2023/7/5

担当：高野晃治

第4章　リストとタプル

目次

EP1.プログラム構文

1.リスト

2.タプル

3.**enumerate**

EP2.理論

4.メモリ上でのリストの表現

5.参照の値引き渡し

6.**リスト内包表記**

説明(EP1･2)：計15分

EP3.課題演習

7.課題：コッホ曲線

8.できればやろう課題：素数の判定

課題演習：計15分

本章でのポイント

□通常の変数とリストとタプルの違いを理解する。

□変数やリストでのメモリ管理を把握する。

□リスト内包表記での書き方と注意点を理解する。

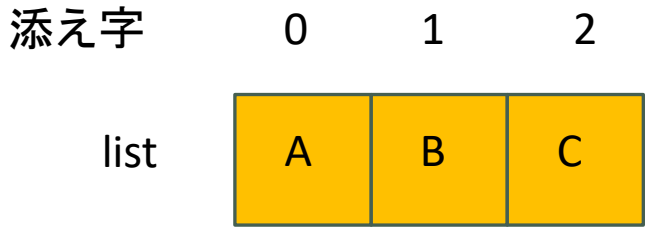
□課題でのリストの使い方，リスト内包表記を理解する。

※本章で使用するプログラム（教科書内プログラム，[穴埋め]課題，[解答]課題)・説明資料は
GoogleDrive内にて共有してある。
各自のドライブにダウンロードして進めること（共有フォルダ内で使用しないこと）
各種プログラムには番号が付いている。
演習説明内では，★が付いているもののみ実行することとする。

1.リスト

リスト
　・・・データ構造を表現するもの。C言語でいう「配列」

＜特徴＞
・複数のデータの管理が可能
・0から始まる添え字によってデータを呼び出す



参考：https://aiacademy.jp/media/?p=1819

1.リスト

＜書き方＞　※格納するlistの変数に型の定義は必要ない。

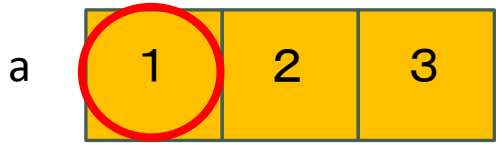
・[ ]の中に,(カンマ)で区切って表記する。

・数字（整数・実数）を入れるときはそのままOK

★

```
#①　リストの作成
a=[1,2,3]
```

Cでは　int a[3]=[1,2,3];



・呼び出すときは添え字を指定する

```
#②　添え字を使って値を取得する
#なおa=[1,2,3]
a[0]
```

```
1
```

1.リスト

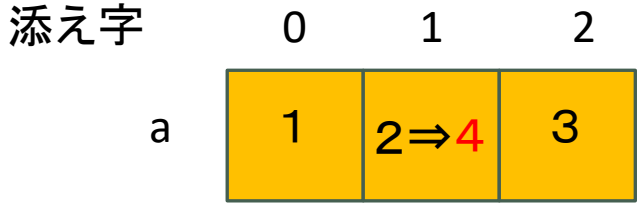
＜書き方＞

・定義後，代入することが可能

★

```
#③　添え字を使って値を代入する
#なおa=[1,2,3]
a[1]=4
a
```

```
[1, 4, 3]
```



1.リスト

＜書き方＞

・リストの入れ子

★

```
#④-1　リストの入れ子
a=[[1,2],[3,4],5]
a[0]
```

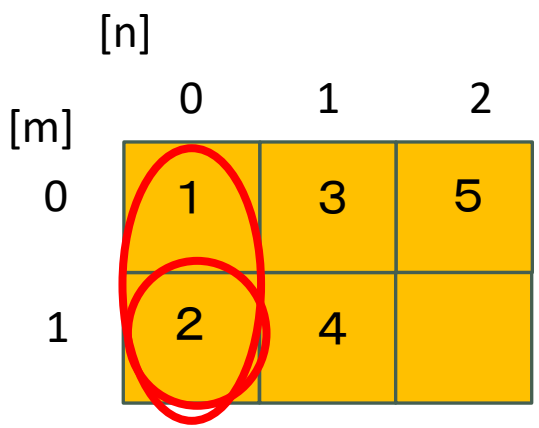
```
[1, 2]
```

★

```
#④-2　リストの入れ子　添え字を複数指定する
a[0][1]
```

```
2
```

a[n][m]



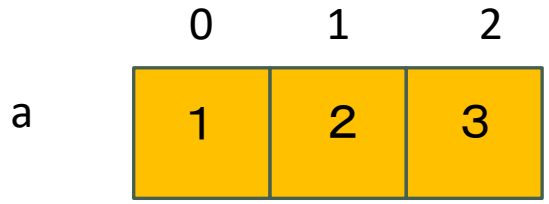
1.リスト

＜書き方＞

・リストの長さ

```
#⑤　リストの長さ（要素数の指定）
a=[1,2,3]
len(a)
```

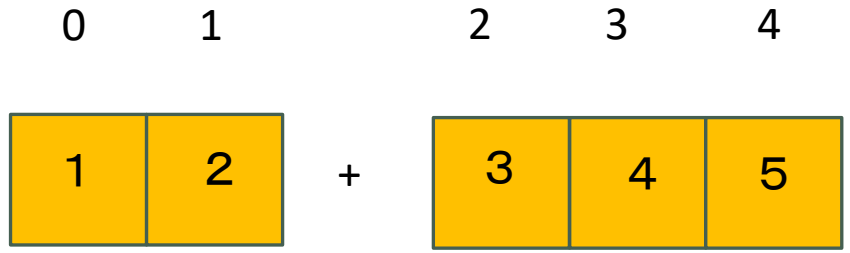
```
3
```



・リストの結合

```
#⑥　2つのリストの結合
[1,2]+[3,4,5]
```

```
[1, 2, 3, 4, 5]
```



1.リスト

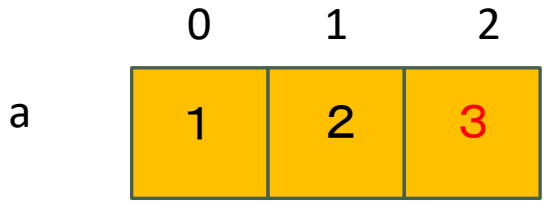
＜書き方＞

・リストの要素追加

追加前

追加後

```
#⑦-1　要素の追加
a=[1,2]
print(a)
a.append(3)
a
```



1.リスト

＜書き方＞

・リストの要素追加　注意！

```
#⑦-2　appendの注意点　準備
a=[]
b=[1,2]
a.append(b)
a.append(b)
print(a)
```

```
[[1, 2], [1, 2]]
```

★

```
#⑦-3 appendの注意点　代入
a=[]
b=[1,2]
a.append(b)
a.append(b)
a[0][0]=4
print(a)
```

```
[[4, 2], [4, 2]]
```

a[ ][ ]

	0	1
0	1	1
1	2	2

1.リスト

＜書き方＞

・リストの真偽

```
#⑧-1　リストに要素が含まれているかの確認 in　（真の時）
a=[1,2,3]
1 in a
```

```
True
```

```
#⑧-2　リストに要素が含まれているかの確認 in　（偽の時）
a=[1,2,3]
4 in a
```

```
False
```

