

情報処理（13週目） Pythonプログラミング発展

王 忠奎

立命館大学 ロボティクス学科

2024.07.08

講義の流れ

- クラス
- NumPy
- Matplotlib
- OpenCVの紹介
- 画像の読み込み/表示/サイズ変更/保存
- レポート

レポート2 説明

自然対数の底 $e = 2.7182818 \dots$ は、以下の式で求めることができる。

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$$

上式を使って、 $n = 10$ のときの e を求めてみよう。

ここで、 $!$ は階乗を表し、 $n! = 1 \times 2 \times 3 \times \dots \times n$ である。

- 関数を作って、階乗を計算してみよう

オブジェクト指向言語

Object-Oriented Programming (OOP)

$X = xx$

$Y = yy$

$f(X)$

$f(Y)$

$f(X) + f(Y)$

手続き型プログラミング

Procedural Programming

$X = xx$

a

$f(X)$

$Y = yy$

b

$f(Y)$

$a.f(X) + b.f(Y)$

オブジェクト指向言語

Object-Oriented Programming

オブジェクト指向言語のメリット

- データと処理の仕方が一緒にカプセルに入れてまとめているので、自分の処理の仕方を自分で知っている
- 外側からデータを処理する際、中身が何であっても、単一の記述で違う種類のデータを処理することができる
- コードを拡張するのが容易になって、作業分散に有利
- データが変わったときにそこだけ修正すれば良いので、保守性が向上

クラス (class) の定義

```
class SimpleData: # クラスの名前

    def __init__(self): # クラスを初期化するコンストラクタ、引数無し
        self.a = 0
        self.b = 0

    def sum(self): # メソッドsumの定義
        return self.a + self.b

    def set(self, a, b): # メソッドsetの定義
        self.a = a
        self.b = b
```

selfは「自分自身」という意味である。

メソッドは**第一引数にselfを書く**というルールが決まっている。

classの利用

```
data1 = SimpleData() # インスタンスを作成  
print(data1.sum()) # 初期化されたデータの合計  
  
data1.set(5, 6) # メソッドを用いたデータ操作  
print(data1.sum()) # セットしたデータの合計  
  
data1.set(14.56, 34.89)  
print(data1.sum())
```



```
0  
11  
49.45
```

引数ありの初期化

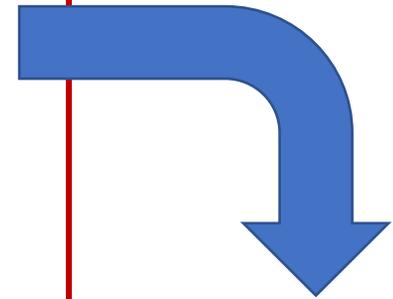
```
class SimpleData: # クラスの名前

    def __init__(self, a, b): # クラスの初期化、引数あり
        self.a = a
        self.b = b

    def sum(self): # メソッドsumの定義
        return self.a + self.b

    def set(self, a, b): # メソッドsetの定義
        self.a = a
        self.b = b

data1 = SimpleData(1, 2)
print(data1.sum())
```



3

classの継承

「継承」とは、「より一般的なclassを受け継いで、より専門的なclassを作ること」です

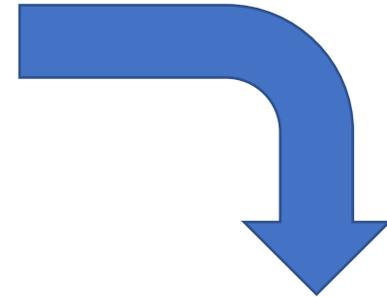
```
class ComplexData(SimpleData): # SimpleDataを継承する

    def __init__(self, a, b):
        super().__init__(a, b) # SimpleDataのメソッドを呼べる
        self.c = 1

    def sum(self):
        return self.a + self.b + self.c

data3 = ComplexData(1,2) # インスタンスを定義する
print(data3.sum()) # 初期化された引数で計算

data3.set(4,5) # 継承されたメソッドsetを呼べる
print(data3.sum())
```



```
4
10
```

