

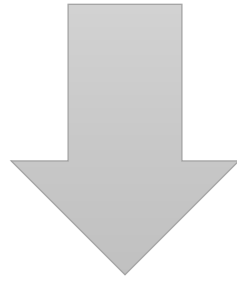
## 当コースの目的

データの前処理のほぼすべてに対応できること

## 当コースの対象者

- AIや機械学習に興味がある方
- Pythonを勉強したい方
- NumpyやPandasの使い方を学習したい方
- データの前処理を効率的にしたい方

なんでデータの前処理が対象なの??



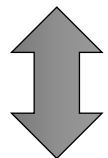
前処理は、データ分析プロジェクトのどのような仕事でも大部分を占める重要な部分だから！！

## ①要件定義

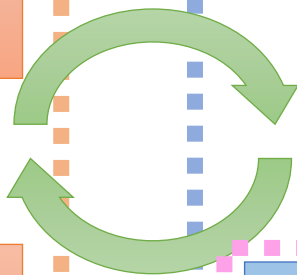
## ②データ分析

## ③実装

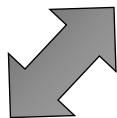
ヒアリング



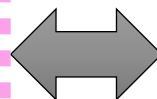
課題設計



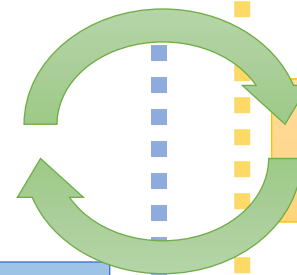
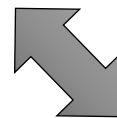
可視化



前処理



モデリング



システム開発

当コースのメイン！！

## 当コースの特徴

- とにかく現場主義！！
- コーディングはライブ形式！！
- コードだけでなく、イメージも！！



前処理スキル向上！！



## 当コース受講における注意

- 当コースは、前作「AIエンジニアが教えるRとtidyverseにおけるデータの前処理講座」の内容をPythonで書き換えたもの！！



同じ処理内容をRとPythonでコーディングすることにより、両者のメリット、デメリットを比較することができる！！

データサイエンス領域においてRとPythonは必須！！



VS



どちらがいいの??



一概に決めることはできない！！

自身で判断！！

# Python前処理コース

## 1. 環境構築



Googleアカウント  
が別途必要！！

## 2. Python



## 3. パッケージ





### 3. パッケージ



No	コアパッケージ	用途
1	numpy	数値計算
2	pandas	データフレーム処理
3	plotnine	可視化

# Python前処理コース

## 1. 環境構築



Googleアカウント  
が別途必要！！

## 2. Python



## 3. パッケージ



# Google Colaboratory

Google Colaboratoryってなんですか？？



Google Colaboratoryっていうのは、ブラウザ上でPythonを実行できるクラウドサービスのことだよ！！

Google Colaboratoryは、難しい環境構築がいらないからとても便利なんだ！！

Google Colaboratoryを略してColabと呼ぶこともあるよ！！

# Colabのメリット



1. 環境構築がほぼ不要
2. GPUが利用できる
3. コードとテキストを1つのドキュメントとして記述できる
4. 基本, 無料

より便利な有料プランもあるが, 多くの場合, 無料プランで事足りる

# Colabを利用するための準備

1. Udemyのリソースからzipファイルをダウンロード
2. 1.のzipファイルをデスクトップに解凍
3. Udemyのリソースからgoogleにアクセス  
(ご自身のgoogleアカウントにログインする.  
もしgoogleアカウントがない場合, 作成する.)
4. googleドライブにアクセス
5. 2.のデスクトップにあるフォルダをgoogleドライブにコピー

# Colabの基本的な使い方

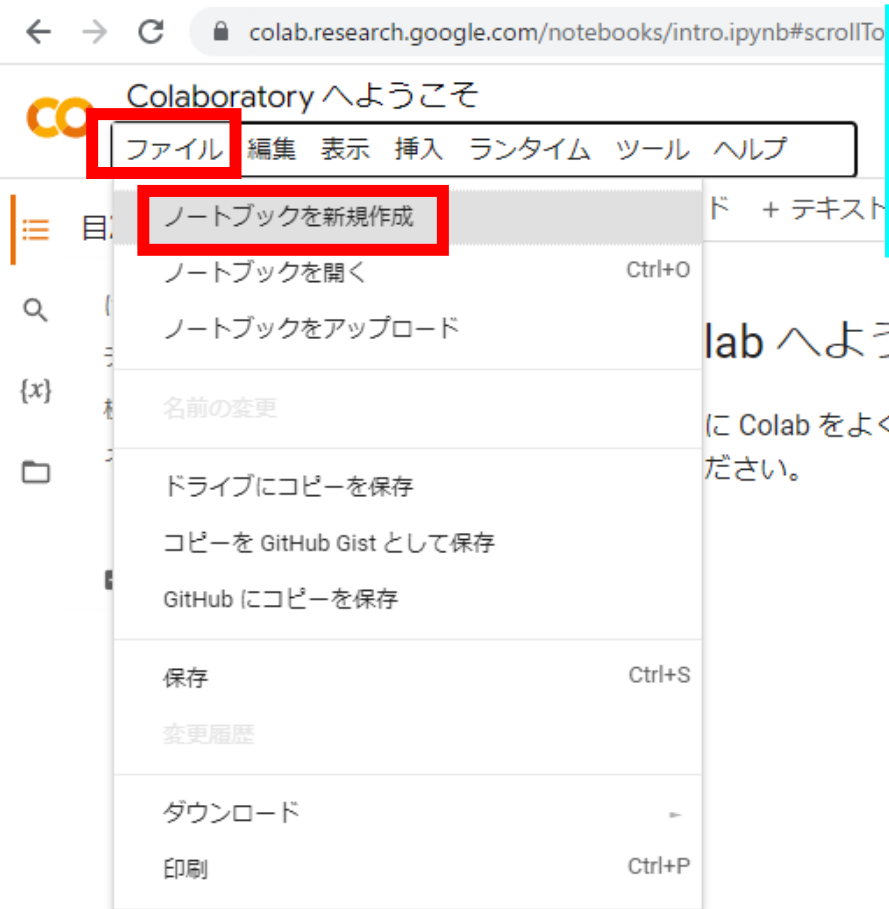


1. ノートブック作成
2. コーディング
3. スクラッチセル利用
4. ノートブック保存
5. ノートブック読み込み

# ノートブックの作成

ColabのURL:

<https://colab.research.google.com/notebooks/intro.ipynb>  
(Udemyのリソースに記載)



Colab へようこそ

に Colab をよくご存じの場合は、この動画でインタラクティブなテーブル、実行されたコードの履歴表示、コマンドパレットについてご  
ださい。



## Colab とは

Colab（正式名称「Colaboratory」）では、ブラウザ上で Python を記述、実行できます。以下の機能を使用できます。

- 環境構築が不要
- GPU への無料アクセス
- 簡単に共有

Colab は、**学生**から**データサイエンティスト**、**AIリサーチャー**まで、皆さんの作業を効率化します。詳しくは、[Colab の紹介動画](#)をご覧ください 15

# コーディング

CO Untitled0.ipynb ☆  
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

+ コード + テキスト

Pythonで  $1 + 1$  をしてみる.



0 秒  
1 1 + 1  
2

テキストセル

コードセル

以下のいずれかで実施

①コードセルの左上のボタンクリック

②Ctrl + Enter



# スクラッチセル利用

The screenshot shows the JupyterLab interface. At the top, there's a header with the Jupyter logo, the file name 'Untitled0.ipynb', and a star icon. Below this is a menu bar with options: ファイル, 編集, 表示, 挿入, ランタイム, ツール, ヘルプ, and a status message 'すべての変更を保存しました'. The main area has two tabs: '+ コード' and '+ テキスト'. The 'テキスト' tab is active, showing a text cell with the text 'Pythonで 1 + 1 をしてみる.' and a Python logo. This cell is highlighted with a yellow border and labeled 'テキストセル' in orange. Below it is a code cell with the code '[1] 1 1 + 1' and the output '2'. This cell is highlighted with a green border and labeled 'コードセル' in green. To the right, a 'スクラッチセル' (Scratch Cell) is shown in a blue-bordered box, containing a play button icon and the number '1', labeled 'スクラッチセル' in blue.

Pythonで 1 + 1 をしてみる.

python

テキストセル

[1] 1 1 + 1

2

コードセル

スクラッチセル x

1

スクラッチセル

以下のいずれかで表示される  
①挿入⇒スクラッチコードセル をクリック  
②Ctrl + Alt + N

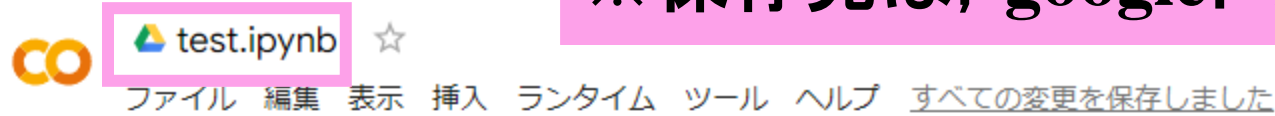
# ノートブック保存

ノートブック名を変更後、以下のいずれかで保存される。

①ファイル⇒保存 をクリック

②Ctrl + S

※保存先は、googleドライブの「Colab Notebooks」フォルダ



+ コード + テキスト

Pythonで  $1 + 1$  をしてみる.

 python テキストセル

コードセル

✓ [1] 1 1 + 1  
0 秒  
2

スクラッチセル ×

1

スクラッチセル

# ノートブック保存

The image shows the Google Drive web interface. On the left sidebar, there are navigation options: '新規' (New), 'マイドライブ' (My Drive), 'パソコン' (Computer), '共有アイテム' (Shared items), '最近使用したアイテム' (Recently used items), 'スター付き' (Starred), and 'ゴミ箱' (Trash). Below these is a '保存容量' (Storage) section showing '15 GB 中 2.33 GB を使用' (15 GB of 2.33 GB used) and a button '保存容量を購入' (Purchase storage). The main area shows the 'マイドライブ' (My Drive) view with a search bar 'ドライブで検索' and a list of folders. The folders listed are '無題のフォルダ' (Untitled folder), 'Colab Notebooks' (highlighted with a red box), and 'AIエンジニアが教えるPythonによるデータの前処理' (Data preprocessing with Python taught by AI engineers). A red arrow points from a red callout box containing the text 'ここに保存される。' (Saved here.) to the 'Colab Notebooks' folder.

ドライブ

ドライブで検索

新規

マイドライブ

パソコン

共有アイテム

最近使用したアイテム

スター付き

ゴミ箱

保存容量

15 GB 中 2.33 GB を使用

保存容量を購入

マイドライブ

名前 ↓

無題のフォルダ

Colab Notebooks

AIエンジニアが教えるPythonによるデータの前処理

ここに保存される。

# ノートブック読み込み



ドライブで検索



新規

マイドライブ

パソコン

共有アイテム

最近使用したアイテム

スター付き

ゴミ箱

保存容量

15 GB 中 2.33 GB を使用

保存容量を購入

マイドライブ > Colab Notebooks

名前 ↓

test.ipynb

プレビュー

アプリで開く

共有

リンクを取得

ファイルの場所を表示

ドライブにショートカットを追加

指定の場所へ移動

スターを追加

名前を変更

Google Colaboratory

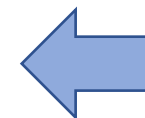
アプリを追加

パソコンにアプリ

ここをクリック

# 当コースの受講方法

1. 画面に従い、実際にコーディングする.



オススメ！！

2. 実際にコーディングせず、既存コードの実行をする.

3. 実際にコーディングも実行もせず、画面を見る.

# Python前処理コース

## 1. 環境構築



Googleアカウント  
が別途必要！！

## 2. Python



## 3. パッケージ





No.	内容
1	データ型
2	条件分岐処理
3	繰り返し処理
4	リスト
5	辞書
6	関数
7	クラス

# データ型

データ型ってなんですか？



データ型っていうのは、データの種類のことだよ！  
データの種類が明確になることで色々な処理が可能となるんだ！

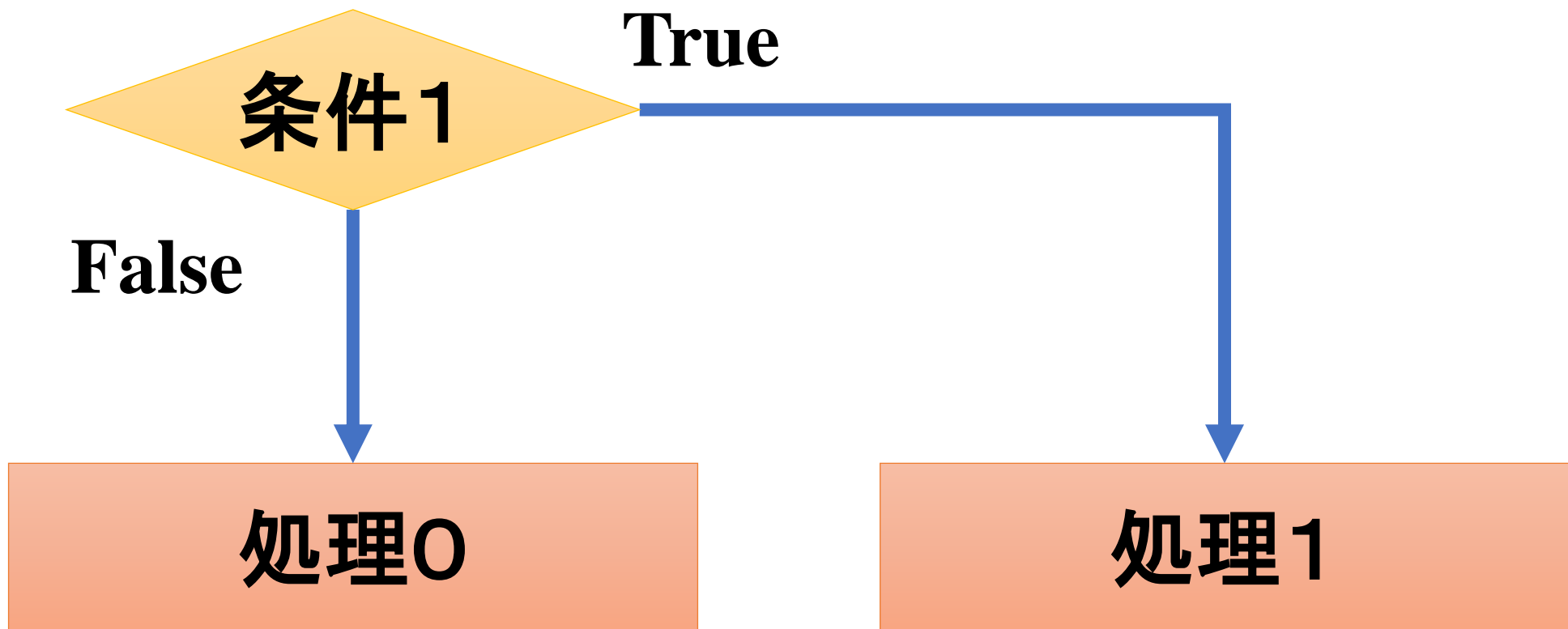




# 主なデータ型

データ型	意味	具体例
int	整数	1 200 9999
float	小数	1.36 3.14 33.9
str	文字列	“apple” “fish” “TOEIC”
bool	論理値	True False

関数typeを使用すると  
データ型を確認できる



# リスト

リストってなんですか？



リストはデータを格納する箱のようなものだよ！！  
プログラミングをする際、データをまとめて格納しておくとな便利ながことが多いんだ！！

基本, 同じデータ型

int型

1

5

8

13

4

float型

1.2

3.4

5.6

7.8

9.0

str型

e

i

g

h

t

bool型

True

False

False

True

False

複合型

1

3.4

g

True

False

# リストの作成

[データ1, データ2, データ3, データ4, データ5]

# リストの要素抽出

インデックス(前)

0

1

2

3

4

リスト

1

5

8

13

4

インデックス(後)

-5

-4

-3

-2

-1

## ①リスト[インデックス]

## ②リスト[スライス]

インデックス(前)

0

1

2

3

4

リスト

1

5

8

13

4

①リスト[4]

4

②リスト[1:4]

5

8

13

インデックスの1～3を表す。  
指定した終点(この場合, 4)が含まれないので注意！！

# 辞書

辞書ってなんですか？



辞書はリストと同様，データを格納する箱のようなものだよ！！  
リストと違い，インデックスではなく，キーでデータを管理するよ！！



# リストと辞書の違い

インデックス

リスト

0

1

2

3

4

データ

1

5

8

13

4

辞書

a

b

c

d

e

キー

# 辞書の作成

## 辞書

{キー1:データ1, キー2:データ2, キー3:データ3}

---

## リスト

[データ1, データ2, データ3]

基本, str型

a

b

c

d

e

int型

1

5

8

13

4

float型

1.2

3.4

5.6

7.8

9.0

str型

e

i

g

h

t

bool型

True

False

False

True

False

複合型

1

3.4

g

True

False

基本, 同じデータ型

# 辞書の要素抽出

リストと違い、  
スライスはない

キー

a

b

c

d

e

データ

1

5

8

13

4

辞書[e]

4

# オブジェクト作成時の注意

- listやdictは, Pythonの組み込み関数にすでに存在している.
- そのようなものをオブジェクト名に使用しない.
  - ※使用すると元の機能を上書きしてしまう...
- 判断に迷ったらオブジェクト名の後ろに\_1など付与すればOK

# 条件分岐処理

条件分岐処理ってなんですか？



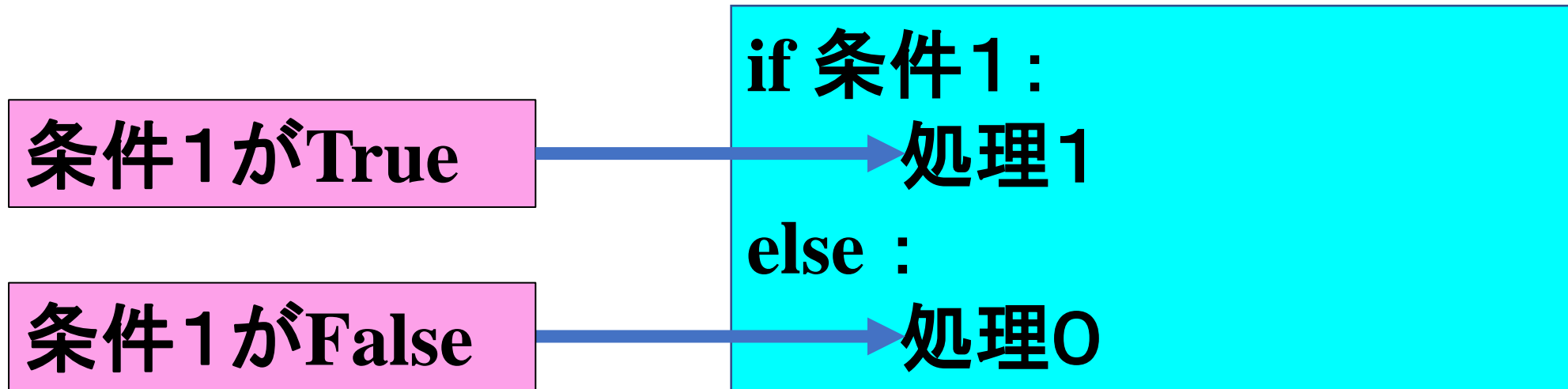
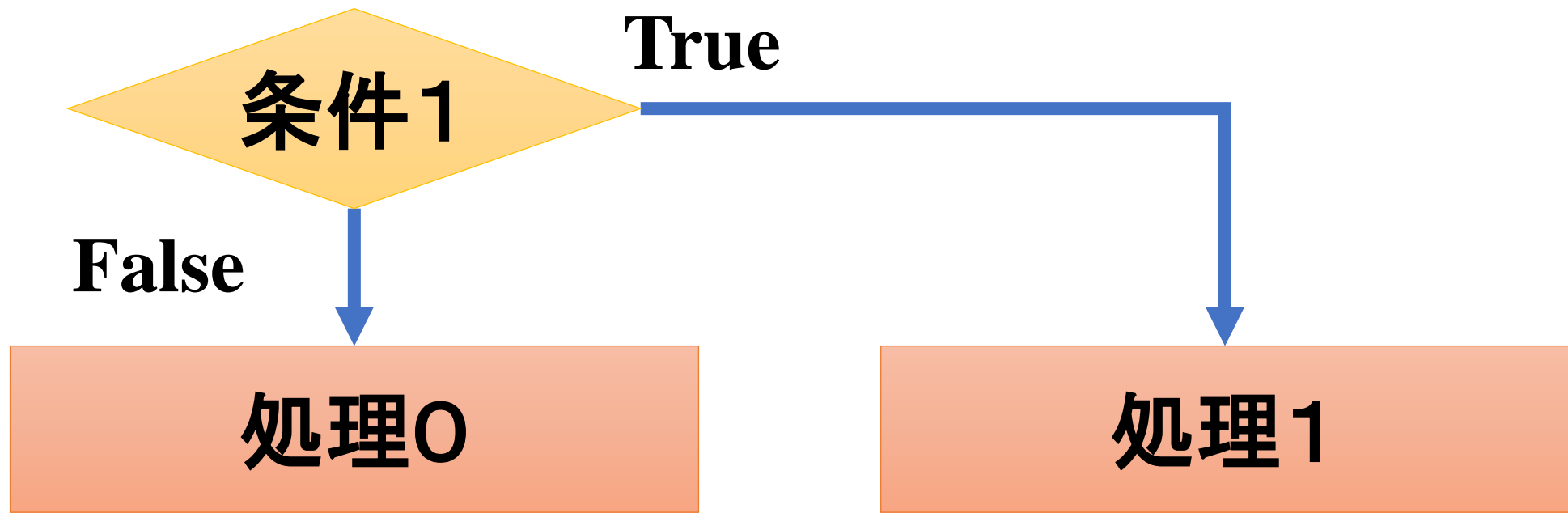
条件分岐処理っていうのは、条件によって処理内容を変えることだよ！！  
条件を設定することで様々な処理をプログラミングすることができるんだ！！  
条件分岐処理は、基本、if, elif, elseを使用して記述するよ！！