・リストの要素すべてについて処理

```
#⑧-3　リストの要素を順番に取り出す＋すべての要素について処理する　for,in
a=["A","B","C"]
for i in a:
  print(i)
```

```
A
B
C
```

1.リスト

＜書き方まとめ＞

☑ [ ]の中に,(カンマ)で区切って表記する。

☑格納するlistの変数は型の定義は必要ない。

☑数字（整数・実数）を入れるときはそのままOK

☑文字を代入するときは” ”で挟む。

☑いろいろな種類を混ぜてもOK

☑添え字を使って取得や代入を行う（0から始まる）

☑リストを入れ子にすることが出来る

☑長さの取得は　len(リスト名)

☑ +　でリストを結合できる

☑要素追加は　リスト名.append(追加要素)

Next>>>

2.タプル

2.タプル

タプル
　･･･複数の値の組を表現するデータ構造。
　　基本は , (カンマ)で区切るが，紛らわしいので(　)でくくる。

```
#⑨　タプルの定義
a=1,2,3
a
```

```
(1, 2, 3)
```

2.タプル

・リストと同じ動作 ⇒長さの取得，要素の取得，タプル同士の結合

```
#⑩-1　リストと同じ動作＜タプルで行うlenの長さ取得，添え字を使った要素の取得
a=1,2,3
print(a[0])
len(a)
```

```
1
3
```

```
#⑩-2　リストと同じ動作＜タプルの結合
(1,2)+(3,4)
```

```
(1, 2, 3, 4)
```

2.タプル

・タプルの修正

⇒タプルは修正することが出来ない

★

```
#①　タプルの修正
a=(1,2,3)
a[1]=4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-dd8ad83a6b3b> in <cell line: 3>()
      1 #①　タプルの修正　※エラーになることを確認せよ。
      2 a=(1,2,3)
----> 3 a[1]=4

TypeError: 'tuple' object does not support item assignment
```

SEARCH STACK OVERFLOW

2.タプル

・タプルの利用用途

⇒複数の値に対して処理を行える

```
#⑫-1　タプルの利用用途　関数で複数の値を返す
def func():
  return 1,2

func()
```

```
(1, 2)
```

```
#⑫-2　タプルの利用用途　複数の変数の初期化
a,b=1,2
print(a)
print(b)
```

```
1
2
```

2.タプル

・タプルでの値交換

★

```
#⑬　変数の値の交換
a=1,2,3
b=4,5,6
print(a,b)#交換前
a,b=b,a
print(a,b)#交換後
```

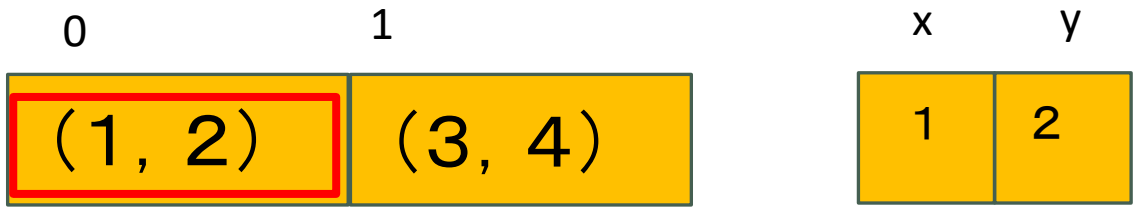
```
(1, 2, 3) (4, 5, 6)
```

⇒aの中身がbへ
bの中身がaへ
代入される

・タプルのリスト

```
#⑭　タプルのリスト
a=[(1,2),(3,4)]
x,y=a[0]#0番目の要素をx,yにそれぞれ代入
print(x)
print(y)
```

```
1
2
```



2.タプル

＜まとめ＞

☑ ,(カンマ)で区切って表記する。
　紛らわしいので(　)を付けることもある。

☑リスト同様，添え字を使って取得や代入，lenでの長さ取得，　＋でのリストの結合が行える。

☑リストと違い，タプルの要素は修正できない。

☑複数の値に対しての処理に用いられる。

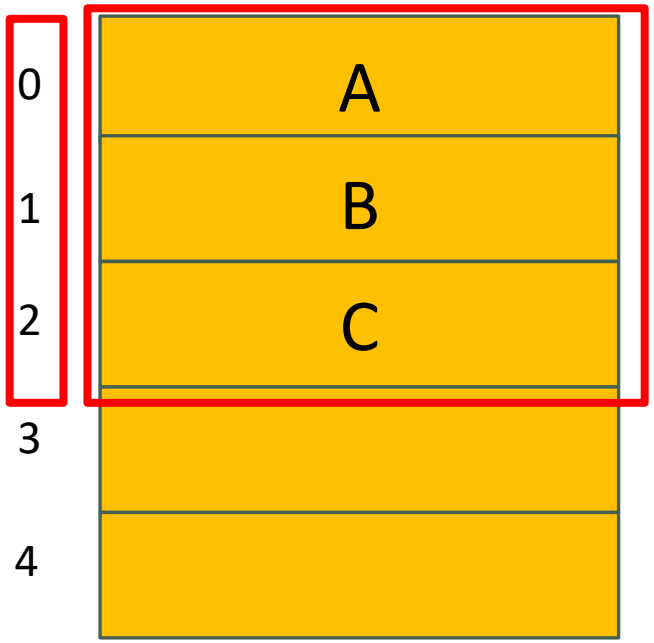
Next>>>
3.enumerate

3.enumerate

enumerate

・・・「要素の値」と「その要素がリストの何番目にあるか」
　の両方の情報を取得する。

```
#⑮　enumerateの使い方　番号と要素の値の取得
a=["A","B","C"]
for i, x in enumerate(a):
  print(i,x)
```



