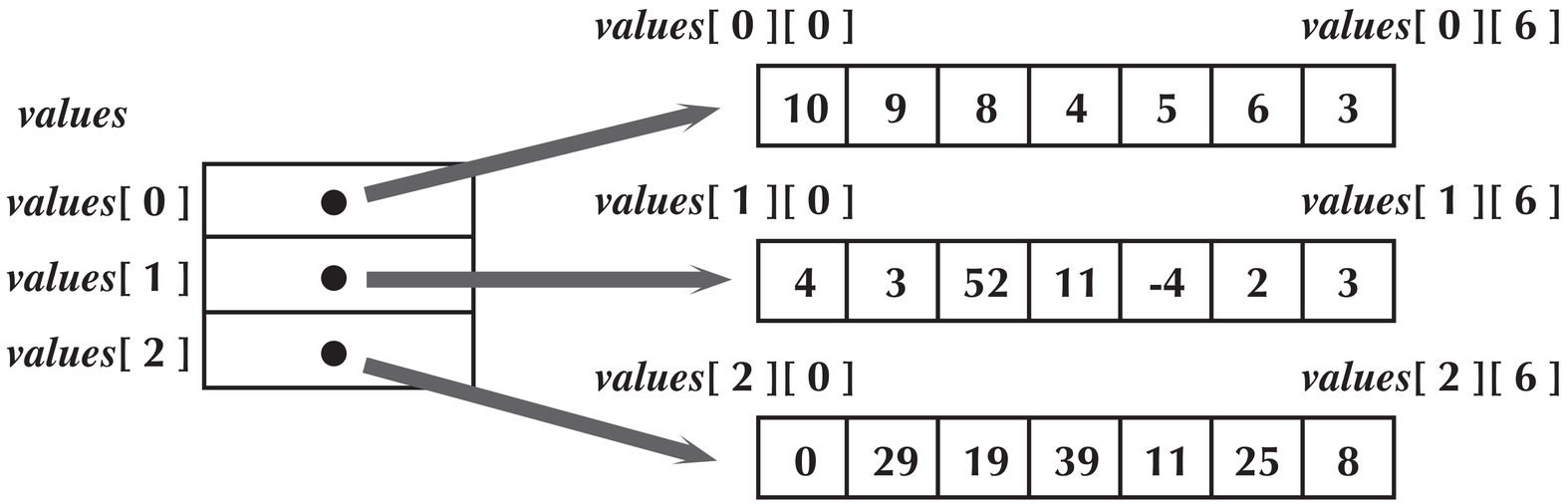
- リスト.append(value)…リストの最後に値を追加する
- リスト.remove(value)…リストから値を削除する（最初に見つかったものだけ）
- リスト.insert(i, value)…リストのi番目の要素の前に値を追加
- リスト.sort()…リストの整列する
- リスト.reverse()…リストを逆順にする
- リスト.index(value)…指定された値が何番目にあるか検索する
- リスト.count(value)…指定された値が何個あるか返す
- リスト.extend(リスト)…リストの後にリストを追加する リスト = リスト + リストと同様の効果
- リスト.copy( )…リストのコピーを返す
- リスト.clear( )…空リストにする

del文による要素の削除

- del s[i]　　… i番目の要素を削除する
- del s[i:j]　　… i番目からj番目の手前までの要素を削除
- del s[i:j:k]　…上記に対してステップkで削除

２次元リスト

- パズルなどのゲームやデータの統計を行なうには、必要



2次元リストの宣言

- 空の２次元リストの例
  - values = [[ ]]
- 領域を確保した２次元リスト
  - values = [[0] * 10] * 10 # ２次元目が共有されてしまう
  - values = [[ ]] * 10　　# １次元目だけを確保する
- 共有されない２次元リストの確保
  - values = [[0 **for** n **in** range(10)] **for** m **in** range(10)]

2次元リストの初期値代入

- 初期値代入の例
  - values = [\
  - [10, 9, 8, 4, 5, 6, 3], \
  - [4, 3, 52, 11, -4, 2, 3], \
  - [0, 29, 19, 39, 11, 25, 8] \
  -]
- 長さが違う２次元リスト
  - values = [[10, 20], [7], [47, 27, 18], [8, 2]]