# 条件の判定

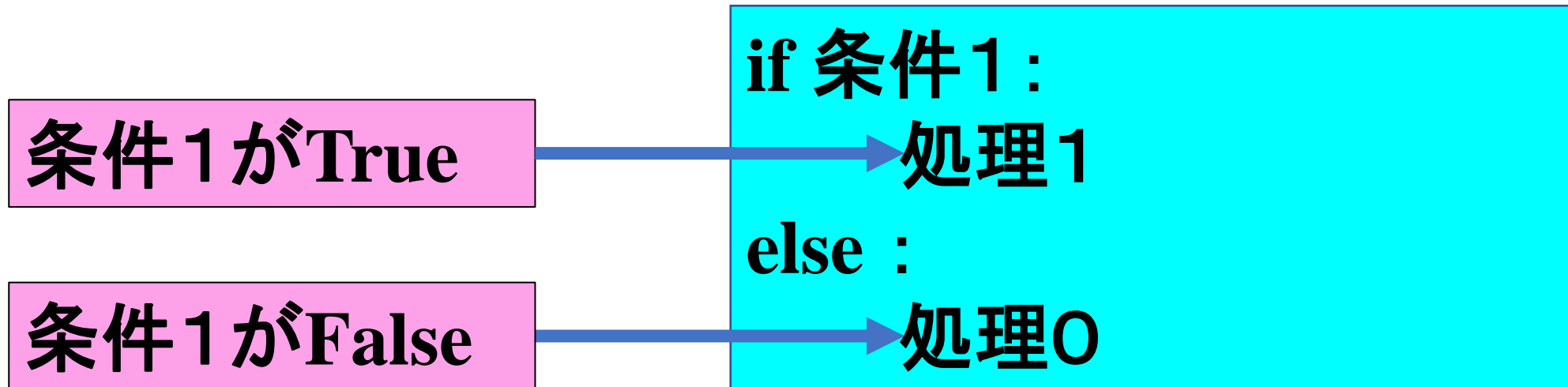
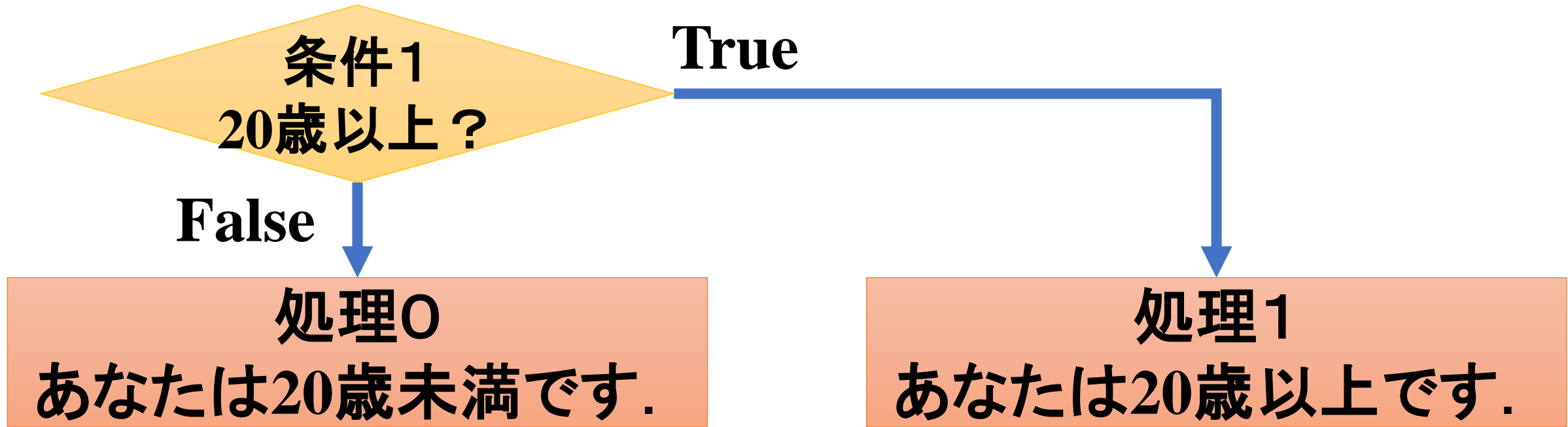
一部のみ抜粋！！

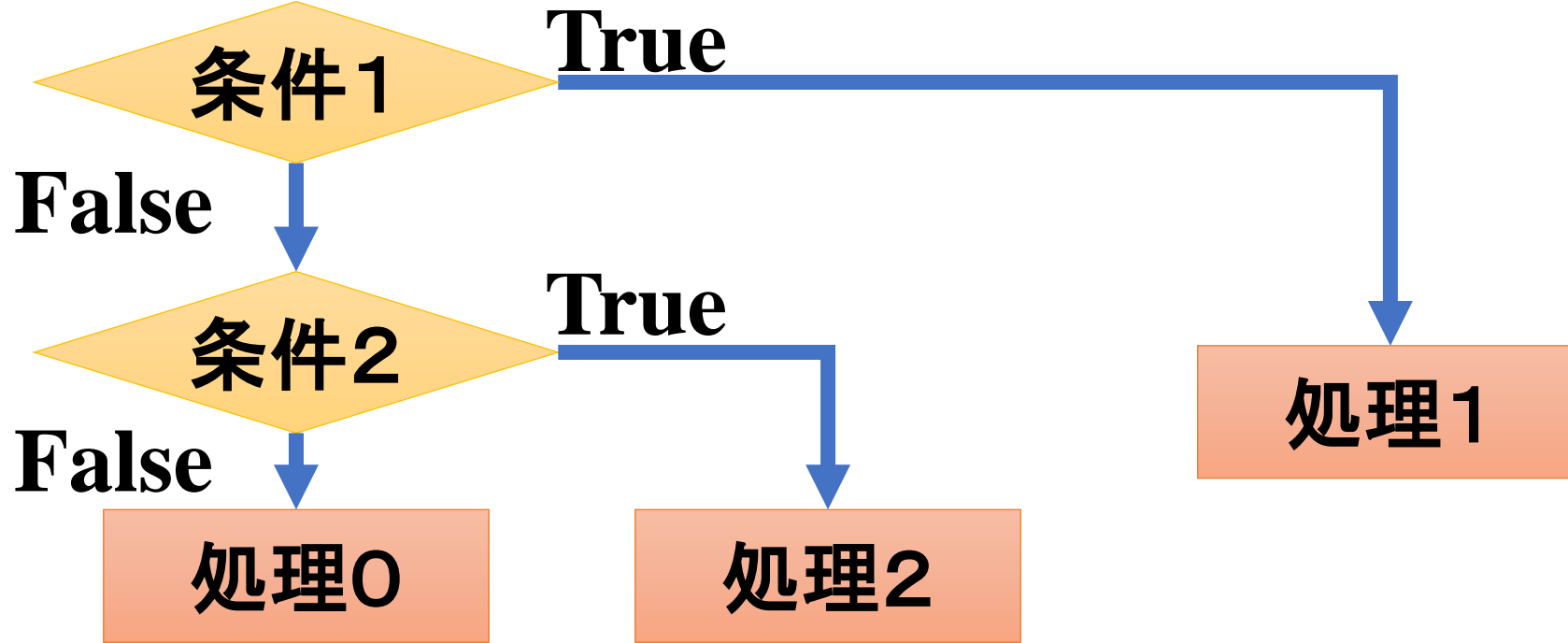
比較演算子	構文	意味
==	$x == y$	xとyは、等しいか？
!=	$x != y$	xとyは、等しくないか？
>=	$x >= y$	xはy以上か？
>	$x > y$	xはyより大きいのか？
<=	$x <= y$	xはy以下か？
<	$x < y$	xはyより小さいか？

True or False







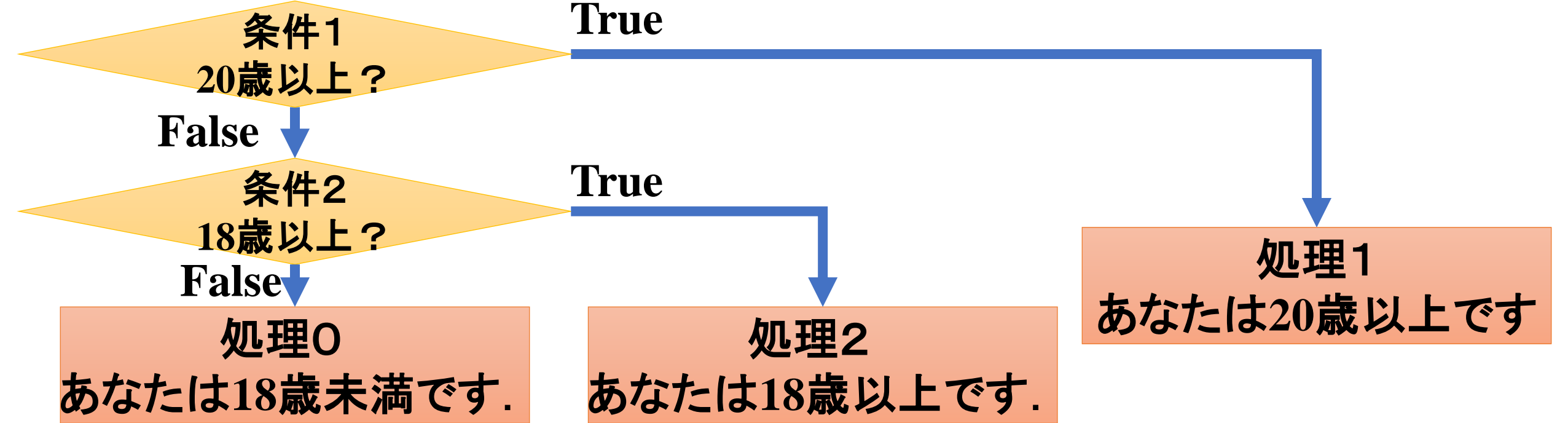


条件1がTrue

条件1がFalse, 条件2がTrue

条件1がFalse, 条件2がFalse

```
if 条件1:  
    処理1  
elif 条件2:  
    処理2  
else:  
    処理0
```



条件1がTrue

条件1がFalse, 条件2がTrue

条件1がFalse, 条件2がFalse

```
if 条件1:  
    処理1  
elif 条件2:  
    処理2  
else:  
    処理0
```

# 繰り返し処理

繰り返し処理ってなんですか？



繰り返し処理っていうのは、同じような処理を繰り返し実施することだよ！！  
繰り返し処理を書くことでコードを短く、きれいに書くことができるんだ！！  
繰り返し処理は基本，forを使用して記述するよ！！

## for 不使用

```
▶ 1 # forを使用しない
2 print("おはようございます！")
3 print("おはようございます！")
4 print("おはようございます！")
5 print("おはようございます！")
6 print("おはようございます！")
7 print("おはようございます！")
8 print("おはようございます！")
9 print("おはようございます！")
10 print("おはようございます！")
```

おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！

## for 使用

```
▶ 1 # forを使用する
2 for i in range(10):
3     print("おはようございます！")
```

おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！  
おはようございます！

**for**

処理 0

処理 1

処理 i

処理 n

リスト

i

0

...

i

...

n

data

data 0

...

data i

...

data n

**for data in リスト:**  
**処理 i**

リスト

i

data

0

1

2

3

4

“zero”

“one”

“two”

“three”

“four”

```
for data in リスト:  
    print(data)
```

0

print(“zero”)

1

print(“one”)

2

print(“two”)

3

print(“three”)

4

print(“four”)

# rangeとappend

連番のリスト(のようなもの)を作成

`range(4)`



`[0, 1, 2, 3]`

`[0, 1, 2, 3].append(4)`



`[0, 1, 2, 3, 4]`

リストの最後に要素を追加



# リスト内包表記

リストを生成するための記法

[0, 2, 4, 6, 8]というリストを作成する.

## リスト内包表記 不使用

```
1 list_3 = []  
2 for i in range(5):  
3     list_3.append(i*2)
```

## リスト内包表記 使用

```
1 [i*2 for i in range(5)]
```

# 関数

関数ってなんですか？？



関数は、入力に対してあらかじめ定めた処理を適用することで出力を返す命令のことだよ！！

関数を使用することで、コードをきれいに書けたり、コードの修正が簡単になったりするんだ！！

# 関数

入力 =  $x$

1



2



⋮

⋮

9



処理

$$x + 1$$

出力 =  $y$

2



3



⋮

⋮

10



# 関数: add\_1

入力 =  $x$

処理

出力 =  $y$

1



$x + 1$



2

2



3

```
def add_1 (x) :  
    y = x + 1  
    return(y)
```

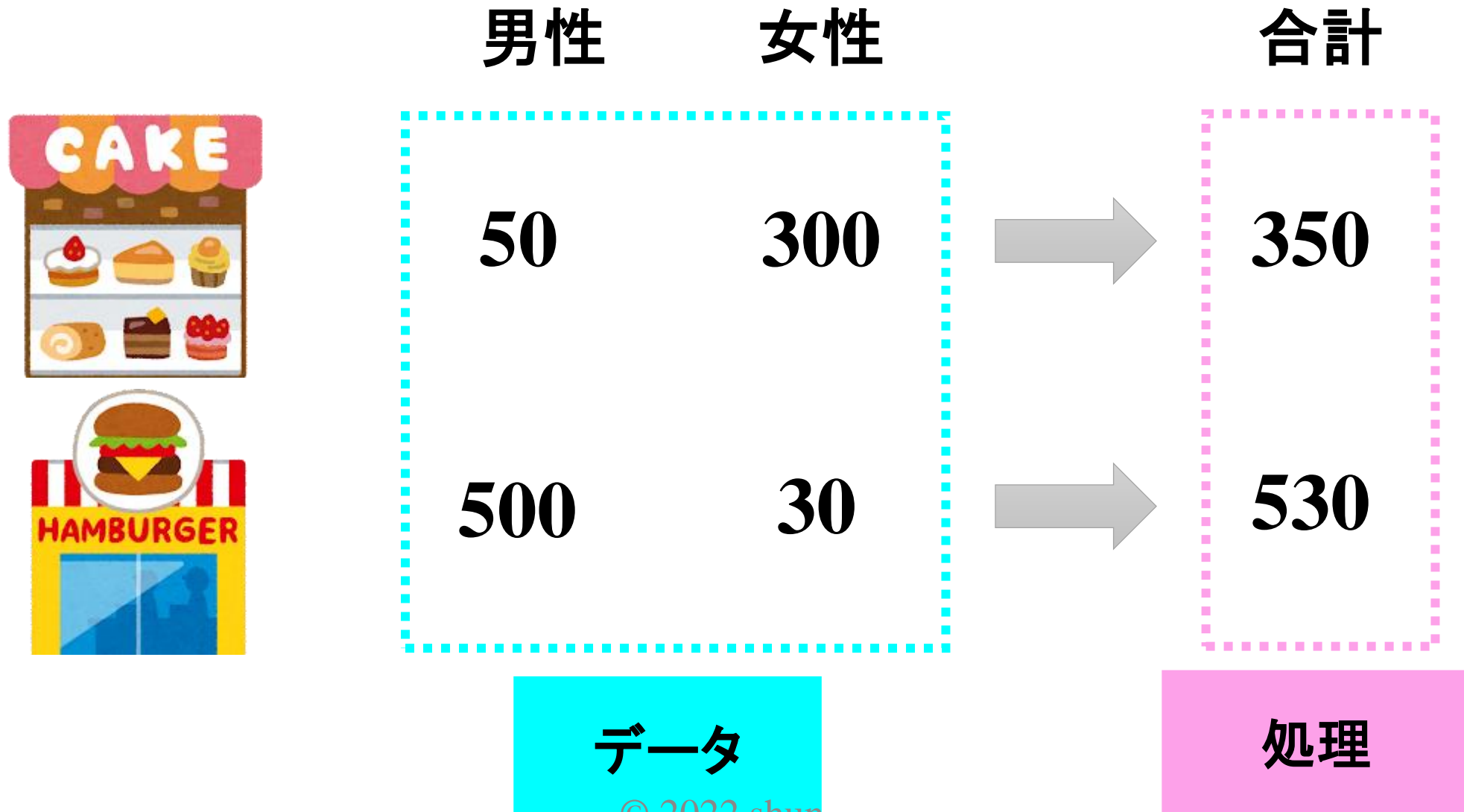
# クラス

クラスってなんですか？？



クラスはデータと処理をまとめた機能のことだよ！！  
クラスを使用することで、同じような処理をきれいに書くことができるんだ！！

# 1日当たりの男女別来客数



## クラス 不使用

```
1  # ケーキ屋
2  cake_male = 50
3  cake_female = 300
4  cake_total = cake_male + cake_female

1  # ハンバーガーショップ
2  burger_male = 500
3  burger_female = 30
4  burger_total = burger_male + burger_female
```

## クラス使用

```
1  class Shop:
2      # コンストラクター
3      def __init__(self, male, female):
4          # インスタンス変数
5          self.male = male
6          self.female = female
7      # メソッド
8      def total(self):
9          return(self.male + self.female)

1     # インスタンス
2     cake = Shop(male = 50, female = 300)
3
4     burger = Shop(male = 500, female = 30)
```

設計物 1

インスタンス 1

設計図

クラス



男性	女性	合計
----	----	----



50	300	→ 350
----	-----	-------



500	30	→ 530
-----	----	-------

インスタンス変数

メソッド

データ

処理

設計物 2

インスタンス 2

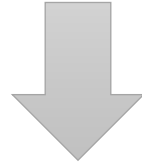
```
1 class Shop:
2     # コンストラクター
3     def __init__(self, male, female):
4         # インスタンス変数
5         self.male = male
6         self.female = female
7     # メソッド
8     def total(self):
9         return(self.male + self.female)
```

```
1 # インスタンス
2 cake = Shop(male = 50, female = 300)
3
4 burger = Shop(male = 500, female = 30)
```