Next>>>EP2
4.メモリ上でのリストの表現

4.メモリ上でのリストの表現

メモリ上での変数の扱い方について

リストを表すラベル⇒**「リストの先頭位置」**

※aは「リストの先頭そのもの」ではない

```
a=[1,2,3]
```

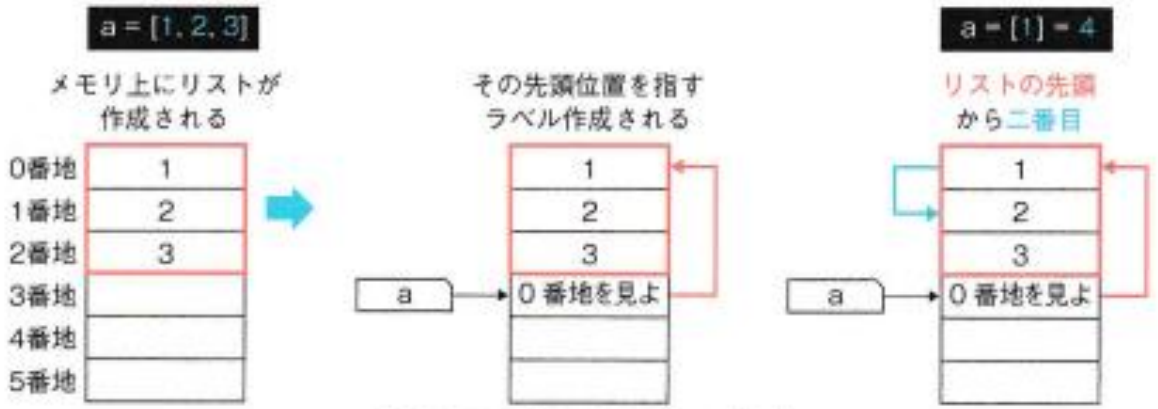


図 4.1　リストのメモリ上での表現

①メモリ上に代入

↓

②代入された値の先頭位置のラベルが作成される

```
a[1]
```

4.メモリ上でのリストの表現

コピーしたときのメモリ上での仕組み

・通常の変数のコピーの時

例）値10を指す変数aをbにコピー後，bに20を代入

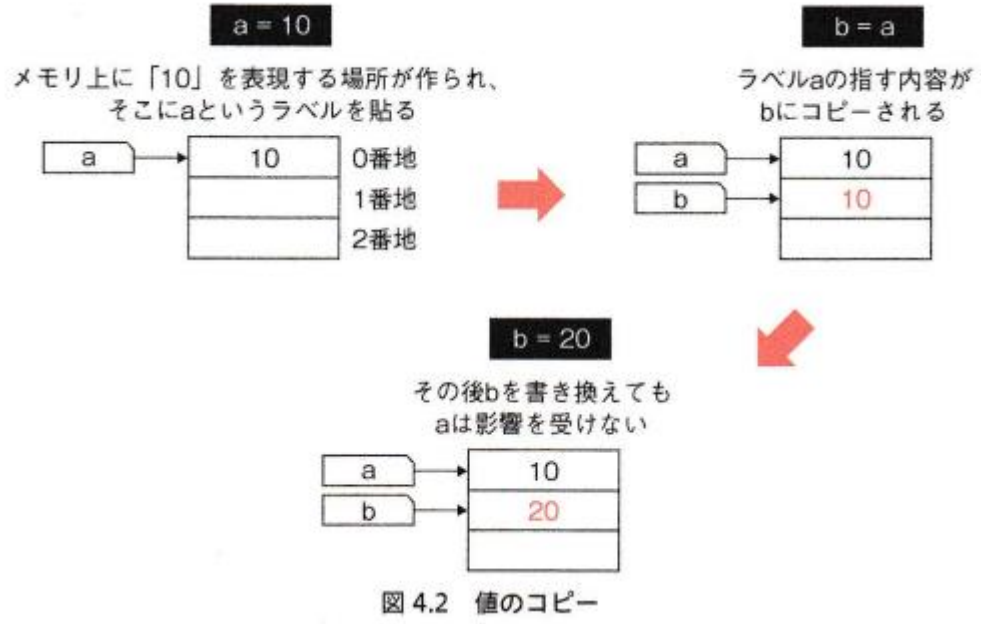


図 4.2　値のコピー

①メモリ上に「10」の場所が作られ，
そこにaというラベル

↓

②aの指す値をコピーし，
そこにbというラベル

↓

③bの指す値を20に書き換え

```
#⑯　普通の変数のコピー
a=10
b=a
b=20
print(a)
print(b)
```

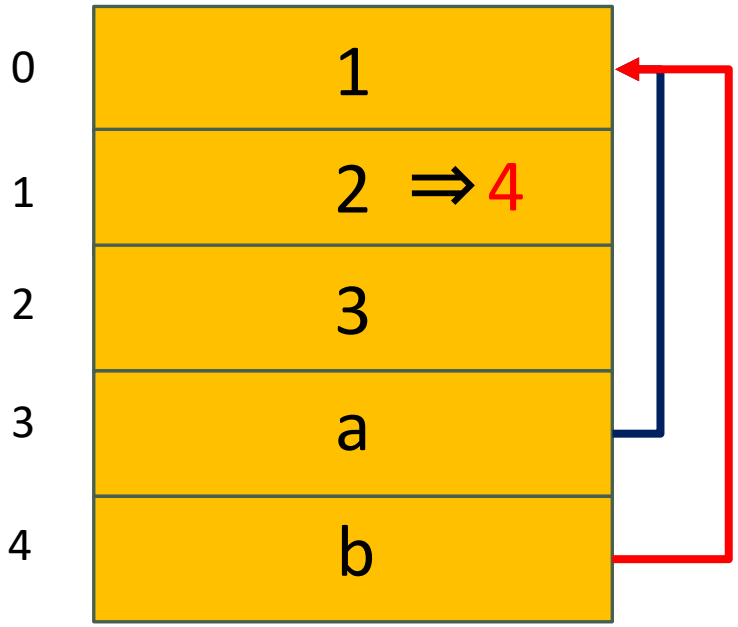
```
10
20
```

4.メモリ上でのリストの表現

コピーしたときのメモリ上での仕組み

・リストのコピーの時

例）a=[1,2,3]を作成し，それをbにコピー後，b[1]=4を代入



①a=[1,2,3]がメモリ上に代入

②リスト名”a”がメモリ上に含まれる

⇒aはあくまで「ラベル」

③bの作成（aをコピーする）

⇒リスト名”b”がメモリ上に含まれる

④b[1]=4を実施

Q. TIME

a,bそれぞれの出力結果はどうなる？

4.メモリ上でのリストの表現

コピーしたときのメモリ上での仕組み

・リストのコピーの時

例）a=[1,2,3]を作成し，それをbにコピー後，b[1]=4を代入

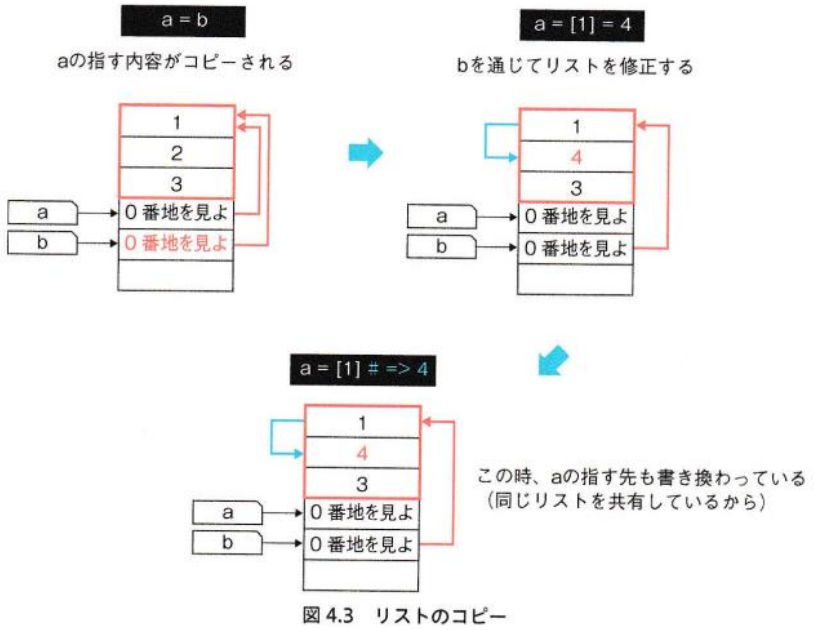
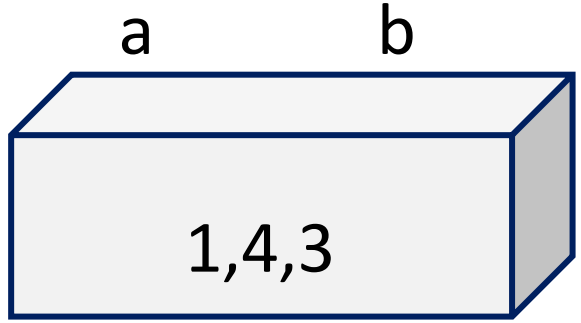