２次元リストの長さ

- 例題のリスト
  - matrix=[[0] * 5] * 10
- len(リスト名) … 1 次元目のリストの長さが求まる
  - len(matrix) ⇒ 10
- len(リスト名[インデックス]) … ２次元目のリストの長さ
  - len(matrix[2]) ⇒ 5
- ２次元目の長さが異なるような場合にも対処する必要

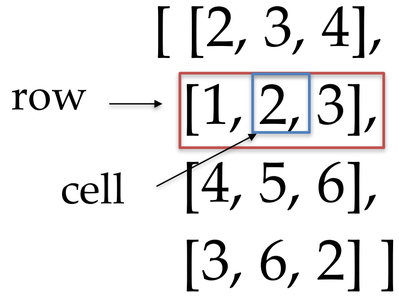
繰返しで２次元リストの要素を参照

- リストの長さが違うとき C/C++/Java的な記述の仕方

```
for i in range( len( values ) ):
    for j in range( len( values[ i ] ) ):
        printf( “%d  ” % ( values[ i ][ j ] ) )
```

- Python流

```
for i, row in enumerate( values ) :
    for j, cell in enumerate( row ):
        values[ i ][ j ] = cell ** 2
```



2次元リストの代入とコピー

- 代入は、共有される
  - a = [[1, 2, 3], [4, 5, 6], [7, 8]]
  - a = b # リストは同一のものとなる
- copyモジュールにある→**import** copyが必要
- copy関数は、浅いコピーなので2次元目は共有されてしまう。
- deepcopy関数は、2次元目もコピーが作られる
- 例：

```
import copy
amat = [ [1, 2, 3], [5, 6, 8], [4, 9, 7] ]
bmat = copy.copy( amat )  # bmat = amat.copy( )
cmat = copy.deepcopy( amat )

cmat[ 0 ][ 0 ] = 100       # amatの要素には何も影響がない
bmat[ 0 ][ 0 ] = 300       # amat[ 0 ][ 0 ]も300になってしまう
```

リストの同一性の判断

- ==は、値が等しいことを示す
  - 例：

    a = [1, 2, 3, 4]
    b = a.copy( )
    a == b # True
- **is**は、同一のオブジェクトを指しているかどうかを示す
  - 例：a **is** b # False
- id( )組込み関数で、オブジェクトの識別子番号（整数）を得ることができる
  - id(a) == id(b) # False

2次元リストの平坦化

- ２次元リストを１次元リストにする方法
  - mat = [[1,2,3], [4,5,6], [7,8]]を
  - [1, 2, 3, 4, 5, 6, 7, 8]にする
- reduce高階関数を使う
  - **import** functools
  - alist = functools.reduce(lambda a, b: a+b, mat)
- sumで、総和の初期値を空リストにする
  - sum(mat, [ ])

1次元リストの2次元リスト化

- 組込み関数などはないので、自分で関数などを作成する必要がある
- 作成例：

```
def nesting( alist, n ):
    result = [ ]
    for i in range( 0, len(alist), n ):
        result.append( alist[i:i+n] )
    return result
```

2次元リストの走査

- max, min, sortedのキー関数を使って、比較対象を選ぶようにする
- 利用例：

```
nestedlist = [ [7, 3], [4, 5], [3, 8], [2, 9] ]
print( sorted( nestedlist, key=lambda x:x[0] ) )
print( max( nestedlist, key=lambda x:x[1] ) )
print( min( nestedlist, key=lambda x:x[0] ) )

sum( x for x in range( 1, 11 ) )
sum( x%2==0 for x in range( 1, 11 ) )
```

オブジェクトのリストの要素

- オブジェクトのリストの確保
  - それだけで、オブジェクトが生成される訳でない
- オブジェクトのリストの要素
  - 繰返しなどを使って、オブジェクトを生成する必要がある
  - 例： **for** i **in** range(len(carray)):

```
carray[ i ] = Canvas( win, width=100, height=100 )

# 要素のオブジェクトを作る
```

オブジェクトのリストのフィールド・メソッド

- オブジェクトリスト[インデックス].フィールド名
- rect[5].width

- オブジェクトリスト[インデックス].メソッド名(実パラメータ)
- fonts[i].getName( )

オブジェクトリストの初期化

- オブジェクトのリストの宣言 = [初期値, ..., 初期値]

