Python入門

東海大学　情報通信学部　組込みソフトウェア工学科
今村研究室

今日の内容

- pythonの基礎　変数・演算・リストと辞書型
- 条件分岐とループ
- 関数
- クラスとインスタンス
- よく使うライブラリの基礎（科学計算、データ加工、グラフ描画）
  - numpy
  - pandas
  - matplotlib

Pythonについて

・プログラミング言語のひとつ

・広く扱われているため、本やサイトも豊富な言語

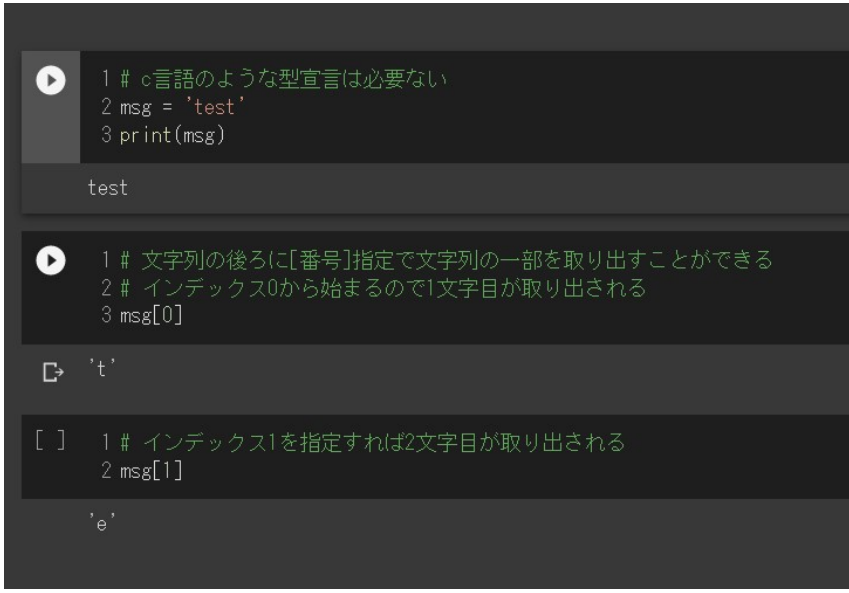
・組込みアプリ開発やwebサイト構築から、ディープラーニングまで様々な分野で利用可能

pythonの基礎　–変数–

Int や float などのような宣言なしで使える。
型は変数に応じて動的に決定される

インデックス指定をすると文字列を取り出すことができる

＊変数名や配列の名前はわかりやすいものにする
例えば時系列データを格納したものならtime_series
for などで使う場合はcountなど
aやbなどの文字のみはNG
他の人が見て直感的にわかるようにする

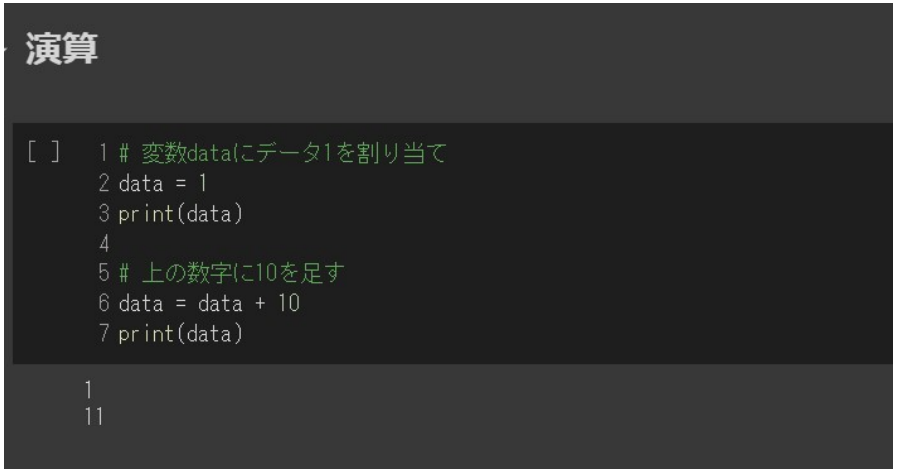


Python 基礎 –演算–

右の図は加算の例である

変数は無論数字も格納可能

＝は同値の意味ではなく、右の値を左の値へ割り当てるという意味



Pythonの基礎　–リスト–

リストとは、複数の値をひとまとめにして扱うための仕組み

変数と同じく、中身の取り出しは

リスト名[インデックス]で可能

インデックス指定では**[1:4]**などで指定するとインデックスの**1**番目から**4**番目の手前まで取り出すという意味になる

リストの大きさよりも大きい数字を指定するとエラー

また、作成の際に**range**というメソッドを使用する方法などある

リストと辞書型

```
[ ]  1 # リストを作る
     2 data_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
     3 print(data_list)
     4
     5 # オブジェクトの種類がわかる
     6 print('変数のタイプ：', type(data_list))
     7
     8 # 1つの要素を取り出す。　0から始まるので、[1]は2番目
     9 print('2番目の数：', data_list[1])
    10
    11 # len関数で要素の数を出力。　今回は1～10の10個なので結果は10を返す
    12 print('要素数：', len(data_list))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
変数のタイプ： <class 'list'>
2番目の数： 2
要素数： 10
```

Pythonの基礎　–リスト–

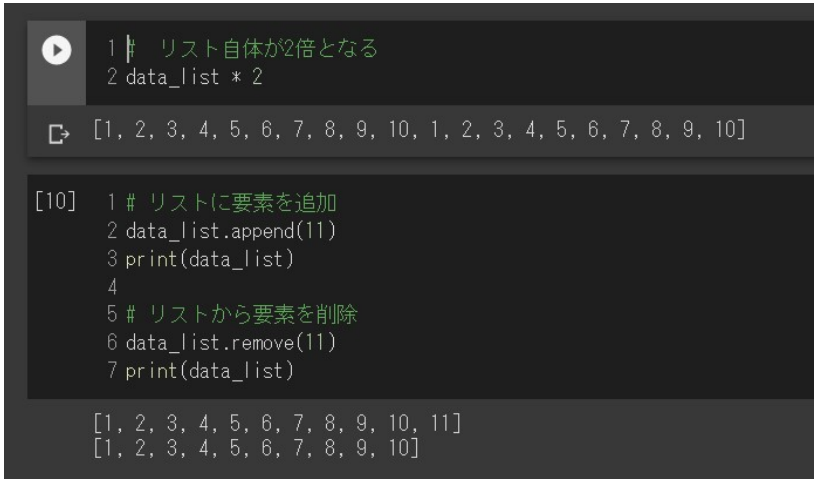
リスト自体への掛け算はリスト全体がもう一つ増えるだけになる。

要素の追加にはappendメソッドを使う

appendでは要素が最後に追加される

挿入したい場合にはinsertメソッドを使う

削除にはremoveを使う



Pythonの基礎　–辞書型–

辞書型では、キーと値をペアにすることができる

図のように値を参照するときは
辞書データ[キー名]と表記する。

追加したいときは、
辞書データ[追加したいキー] = 値
のようにすると追加可能。

削除にはdelメソッドや、popメソッドを使う

```
# 辞書型
dic_data = {'apple': 100, 'banana': 100, 'orange': 300, 'mango': 400, 'melon': 500}

# melonの値を表示
print(dic_data['melon'])

# orangeの値を表示
print(dic_data['orange'])
```

```
500
300
```

```
[12]  # 辞書に要素を追加
      dic_data['peach'] = 600
      print(dic_data)
```

```
{'apple': 100, 'banana': 100, 'orange': 300, 'mango': 400, 'melon': 500, 'peach': 600}
```