図 4.3　リストのコピー

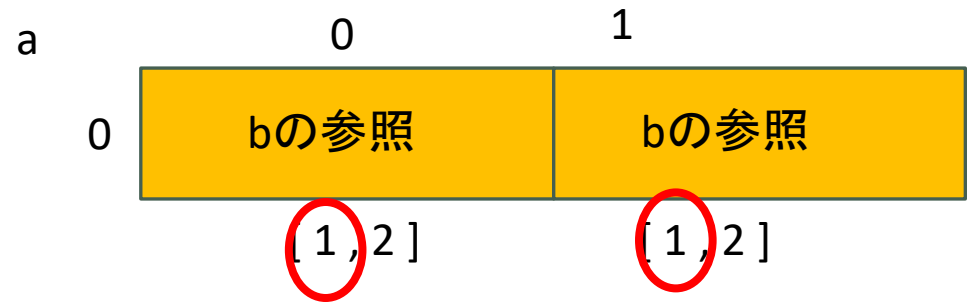



```
#⑰-1　リストのコピー
a=[1,2,3]
b=a
b[1]=4
print(a)
print(b)
```

4.メモリ上でのリストの表現

コピーしたときのメモリ上での仕組み

※appendの時はどうなっていたのか？



1.リスト

＜書き方＞

・リストの要素追加　注意！

```
#⑦-2　appendの注意点　準備
a=[]
b=[1,2]
a.append(b)
a.append(b)
print(a)
```

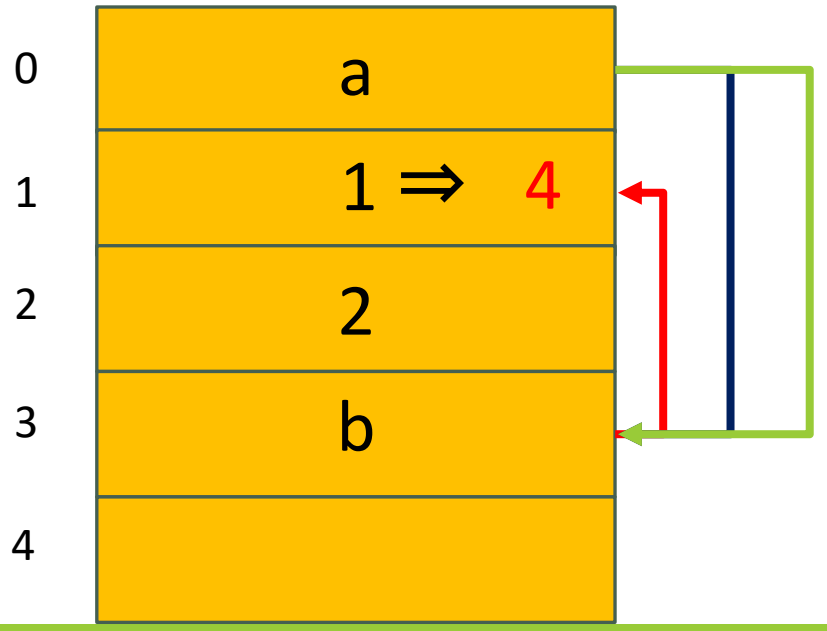
```
[[1, 2], [1, 2]]
```

★

```
#⑦-3 appendの注意点　代入
a=[]
b=[1,2]
a.append(b)
a.append(b)
a[0][0]=4
print(a)
```

a[ ][ ]

	0	1
0	1	1
1	2	2



4.メモリ上でのリストの表現

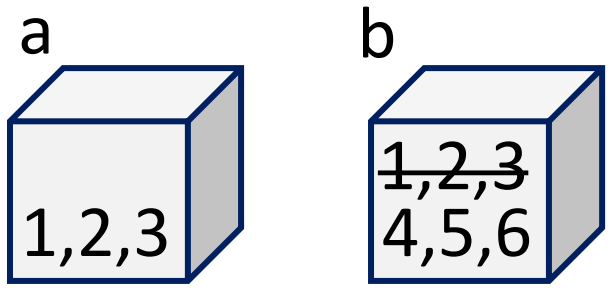
コピーしたときのメモリ上での仕組み

・リストのコピーの時

⇒異なるリストを代入すると別として扱われる。

★

```
#⑦-2　リストのコピー＜異なるリストを代入する
a=[1,2,3]
b=a
b=[4,5,6]
print(a)
print(b)
```



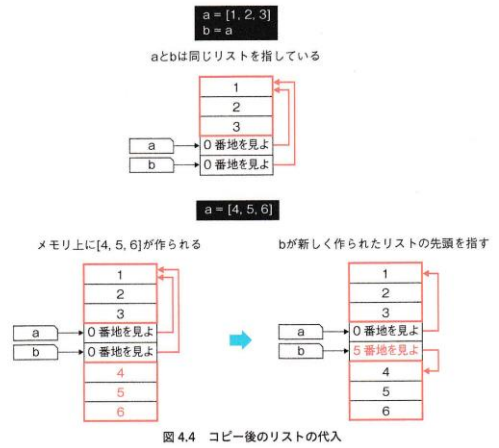
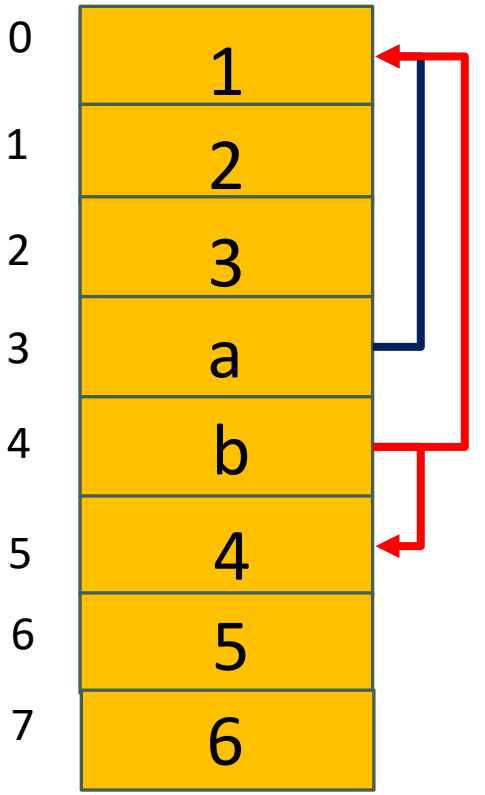