NumPy

Python で数値計算を高速に行うためのライブラリ

NumPyのインストール：

File → Setting → 「Project:<プロジェクト名>」 →

Project Interpreter → +マーク → 「NumPy」を検索

し、「Install Package」をクリックしてインストールする

NumPyの利用

```
import numpy as np # NumPyをnpとしてインポート

x = np.array([1, 2, 3]) # ベクトルを定義

print(x, '¥n')
print(x.shape, '¥n') # ベクトルの形状
print(x.ndim, '¥n') # ベクトルの次元数

y = np.array([ # 2次元行列を定義
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print(y, '¥n')
```



```
[1 2 3]

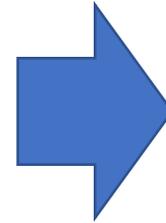
(3,)

1

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

行列操作

```
print(y.shape, '¥n') # 行列の形状  
print(y.ndim, '¥n') # 行列の次元数  
print(y.mean(), '¥n') # 行列の平均値  
print(y.mean(axis=1), '¥n') # 各行の平均  
print(y.var(), '¥n') # 行列のばらつき  
print(y.std(), '¥n') # 行列の標準偏差  
print(y.max(), '¥n') # 行列の最大値  
print(y.max(axis=1), '¥n') # 各行の最大値
```



```
(3, 3)  
  
2  
  
5.0  
  
[2. 5. 8.]  
  
6.666666666666667  
  
2.581988897471611  
  
9  
  
[3 6 9]
```

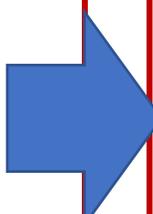
行列操作

```
z = np.array([
    [11, 12, 13],
    [21, 22, 23],
    [31, 32, 330]
])
```

```
print(z.T, '¥n') # 行列の転置
```

```
print(np.linalg.inv(z), '¥n') # 逆行列の計算
```

```
print(np.dot(y, z), '¥n') # 行列積の計算
```



```
[[ 11  21  31]
 [ 12  22  32]
 [ 13  23 330]]
```

```
[[ -2.196633    1.19326599  0.003367   ]
 [  2.09326599 -1.08653199 -0.00673401]
 [  0.003367   -0.00673401  0.003367   ]]
```

```
[[ 146  152 1049]
 [ 335  350 2147]
 [ 524  548 3245]]
```

特殊行列

```
print(np.zeros((3,3)), '¥n') # すべて要素が0
```

```
print(np.ones((3,3)), '¥n') # すべて要素が1
```

```
print(np.eye(3), '¥n') # 単位行列
```

```
print(np.random.random((3,3)), '¥n')  
# ランダム行列
```



```
[[0. 0. 0.]  
 [0. 0. 0.]  
 [0. 0. 0.]]
```

```
[[1. 1. 1.]  
 [1. 1. 1.]  
 [1. 1. 1.]]
```

```
[[1. 0. 0.]  
 [0. 1. 0.]  
 [0. 0. 1.]]
```

```
[[0.7887845  0.15884651  0.33635022]  
 [0.2736158  0.3030747  0.26566676]  
 [0.64561389  0.43007767  0.55651081]]
```

Matplotlib

matplotlibはグラフを使った可視化の際によく使われるライブラリで、Python上でMatlabのようなグラフ描画機能を持たせることを目的として作られた。

Matplotlibのインストール：

File → Setting → 「Project:<プロジェクト名>」 →

Project Interpreter → +マーク → 「matplotlib」を検

索し、「Install Package」をクリックしてインストールする

matplotlibの使い方