Pythonの基礎　–演算子と真偽判定–

Pythonでよく使用する演算子を示す

他にも!=で否定や<や>などの大小記号などを使用する。

また、and, or, not などを使って条件を追加することも可能

演算子	意味	使用例
＋	加算	num1 + num2
－	減算	num1 – num2
*	乗算	num1 * num2
／	除算	num1 / num2
%	剰余	num1 % num2
**	べき乗計算	num1 ** num2
//	除算（切り捨て）	num1 // num2

Pythonの基礎　−if文−

条件分岐を行う

C言語のように if (a > 5): のように書くこともできる

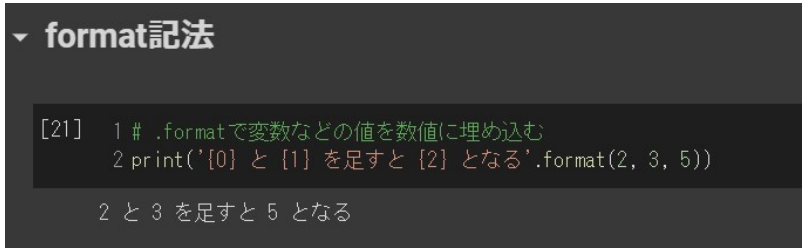
右図の場合だと in data_list で　data_list にあるかどうかの判定を行い分岐する

複数の分岐を使う場合はelse if ではなくelif を使う

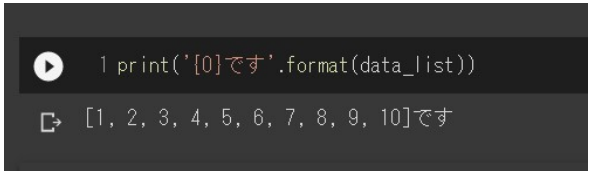
*python はif やfor の始まりと終わりはすべてインデントで判断している

Pythonの基礎　−format文−

変数などの値を文字列に埋め込むための機能



リストなども埋め込み可能



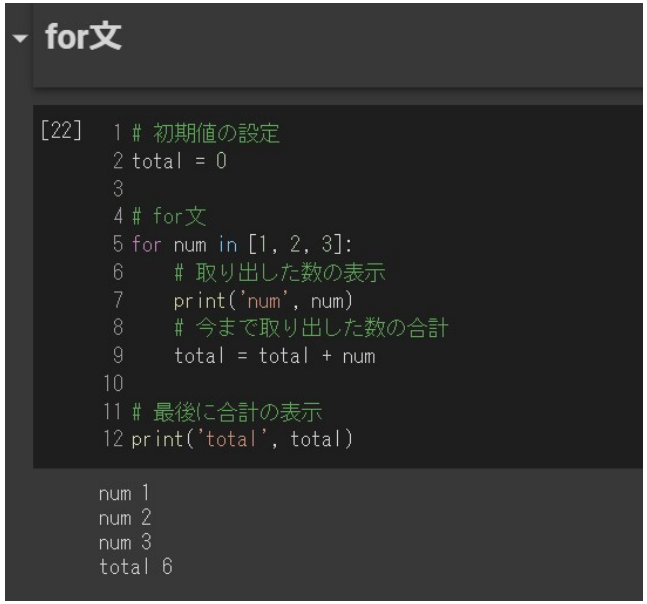
Pythonの基礎　−for文−

繰り返し処理の構文

右図はリストからデータを一つずつ取り出す例

for num in [リスト]:

で、リストの中身をひとつずつ取り出す。



Pythonの基礎　−rangeを使ったfor文−

連続した整数のリストを作りたいとき

range関数を使うと楽ができる

range(n,m,l)でnが開始の値

mが最後の値-1

lが飛ばす値　となっている

for i in range(0, 101, 2):であれば、0から100まで2個飛ばしで繰り返すということ

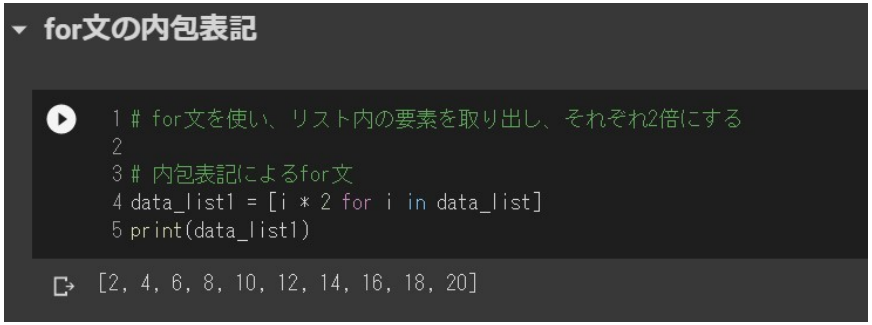
Pythonの基礎　–for文の内包表記–

新しいリストの作成などをするときにシンプルに書く手法

for 以外にもif による条件分岐なども可能である

例: even = [i for i in range (10) if i % 2 == 0] # 偶数のリストの作成

else をつかうこともできる



Pythonの基礎　−while文−

while文は　while (条件式)が成り立っている間は繰り返すという構文

図では　numが10以下が真の場合は繰り返す

＊こういったループを行う際にループの最後でnumのような変数に加算をする

忘れると無限ループに入りエラーとなる

Pythonの基礎　−関数−

関数の記述はdef 関数名(引数):で行う

引数が入力となって,returnで返り値を返す

実際の実装では後述するクラス内で関数を定義したりする

深層学習などになると、学習結果の評価に使う損失関数と呼ばれる関数の定義などにも使う

例:softmax 関数　ReLu関数　など

関数

```
[ ]  1 # 掛け算を行う関数
     2 def calc_multi(a, b):
     3     return a * b
```

```
[ ]  1 calc_multi(3, 10)
```

```
30
```

```
[ ]  1 # 再帰関数の例（フィボナッチ数列）
     2 def calc_fib(n):
     3     if n == 1 or n == 2:
     4         return 1
     5     else:
     6         return calc_fib(n - 1) + calc_fib(n - 2)
```

```
[ ]  1 print('フィボナッチ数：', calc_fib(10))
```

```
フィボナッチ数： 55
```