図 4.4　コピー後のリストの代入



4.メモリ上でのリストの表現

※補足（教科書には載っていない内容）

・リストのコピーの時，**全く干渉しないコピーを作りたい！**

copy関数を使う！

<書式>　**コピー先＝copy.copy(コピー元)**

※copyには浅いコピーと深いコピーがある…！？
この方法は浅いコピー，深いコピーは多重リストの時に使う！

参考：https://www.headboost.jp/python-list-copy/

```
#⑰-3ex　リストのコピー＜⑰-1のような場合
import copy

a=[1,2,3]
b=copy.copy(a)
b[1]=4
print(a)
print(b)
```

```
[1, 2, 3]
[1, 4, 3]
```

Next>>>
5.参照の値渡し

5.参照の値渡し

おさらい！

前回：第3章で「関数」

今回：第4章で「リスト」

⇒関数の引数にリストを使用できる！

※注意

関数の引数⇒ブロック内だけで有効なローカル変数

グローバル変数と同じ名前をつけても別の変数

5.参照の値渡し

例）

```
#⑱　参照の値渡し例1
def func(a):
  a=2
```

```
#⑱　参照の値渡し例1
a=1
func(a)
print(a)
```

```
1
```

⇓

関数funcの**a**

通常の　**a**　は**別の変数**

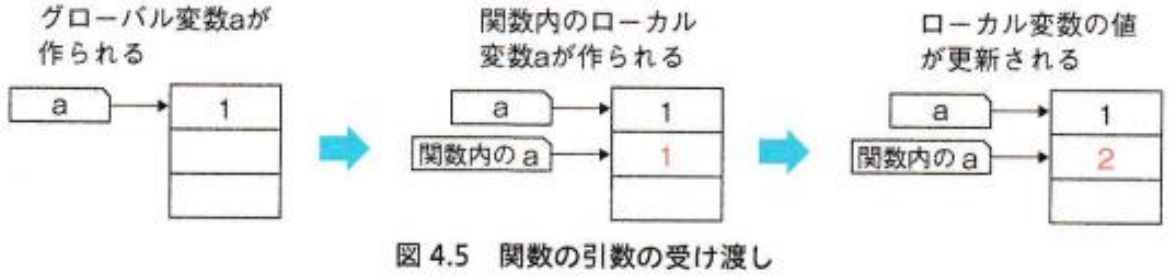


図 4.5　関数の引数の受け渡し

関数内で値を変更しても

外部の変数には影響を与えない

⇒このような情報の渡し方が

値渡し(call by value)

※同じ変数名はバグの原因となるのでなるべくやめよう。

5.参照の値渡し

関数の引数がリストなら？

⇒4.メモリ上でのリストの表現　と同様，コピー時と全く同じ

```
#⑲-1　参照の値渡し例2 関数の引数をリストにする
def func(a):
  a[1]=4

a=[1,2,3]
func(a)
print(a)
```

```
[1, 4, 3]
```

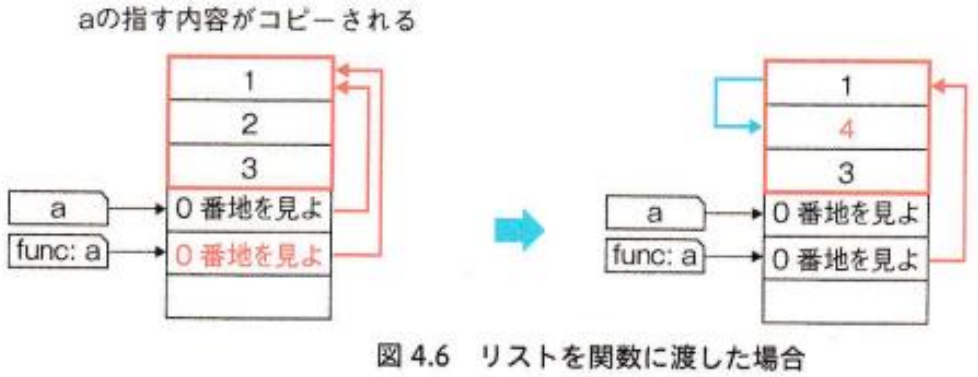


図 4.6　リストを関数に渡した場合

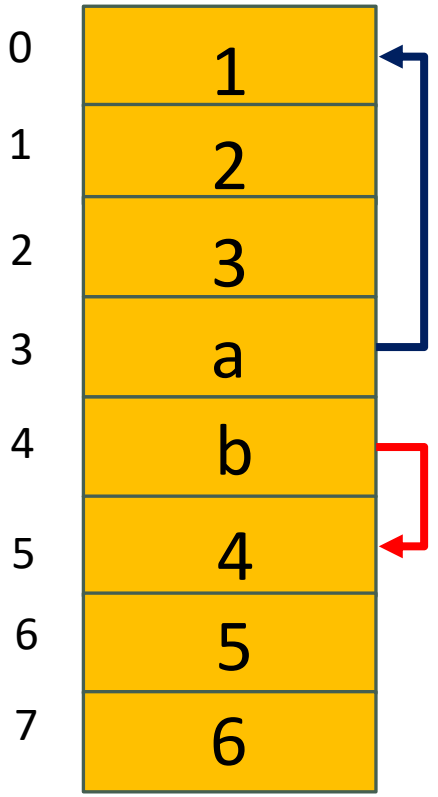
func関数で定義されたa

グローバル変数のリストa　は**同じところを指す**

5.参照の値渡し

関数の引数がリストなら？

⇒4.メモリ上でのリストの表現　と同様，コピー時と全く同じ



受け取ったリストに，別の値を代入することで関数ローカルだけで有効になる。

```
#⑲-2　参照の値渡し例2 引数として受け取ったリストに新たに代入する
def func(b):
  b=[4,5,6]

a=[1,2,3]
func(a)
print(a)
```

```
[1, 2, 3]
```