# 関数やクラスのオススメ作成方法

関数もクラスも難しい！！



まずは関数やクラスを使用しないでコーディングして、  
長くなったら関数化，もしくはクラス化を検討する！！

気楽に使用していこう！！

# Python前処理コース

## 1. 環境構築



Googleアカウント  
が別途必要！！

## 2. Python



## 3. パッケージ



### 3. パッケージ



No	コアパッケージ	用途
1	numpy	数値計算
2	pandas	データフレーム処理
3	plotnine	可視化



パッケージ

ライブラリ

どっちでもOK！！

# numpy



numpyってなんですか？？



numpyは数値計算をするためのパッケージのことだよ！！  
numpyを使用することで、ベクトルや行列計算ができるようになるんだ！！

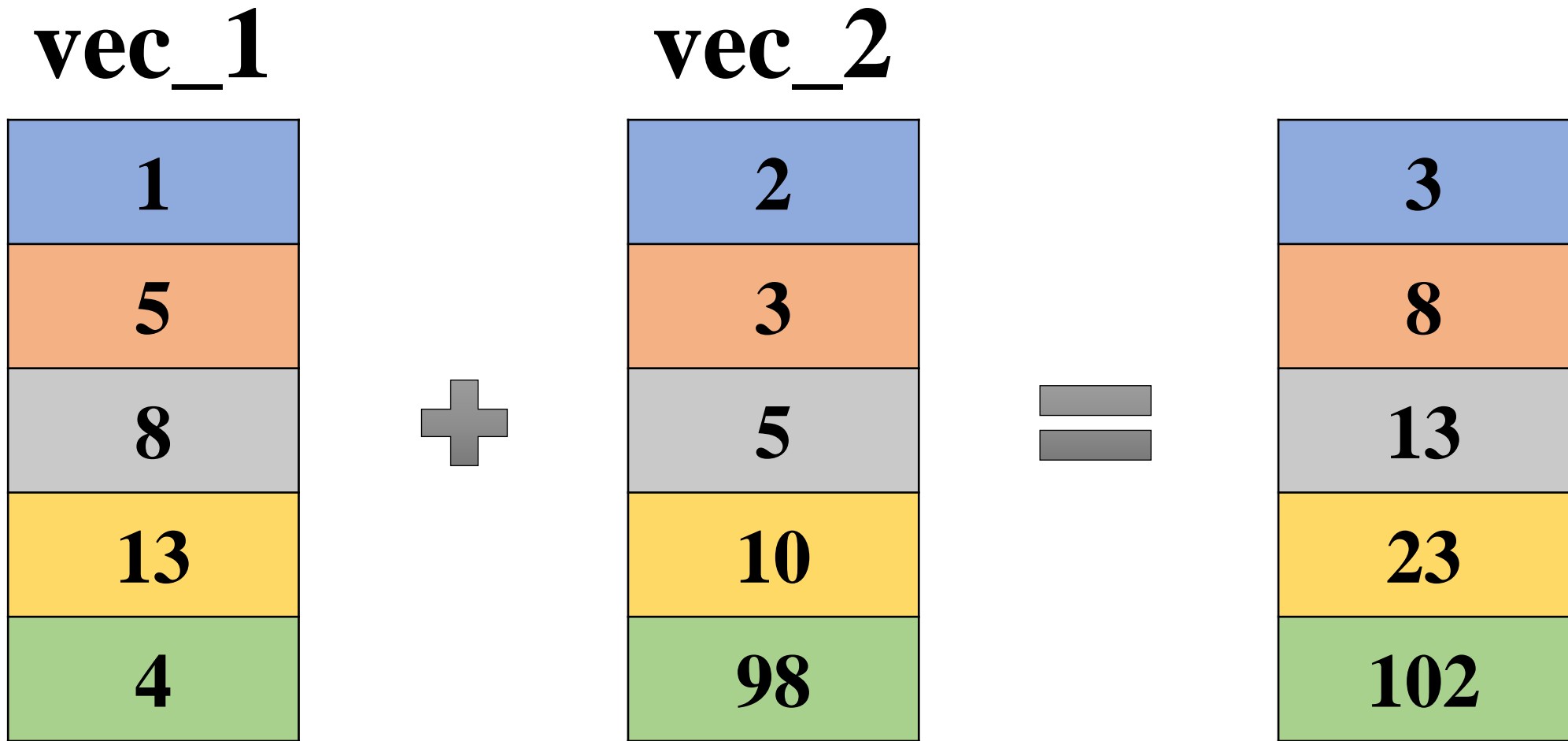
## ベクトル

メイン

1	5	8	13	4
---	---	---	----	---

## 行列

1	5	8	13	4
2	10	16	26	8
3	15	24	39	12



# 四則演算

算術演算子	説明
+	足し算
-	引き算
*	掛け算
/	割り算

一部のみ抜粋！！



# パッケージのインポート

## 基本

**import パッケージ as 名前**

```
1  import numpy as np
2  import pandas as pd
```

# ベクトルの作成方法

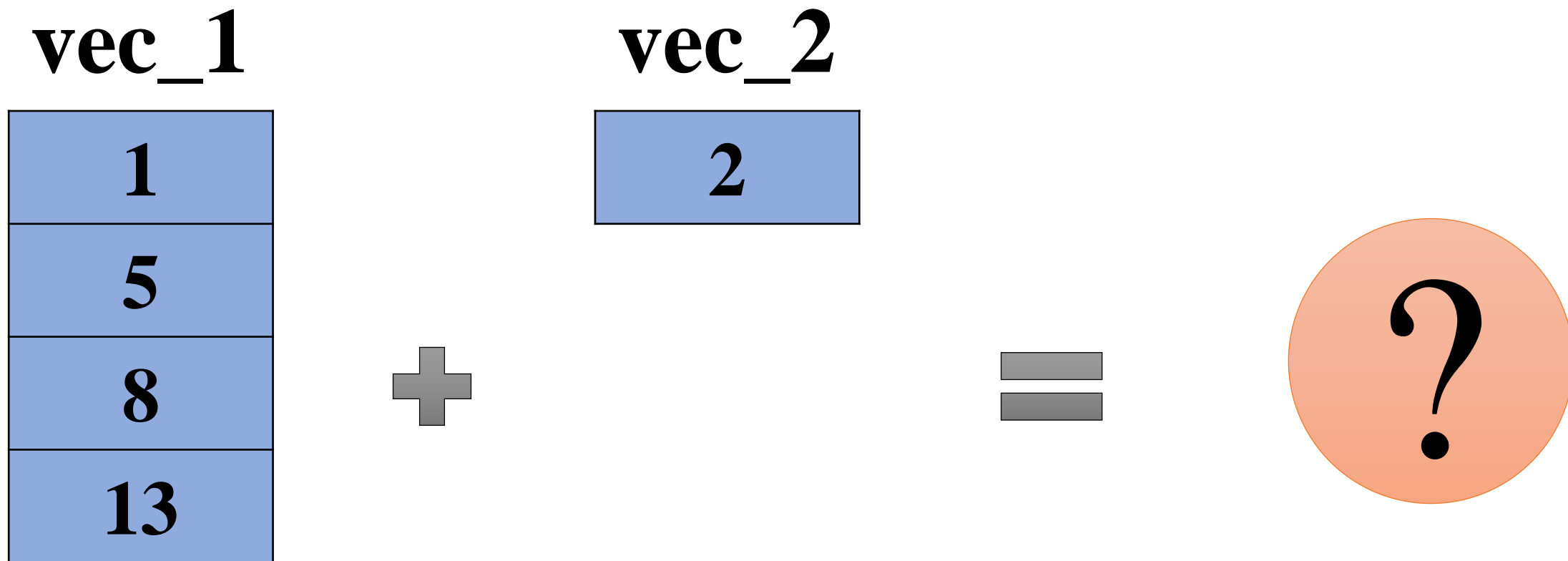
## 基本

**np.array(リスト)**

```
np.array([1, 5, 8, 13, 4])
```

```
np.array([2, 3, 5, 4, 98])
```

# ベクトルのブロードキャスト



before

1
5
8
13

+

2

ただの数値でもOK

足りない...

ブロードキャスト

after

1
5
8
13

+

2
2
2
2

上のベクトルで補完！！

**vec\_1**

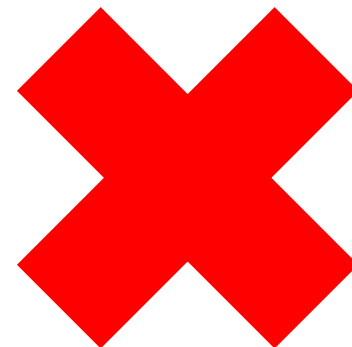
1
5
8
13

+

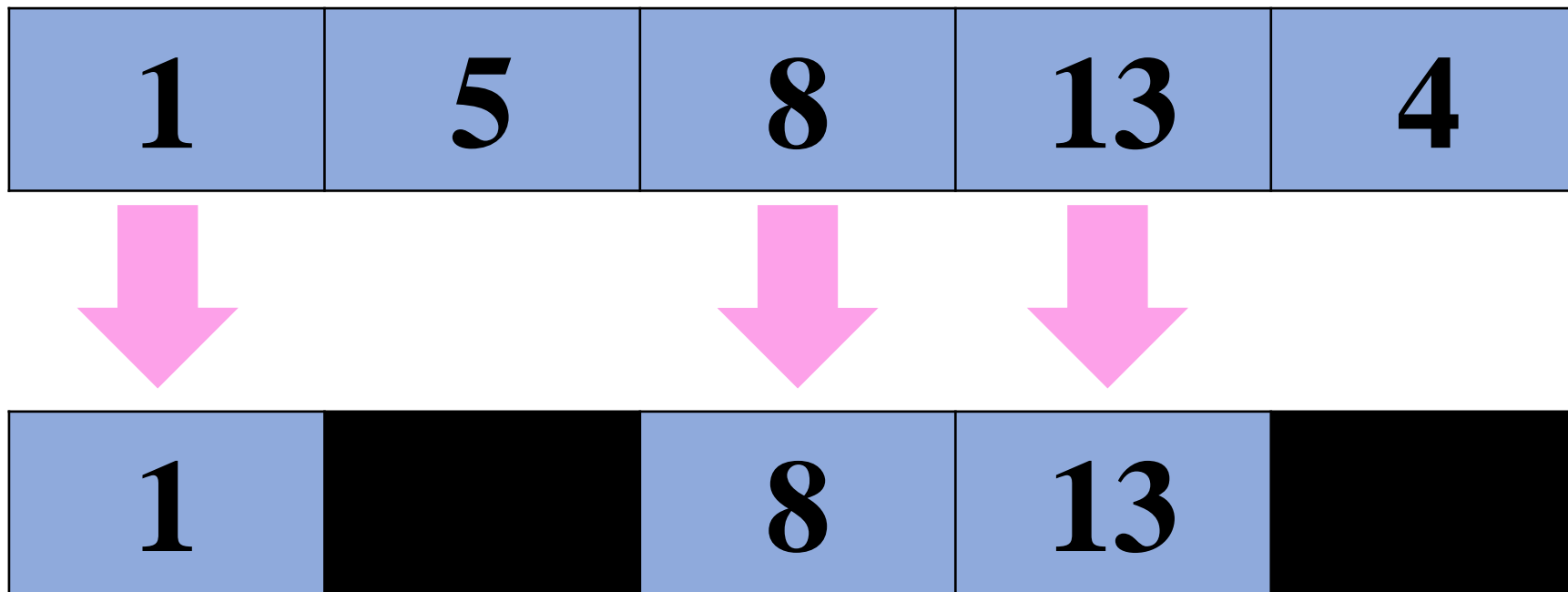
**vec\_3**

2
3

=



# ベクトルの要素抽出



- ①インデックスで指定
- ②スライスで指定
- ③論理値ベクトルで指定

## ①ベクトル[インデックス]

## ②ベクトル[スライス]

インデックス(前)

0

1

2

3

4

ベクトル

1

5

8

13

4

①ベクトル[4]

4

②ベクトル[1:4]

5

8

13

インデックスの1～3を表す。  
指定した終点(この場合, 4)が含まれないので注意！！

インデックス

0

1

2

3

4

ベクトル

1

5

8

13

4

論理値ベクトル

T

F

T

T

F



1

8

13



# ベクトルの便利関数

No.	関数	意味
1	np.sum	総和
2	np.average	平均
3	np.median	中央値
4	np.max	最大値
5	np.min	最小値
6	np.var	分散
7	np.std	標準偏差

論理値ベクトル

T

F

T

T

F

1

0

1

1

0

np.sum



3

Tの個数

np.average



0.6

Tの割合

**n: 整数**  
**x,y,z: 数値**  
**vec: ベクトル**

# 規則的なベクトルの作成

関数	使い方	意味
np.arange	np.arange(x, y, z)	xからyまでの等差zの等差ベクトル
np.linspace	np.linspace(x, y, n)	xからyまで長さnの等差ベクトル
np.repeat	np.repeat(vec, n)	vecの要素をn回繰り返す.
np.tile	np.tile(vec, n)	vecをn回繰り返す.

どのように論理値ベクトルを  
作成するのか??



条件を満たしたかどうかでTRUEと  
FALSEを判定し, その判定結果を論理値  
ベクトルとする.

季節は春か?

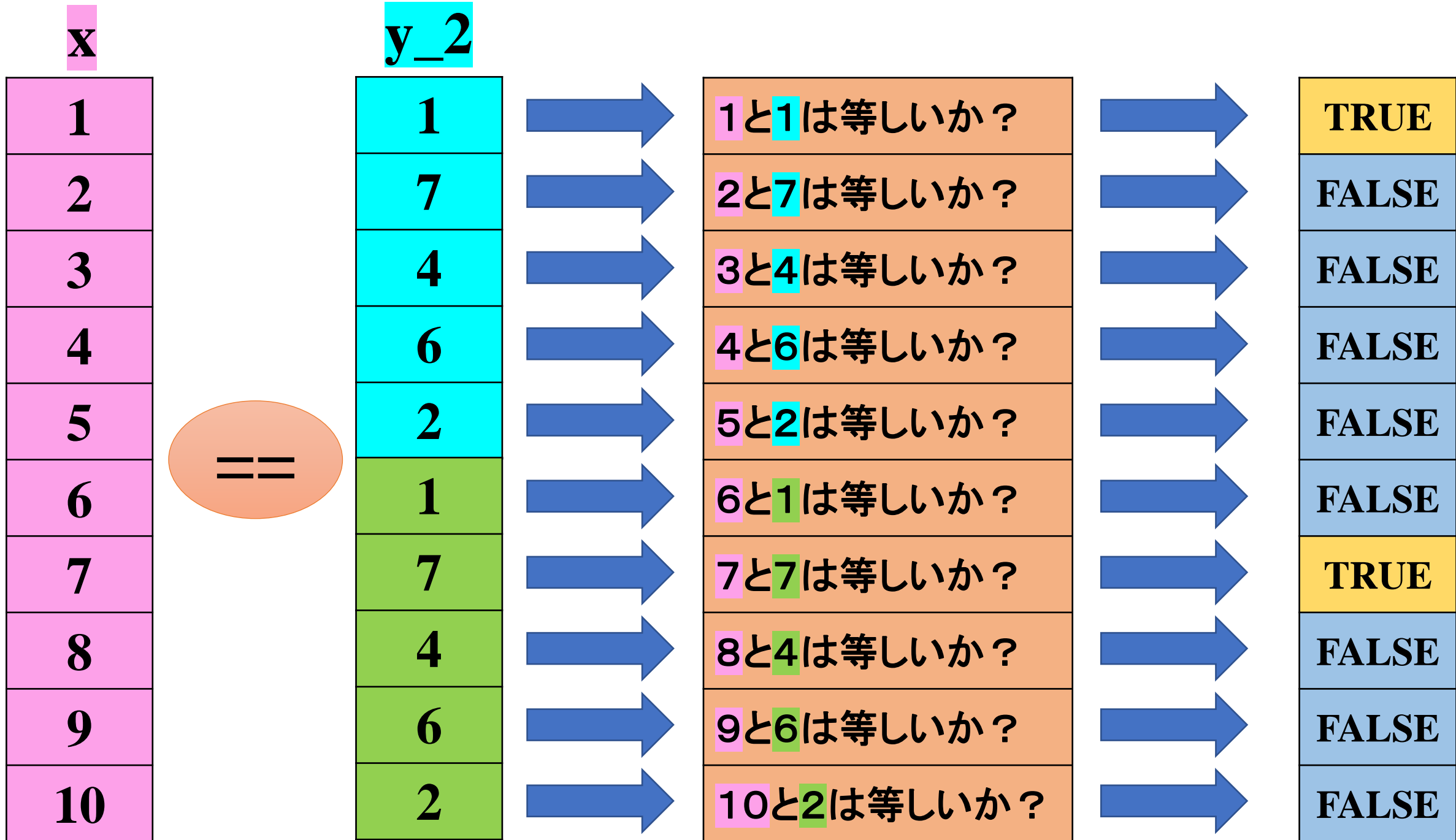
気温は5°C以上か?

年齢は25歳以下か?

$x, y$ : ベクトル

# 比較演算子

比較演算子	構文	意味
<code>==</code>	<code>x == y</code>	$x$ と $y$ は, 等しいか?
<code>!=</code>	<code>x != y</code>	$x$ と $y$ は, 等しくないか?
<code>&gt;=</code>	<code>x &gt;= y</code>	$x$ は $y$ 以上か?
<code>&gt;</code>	<code>x &gt; y</code>	$x$ は $y$ より大きいのか?
<code>&lt;=</code>	<code>x &lt;= y</code>	$x$ は $y$ 以下か?
<code>&lt;</code>	<code>x &lt; y</code>	$x$ は $y$ より小さいか?
<code>np.isin</code>	<code>np.isin(x, y)</code>	$x$ は $y$ に含まれているか?





# 論理値ベクトルの計算

論理値ベクトルの計算？  
ということ？



論理値ベクトルは、組み合わせたり、反転させたり、TRUEの有無を確認したり、いろいろな計算方法があるんだ。

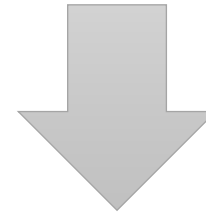
論理値ベクトルを組み合わせるというのは、2つの論理値ベクトルに対して、とある計算をすることで、1つの新たな論理値ベクトルを作成することをいうよ。



## キャベツのサイズ

7
13
5
25
38
57
18
9
32
28

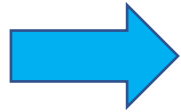
10以上30以下のサイズを  
抽出したい！！



- ① 10以上のキャベツかどうかを判定する.
- ② 30以下のキャベツかどうかを判定する.
- ③ ①と②を両方満たしたキャベツを市場に出せると判定する.

## キャベツのサイズ

7
13
5
25
38
57
18
9
32
28

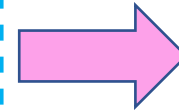


①10以上か？

F
T
F
T
T
T
T
F
T
T

②30以下か？

T
T
T
T
F
F
T
T
F
T



③市場出荷OKか？

F
T
F
T
F
F
T
F
F
T

※ xとyは, 論理値ベクトルを表す.

論理演算子	構文	意味	返り値
&	x & y	両方TRUE $\Rightarrow$ TRUE それ以外 $\Rightarrow$ FALSE	論理値ベクトル
	x   y	1つがTRUE $\Rightarrow$ TRUE 全部FALSE $\Rightarrow$ FALSE	論理値ベクトル
^	x ^ y	1つがTRUE, 1つがFALSE $\Rightarrow$ TRUE 両方ともTRUEもしくはFALSE $\Rightarrow$ FALSE	論理値ベクトル
~	~ x	TRUE $\Rightarrow$ FALSE FALSE $\Rightarrow$ TRUE	論理値ベクトル
np.any	np.any(x)	1つ以上TRUE $\Rightarrow$ TRUE 全部FALSE $\Rightarrow$ FALSE	論理値スカラー
np.all	np.all(x)	全部TRUE $\Rightarrow$ TRUE 1つ以上FALSE $\Rightarrow$ FALSE	論理値スカラー

### 3. パッケージ



No	コアパッケージ	用途
1	numpy	数値計算
2	pandas	データフレーム処理
3	plotnine	可視化

# pandas



pandasってなんですか？？

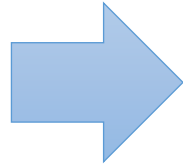


pandasはデータフレーム処理をするためのパッケージのことだよ！！  
pandasを使用することで、スムーズにデータ分析ができるようになるんだ！！

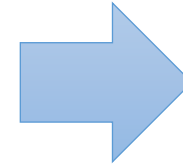
# データフレーム処理




①行の抽出




②列の追加




データフレーム

# シリーズとデータフレーム

シリーズ


データフレーム


メイン

# pandasを利用する際の注意点

## pandasの注意点



同じ処理でもコーディングの仕方が色々ある



あまり良くないコーディング方法 ⇒ bad  
オススメのコーディング方法 ⇒ good



# badとgoodの判定基準

コードの可読性

常に正しいわけではない

bad:読みづらい  
good:読みやすい

badとgoodの切り分けは独断と偏見

システム実装時は、判定  
が入れ替わることもある

ただの感想で一般的な  
基準ではない

badでもgoodでも結果は同じ⇒どちらでもOK  
色々試して自分なりのベストを見つけよう！！

# データの入出力

データ

データフレーム

加工前



入力


加工

加工後



出力


# csv入出力

csv



入力

```
df = pd.read_csv(csvのパス)
```

df


加工

index=False

df\_rev


csv\_rev



出力

```
df_rev.to_csv(csv_revのパス)
```

# データフレームの作成

## データフレーム

```
pd.DataFrame(  
{キー1: リスト1, キー2: リスト2, キー3: リスト3}  
)
```

キー1	キー2	キー3
リスト1	リスト2	リスト3

ベクトルなどの  
他の形式でも  
OK

# メソッドチェーン

メソッドチェーンってなんですか？？



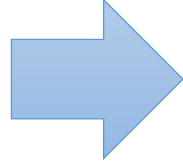
メソッドチェーンとは、メソッドを重ね合わせることでデータフレームを連続して処理することだよ！！

メソッドチェーンを使用することで可読性と保守性が高いきれいなコードを書くことができるんだ！！

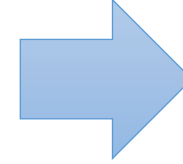
# df

# df\_rev

①列の更新



②行の抽出



メソッドチェーン

df.メソッド1.メソッド2.メソッド3.・・・.メソッドn

メソッドを鎖(チェーン)のように繋いでいく  
⇒メソッドチェーン

①列の更新

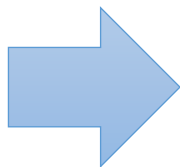
②行の抽出

```
1 df_rev = ¥
2 df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"]).query("sepal_length > 8")
```

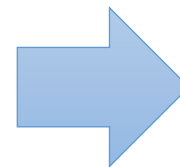
# df

# df\_rev

①列の更新



②行の抽出



メソッドチェーン

good

```
1 df_rev = df
2 df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"]).query("sepal_length > 8")
```

再帰代入

bad

```
1 df = df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"])
2 df_rev = df.query("sepal_length > 8")
```

# 再帰代入のデメリット

## 1回目

```
1 df = df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"])
2 df_rev = df.query("sepal_length > 8")
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	8.6	3.5	1.4	0.2	setosa

## 2回目

```
1 df = df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"])
2 df_rev = df.query("sepal_length > 8")
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	12.1	3.5	1.4	0.2	setosa

dfが書き変わっている  
ので結果が変わる...