Pythonの基礎　–クラスとインスタンス–

class クラス名: で作成する

オブジェクトを作成する際のデータやメソッドを定義する

__init__ はコンストラクタである

どのように生成するか、どのようなデータを持たせるかという情報を定義する

Pythonの基礎　–クラスとインスタンス–

class クラス名: で作成する

オブジェクトを作成する際のデータやメソッドを定義する

__init__ はコンストラクタである

どのように生成するか、どのようなデータを持たせるかという情報を定義する

ライブラリのインポート

データサイエンスでは大量のデータを加工・分析する

→ライブラリを使用して作業効率を上げる

import ライブラリ名 as 識別名 のように記述する

識別名を使うことで、以降ライブラリの呼び出しはその識別名で行える

よく使用するライブラリ

・Numpy・・・配列処理や数値計算のライブラリ

・Pandas・・・データフレーム形式で様々なデータを閲覧・加工するライブラリ

・Matplotlib・・・データや画像の可視化を行うライブラリ

・Scikit-learn・・・機械学習のライブラリ　多くの機械学習アルゴリズムが実装されている　また、サンプルデータセットも使うことができる

・pytorch・・・機械学習(deep learning)のライブラリ

などがある

ライブラリのインポート

データサイエンスでは大量のデータを加工・分析する

→ライブラリを使用して作業効率を上げる

import ライブラリ名 as 識別名 のように記述する

識別名を使うことで、以降ライブラリの呼び出しはその識別名で行える

Numpy −配列操作−

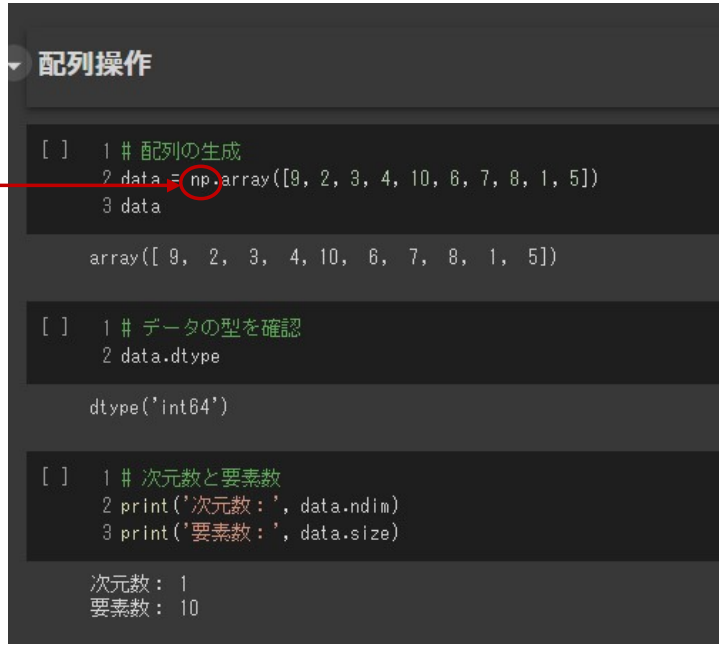
arrayメソッドを使用することで配列を作れる

＊先ほど識別名npでimport したためnpで呼び出している

dtypeでデータ型、ndim,sizeで次元数、要素数を確認

ほかにもshapeで行列数、sizeで大きさを確認可能など様々なメソッドが存在する

＊numpy メソッドなどで検索すると大量にヒットするので調べてみましょう



Numpy −配列の計算−

リストの時とは違い、単純に配列に数を掛けるとそれぞれの要素を係数倍する

同様に+ n　でそれぞれの要素にnを加算する

また、for文などを使わなくても配列のそれぞれの要素同士での計算が可能

```
[34]  1 # 配列内のそれぞれの要素を係数倍
      2 data * 2

      array([18,  4,  6,  8, 20, 12, 14, 16,  2, 10])

[35]  1 # それぞれの要素同士での計算
      2 print('掛け算：', np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) * np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]))
      3 print('累乗：', np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) ** 2)
      4 print('割り算：', np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) / np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]))

      掛け算： [10 18 24 28 30 30 28 24 18 10]
      累乗： [  1   4   9  16  25  36  49  64  81 100]
      割り算： [ 0.1    0.222  0.375  0.571  0.833  1.2    1.75   2.667  4.5   10.   ]
```

Numpy −配列のソート−

配列の中身を並べ替えることができる

sortメソッドを使うと元の配列を置き換えて並べ替える

また、スライス表記により降順にソートすることもできる

＊スライス　配列内の指定をする際に[n:m:s]と表記することで「n番目からm番目をs飛ばしで」取り出せる

[:-1]のようにすると最後尾からひとつ前までとなる

```
[120]  1 # 並べ替え
       2 print('並べ替え無し：', data)
       3
       4 # ソートした結果
       5 data.sort()
       6 print('ソート後：', data)
       7
       8 # sortメソッドは元の値を置き換えるのでdataはsort後のものとなる
       9 data

       並べ替え無し： [ 9  2  3  4 10  6  7  8  1  5]
       ソート後： [ 1  2  3  4  5  6  7  8  9 10]
       array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

[121]  1 # スライスについて [n:m:s]で「n番目からm番目までをs飛ばしで取り出す」
       2 print(data[1:3])
       3 print(data[:5])
       4 print(data[:-1])

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

       1 # 降順にsort
       2 data[::-1].sort()
       3 print('ソート後：', data)

       ソート後： [10  9  8  7  6  5  4  3  2  1]
```

Numpy −配列内での計算−

配列内の最小値、最大値、合計、積み上げなどをメソッドを使い計算できる

また、mean()やaverage()で平均の計算などがあります（どちらも平均ですが、データ型を指定できたり、重み付き平均の計算であるなどの違いがある）

これらのメソッドはデータ分析の際によく使用する

```
[123]  1 # 最小、最大、合計、積上の計算
       2
       3 # 最小値
       4 print('Min:', data.min())
       5
       6 # 最大値
       7 print('Max:', data.max())
       8
       9 # 合計
      10 print('Sum:', data.sum())
      11
      12 # 積み上げ
      13 print('Cum:', data.cumsum())
      14
      15 # 積み上げ割り当て
      16 print('Ratio:', data.cumsum() / data.sum())

    Min: 1
    Max: 10
    Sum: 55
    Cum: [10 19 27 34 40 45 49 52 54 55]
    Ratio: [0.182 0.345 0.491 0.618 0.727 0.818 0.891 0.945 0.982 1.   ]
```