```
import numpy as np
import matplotlib.pyplot as plt # pltとしてインポートする

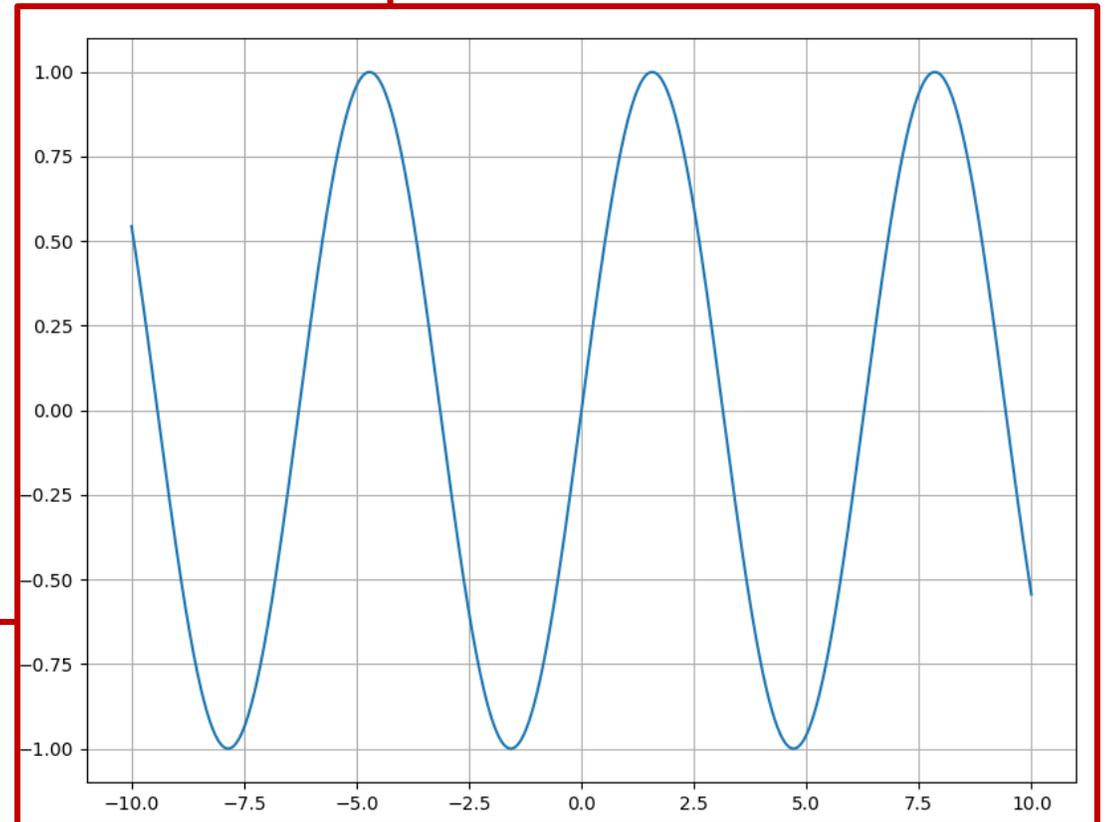
x = np.linspace(-10, 10, 1000) # x軸データを用意

y = np.sin(x) # サインを計算する

plt.plot(x, y) # プロットする

plt.grid(True) # グリッドを入れる

plt.show() # グラフの表示
```



ラベルなどの追加

```
import numpy as np
import matplotlib.pyplot as plt # pltとしてインポートする

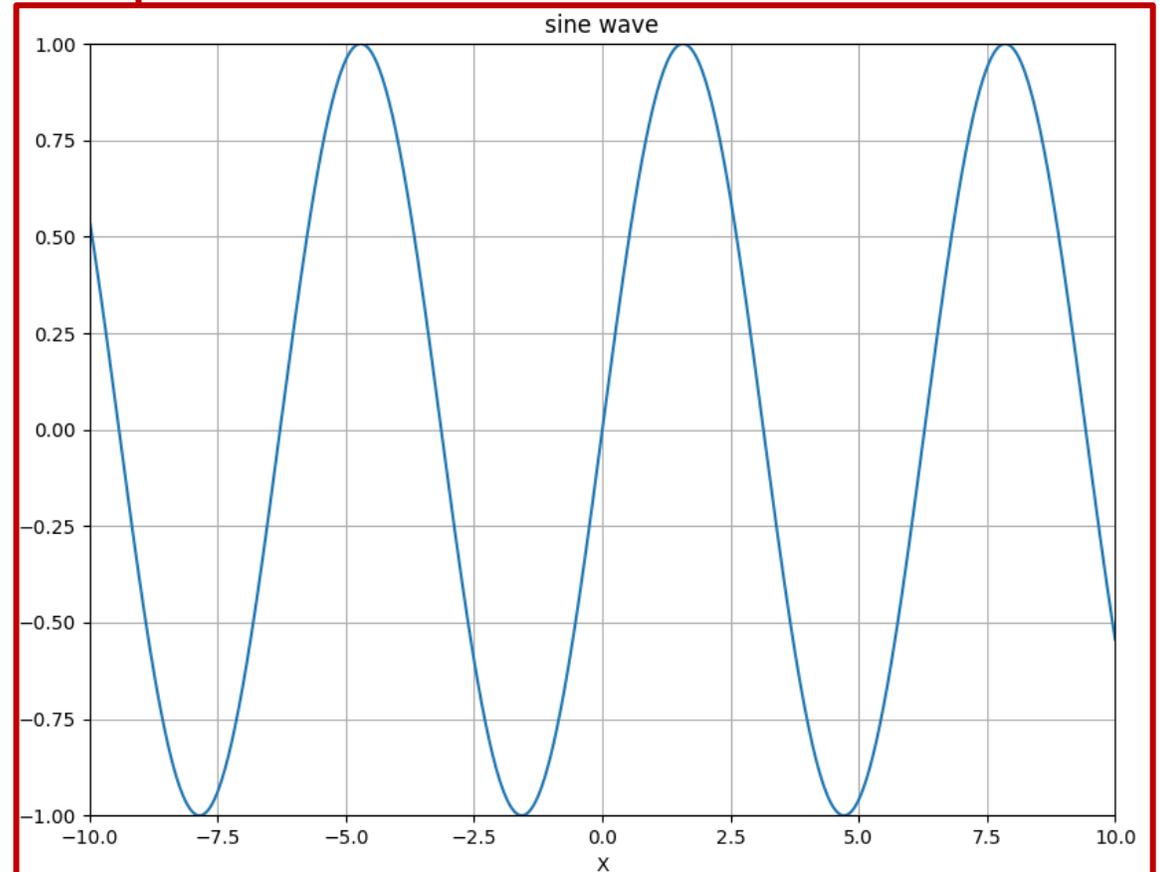
x = np.linspace(-10, 10, 1000) # x軸データを用意