5.参照の値渡し

＜まとめ＞

関数の引数に値をコピーして渡す方法･･･**値渡し**

リストを指す変数⇒**リストの値そのもの　×**

リストの先頭の場所を指す･･･参照

↓

リストを指す変数　を　**関数に渡す**　ということ

⇒「**リストの先頭の場所**」という「**参照**」の「**値**」をコピーして渡す

･･･**参照の値渡し**

Next>>>

6.リスト内包表記

6.リスト内包表記

リスト内包表記とは　～前提～

そもそもこのリスト内包表記の有用性とは？

例）業務を行う

まとまったデータを受け取り，
それに何か処理をして，
まとまったデータを返す

↓リストも同様

例）リストaの値を2倍にした
リストbを作成する

図 4.8　大量のレポートを採点する教員

6.リスト内包表記

リスト内包表記とは　～リスト内包表記にしてみる～

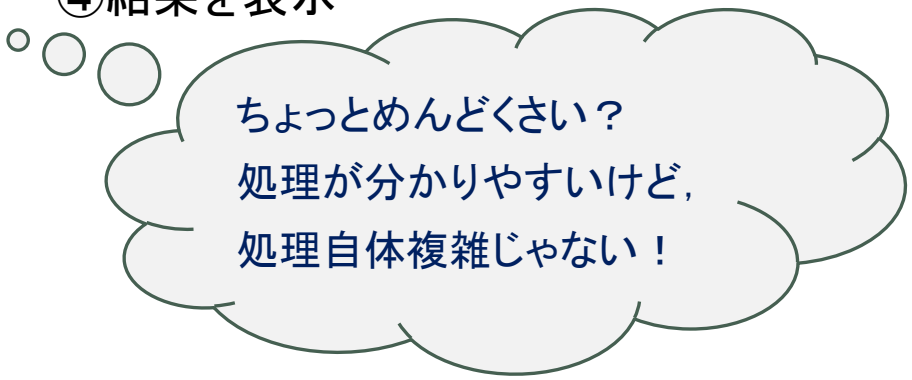
例）整数のリストの要素を2倍にしたリスト

```
#⑳-1　整数のリストを受け取って，要素を二倍にする　通常の例
source=[0,1,2]
result =[]
for i in source:
  result.append(i*2)

print(result)
```

```
[0, 2, 4]
```

①元のリスト(source)を作成
②空の結果用リスト(result)を作成
③for文でsourceの最後までループさせる
④結果を表示



6.リスト内包表記

リスト内包表記とは　～リスト内包表記にしてみる～

例）整数のリストの要素を2倍にしたリスト

```
#⑳-2　整数のリストを受け取って，要素を二倍にする　リスト内包表記の例
source=[0,1,2]
#リスト内包表記の書き方　[新しいリストの要素 for 元のリストの要素 in 元のリスト]
result=[2*i for i in source]
print(result)
```

```
[0, 2, 4]
```

①元のリスト(source)を作成
②結果用リスト(result)に
　for文でsourceの最後まで2倍させた値
　を代入するループを行う。
③結果の表示

簡潔なプログラムになった！

6.リスト内包表記

リスト内包表記とは

リスト内包表記の構文

[新しいリストの要素␣for␣元のリストの要素␣in␣元のリスト]

前ページの該当箇所

```
result=[2*i for i in source]
```

理解するコツ　後ろから読むこと！

⇒リストsource に含まれる

それぞれの要素 i について

2*i を要素とするような新しいリストを作る

6.リスト内包表記

リスト内包表記とは

リスト内包表記の構文

[新しいリストの要素␣for␣元のリストの要素␣in␣元のリスト]

前ページの該当箇所

```
result=[2*i for i in source]
```

```
#⑳-2　リスト内包表記　別の書き方その1
result = [2*i for i in [0,1,2]]
```

リスト名でなくてもOK
直接，要素を入れてよい

```
#⑳-2　リスト内包表記　別の書き方その2
result = [2*i for i in range(3)]
```

range()を使って
規則的な数値の入力でもOK

Next>>>EP3

課題にチャレンジ！

7.課題：コッホ曲線

Introduction コッホ曲線とは？

以下のような手続きで描かれる曲線のこと。

①線分を用意する。
②線分を3等分する。
③中央の線分を正三角形の形に盛り上げる。

④②～③を一定回数（N）繰り返す。

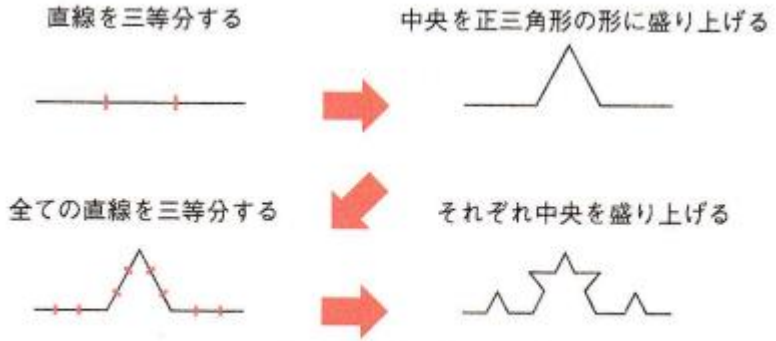


図 4.11　コッホ曲線の作り方

通常は再帰を使って描画するが，今回はリストとタプルで描画しよう！

7.課題：コッホ曲線

Introduction コッホ曲線とは？

コッホ曲線　⇒　すべてつながった線分

↓

ある点から次の点へのベクトルの集合　と考えられる！

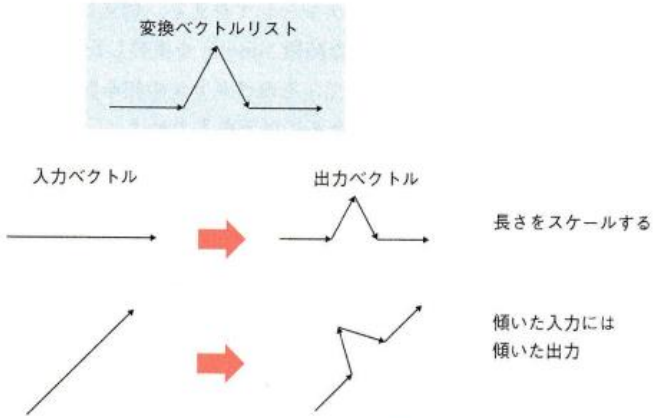


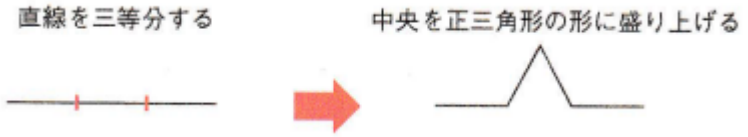
図 4.12　ベクトルの変換

あるベクトルが与えられたとき，

どのように変換したいかを表現したベクトル　の**リスト**を与えて変換する

7.課題：コッホ曲線

コッホ曲線　を描画するのに必要なものは？



①x方向に伸びた長さ1の線分がある。⇒2次元ベクトル(1,0)

②4つのベクトルに変換したい！⇒(1/3 , 0) (1/6 , V3/6) (1/6 , -V3/6) (1/3 , 0)

※**最初に与えたベクトルの始点・終点間の長さ＝変換で与えたベクトルの始点・終点間の長さ**

↓

いちいち変換はだるい。

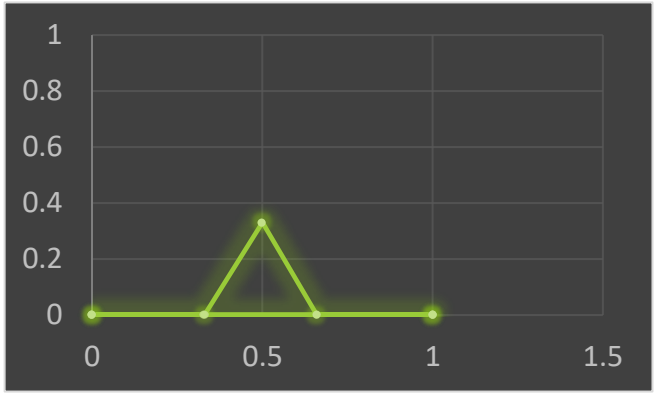
↓

長さを自動調節したい　＋　傾いたベクトルに対応させたい

↓

必要なこと

・長さの自動調節　　・傾きを考慮　　・描画



7.課題：コッホ曲線

Let's Try!