Numpy −行列−

配列は1次元のみですが、行列で多次元のデータを作成したり、扱うこともできる

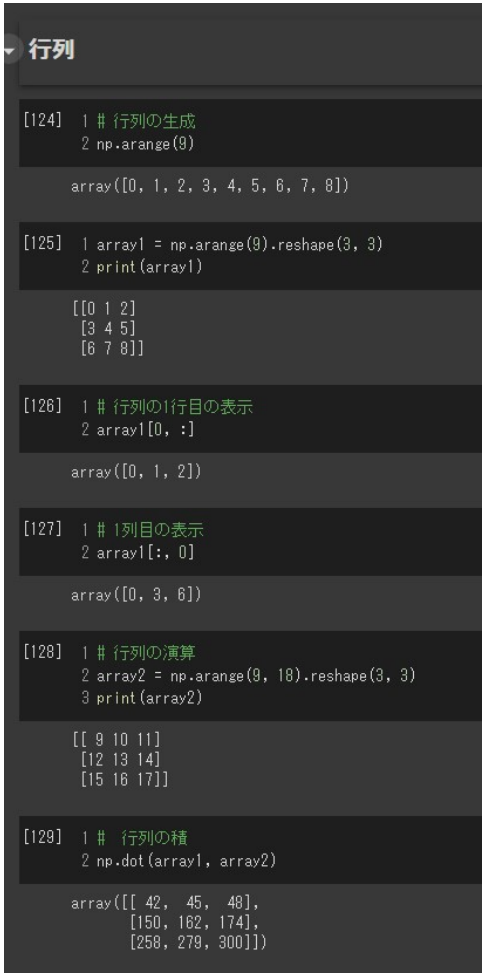
行列の場合はarange()を使う

また、reshapeにより行列の形状を変える

今回reshape(3, 3)で3行3列になっているがもとの行列と要素数が一致しないとエラーとなる

dotメソッドを使用し行列の積を計算する

そのまま行列同士を演算させると要素同士の演算となる



Numpy –要素が0や1の行列–

zeros(), ones()で要素が0や1の行列を作る

こういった行列や配列は機械学習の計算で用いられることがある

```
[131]  1 # 要素が0や1の行列を作成
       2 print(np.zeros((2, 3), dtype = np.int64))
       3 print(np.ones((2, 3), dtype = np.float64))

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

Pandas −Series−

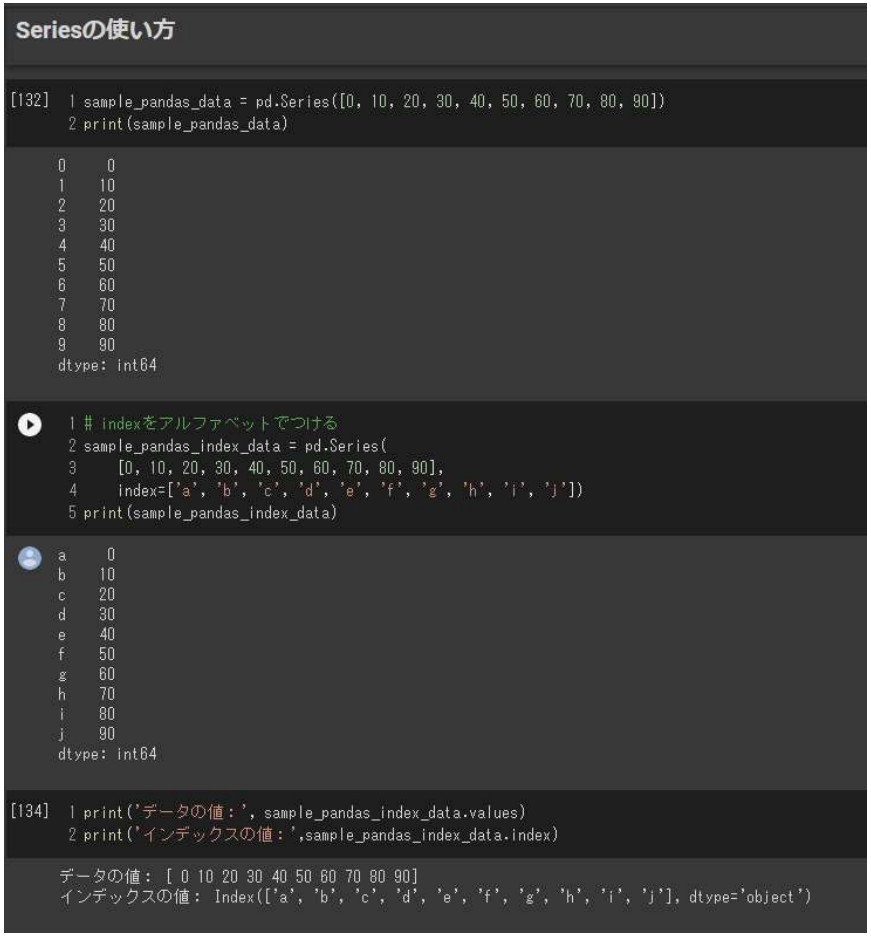
Seriesは1次元のデータを格納するために使われる

（高機能な1次元配列みたいなもの）

インデックスやカラムに名前を付けることができたり多くのメソッドが存在する

例では0から90までのデータを格納したseriesを作り、インデックス名をアルファベットにしている

＊実際にはseriesよりも後述のdataframeの方が多く使う印象がある

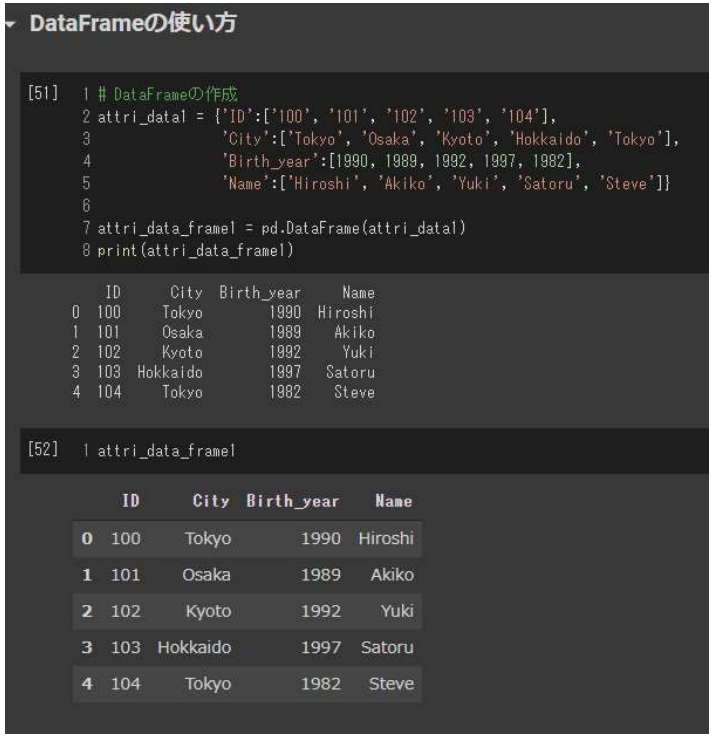


Pandas −DataFrame−

Seriesは1次元のデータであるが、DataFrameは二次元のデータである

Seriesと同様にインデックス名、カラム名を指定でき、行ごと、列ごとに様々な操作ができる

作成するほかに、CSVファイルなどから読み込むこともできる

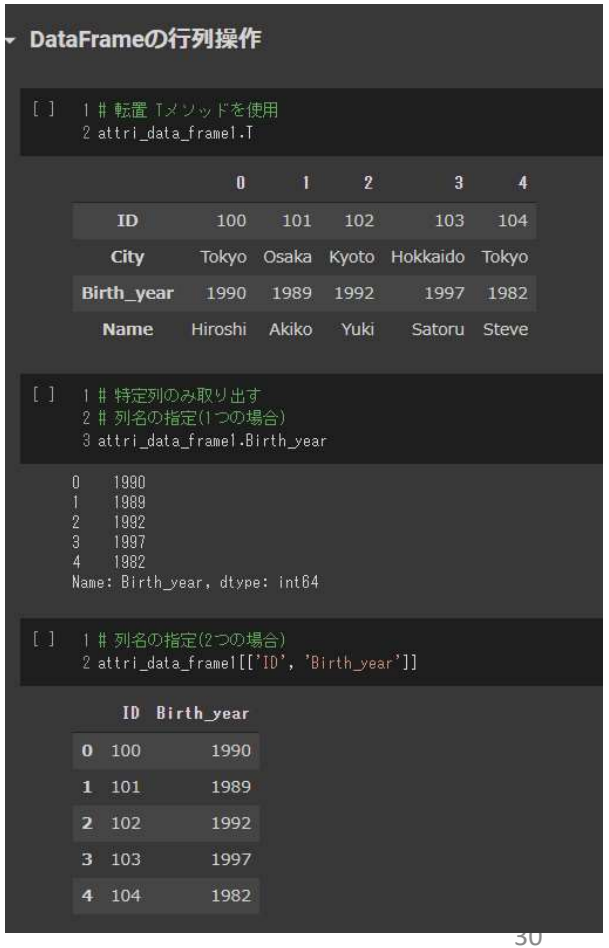


Pandas −DataFrame 行列操作−

Tメソッドを使用してカラムとインデックスを入れ替えることができる

列名の指定及び複数の列名を指定し、操作を行うことができる

今回行ったもの以外にもilocメソッドなどで行、列を指定することもできる



DataFrameの行列操作