- carray = [Color(255, 0, 0), Color(0, 255, 0),
  Color(244, 122, 0), Color(0, 0, 255)]

- carray =[None] * 4
- carray[0] = Color(255, 0, 0)
- carray[1] = Color(0, 255, 0)
- carray[2] = Color(244, 122, 0)
- carray[3] = Color(0, 0, 255)

リストを使った高階関数

- filter(関数, リスト)…関数を適用してTrueの要素だけ抜き出したリストのオブジェクトを作る
- map(関数, リスト)…各要素に関数を適用した新しいリストのオブジェクトを作る
- functools.reduce(関数, リスト)…各要素の前後に関数を適用して、計算をする
  import functoolsが必要

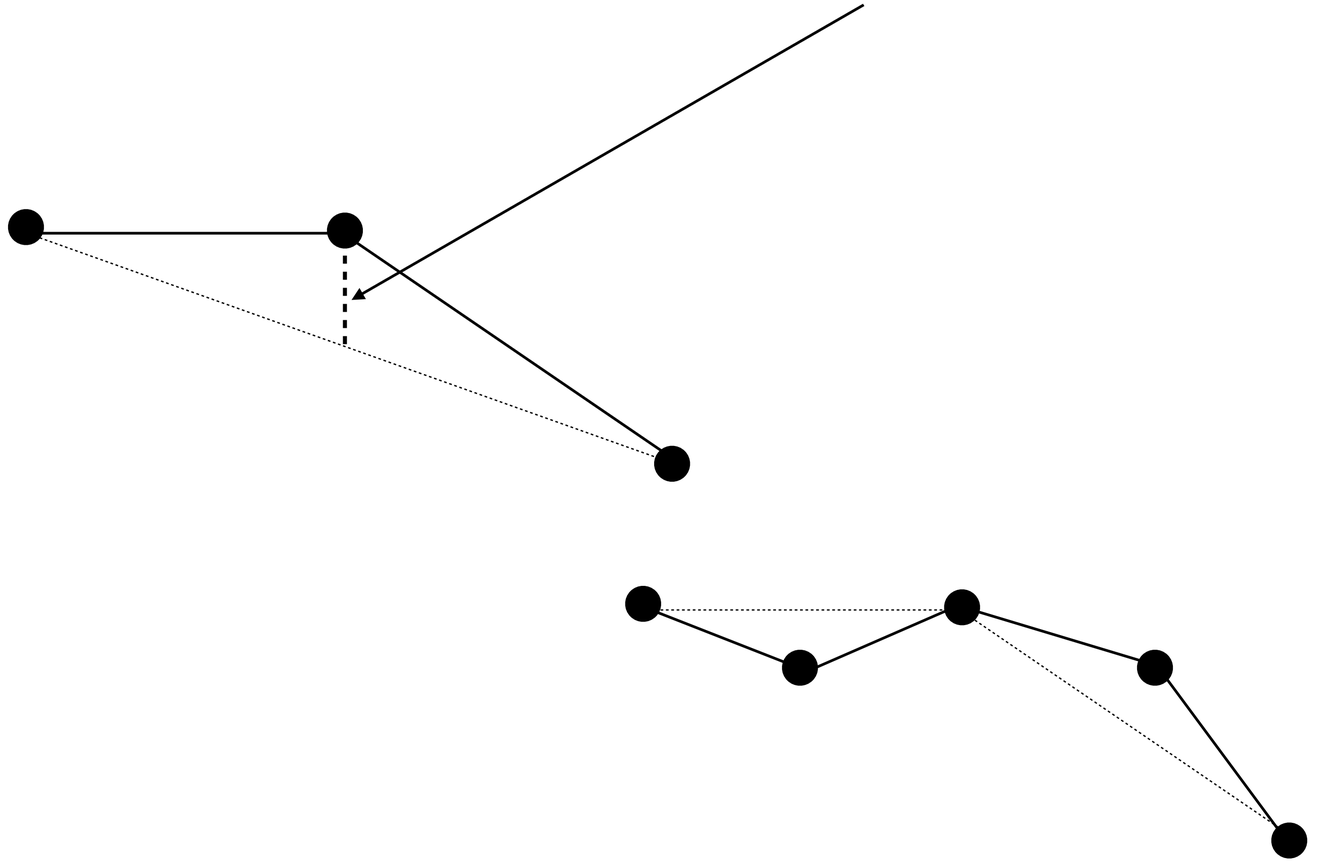
- sorted(リスト, key=functools.cmp_to_key(比較関数))
- max(リスト, key=functools.cmp_to_key(比較関数))
- min(リスト, key=functools.cmp_to_key(比較関数))