# メソッドチェーンの改行

df.メソッド1.メソッド2.メソッド3....メソッドn

①列の更新

②行の抽出

```
2 | df.assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"]).query("sepal_length > 8")
```

¥(もしくはバックスラッシュ)で改行

改行

df¥  
.メソッド1¥  
.メソッド2¥  
.メソッド3¥  
....¥  
.メソッドn

```
2 df¥ ①列の更新  
3 .assign(sepal_length = lambda x:x["sepal_length"] + x["sepal_width"])¥  
4 .query("sepal_length > 8") ②行の抽出
```

# データフレーム処理



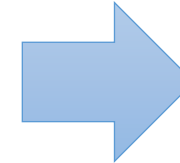
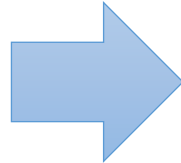
様々な加工の一例

df

new\_df

①行の抽出

②列の追加





No.	メソッド	加工内容
1	query	行の抽出
2	filter	列の抽出
3	assign	列の追加・更新
4	merge	キー結合
5	concat	縦横結合
6	group_by	グルーピング
7	sort_values	ソート
8	drop_duplicates	重複削除
9	melt, pivot	縦横変換

# query: 行の抽出

**df**



**df.query(条件)**

**query**

条件を満たした行が  
抽出される。

# queryの内部処理

- ① 条件を判定し, 行ごとに論理値ベクトル(True, False)を作成する.
- ② ①の論理値ベクトルをデータフレームに記録する.
- ③ ②でTrueであった行のみを抽出する.

df		
No	gender	height
1	M	165
2	F	150
3	F	170
4	M	175
5	F	165
6	M	195
7	M	180

```
df.query(
  "gender == 'F'"
)
```

query

①

gender	条件	判定結果
M	F	False
F	F	True
F	F	True
M	F	False
F	F	True
M	F	False
M	F	False

②

No	gender	height
1	M	165
2	F	150
3	F	170
4	M	175
5	F	165
6	M	195
7	M	180

判定結果
False
True
True
False
True
False
False

③

No	gender	height	判定結果
2	F	150	True
3	F	170	True
5	F	165	True

左側が抽出される

# 比較演算子

比較演算子	構文	意味
<code>==</code>	<code>x == y</code>	xとyは, 等しいか?
<code>!=</code>	<code>x != y</code>	xとyは, 等しくないか?
<code>&gt;=</code>	<code>x &gt;= y</code>	xはy以上か?
<code>&gt;</code>	<code>x &gt; y</code>	xはyより大きいのか?
<code>&lt;=</code>	<code>x &lt;= y</code>	xはy以下か?
<code>&lt;</code>	<code>x &lt; y</code>	xはyより小さいか?
<code>in</code>	<code>x in y</code>	xはyに含まれているか?

## good

① `df.query("price == 337")`

② `df.query("depth >= 62 & color == 'H'")`

## bad

① `df[df["price"] == 337]`

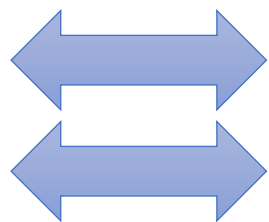
② `df[(df["depth"] >= 62) & (df["color"] == "H")]`

- ・コードが長い.
- ・可読性が低い.
- ・名前が変わったときに修正が大変.



## ベクトル1

I
J
J
I
H
H
J
J



•  
•  
•



IかJはあるか？

## ベクトル2

I	J
I	J
I	J
I	J
I	J
I	J
I	J
I	J



•  
•  
•



ある⇒True  
ない⇒False

## 論理値ベクトル

True
True
True
True
False
False
True
True

ここが  
作成

※ xとyは, 論理値ベクトルを表す.

論理演算子	構文	意味	返り値
&	$x \ \& \ y$	両方TRUE $\Rightarrow$ TRUE それ以外 $\Rightarrow$ FALSE	論理値ベクトル
	$x \   \ y$	1つがTRUE $\Rightarrow$ TRUE 全部FALSE $\Rightarrow$ FALSE	論理値ベクトル
^	$x \wedge y$	1つがTRUE, 1つがFALSE $\Rightarrow$ TRUE 両方ともTRUEもしくはFALSE $\Rightarrow$ FALSE	論理値ベクトル
~	$\sim x$	TRUE $\Rightarrow$ FALSE FALSE $\Rightarrow$ TRUE	論理値ベクトル

# filter:列の抽出

df



df.filter(条件)

filter

条件で指定した列が  
抽出される.

# filterの使い方

No.	指定方法	引数	説明
1	完全指定	items	列名を指定する. 引数itemsは省略可
2	部分指定	like	列名の一部を指定する.
3	正規表現指定	regex	列名の一部を正規表現で指定する.

**参考: データ型指定**  
**select\_dtypes(データ型)**  
**で指定したデータ型に該当する列を抽出**

**good**

```
df.filter(["sepal_length"])
```

**bad**

```
df[["sepal_length"]]
```

・シリーズと区別するのが難しい。  
参考: シリーズの抽出

```
df["sepal_length"]
```

データフレームを抽出⇒filter  
シリーズを抽出⇒[]  
と覚えるのがオススメ！

# assign:列の追加・更新

df

列0		

```
df.assign(  
  列0 = ベクトル0,  
  列1 = ベクトル1,  
  列2 = ベクトル2  
)
```

assign

列0⇒  
既存の列を**更新**  
列1, 列2⇒  
新規の列を右側に**追加**

列0			列1	列2
ベクトル0			ベクトル1	ベクトル2

df



```
def 関数(df):  
    return ベクトル
```

```
df.assign(列 = 関数)
```

lambda式が一般的

assign

				列
				ベクトル

# 関数: `add_1`

入力 = `x`

処理

出力 = `y`

1



`x` + 1



2

2



3

普通

```
def add_1 (x):  
    y = x + 1  
    return(y)
```

ちょっとした処理に対し  
関数を定義しなくてよい.

lambda式

```
lambda x:x+1
```



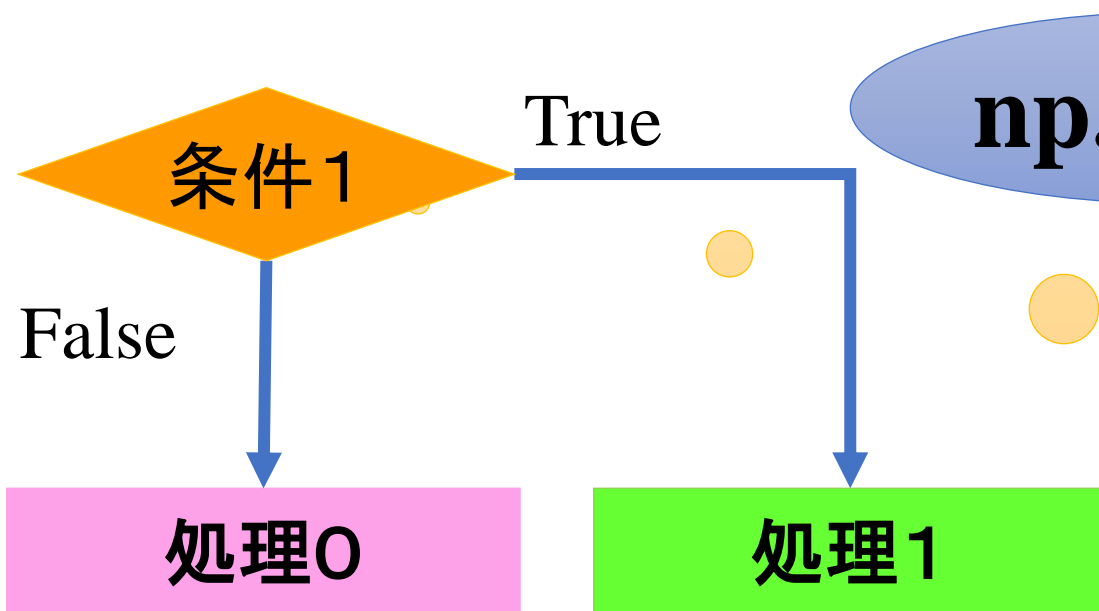
# df



```
df.assign(  
  列名 = lambda x: np.where(  
    条件1,  
    処理1,  
    処理0  
  )  
)
```

assign

				列名
				処理1
				処理0
				処理1
				処理1
				処理0



np.where

条件が1つ  
⇒  
単数条件

good

```
df.assign(table = lambda x:x["table"] + 1)
```

bad

```
df["table"] = df["table"] + 1
```

こちらと一緒に

```
df = df.assign(table = lambda x:x["table"] + 1)
```

・列の追加・更新と同時にdfを書き換えてしまう.

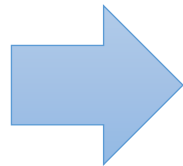
⇒

再帰代入

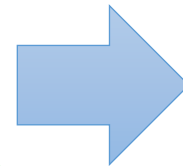
# df

# df\_rev

①列の更新



②行の抽出



再帰代入すると  
こっちがdfに...

メソッドチェーン

good

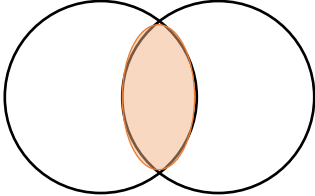
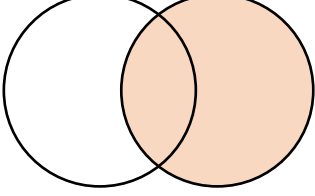
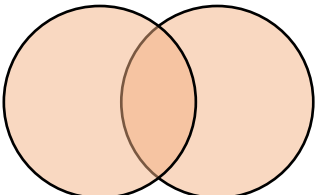
```
1 df_rev = df.assign(sepal_length = lambda x: x["sepal_length"] + x["sepal_width"]).query("sepal_length > 8")
2
```

再帰代入

bad

```
1 df = df.assign(sepal_length = lambda x: x["sepal_length"] + x["sepal_width"])
2 df_rev = df.query("sepal_length > 8")
```

# merge:キー結合

No.	キー結合の種類	使い方	省略可	イメージ
1	inner_join (内部結合)	pd.merge(x, y, how="inner")		
2	left_join (左結合)	pd.merge(x, y, how="left")		
3	right_join (右結合)	pd.merge(x, y, how="right")		
4	outer_join (外部結合)	pd.merge(x, y, how="outer")		