```
[ ]  1 # 転置 Tメソッドを使用
     2 attri_data_frame1.T
```

	0	1	2	3	4
ID	100	101	102	103	104
City	Tokyo	Osaka	Kyoto	Hokkaido	Tokyo
Birth_year	1990	1989	1992	1997	1982
Name	Hiroshi	Akiko	Yuki	Satoru	Steve

```
[ ]  1 # 特定列のみ取り出す
     2 # 列名の指定(1つの場合)
     3 attri_data_frame1.Birth_year
```

```
0    1990
1    1989
2    1992
3    1997
4    1982
Name: Birth_year, dtype: int64
```

```
[ ]  1 # 列名の指定(2つの場合)
     2 attri_data_frame1[['ID', 'Birth_year']]
```

	ID	Birth_year
0	100	1990
1	101	1989
2	102	1992
3	103	1997
4	104	1982

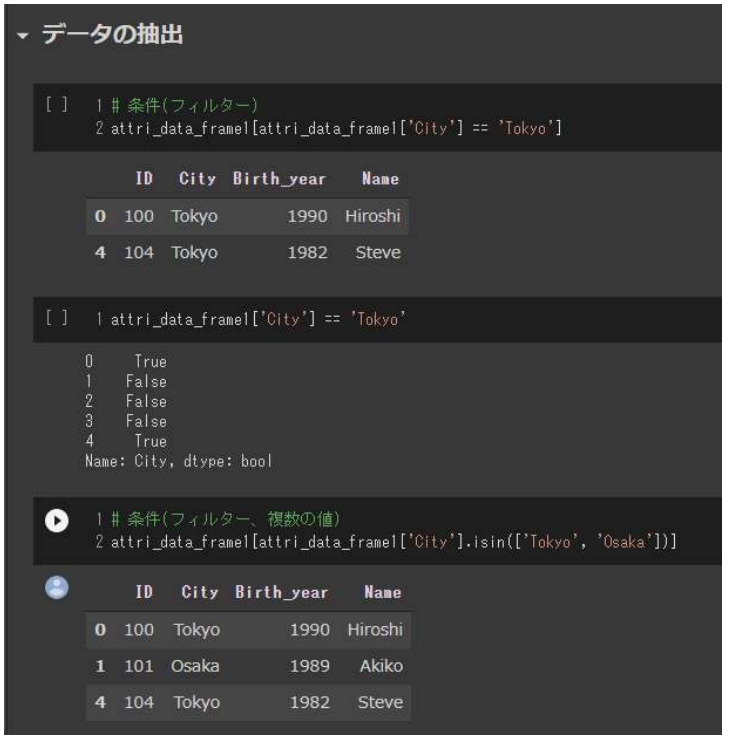
Pandas −DataFrame　抽出−

データフレーム内の特定の条件を満たすもののみの抽出ができる

図の条件はデータフレームの’City’が’Tokyo’であるものを抽出している

また、isinメソッドを使用することで複数条件を指定できる

膨大な量のデータで、特定の条件を満たすデータの数を数えたい時などによく使用する



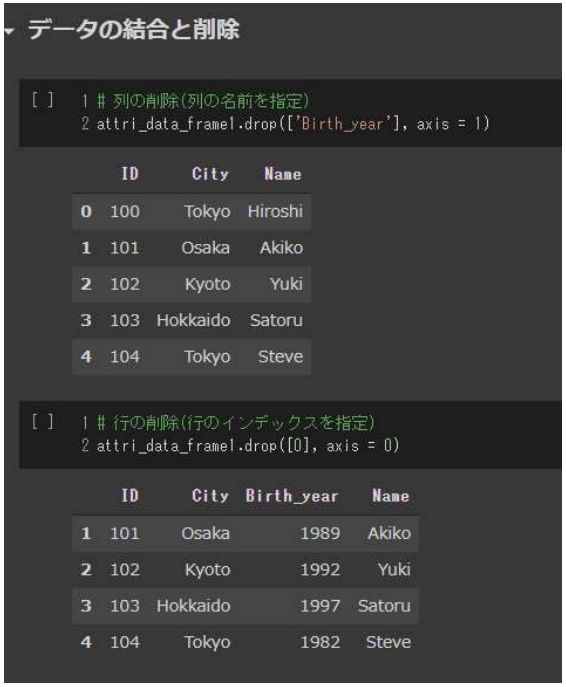
Pandas −DataFrame 削除−

膨大な量のデータを扱う際には、データに歯抜けがある場合が存在する（歯抜けはNaNという表示になる）

歯抜けが数個であった場合などはそれらしい値(平均や中央値など）で埋めることが挙げられる

しかし5割6割歯抜けしているカラムがあるとデータとして使い物にならない場合がある。

その場合などにdrop()やdropna()（NaNを含む行すべて削除）などを使い歯抜けを処理する



Pandas −DataFrame 結合−

データフレームの結合ができる

様々なデータがある場合にそれらを結合して分析することがある

mergeメソッドを使用する

キーを明示しない場合、自動的に同じキーの値であるものを見つけて結合する

結合の方法には全結合や内部結合など様々な種類が存在する