①まずは必要なライブラリを読み込む。

```
#課題　コッホ曲線の描画
from math import sqrt#mathライブラリの平方根sqrt
from PIL import Image,ImageDraw#描画に必要なライブラリ
```

②長さの自動調節を行う関数lengthを作る。

```
def length(a):#長さの自動調節を行う関数length
  x,y = 0,0
  for (dx,dy) in a:
    x += dx
    y += dy
  return sqrt(x**2 + y**2)
```

ここまで実行出来たら・・・

この入力で正しく出力できているか確認しよう！

```
#確認用①
a = [(1,0),(0,1)]
length(a)
```

```
1.4142135623730951
```

7.課題：コッホ曲線

Let's Try!

③入力ベクトルを変換ベクトルリストに基づいて変換する。
→「ベクトルをタプルとして与えられた時，変換ベクトルリストに従って
　タプルのリストに変換する関数」
＜処理の流れ＞
(1)入力ベクトルの長さと変換ベクトルリストの長さの比(scale)を求める。
(2)入力ベクトルの傾き角度θのsinとcosの値を求める。
(3)変換ベクトルリストに含まれるベクトルそれぞれについて，
　scale倍してθだけ傾けたものをリストに追加する。

```
def convert(a,b):
  ax,ay = a
  alen =sqrt(ax**2+ay**2)
  c=ax/alen
  s=ay/alen
  scale = alen/length(b)

  #リスト内包表記不使用の場合
  r=[]
  for(bx,by) in b:
    bx*=scale          #(a)
    by*=scale          #(a)
    nx=c*bx-s*by       #(b)
    ny=s*bx+c*by       #(b)
    r.append((nx,ny))
  return r
```

7.課題：コッホ曲線

Let's Try!

③入力ベクトルを変換ベクトルリストに基づいて変換する。
→「ベクトルをタプルとして与えられた時，変換ベクトルリストに従って
　タプルのリストに変換する関数」
＜処理の流れ＞
(1)入力ベクトルの長さと変換ベクトルリストの長さの比(scale)を求める。
(2)入力ベクトルの傾き角度θのsinとcosの値を求める。
(3)変換ベクトルリストに含まれるベクトルそれぞれについて，
　scale倍してθだけ傾けたものをリストに追加する。

ここまで実行出来たら・・・

この入力で正しく出力できているか確認しよう！

```
#確認用②
a=(0,1)
b=[(1,1),(1,-1)]
convert(a,b)
```

```
[(-0.5, 0.5), (0.5, 0.5)]
```

7.課題：コッホ曲線

Let's Try!

③入力ベクトルを変換ベクトルリストに基づいて変換する。

Q1. □の部分をリスト内包表記に書き換えるとどうなる？

```
#Q1.リスト内包表記にするとどうなる？
#b=[〇〇〇〇 for (x,y) in b]   #(a)
#b=[〇〇〇〇 for (x,y) in b]   #(b)
#return b
```

```
def convert(a,b):
  ax,ay = a
  alen =sqrt(ax**2+ay**2)
  c=ax/alen
  s=ay/alen
  scale = alen/length(b)

  #リスト内包表記不使用の場合
  r=[]
  for(bx,by) in b:
    bx*=scale          #(a)
    by*=scale          #(a)
    nx=c*bx-s*by       #(b)
    ny=s*bx+c*by       #(b)
    r.append((nx,ny))
  return r
```

7.課題：コッホ曲線

Let's Try!

④ ③を使った，「タプルのリスト」が与えられた時のそれぞれの結果をまとめたリストを作る関数を作る。

```
def apply(a,b):
  r=[]
  for i in a:
    r += convert(i,b)
  return r
```

ここまで実行出来たら・・・

この入力で正しく出力できているか
確認しよう！

```
#確認用③
a=[(1,0),(0,-1)]
b=[(1,1),(1,-1)]
apply(a,b)
```

```
[(0.5, 0.5), (0.5, -0.5), (0.5, -0.5), (-0.5, -0.5)]
```

7.課題：コッホ曲線

Let's Try!

⑤ベクトルのリストを使って，線を描画してみる。

```
def draw_line(draw,a,size):
  x1,y1=0,0
  for(dx,dy) in a:
    x2=x1+dx
    y2=y1+dy
    draw.line((x1,size/2-y1,x2,size/2-y2),fill=(255,255,255))
    x1,y1=x2,y2
```

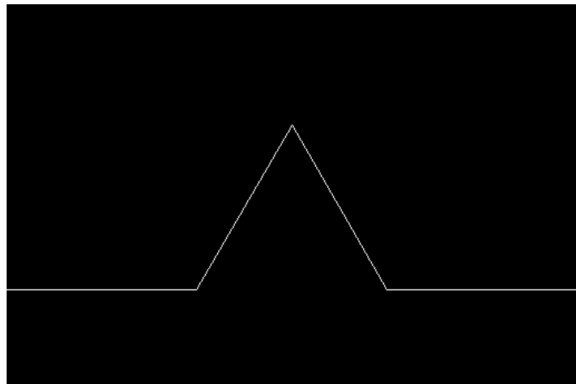
```
size=512
N=1                              #ここの値を変えてみる(最大5ぐらいにすること)
img=Image.new("RGB",(size,size))
draw=ImageDraw.Draw(img)
a=[(size,0)]
b=[(1,0),(0.5,sqrt(3.0)/2),(0.5,-sqrt(3.0)/2),(1,0)]

for _ in range(N):
  a=apply(a,b)
draw_line(draw,a,size)
img
```

ここまで実行出来たら・・・

この入力で正しく出力できているか確認しよう！

N＝1のとき，



b=[(1,0) , (0,1) , (1,0) , (0,-1) , (1,0)]

と変えてみると，描画される図形はどうなる？

7.課題：コッホ曲線

Let's Try!