リストをfor文で作って実引数にする

- max, min, sum, map, filter, list, tupleなど、シーケンス型（リストやタプルなど）を引数に持つ関数に、for文で生成したリストを実引数として渡すことができる（2つ以上の引数を求めるものは、( )で囲む必要がある）
- 例：
  - max(n **for** n **in** range(1, 11)) → 10
  - sum(n**2 **for** n **in** range(1, 11)) → 385
  - list(map(**lambda** a: str(a), (n **for** n **in** range(1, 6)))) → ['1', '2', '3', '4', '5']
  - list(n**0.5 **for** n **in** (m**2 **for** m **in** range(1, 6))) → [1.0, 2.0, 3.0, 4.0, 5.0]
  - list(filter(**lambda** a: a%3==0, (n**2 **for** n **in** range(1, 10)))) → [9, 36, 81]

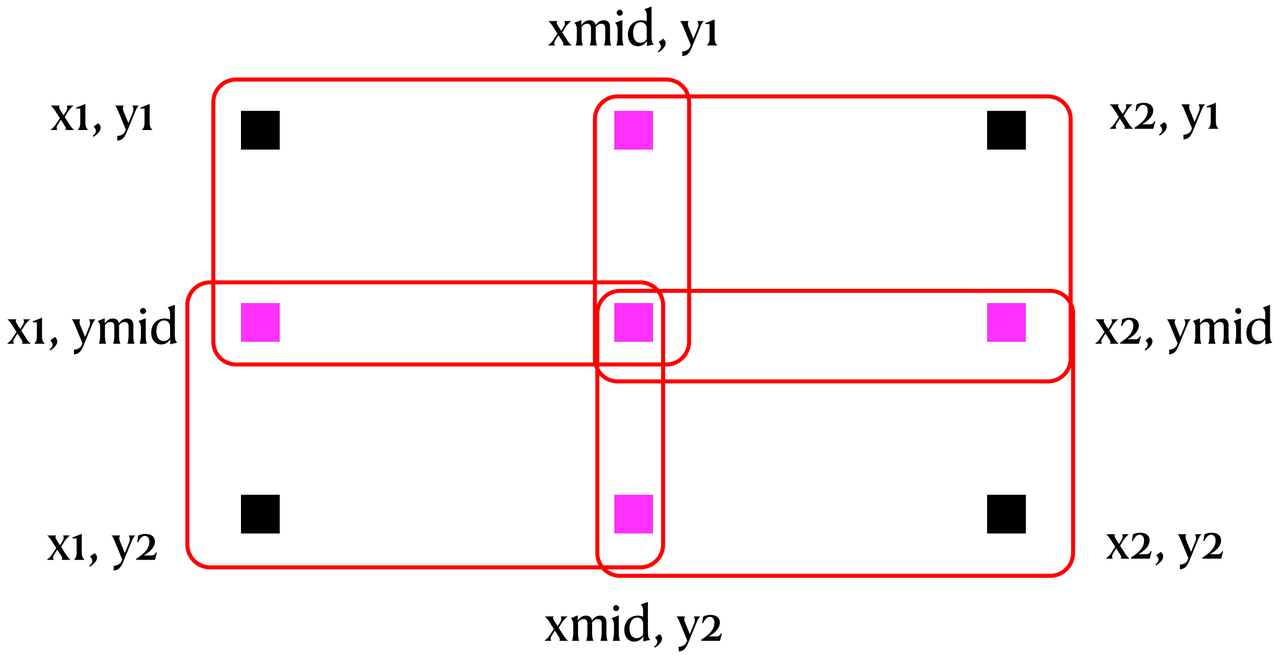
中点変位法

- ２つの点の中点の位置から、一定の割合（乱数を使う）でずらす



２次元上での中点変位法

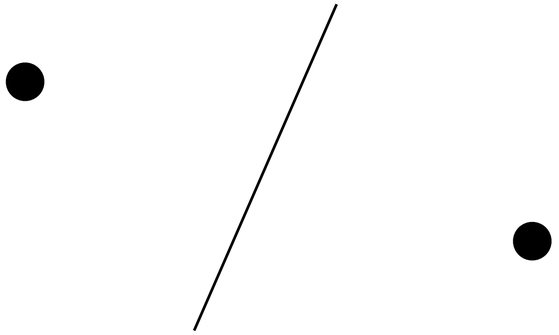
- 四隅の点を与える。そこから中点を５つ求める



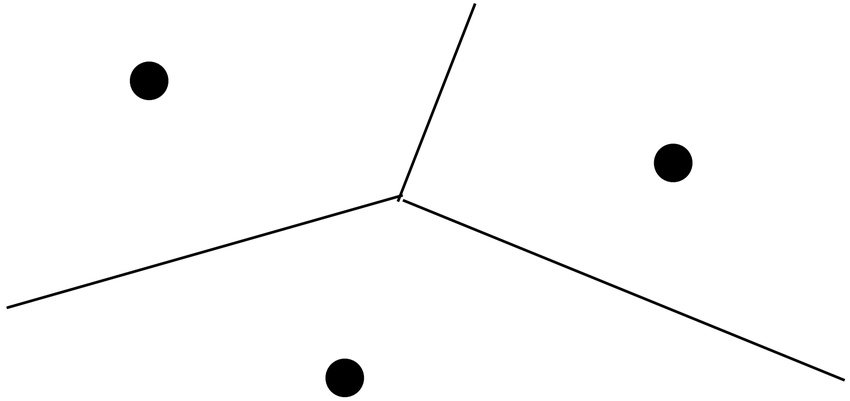
- さらに再帰で４つのパートに分けて、同じことを繰り返す。

ボロノイ図

- 各点について、一番近くにある標本点に属するとする
- ２つだとこんな感じで境界線ができる



- ３つだとこんな感じで境界線ができる



ファイルからの読込み

- f = open(ファイル名, モード)
  - モードは、"r"…読み、"w"…新規作成・上書き
  - "r+"…更新用に読み、"w+"…更新用に書込み、
  - "a"…追加書込み, "b"…バイナリモード
- f.read( )…テキスト全体を読み込む
- f.readlines( )…テキストを改行で区切って 1 行ずつのリストとして読み込む
- f.readline( )…テキストを 1 行ずつ読み込む
- f.close( )…読込み終了

ファイルへの書込み

- f.seek(バイトのオフセット)
  f.seek(バイトのオフセット, 起点)
- ファイル内の任意の場所に書込み（読込み）位置を移動する
- 起点は、0…ファイルの先頭、1…現在の位置、2…ファイルの最後、起点が1の場合は、オフセットの値は負の値でも良い、2の場合は通常負の値