```
[ ]  1 # データの結合
     2
     3 #別のデータの準備
     4 attri_data2 = {'ID':['100', '101', '102', '105', '107'],
     5               'Math':[50, 43, 33, 76, 98],
     6               'Englidh':[90, 30, 20, 50, 30],
     7               'Sex':['M', 'F', 'F', 'M', 'M']}
     8 attri_data_frame2 = pd.DataFrame(attri_data2)
     9 attri_data_frame2
```

	ID	Math	Englidh	Sex
0	100	50	90	M
1	101	43	30	F
2	102	33	20	F
3	105	76	50	M
4	107	98	30	M

```
     1 # データのマージ
     2 pd.merge(attri_data_frame1, attri_data_frame2)
```

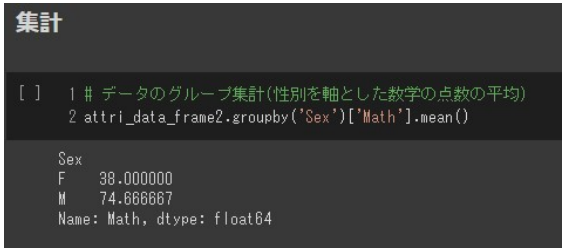
	ID	City	Birth_year	Name	Math	Englidh	Sex
0	100	Tokyo	1990	Hiroshi	50	90	M
1	101	Osaka	1989	Akiko	43	30	F
2	102	Kyoto	1992	Yuki	33	20	F

Pandas −DataFrame 集計−

groupbyを使って特定列を軸とした集計ができる

この場合だと、性別ごとの数学平均の計算をしている

計算系のメソッドはmean()以外にも
max(),min(),mode()などが存在する



Pandas −DataFrame ソート−

インデックスの順番などがバラバラであった場合に
sortメソッドを使い並べ替えができる

インデックスによるソートならsort_index()

データの値でソートしたい場合は

軸としたいデータ.sort_value()でできる

値のソート

```
[ ]  1 # データの準備
     2 attri_data2 = {'ID':['100', '101', '102', '103', '104'],
     3                'City':['Tokyo', 'Osaka', 'Kyoto', 'Hokkaido', 'Tokyo'],
     4                'Birth_year':[1990, 1989, 1992, 1997, 1982],
     5                'Name':['Hiroshi', 'Akiko', 'Yuki', 'Satoru', 'Steve']}
     6
     7 attri_data_frame2 = pd.DataFrame(attri_data2)
     8 attri_data_frame_index2 = pd.DataFrame(attri_data2, index=['e', 'b', 'a', 'd', 'c'])
     9 attri_data_frame_index2
```

	ID	City	Birth_year	Name
e	100	Tokyo	1990	Hiroshi
b	101	Osaka	1989	Akiko
a	102	Kyoto	1992	Yuki
d	103	Hokkaido	1997	Satoru
c	104	Tokyo	1982	Steve

```
[ ]  1 # indexによるソート
     2 attri_data_frame_index2.sort_index()
```

	ID	City	Birth_year	Name
a	102	Kyoto	1992	Yuki
b	101	Osaka	1989	Akiko
c	104	Tokyo	1982	Steve
d	103	Hokkaido	1997	Satoru
e	100	Tokyo	1990	Hiroshi

Pandas −DataFrame 欠損値の処理−

まず、データ内にどれだけnanがあるのかを計算している　isnull()でnanの判定 sum()でnanの合計を計算している

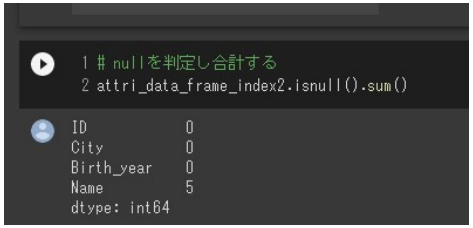
先ほどのスライドでもあったようにこのnanの処理を行わないと後の機械学習などを正しく行えない

今回の場合欠落しているのはName(名前)なので削除しても問題なさそうなため、削除するなどの処理を行うべきである

別の値で埋める場合はfillna()を使う

例:fillna(method = ‘ffill’)で直前の値で埋める

dataframe.fillna(dataframe.mean())で平均値で埋める



Matplotlib –可視化ライブラリ–

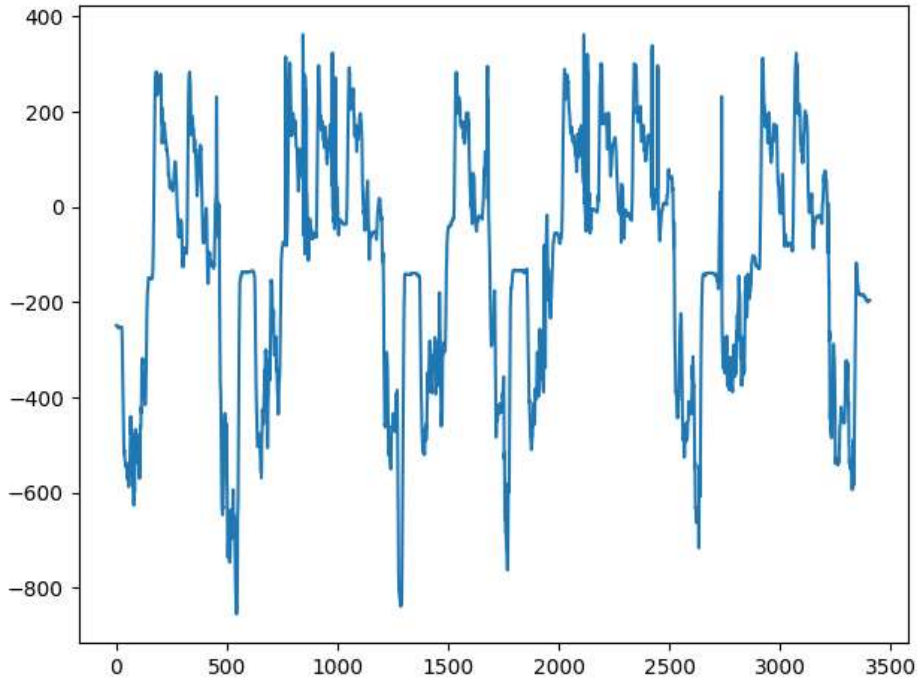
Matplotlibはデータの可視化を行うライブラリである

今村研究室では加速度や座標データを扱う

そういったデータの遷移を可視化する際に頻繁に扱うことになる

右のグラフは実際に実験で取得した右手首の座標データの推移である

ノイズが多いため、平滑化などが必要であるがこのようにデータを可視化することができる



Matplotlib −散布図の例−

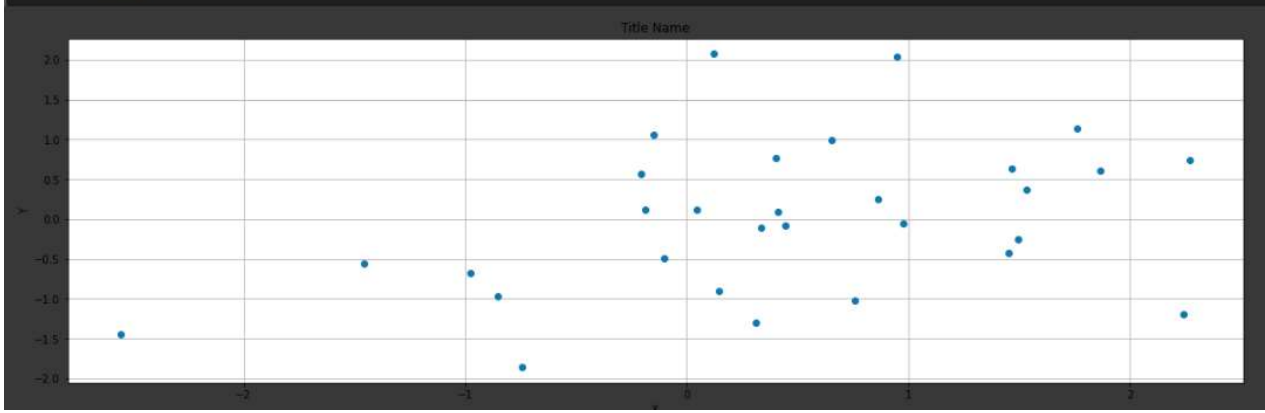
右はランダムに生成したデータを散布図で可視化する例である

figure()ではグラフ自体の大きさを指定している

plotがグラフのプロット部分である
色やマーカーの形、線の種類などを指定できる　今回は‘o’で〇のマーカーになっている

タイトルやラベルなどもつけることができる
gird()をTrueにすればグラフ内のgridを表示する