＜発展＞色付きの線に変えてみる！

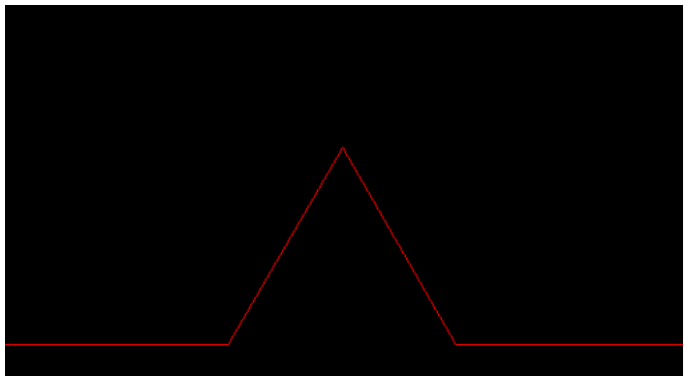
```
#発展課題　色付きフラクタル曲線
def draw_line_color(draw,a,colors,size):
  x1,y1=0,0
  for i, (dx,dy) in enumerate(a):
    x2=x1+dx
    y2=y1+dy
    c=colors[0]
    draw.line((x1,size/2-y1,x2,size/2-y2),fill=c)
    x1,y1=x2,y2
```

```
size=512
N=1
img=Image.new("RGB",(size,size))
draw=ImageDraw.Draw(img)
a=[(size,0)]
b=[(1,0),(0.5,sqrt(3.0)/2),(0.5,-sqrt(3.0)/2),(1,0)]
c=[(255,0,0),(0,255,0),(0,0,255)]
for _ in range(N):
  a=apply(a,b)
draw_line_color(draw,a,c,size)
img
```

ここまで実行出来たら・・・

この入力で正しく出力できているか確認しよう！

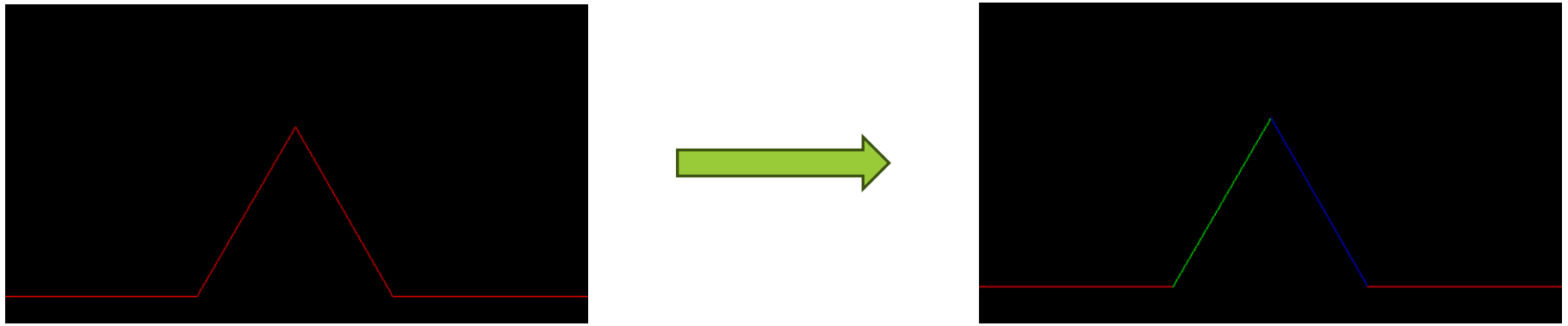
N＝１のとき，



7.課題：コッホ曲線

Let’s Try!

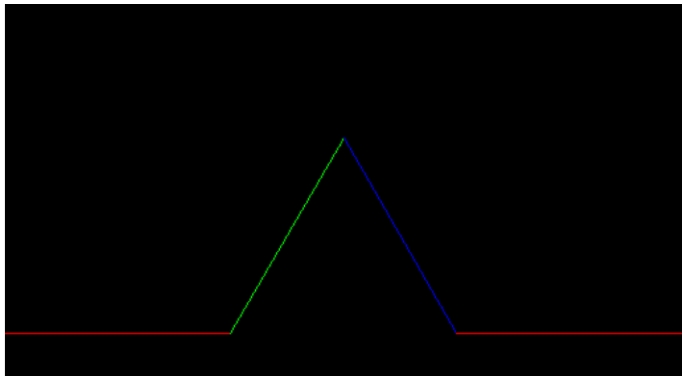
Q2.　線ごとに色を変えるようにしたい！どうする？



7.課題：コッホ曲線

Let's Try!

Q2.　線ごとに色を変えるようにしたい！どうする？



ヒント！

・変更するのはdraw_line_color関数

・変更するのは1か所でOK！

・colorsには色の情報が入ったリストがある！そのリストを使おう！

・今回は3色だけど，できれば何色入ってもいいようにしてほしい

　→リストのサイズが分かるといいな～

・余りの求め方　i％6　→　i÷6の余りの値が求められる！

```
#発展課題　色付きフラクタル曲線
def draw_line_color(draw,a,colors,size):
  x1,y1=0,0
  for i, (dx,dy) in enumerate(a):
    x2=x1+dx
    y2=y1+dy
    c=colors[0]
    draw.line((x1,size/2-y1,x2,size/2-y2),fill=c)
    x1,y1=x2,y2
```

Next>>>

オリジナル課題！

8.できればやろう課題：素数の判定

Let’s Try!

問題：以下のプログラムを穴埋めを行い，適切な出力結果を表示せよ。

＜与えられた整数のリストが素数か素数で無いかを判定し，結果を表示する。＞

※注意事項

・main関数内で指定したデータリストが素数であるかの判定は，judge関数内で行うこと。

・judge関数では，prime関数で作成された素数のリストを使用して素数の判定を行うこと。

```
[4, 9, 11, 15, 17]
['素数ではない', '素数ではない', '素数', '素数ではない', '素数']
```

8.できればやろう課題：素数の判定

Let's Try!

問題：以下のプログラムを穴埋めを行い，適切な出力結果を表示せよ。

＜関数prime＞　2～limitまでの範囲にある素数のリストを作成する。

```
#関数prime 素数のリストを作成する。
#参考：https://ictsr4.com/py/m0130.html

def prime(limit):
    prime = []                                  #素数のリスト
    for i in range(2,limit):                    #2～limitまでの数に対してfor
        for j in range(2, int(limit**0.5)+1):
            if i % j == 0:                      #素数でなければ次の数字へ
                break
        else:
            #------------                      #素数だったらprimeに追加して次の数字へ

    return #------------
```

8.できればやろう課題：素数の判定

Let's Try!

問題：以下のプログラムを穴埋めを行い，適切な出力結果を表示せよ。

＜関数judge＞関数primeで作成した素数リストを使って，データリスト要素が素数かを判定する。

```
#関数judge 素数かどうかの結果リストを作成する。

def judge(d1,d2):

  answer = []                    #解答保管用リスト（mainに返す）
  p_num_list = prime(20)         #素数のリスト，()の数字までの素数のリストが入る
                                 #関数primeにて素数のリストを作成する
  for i in d2:                   #for・if文で与えられたデータが素数のリストに含まれているか判別する
    if i in p_num_list:
      #-------------       #"素数"を格納
    else:
      #-------------       #"素数ではない"を格納

  return #--------
```

8.できればやろう課題：素数の判定

Let's Try!

問題：以下のプログラムを穴埋めを行い，適切な出力結果を表示せよ。

＜mainのブロック＞データの定義，関数の呼び出し，結果の表示

```
#mainのブロック

data1 = ["素数","素数ではない"] #テキスト
data2 = [4,9,11,15,17]          #データリスト
p_number = []                   #結果の解答を入れるリスト

p_number = #------------    #素数かどうかの判別を行った結果を代入

print(data2)                    #結果の表示（数字データ）
print(p_number)                 #結果の表示（テキストの解答）
```

```
[4, 9, 11, 15, 17]
['素数ではない', '素数ではない', '素数', '素数ではない', '素数']
```

第4章　リストとタプル

FIN