- f.write(データ)
- ファイルの現在の位置に、データを書き込む

CSVファイルの読み書き

- csvのモジュールだと１次元リストとCSVファイルの１行データを直接読み書きできる。文字列データで、途中に空白が入ってもOKなのが特徴
- 読込み
  - **import** csv
  - matrix = [ ]
  - f = open(filename, "r")
  - reader = csv.reader(f)
  - **for** row **in** reader: matrix.append(row)
  - f.close( )
- 書込み
  - f = open(filename, "w")
  - writer = csv.writer(f)
  - **for** row **in** matrix: writer.writerow(row)
  - f.close( )

インターネットのwebページの読込み

- urllibライブラリとsslライブラリを使う
- # インターネットへのアクセス

```
from urllib.request import *
import ssl
gcontext = ssl.SSLContext()
url = urlopen( "https://www.keio.ac.jp",  context=gcontext )
text = url.read().decode( "utf-8" )
for n, line in enumerate( text.split( "\n" ), 1 ):
    print( "%04d: %s" % ( n, line ) )
url.close()
```

Beautiful Soapを用いる

- インストール
  - pip3 install bs4 (Windowsの場合はpipで大丈夫)
- 利用例：

```
import urllib.request
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

responce = urllib.request.urlopen( "https://www.sfc.keio.ac.jp")
content = responce.read( )

from bs4 import BeautifulSoup

webtext = content.decode( "utf-8" )

soup = BeautifulSoup( webtext, "html.parser" )

print( soup.title )

items = soup.find_all( "h2" )

for item in items:

    print( item )
```