```python
# シード値の固定
np.random.seed(0)

# x軸のデータ
x = np.random.randn(30)

# y軸のデータ
y = np.sin(x) + np.random.randn(30)

# グラフの大きさの指定
plt.figure(figsize=(20, 8))

# グラフの描画
plt.plot(x, y, 'o')

# 以下でも散布図が描ける
# plt.scatter(x, y)

# タイトル
plt.title('Title Name')
# x軸の座標名
plt.xlabel('X')
# y軸の座標名
plt.ylabel('Y')

# gridの表示
plt.grid(True)
```



Matplotlib −連続曲線の例−

連続するデータをplotする場合はこのような連続曲線となる

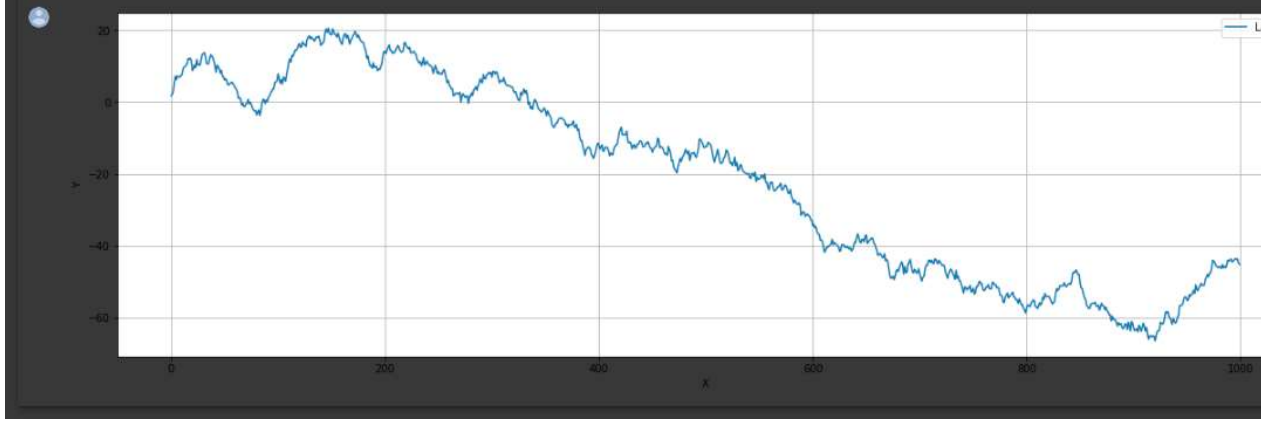
legend()は凡例の表示である

plot()の引数に関数を持ってきて描画することもできる

例：plt.plot(x, my_function(x))
my_functionの関数を描画できる

連続曲線

```
# シード値の固定
np.random.seed(0)