y = np.sin(x) # サインを計算する

plt.plot(x, y) # プロットする

plt.grid(True) # グリッドを入れる
plt.title('sine wave') # タイトルを追加
plt.xlabel('X') # X軸ラベル
plt.ylabel('y') # Y軸ラベル
plt.xlim(-10, 10) # X軸範囲
plt.ylim(-1.0, 1.0) # Y軸範囲

plt.show() # グラフの表示
```



```
import numpy as np
import matplotlib.pyplot as plt # pltとしてインポートする
```

```
# plot 1
```

```
x = np.array([0, 1, 2, 3]) # x軸データ
y = np.array([3, 8, 1, 10]) # y軸データ
```

```
# 1行2列のサブプロットを作成し、1列目を利用
```

```
plt.subplot(1, 2, 1)
plt.plot(x, y)
```

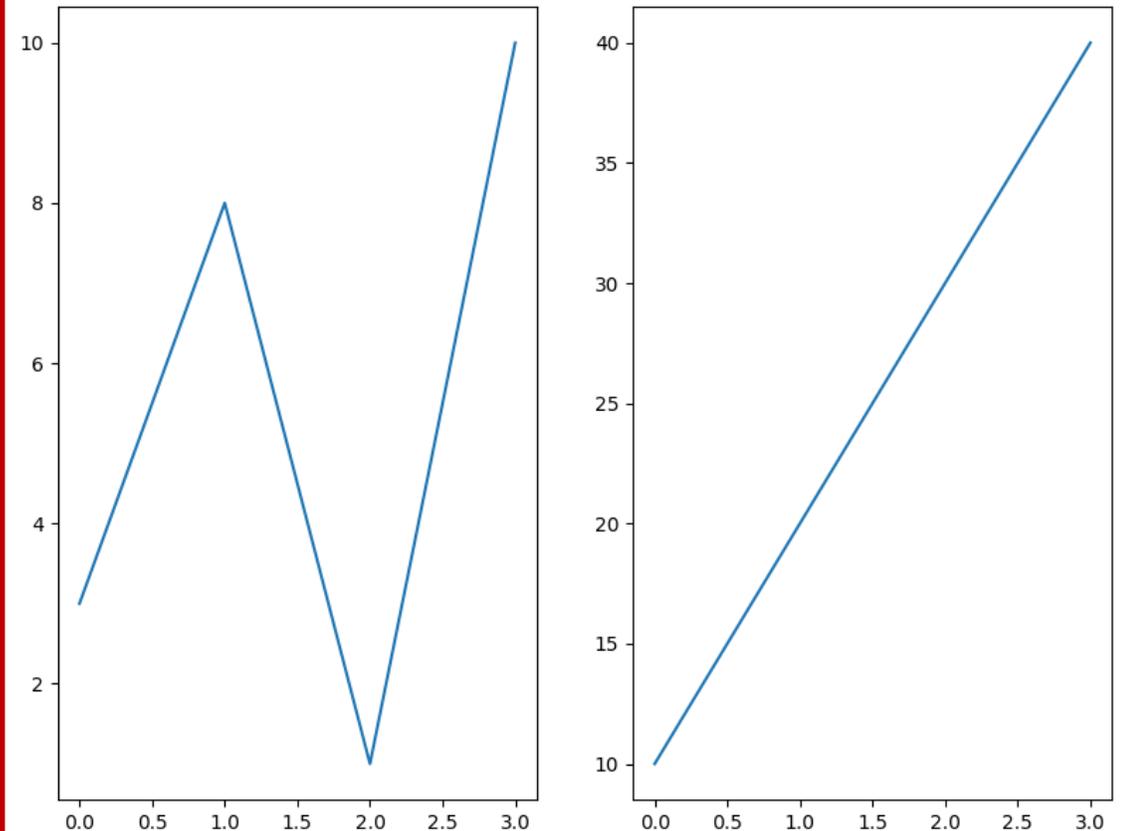
```
# plot 2
```