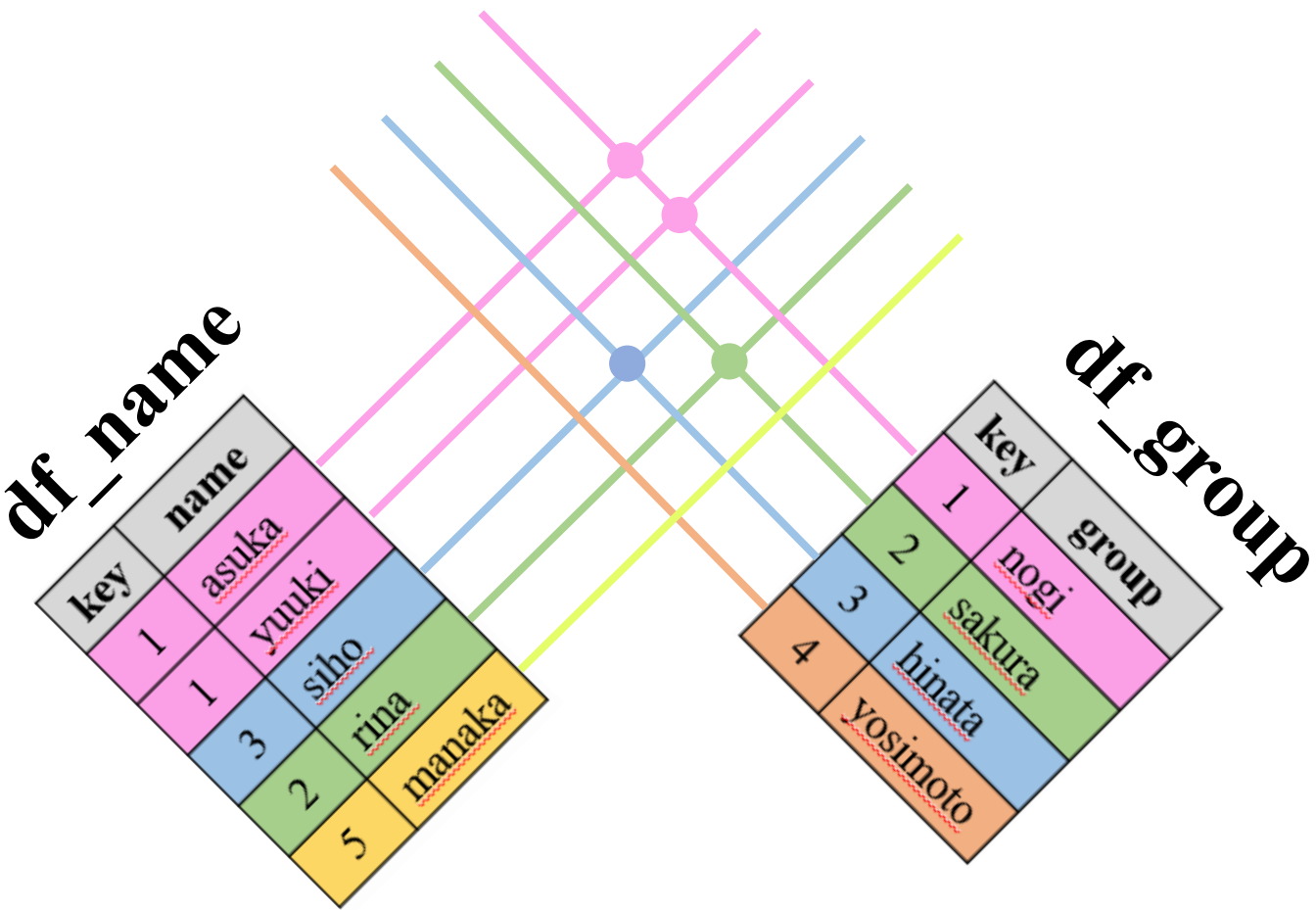
## df\_name

key	name
1	asuka
1	yuuki
3	siho
2	rina
5	manaka

## df\_group

key	name
1	nogi
2	sakura
3	hinata
4	yosimoto

# inner\_join

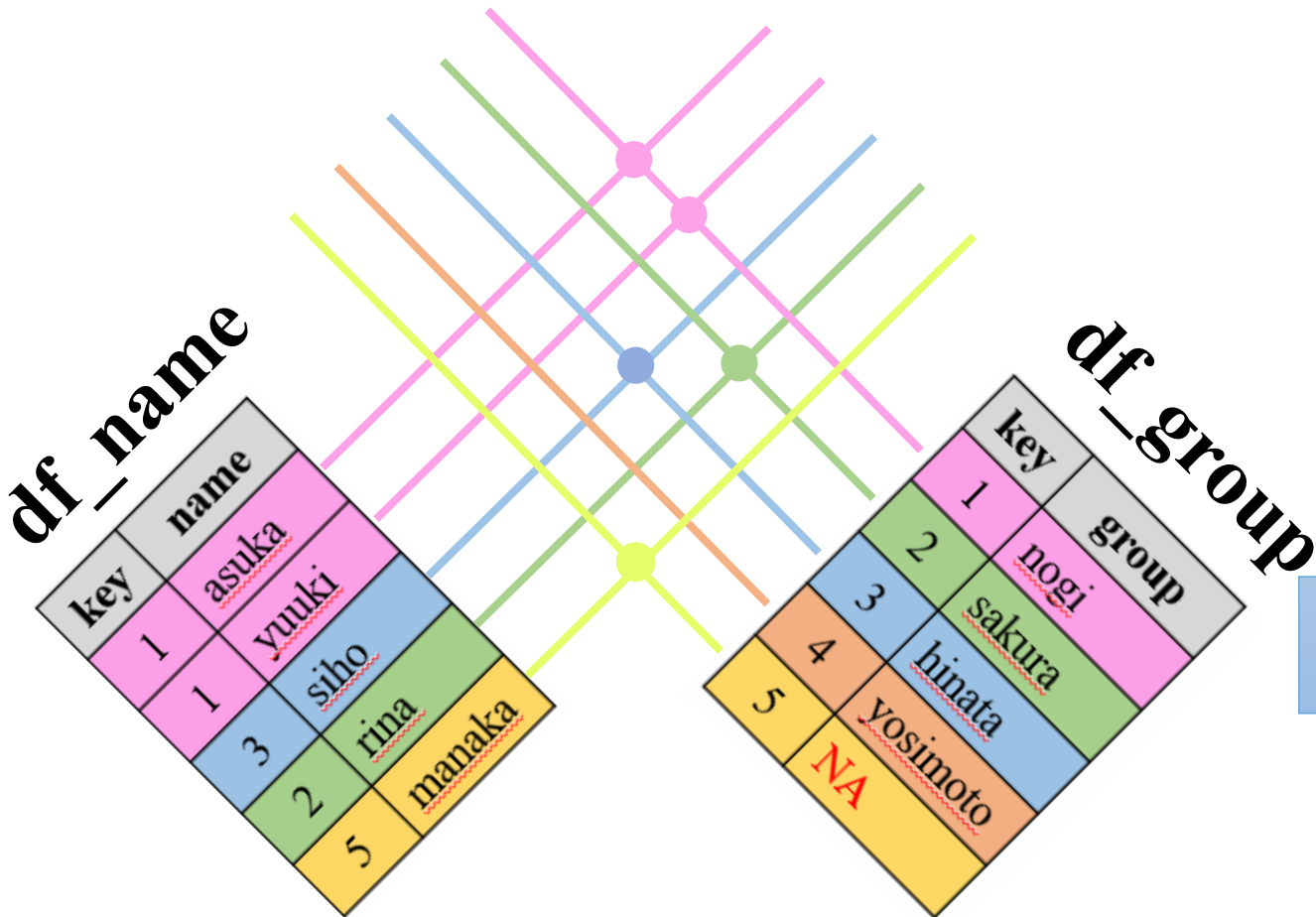


inner\_join

df\_inner\_join

key	name	group
1	asuka	nogi
1	yuuki	nogi
3	siho	hinata
2	rina	sakura

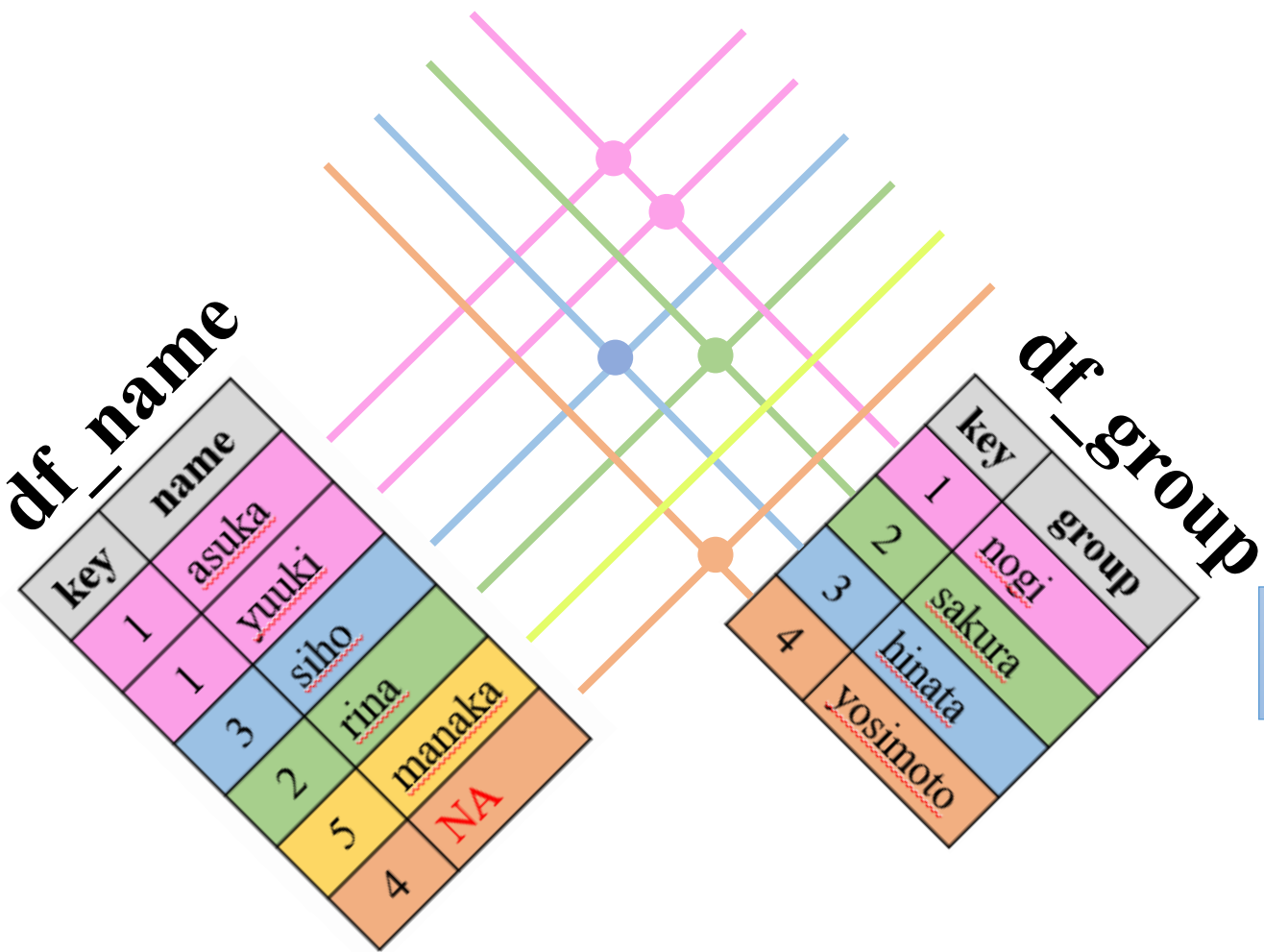
# left\_join



df\_left\_join

key	name	group
1	asuka	nogi
1	yuuki	nogi
3	siho	hinata
2	rina	sakura
5	manaka	NA

# right\_join



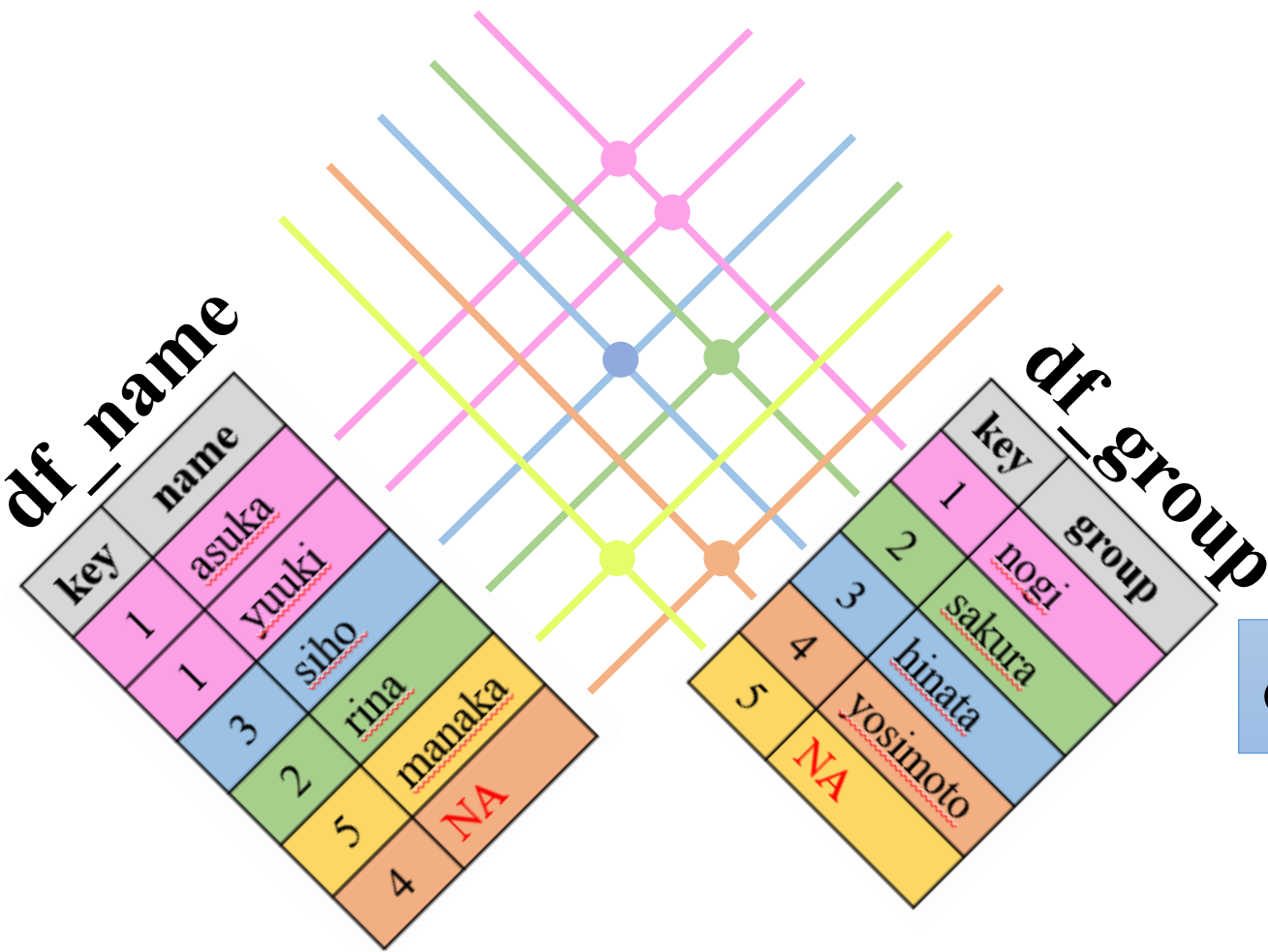
right\_join

## df\_right\_join

key	name	group
1	asuka	nogi
1	yuuki	nogi
2	rina	sakura
3	siho	hinata
4	NA	yosimoto

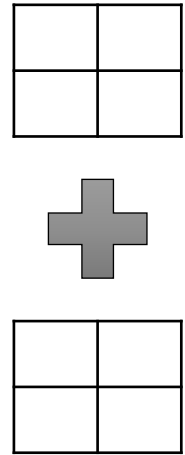
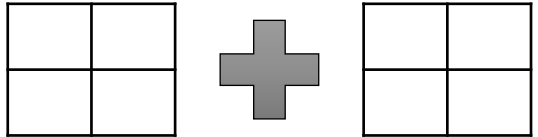


# outer\_join



outer\_join

# concat:縦横結合

No.	縦横結合の種類	使い方	イメージ
1	縦結合 オススメ	pd.concat([x, y], axis=0) 省略可 ignore_index=True	
2	横結合	pd.concat([x, y], axis=1)	

df\_name\_1

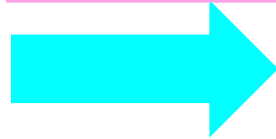
key	name
1	asuka
2	rina



df\_name\_2

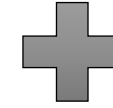
key	name
1	hinako
2	yui
3	kyouko

axis=0



df\_name

key	name
1	asuka
2	rina
1	hinako
2	yui
3	kyouko



df\_group

group
nogi
sakura
nogi
sakura
hinata

axis=1



df\_name\_group

key	name	group
1	asuka	nogi
2	rina	sakura
1	hinako	nogi
2	yui	sakura
3	kyouko	hinata

df

# groupby: グループピング

class	gender	height
a	M	162
b	F	150
c	F	168
c	M	173
a	F	162
c	M	198
b	M	182
a	F	154
c	M	175
b	M	160
a	F	172

①  
グループピング

groupby  
("class")

class	gender	height
a	M	162
a	F	162
a	F	154
a	F	172
b	F	150
b	M	182
b	M	160
c	F	168
c	M	173
c	M	198
c	M	175

②  
計算

mean()

class	height
a	162.5
b	164.0
c	178.5

# 集約メソッド

No.	集約メソッド	意味
1	sum	総和
2	mean	平均
3	median	中央値
4	max	最大値
5	min	最小値
6	var	分散
7	std	標準偏差
8	count	データ数

# sort\_values:ソート

df

指定した列をソートする.

col			
6			
4			
1			
3			
2			
5			
8			
7			

```
df.sort_values("col")
```

sort\_values

col			
1			
2			
3			
4			
5			
6			
7			
8			

# sort\_values:ソート

No.	ソートの種類	使い方	イメージ								
1	昇順 (小さい順)	<div>df.sort_values(     “col”, ascending=True     ) <div>省略可</div></div>	<table><tr><th>col</th><th></th></tr><tr><td>1</td><td></td></tr><tr><td>2</td><td></td></tr><tr><td>3</td><td></td></tr></table>	col		1		2		3	
col											
1											
2											
3											
2	降順 (大きい順)	<div>df.sort_values(     “col”, ascending=False     )</div>	<table><tr><th>col</th><th></th></tr><tr><td>3</td><td></td></tr><tr><td>2</td><td></td></tr><tr><td>1</td><td></td></tr></table>	col		3		2		1	
col											
3											
2											
1											

# drop\_duplicates: 重複削除

df

col				
1				
2				
1				
1				
2				
3				
2				
1				

df.drop\_duplicates("col")

指定した列の重複を削除する.

drop\_duplicates

col				
1				
2				
3				



# melt, pivot:縦横変換

**df\_wide**

果物 ↓

お店	みかん	りんご
A	100	150
B	70	90
C	120	80

↑ お店      ↑ 値段

縦変換  
df\_wide.melt

横変換  
df\_long.pivot

**df\_long**

お店	果物	値段
A	みかん	100
B	みかん	70
C	みかん	120
A	りんご	150
B	りんご	90
C	りんご	80

↑ お店      ↑ 果物      ↑ 値段

tidyデータ

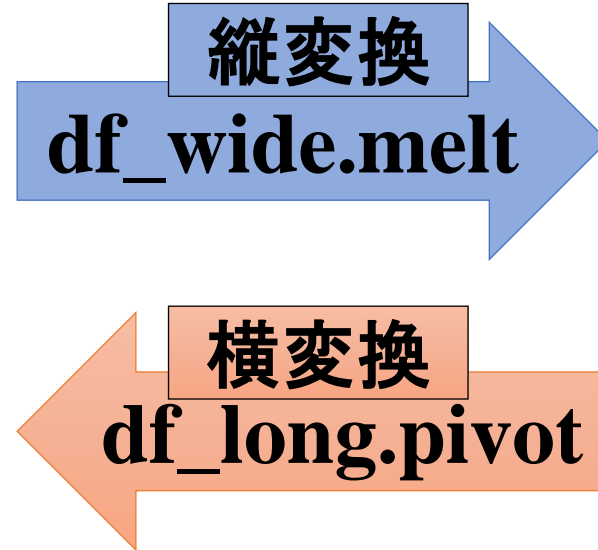
# melt, pivot:縦横変換

**df\_wide**

fruit

store	orange	apple
A	100	150
B	70	90
C	120	80

store price



**df\_long**

store	fruit	price
A	orange	100
B	orange	70
C	orange	120
A	apple	150
B	apple	90
C	apple	80

store fruit price

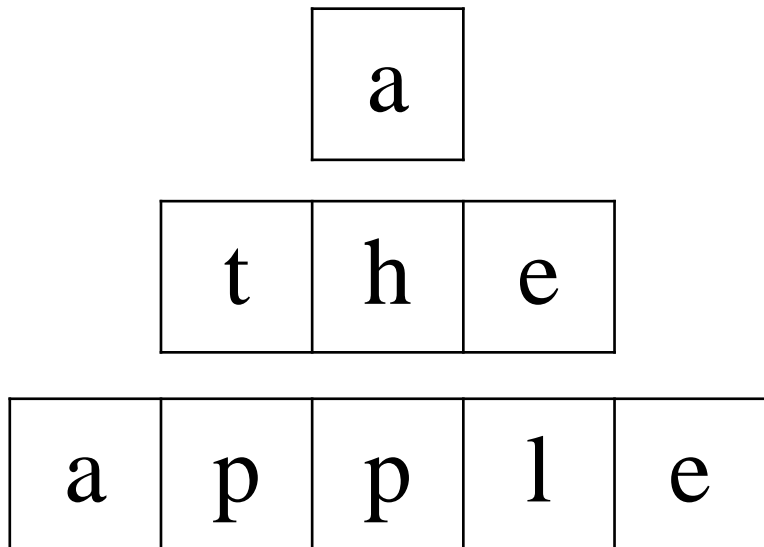
tidyデータ

# 文字列処理

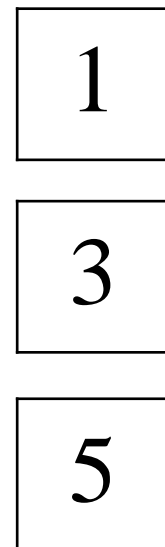


## 様々な文字列処理の一例

**series**



**str.len**





No.	メソッド	内容
1	str	抽出
2	str.len	長さ取得
3	str.cat	連結
4	str.contains	マッチの真偽
5	str.replace	マッチの単数置換
6	str.translate	マッチの複数置換
7	str.split	マッチの分割

series

0	q	-11
1	w	-10
2	e	-9
3	r	-8
4	t	-7
5	y	-6
6	u	-5
7	i	-4
8	o	-3
9	p	-2
10	@	-1

str: 抽出

`series.str[1:5]`

終点は含まれない

指定したインデックス  
の要素を抽出する.

str

`series.str[-10:-6]`

終点は含まれない

w
e
r
t

# str.len : 長さ取得

文字列の長さを  
取得する.

series

a

1

t h e

3

a p p l e

5

str.len

# str.cat: 連結

series

nogi

sakura

hinata

sep

\_

省略可

文字列を連結する.

str.cat

nogi

\_

sakura

\_

hinata

# str.cat: 連結

series\_1

series\_2

nogi

mai

sep

-

省略可

文字列を連結する.

nogi

-

mai

sakura

ponpon

str.cat

sakura

-

ponpon

hinata

sarina

hinata

-

sarina



# str.contains: マッチの真偽

series

マッチの真偽を判定する.

nogi

a

False

sakura

str.contains

True

hinata

True

# str.replace: マッチの単数置換

series

単数

置換前 置換後

asuka

a

-

-suk-

mizuki

mizuki

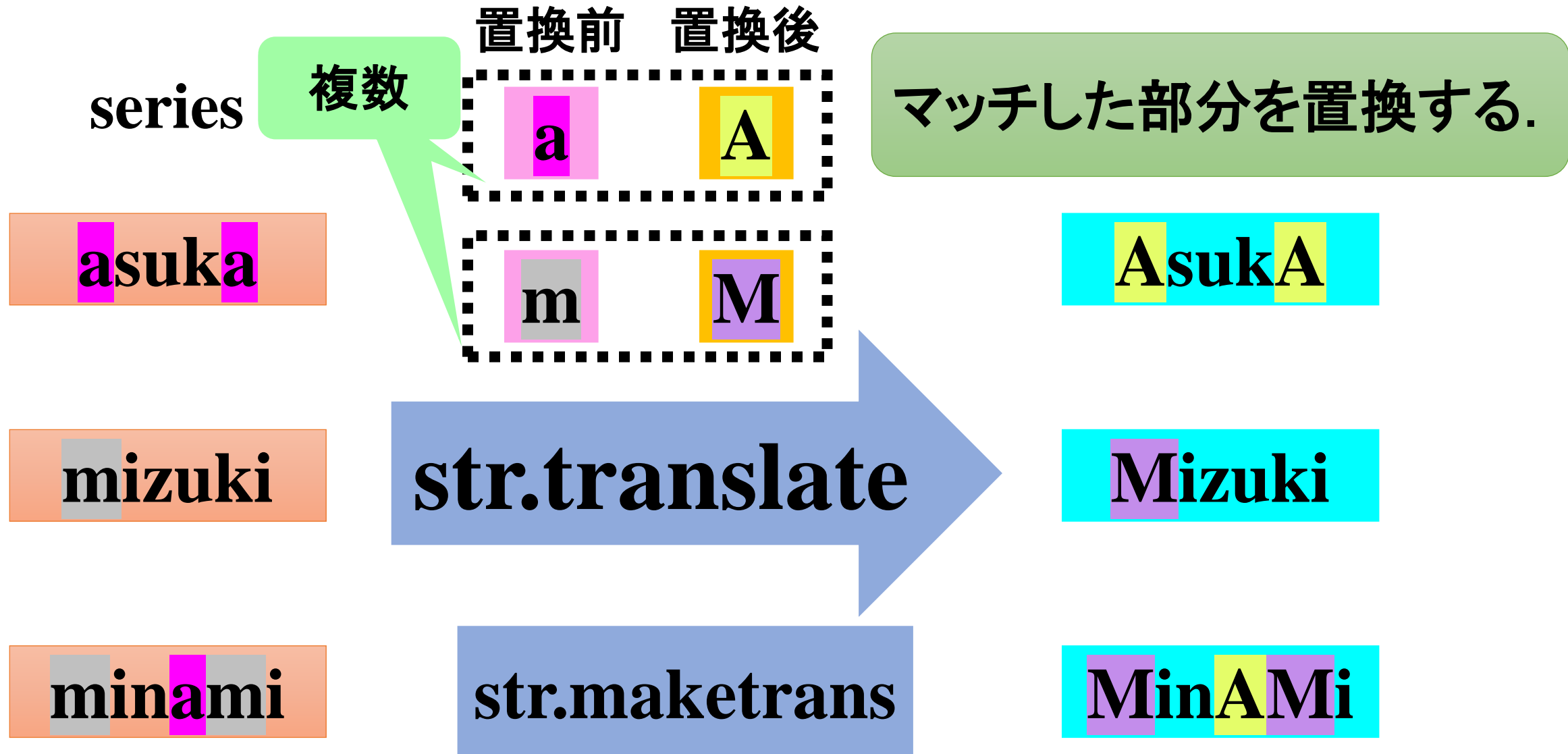
minami

min-mi

str.replace

マッチした部分を置換する.

# str.translate: マッチの複数置換



# str.split: マッチの分割

series

マッチした部分で分割する.

asuka

u

as

ka

mizuki

str.split

miz

ki

minami

expand=True

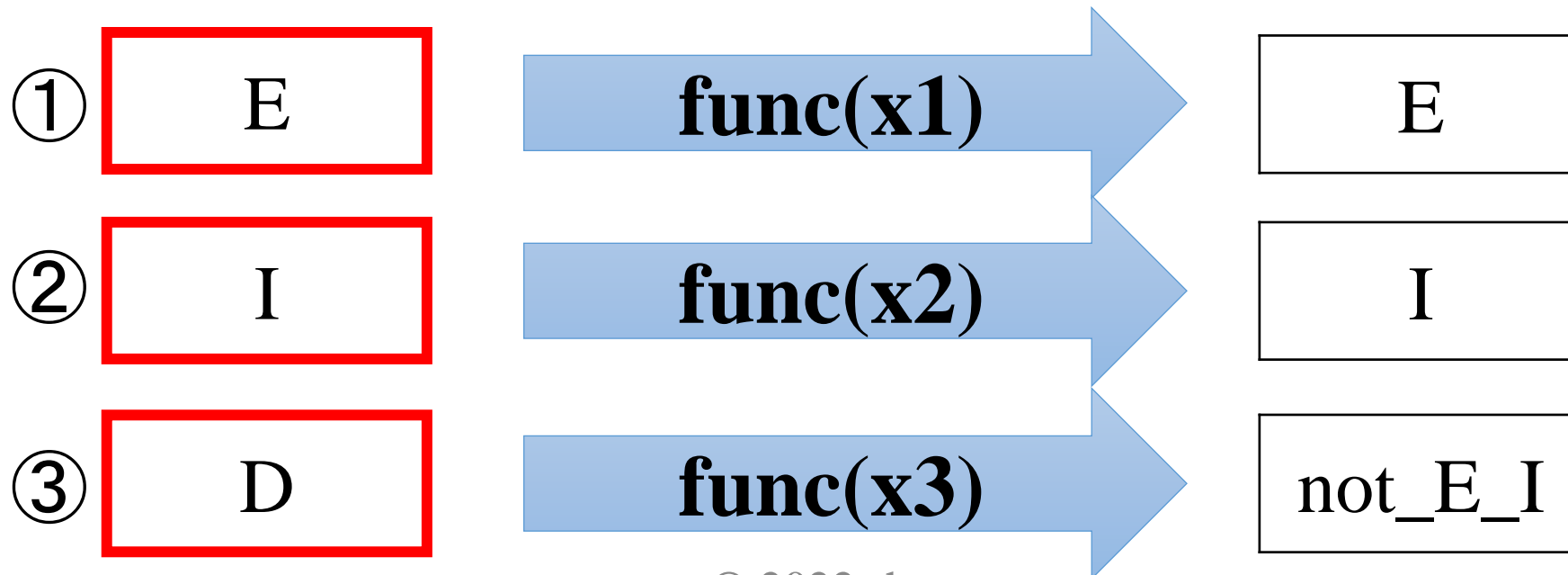
minami

データフレームを出力

# 繰り返し処理



## 繰り返し処理の一例



# 繰り返し処理

for文よりも高速



並列処理と呼ぶこともある

繰り返し処理の一例

①

E

func(x1)

E

②

I

func(x2)

I

③

D

func(x3)

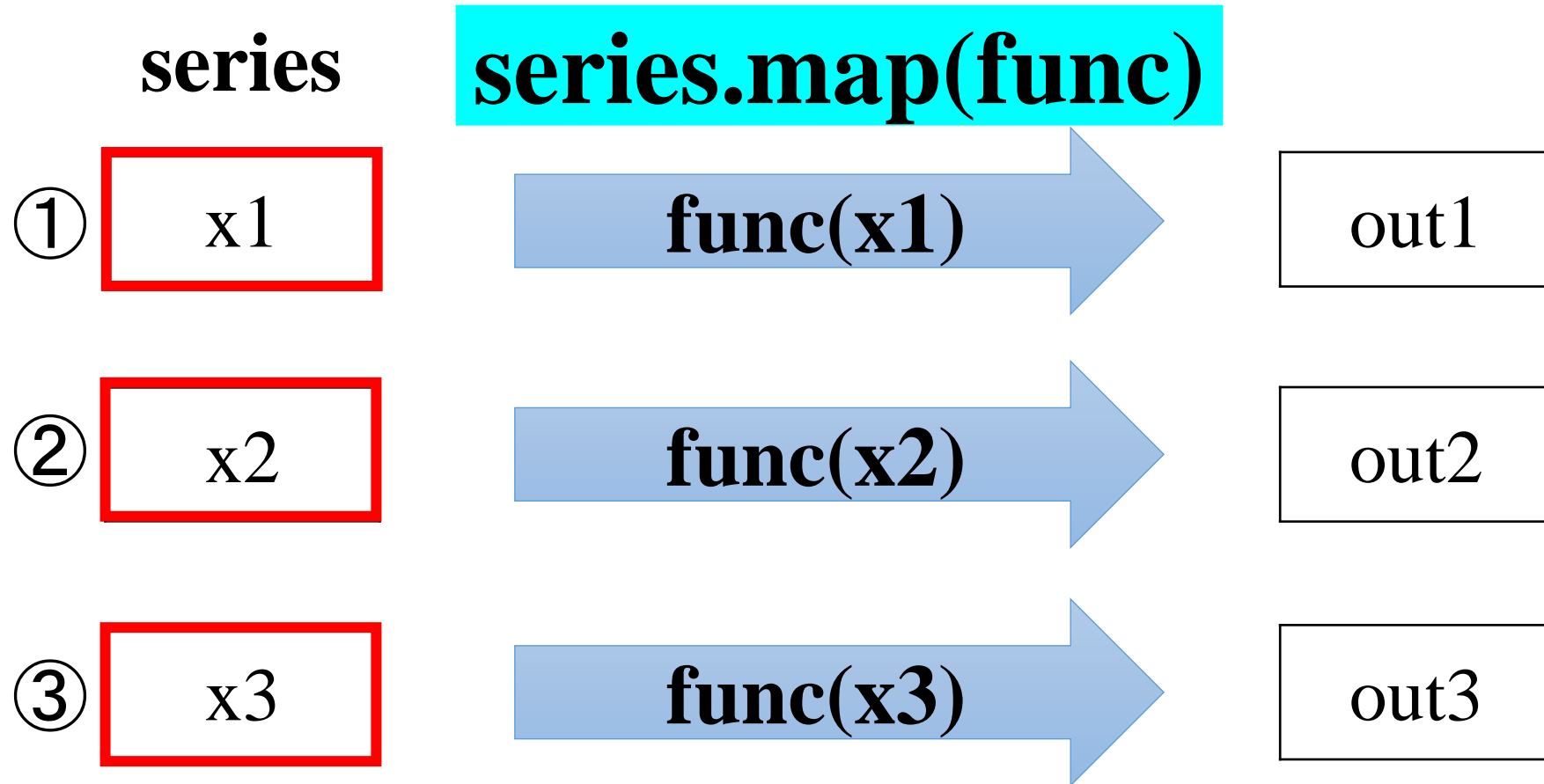
not\_E\_I



No.	メソッド	内容
1	map	シリーズの行処理
2	apply	データフレームの行処理
3	applymap	データフレームの要素処理

# map: シリーズの行処理

シリーズの各行に  
関数を適用する.