# データの範囲
numpy_data_x = np.arange(1000)

# 乱数の発生と積み上げ
numpy_random_data_y = np.random.randn(1000).cumsum()

# グラフの大きさを指定
plt.figure(figsize = (20, 8))

# ラベルを付ける
plt.plot(numpy_data_x, numpy_random_data_y, label='Label')
plt.legend()

plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
```

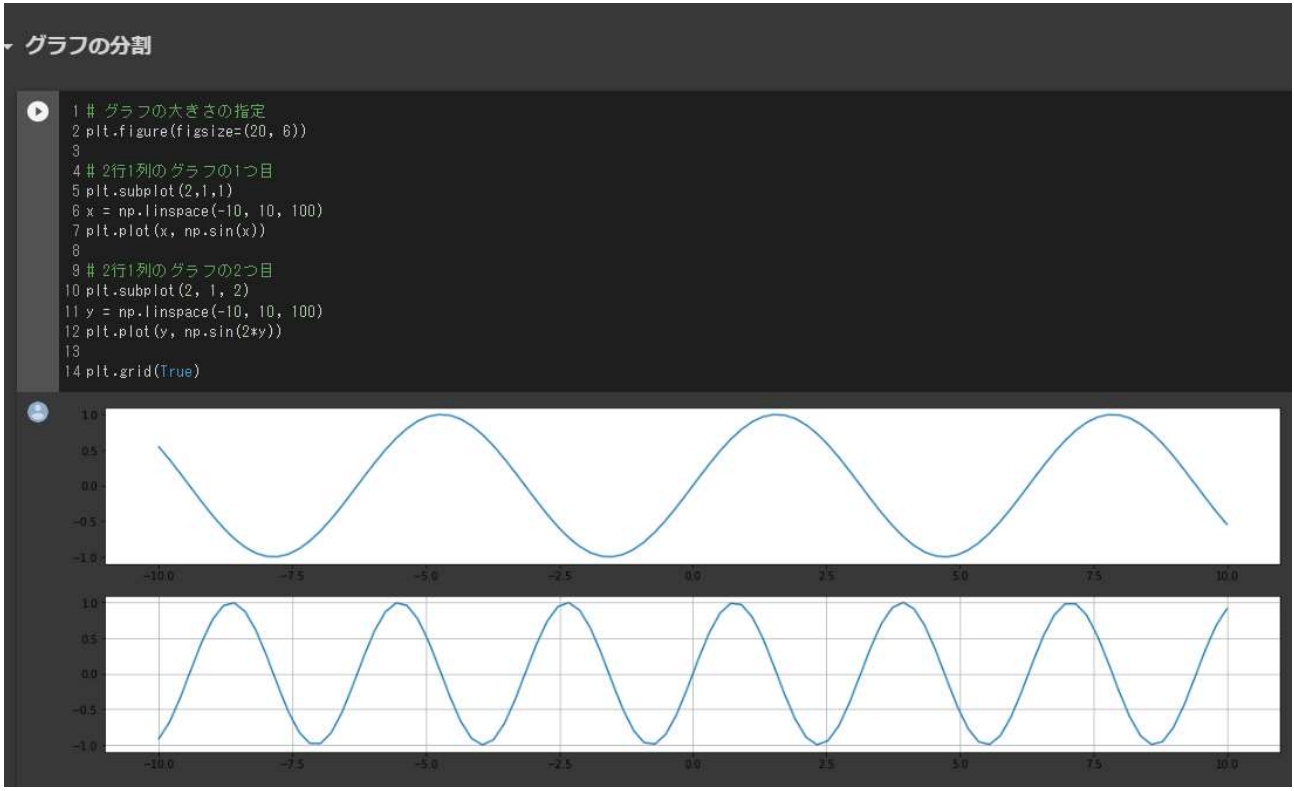


Matplotlib –グラフ分割の例–

subplot()を使い複数のグラフを同時に分割して描画できる

subplot()では引数で描画する場所を指定する

グラフの分割ではいくつかのデータの推移を比較したいときなどに使うことがある



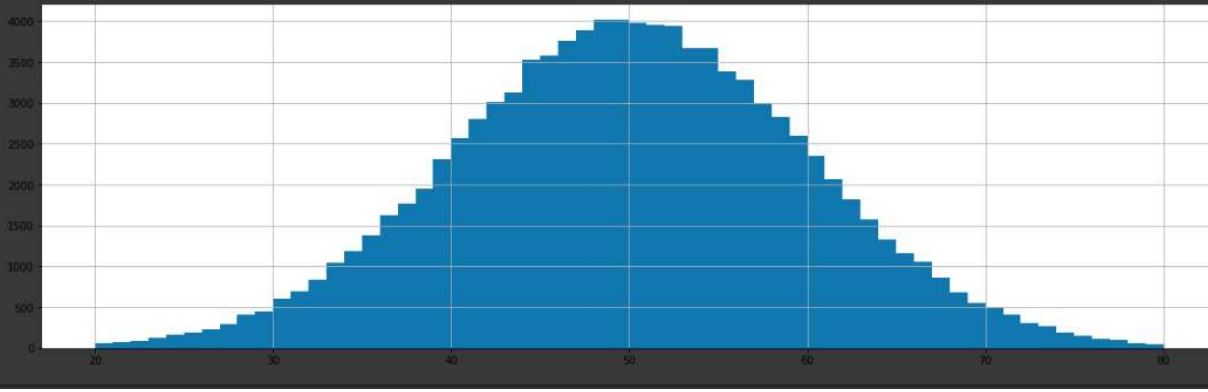
Matplotlib –ヒストグラムの例–

ヒストグラムではそれぞれの値の度数を示す
→データの全体像の把握
（どんな数値が多い？データの偏りは？など）

ヒストグラムはhistメソッドを使って描画する

ヒストグラム

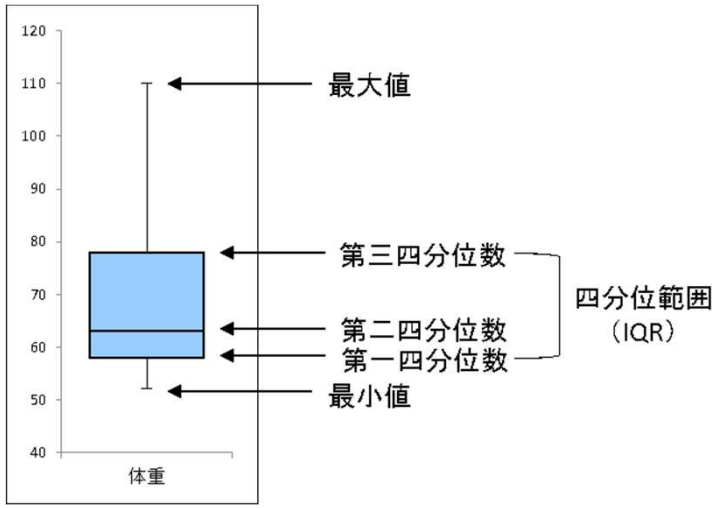
```
1 # シード値の固定
2 np.random.seed(0)
3
4 # グラフの大きさ指定
5 plt.figure(figsize=(20, 6))
6
7 # ヒストグラムの描画
8 plt.hist(np.random.randn(10 ** 5) * 10 + 50, bins=60, range=(20, 80))
9
10 plt.grid(True)
```

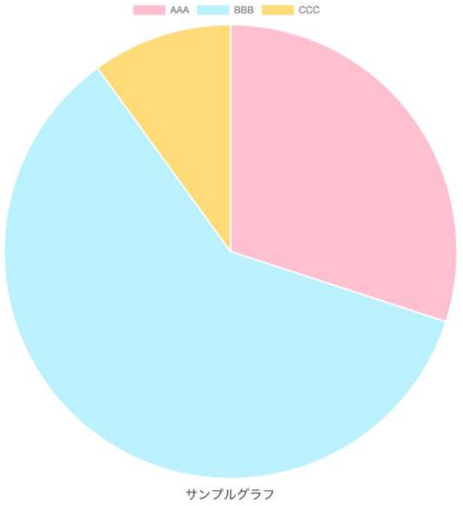


その他の描画について

今回は基本的なグラフのみでしたが、このほかにも
箱ひげ図や円グラフなどもデータ分析をするうえで
有効なものとなります

また、matplotlib以外にもseabornというライブラリ
もあるので、そちらも調べてみてください





おわりに

今回は基礎ということで、簡単な部分の紹介を行いました

しかし、実践では今回紹介したもの以外にも様々なライブラリやメソッドを使用していく必要があります

今村研究室では、様々なセンサデータを扱ったり、機械学習系のテーマを行う人は大量のデータを持ったデータセットなどを使う機会が多くあります

Pythonの講義内だけではレクチャーできないものが多いので各自自分で学習を進めることをお勧めします

参考文献

・東京大学のデータサイエンティスト育成講座　Pythonで手を動かして学ぶデータ分析
著:塚本邦尊、山田典一、大澤文孝

・https://qiita.com/
・https://techacademy.jp/magazine/
・https://note.nkmk.me/