```
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
```

```
plt.subplot(1, 2, 2) # 2列目の利用
plt.plot(x, y)
```

```
plt.show()
```

複数グラフのプロット



散布図

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

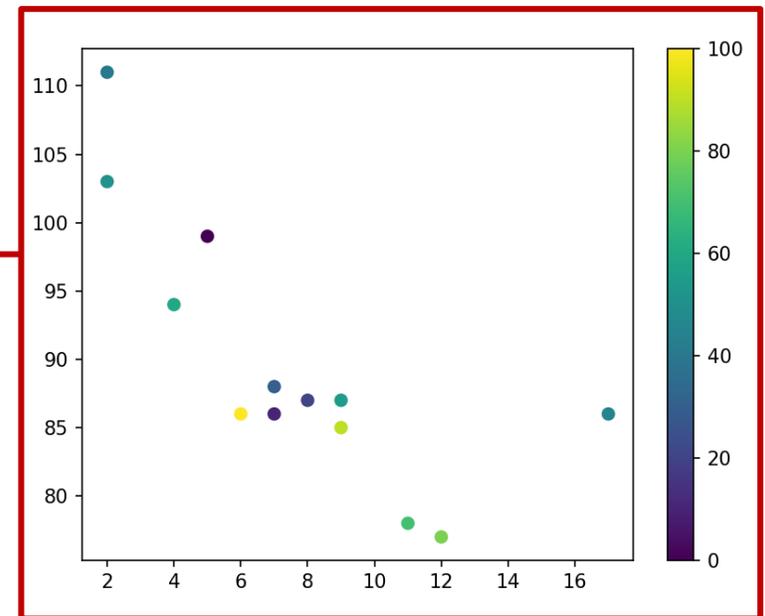
```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```

```
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
```

```
plt.scatter(x, y, c=colors, cmap='viridis')
```

```
plt.colorbar()
```

```
plt.show()
```



OpenCV

「OpenCV」とは、Open Source Computer Vision Libraryの略で、**画像や動画を処理するための機能がまとめて実装されているオープンソースのライブラリ**です。

「opencv-python」はOpenCVのPythonコードのライブラリです。

OpenCVのインストール：

File → Setting → 「Project:<プロジェクト名>」 → Project

Interpreter → +マーク → 「opencv-python」を検索し、

「Install Package」をクリックしてインストールする

画像の読み込みと表示

```
import cv2 as cv  # OpenCVをcvとしてインポート  
  
img = cv.imread('image1.png') # 画像の読み込み  
  
cv.imshow('Gripper', img)    # 画像にタイトルを付けて表示  
  
cv.waitKey(0)    # 任意のキーを押すことを待つ
```

- 用意した画像がプログラムと同じフォルダにある場合
- 同じフォルダではない場合、パスを指定する

画像のサイズ変更と保存

```
import cv2 as cv

img = cv.imread('image1.png') # 画像の読み込み

height = img.shape[0] # 画像の高さを取得
width = img.shape[1] # 画像の幅を取得

resized_img = cv.resize(img, (int(width*0.3), int(height*0.3))) # サイズの変更

cv.imshow('Resized image', resized_img) # 画像の表示
cv.waitKey(0)

cv.imwrite('gripper_resized.jpg', resized_img) # 画像の保存
```

レポート

- 写真の1枚を用意して、画像サイズを「300画素×300画素」に変更し、別名で保存するプログラムを作成してみよう。
- 写真とプログラムをmanaba+Rで提出してください。