# mapの基本

シリーズの各行に  
関数を適用する。

**func**

```
def func(x):  
    if x == "E":  
        return("E")  
    elif x == "I":  
        return("I")  
    else:  
        return("not_E_I")
```

**series**

①

E

②

I

③

D

**series.map(func)**

**func(x1)**

E

**func(x2)**

I

**func(x3)**

not\_E\_I

# mapの繰り返し

シリーズの各行に  
関数を適用する。

**func**

```
def func(x):  
    if x == "E":  
        return("E")  
    elif x == "I":  
        return("I")  
    else:  
        return("not_E_I")
```

**series**

①

E

②

I

③

D

**series.map(func)**

**func(x1)**

E

**func(x2)**

I

**func(x3)**

not\_E\_I

# mapとassign

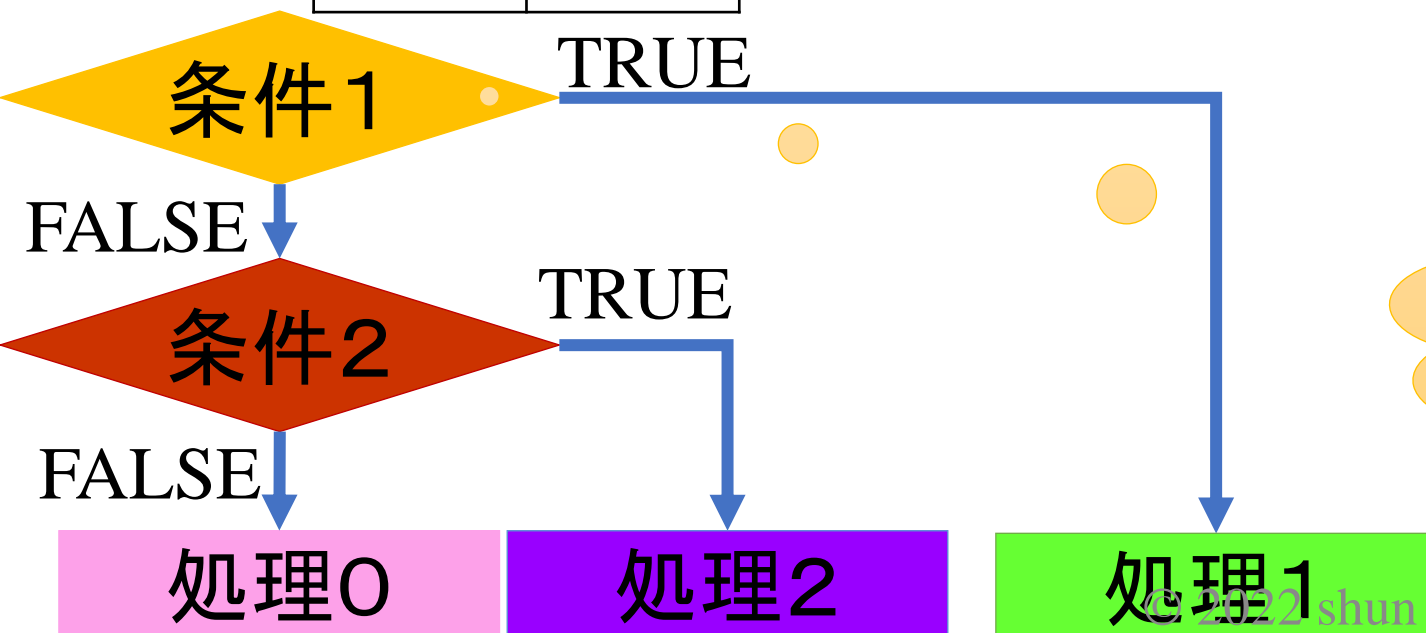
df


assign

map

df\_rev

		列名
		処理1
		処理0
		処理2
		処理1



条件が2つ以上  
⇒  
複数条件

# mapと辞書

```
series.map(  
    {key1:data1, key2:data2, kye3:data3''}  
)
```



# mapと辞書

```
series.map(  
    {"E": "e", "I": "i", "not_E_I": "not_e_i"}  
)
```



# apply: データフレームの行処理

データフレームの各行に  
関数を適用する。

df

	x	y
①	x1	y1
②	x2	y2
③	x3	y3

**df.apply(func, axis=1)**

func(①)

out1

func(②)

out2

func(③)

out3

# applyの基本

シリーズの各行に  
関数を適用する。

**func**

```
def func(x):  
    if x["color"] == "E":  
        return x["depth"]  
    elif x["color"] == "I":  
        return 0  
    else:  
        return 1
```

**df**

	depth	color
①	61.5	E
②	62.4	I
③	63.1	D

**df.apply(func, axis=1)**

**func(①)**

61.5

**func(②)**

0

**func(③)**

1

# applyの繰り返し

シリーズの各行に  
関数を適用する。

**func**

```
def func(x):  
    if x["color"] == "E":  
        return x["depth"]  
    elif x["color"] == "I":  
        return 0  
    else:  
        return 1
```

**df**

	depth	color
①	61.5	E
②	62.4	I
③	63.1	D

**df.apply(func, axis=1)**

**func(①)**

61.5

**func(②)**

0

**func(③)**

1



# applyとassign

df

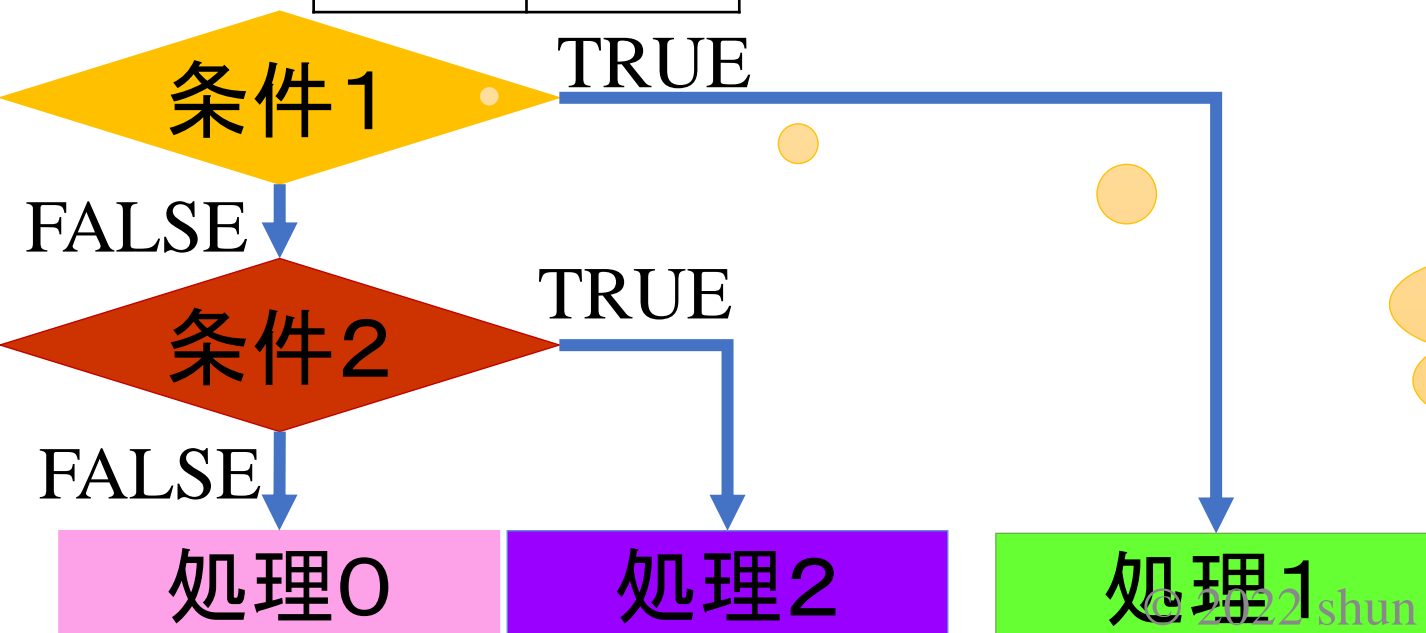


assign

apply

df\_rev

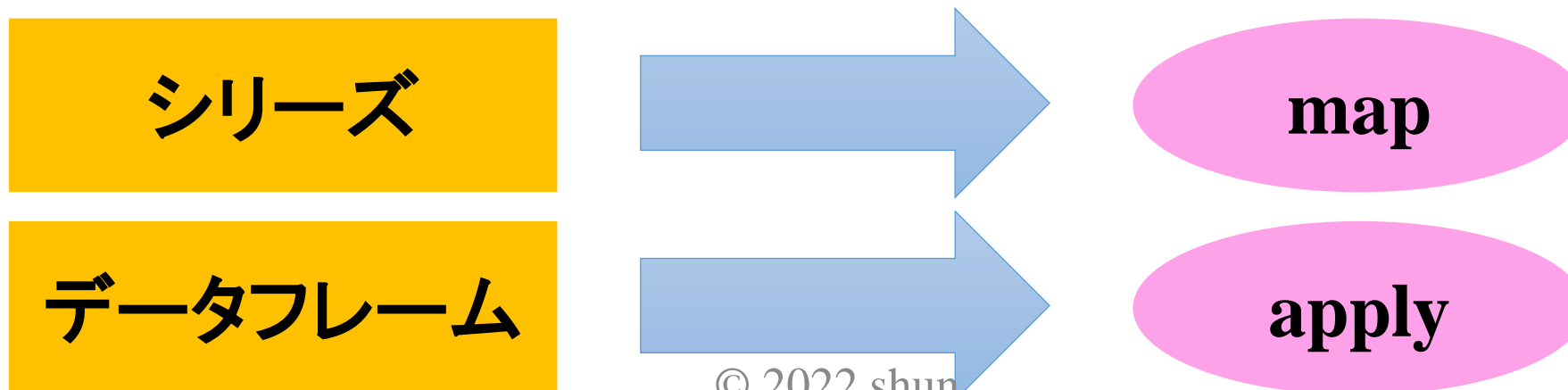
		列名
		処理1
		処理0
		処理2
		処理1



条件が2つ以上  
⇒  
複数条件

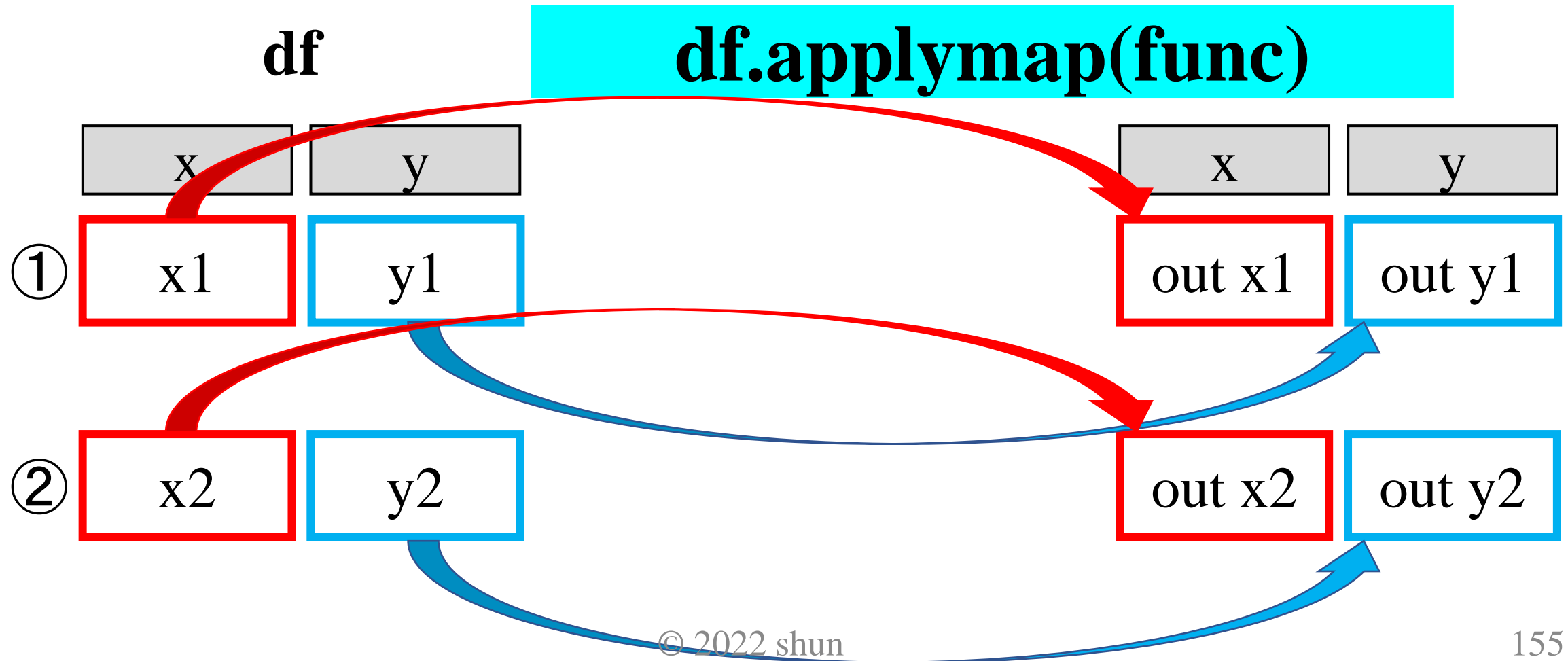
# mapとapply

	map	apply
インプット	・シリーズ	・シリーズ ・データフレーム
辞書	使用可	使用不可
速度	高速	低速



# applymap: データフレームの要素処理

データフレームの各要素に  
関数を適用する.



# applymapの基本

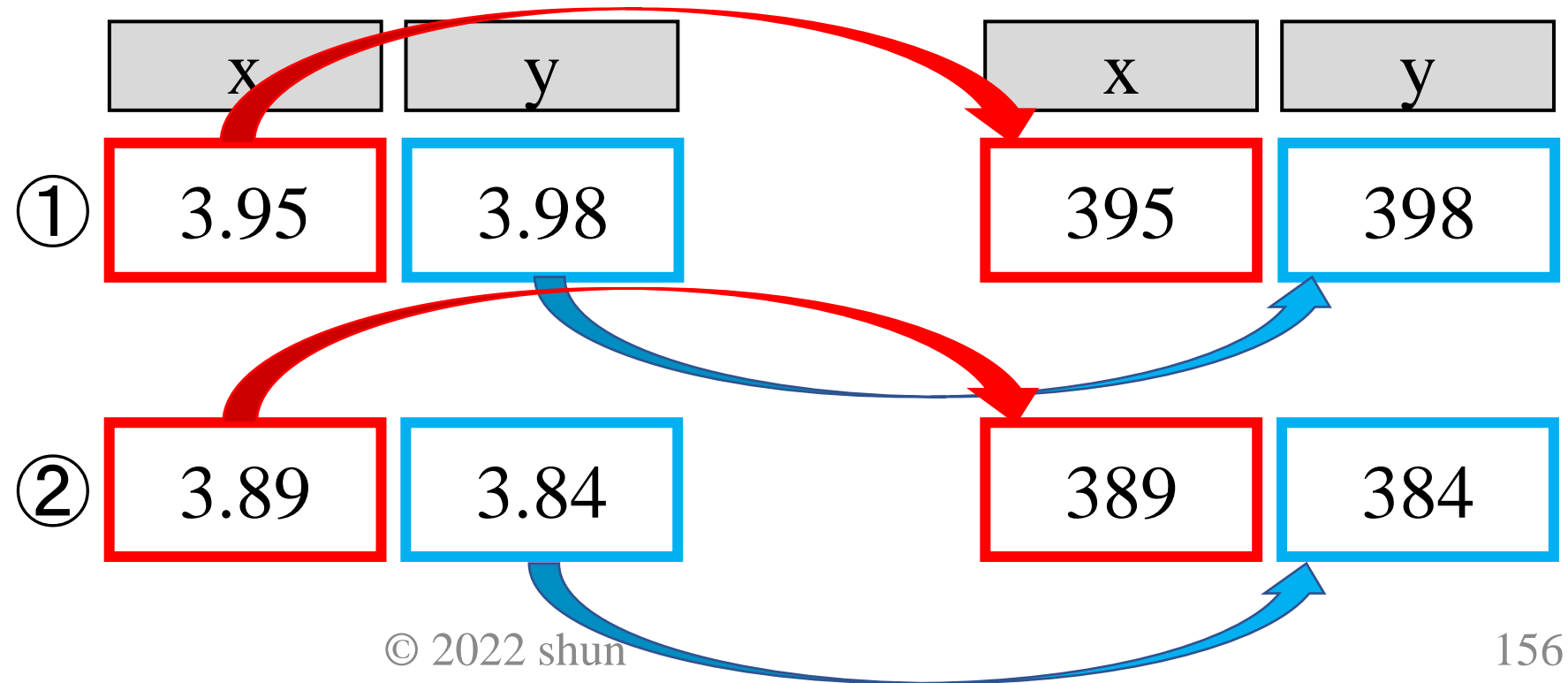
データフレームの各要素に  
関数を適用する。

## df.applymap(func)

df

**func**

```
def func(x):  
    return x*100
```



# applymapの繰り返し

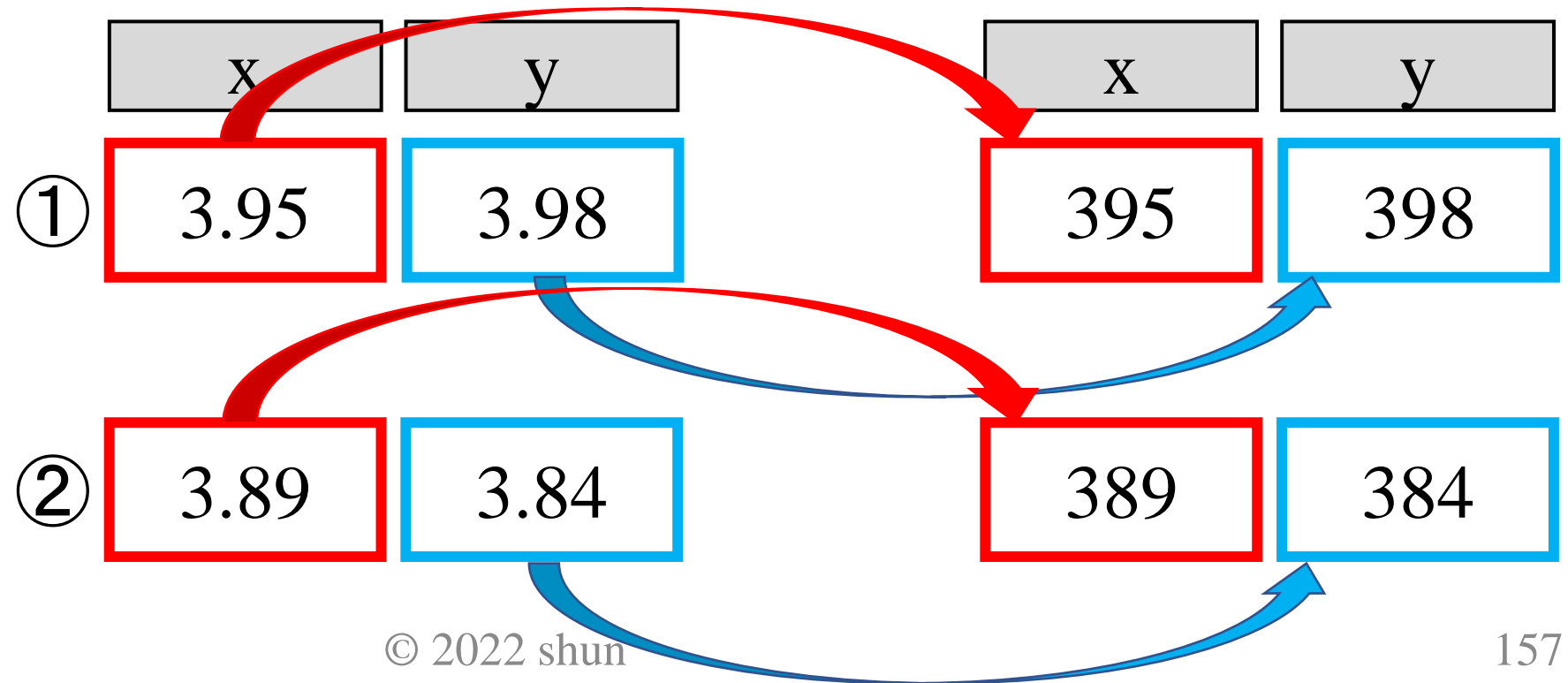
データフレームの各要素に  
関数を適用する。

## df.applymap(func)

df

**func**

```
def func(x):  
    return x*100
```



# applymapとlambda式

普通

**func**

```
def func(x):  
    return x*100
```

**df.applymap(func)**

lambda式

**df.applymap(lambda x:x\*100)**

mapやapplyでもlambda式は使用できるが、  
頻度は少ない。

# 欠損値処理



## 欠損値処理の一例

x	y	z
1	10	NaN
3	4	8
2	NaN	7

`df.fillna(0)`

`fillna`

x	y	z
1	10	0
3	4	8
2	0	7



No.	メソッド	内容
1	dropna	欠損値行削除
2	fillna	欠損値置換



# dropna: 欠損値行削除

欠損値がある行を削除する.

df

x	y	z
1	10	NaN
3	4	8
2	NaN	7
8	6	8
10	NaN	NaN

subsetで列を指定

df.dropna()

dropna

x	y	z
3	4	8
8	6	8

# fillna: 欠損値置換

欠損値を指定した値に置換する。

df

x	y	z
1	10	NaN
3	4	8
2	NaN	7
8	6	8
10	NaN	NaN

df.fillna(0)

fillna

x	y	z
1	10	0
3	4	8
2	0	7
8	6	8
10	0	0

# 置換後の値の指定方法

No.	引数	説明
1	value	<ul style="list-style-type: none"><li>・置換後の値を直接指定する.</li><li>・引数valueは省略可</li><li>・辞書, シリーズで指定可</li></ul>
2	method	<ul style="list-style-type: none"><li>・置換後の値を欠損値の上下の値に指定する.</li><li>・ffill: 欠損値の上の値で置換</li><li>・bfill: 欠損値の下で置換</li></ul>

# plotnine



plotnineってなんですか？



plotnineはデータの可視化のためのパッケージのことだよ！！  
plotnineを使用することで，簡単にデータの可視化ができるようになるんだ！！

# データの可視化

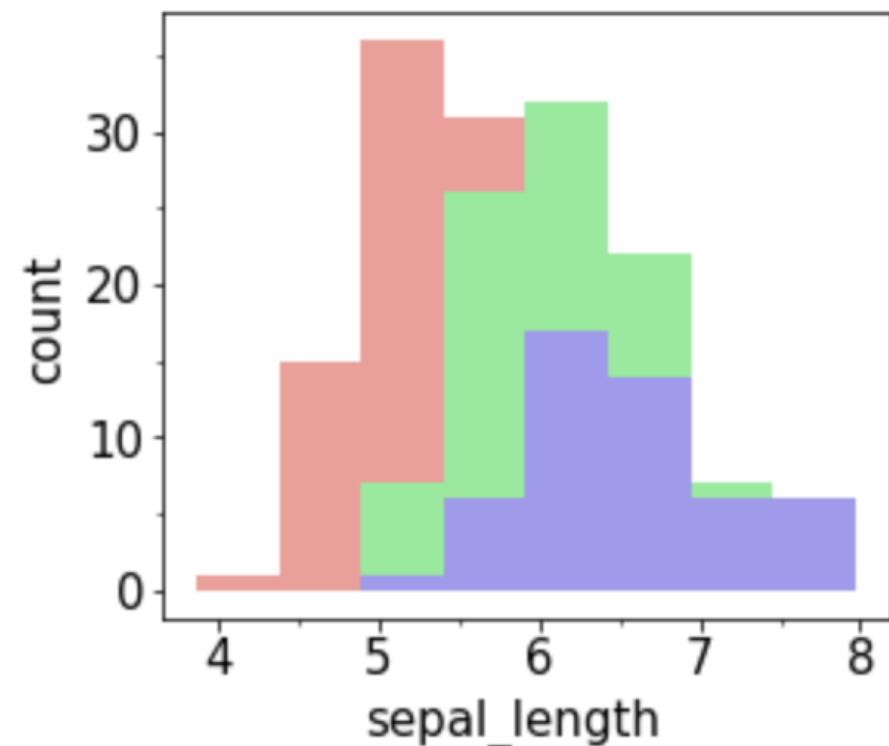
df

sepal_length	species
5.2	setosa
5.6	versicolor
6.7	virginica
4.9	setosa
5.5	setosa
7.2	virginica
5.7	versicolor

傾向がわかりづらい...

可視化

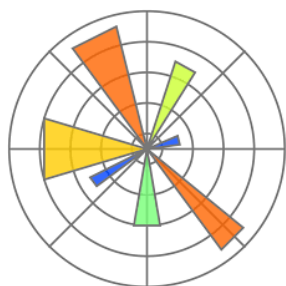
fig



傾向がわかりやすい！！

# なぜplotnineか？

## デファクトスタンダード



Matplotlib



seaborn

構文が複雑...

採用！！



構文が簡単！

Rのggplot2とほぼ同じ！

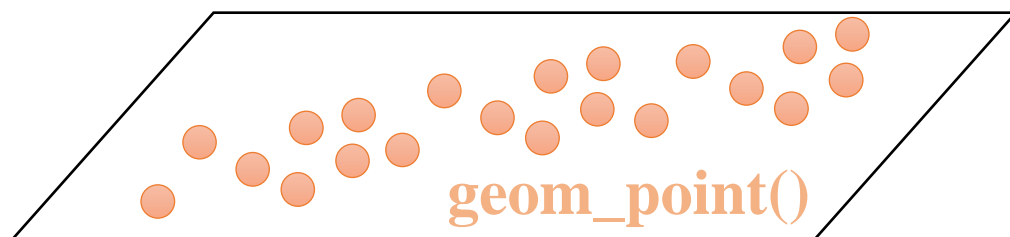
# plotnineの基本

1.キャンパス



+

2.グラフ

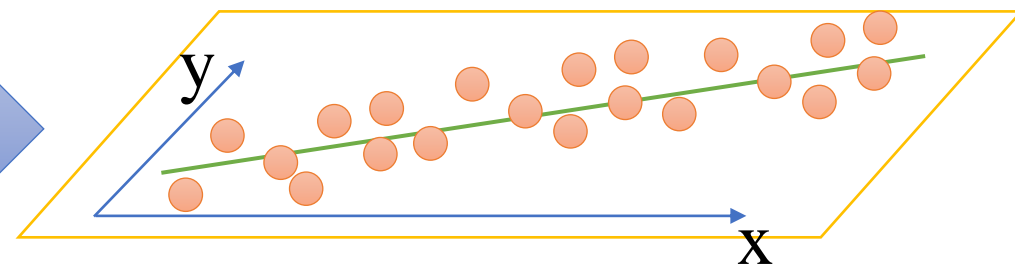


+

3.体裁



全体



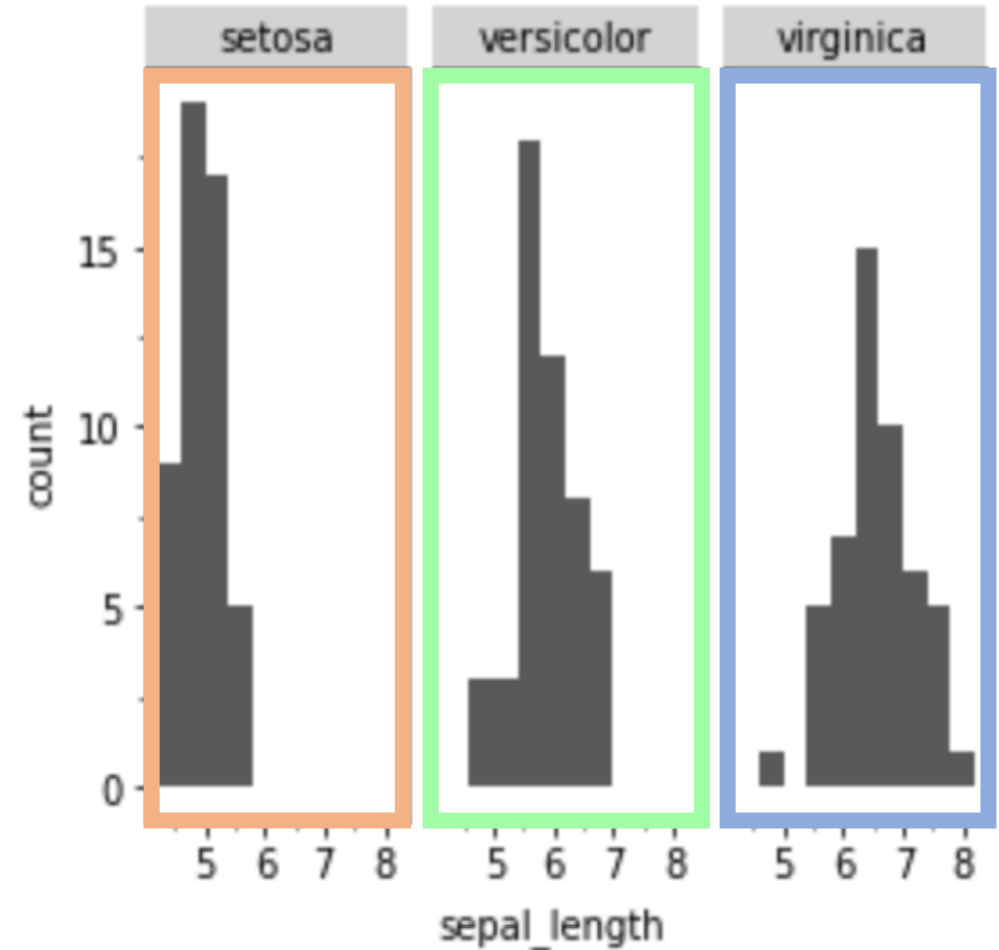
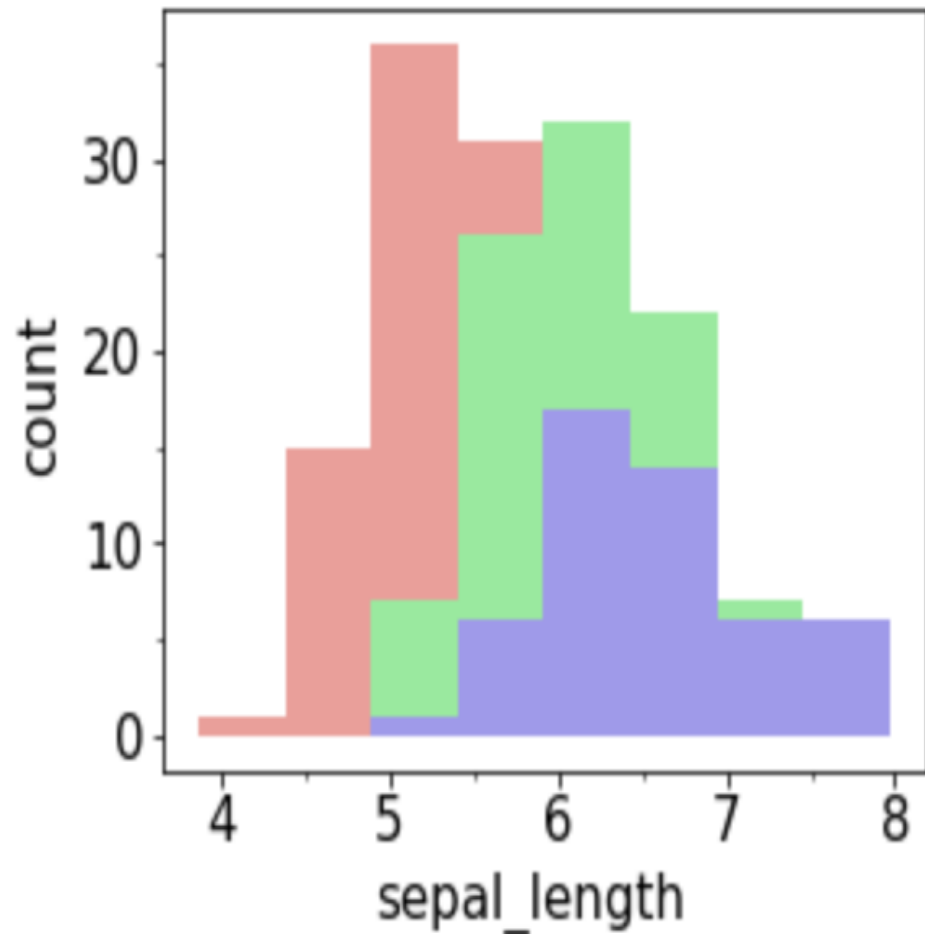
# カテゴリごとに出力

単数

複数

species

- setosa
- versicolor
- virginica





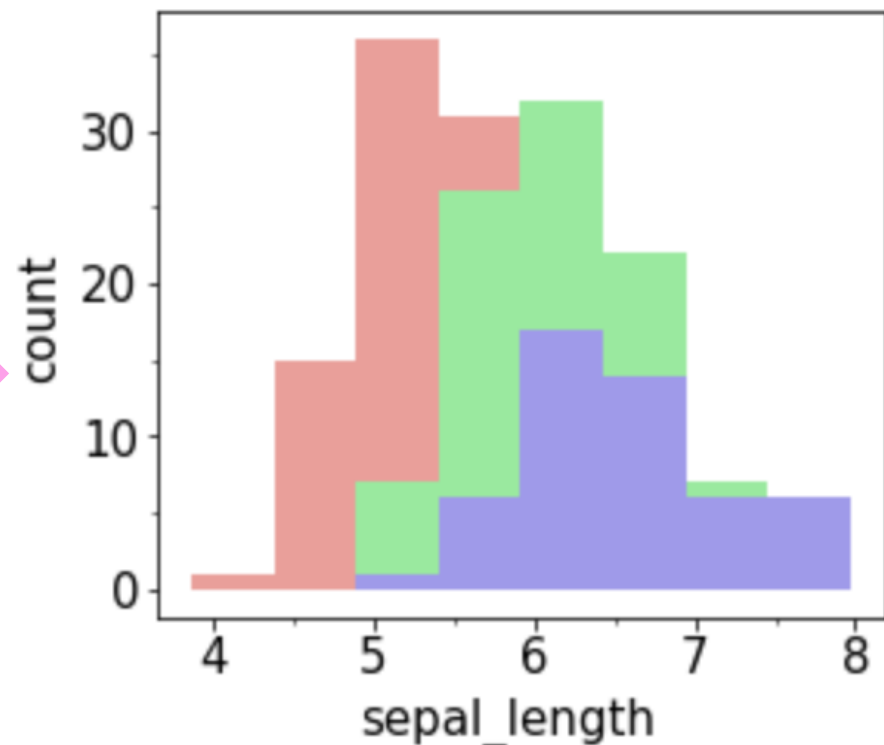
# カテゴリごとに出力(単数)

df

sepal_length	species
5.2	setosa
5.6	versicolor
6.7	virginica
4.9	setosa
5.5	setosa
7.2	virginica
5.7	versicolor

fig

species  
setosa  
versicolor  
virginica



ヒストグラム, 棒グラフ:

```
aes(x="sepal_length", fill="species")
```

散布図:

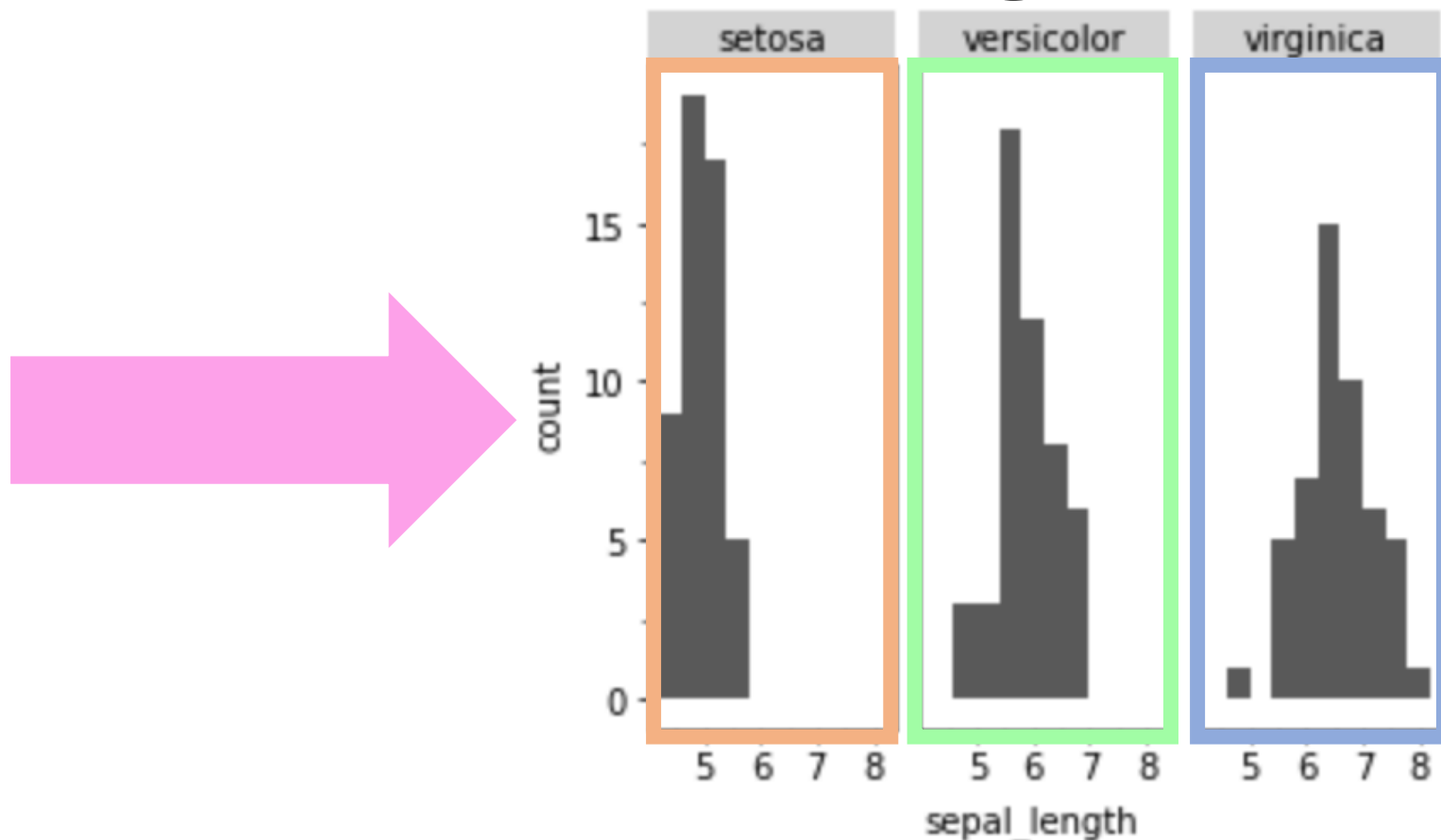
```
aes(x="sepal_length", y="sepal_width", color="species")
```

# カテゴリごとに出力(複数)

df

sepal_length	species
5.2	setosa
5.6	versicolor
6.7	virginica
4.9	setosa
5.5	setosa
7.2	virginica
5.7	versicolor

fig



```
facet_wrap("species")
```



No.	メソッド	用途
1	geom_histogram	ヒストグラム
2	geom_bar	棒グラフ
3	geom_point	散布図

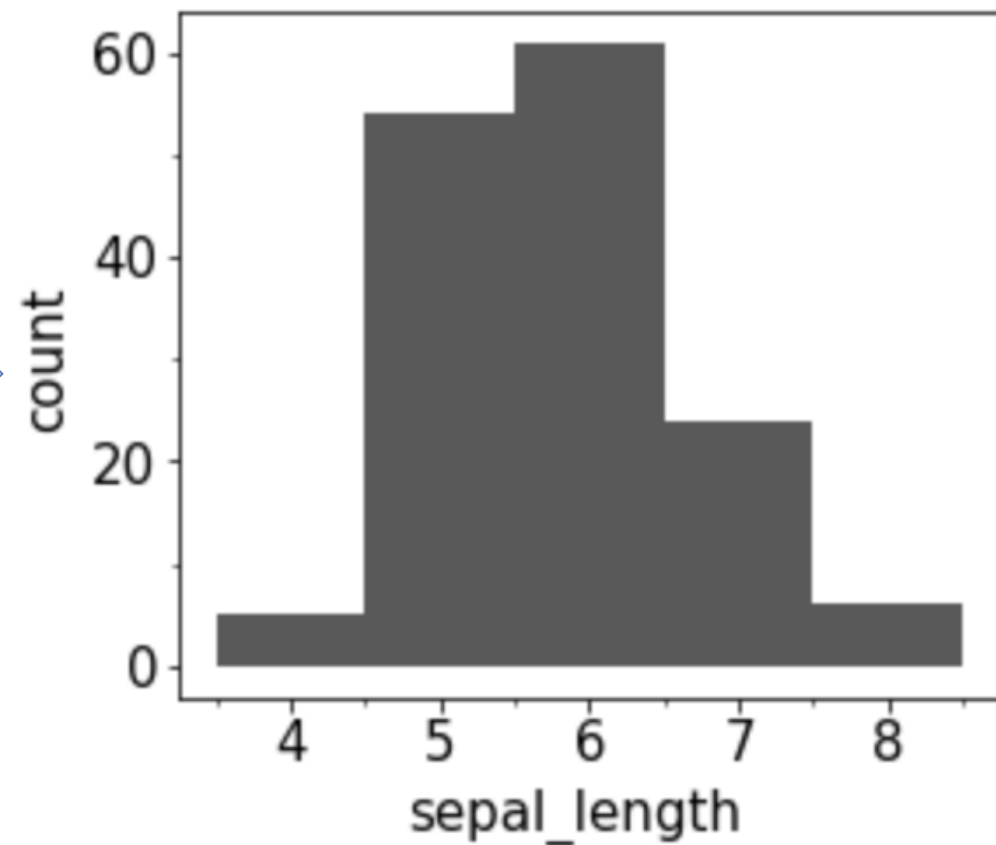
# geom\_histogram: ヒストグラム

df

sepal_length
5.1
4.9
4.7
:
5.0
5.4
4.6

geom\_histogram

fig



# ヒストグラムの基本

1変数(連続値)の分布の傾向を確認する.

ビン数(bins): 5  
ビン幅(binwidth): 1

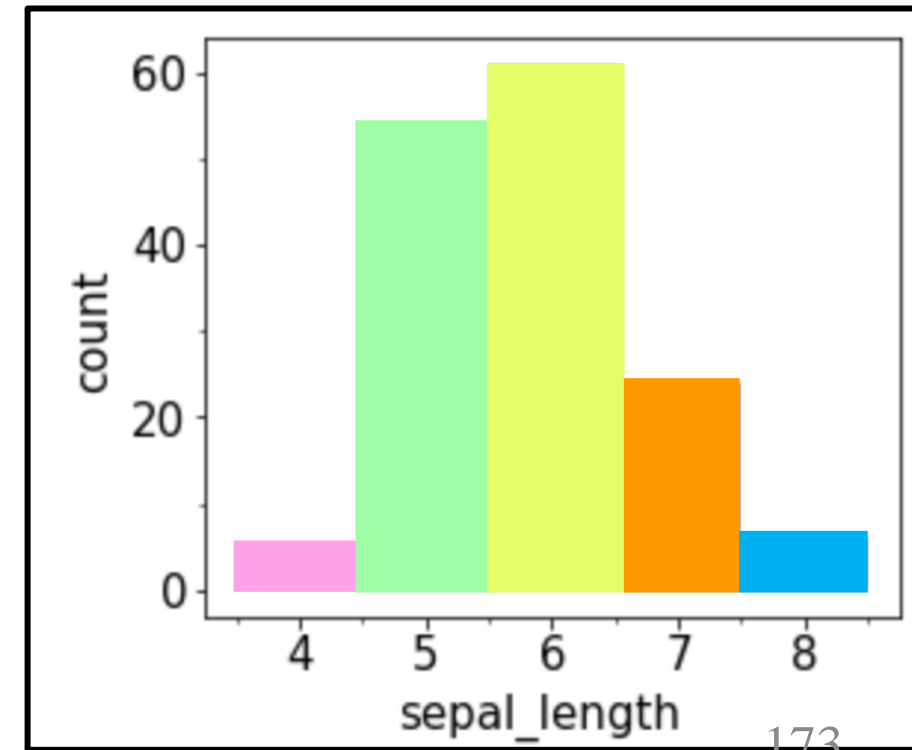
df

sepal_length
5.1
4.9
4.7
:
5.0
5.4

度数分布表

階級	度数
3.5 ~ 4.5	4
4.5 ~ 5.5	55
5.5 ~ 6.5	60
6.5 ~ 7.5	22
7.5 ~ 8.5	5

ヒストグラム



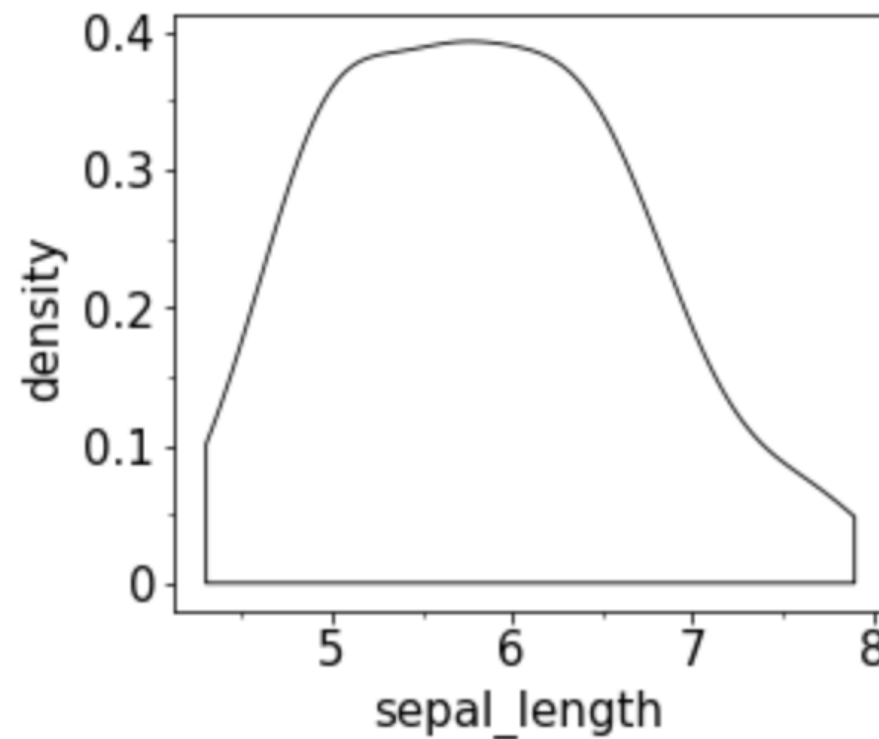
# geom\_density: 密度曲線

df

sepal_length
5.1
4.9
4.7
:
5.0
5.4
4.6

geom\_density

fig



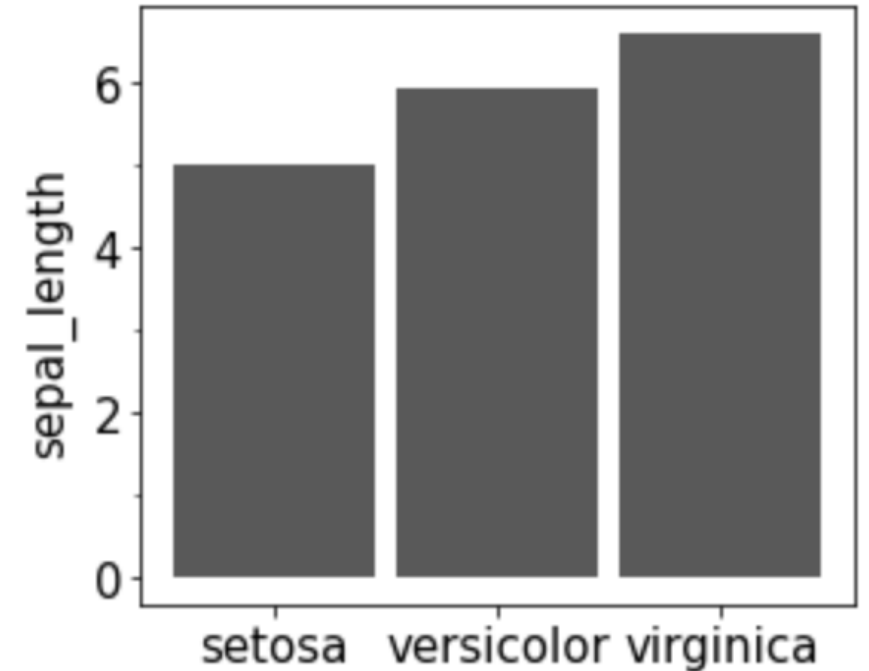
# geom\_bar: 棒グラフ

df\_group

species	sepal_length
setosa	5.006
versicolor	5.936
virginica	6.588

geom\_bar

fig

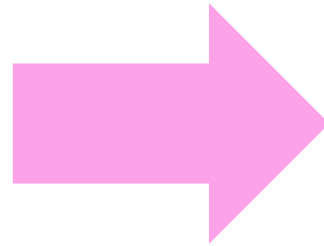


# 棒グラフの基本

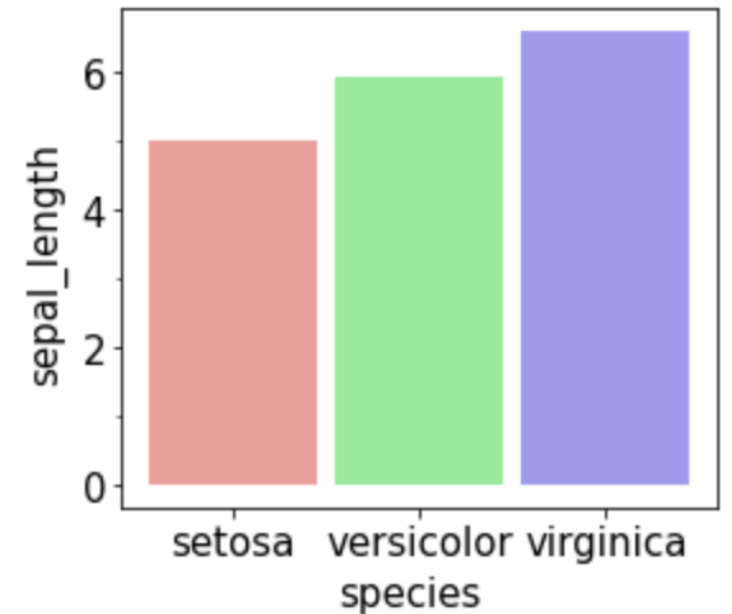
カテゴリの数値の大きさを比較する.

df\_group

species	sepal_length
setosa	5.006
versicolor	5.936
virginica	6.588



棒グラフ





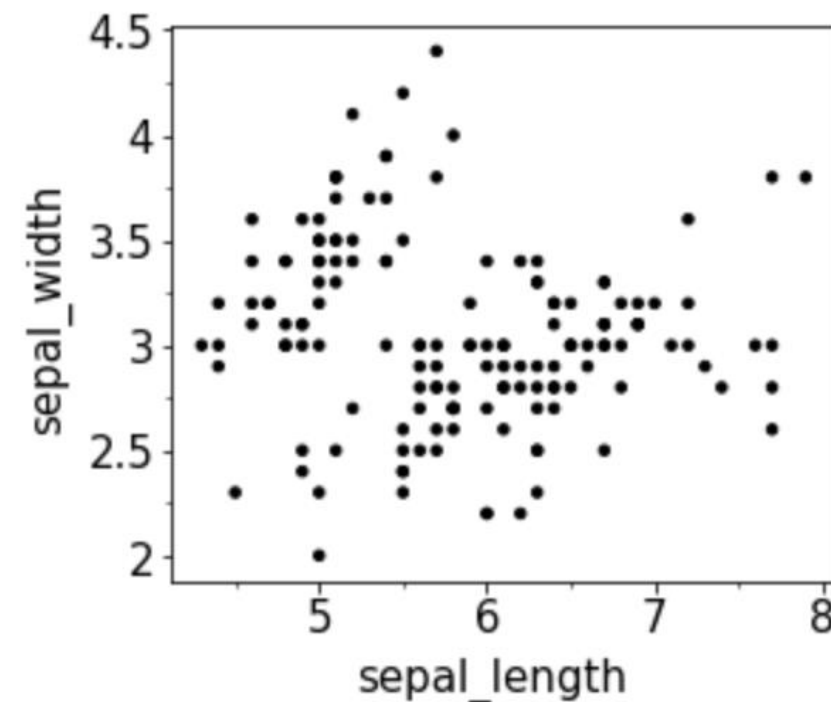
# geom\_point: 散布図

df

sepal_length	sepal_width
5.1	3.5
4.9	3.0
4.7	3.2
:	:
5.0	3.6
5.4	3.9
4.6	3.4

geom\_point

fig

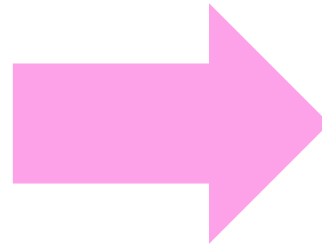


# 散布図の基本

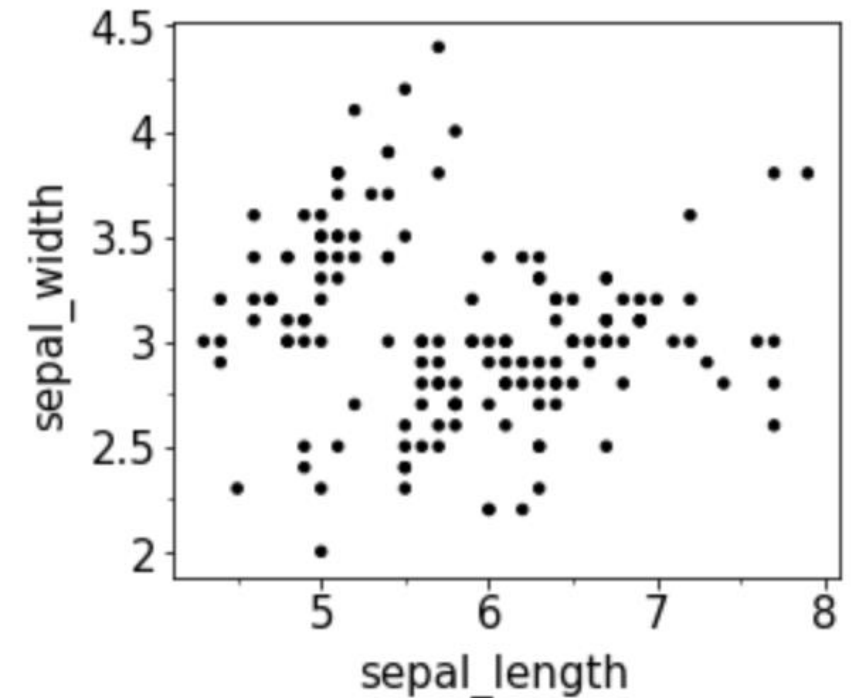
2変数(連続値)の関係の傾向を確認する.

df

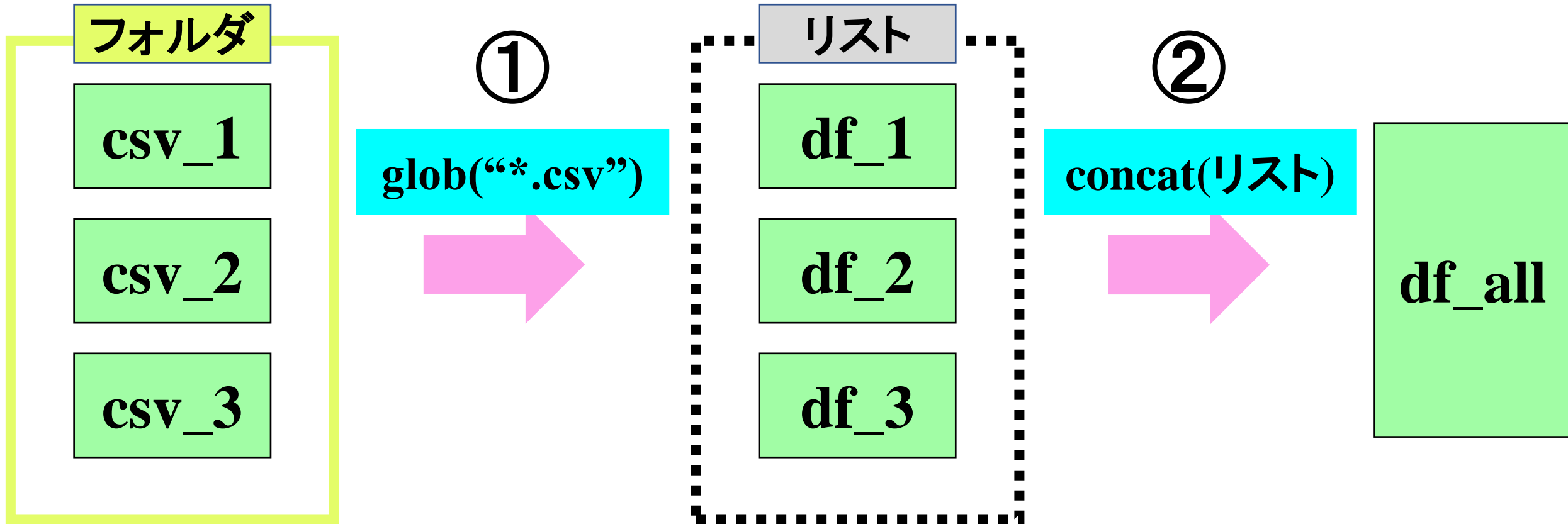
sepal_length	sepal_width
5.1	3.5
4.9	3.0
4.7	3.2
:	:
5.0	3.6
5.4	3.9
4.6	3.4



散布図



# 複数ファイルのマージ



# 散布図行列の作成

各変数(連続値)の関係の傾向を確認する.

hue: カテゴリ別

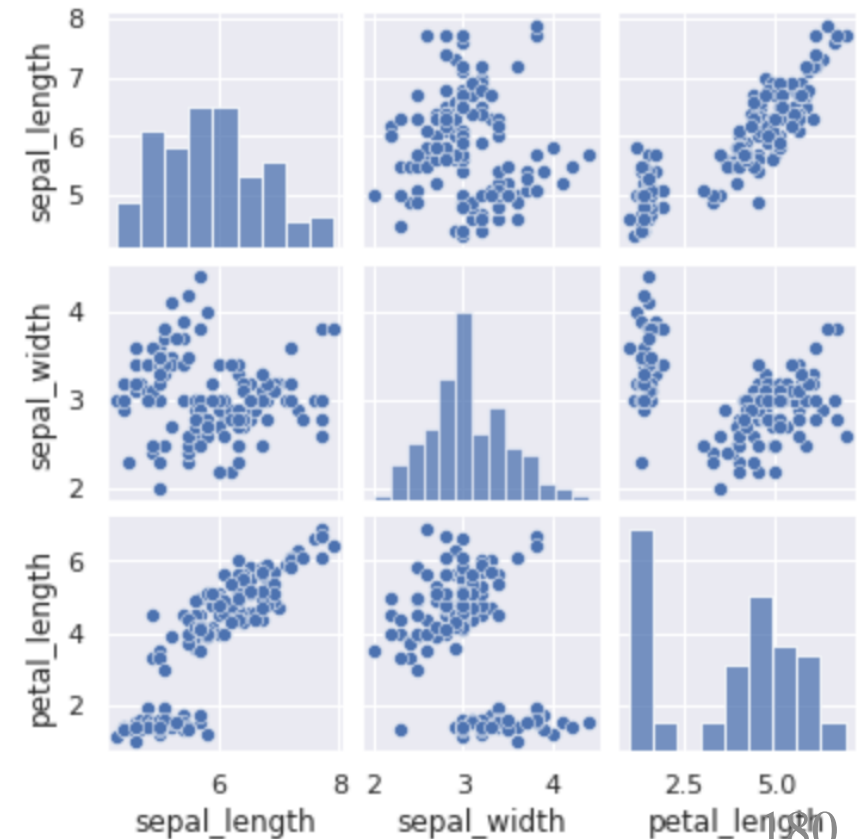
df



`sns.pairplot(df)`

pairplot

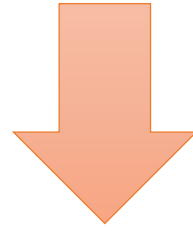
散布図行列



# 総合演習

目的

今までに学習したことを利用し、実務でよく使用する前処理を経験することで、データの前処理に関する理解を深める！！



実践

映画のデータを使用して、総合演習！！

## info\_movies

#	列名	意味
1	ID	主キー
2	Title	タイトル
3	Year	上映年
4	Age	推奨年齢
5	IMDb	映画のデータベースによる点数(0～10で評価)
6	Rotten_Tomatoes	映画評論サイトによる点数(0～100%で評価)
7	Directors	監督
8	Genres	ジャンル
9	Country	上映国
10	Language	言語
11	Runtime	上映時間(単位:分)

## info\_platforms

#	列名	意味
1	ID	主キー
2	Age	推奨年齢
3	Netflix	Netflixで放映しているか.
4	Hulu	Huluで放映しているか.
5	Prime_Video	Prime_Videoで放映しているか.
6	Disney	Disneyで放映しているか.

# 問題

1. 以下を実施し、データを整理しましょう。

- ① info\_movies, info\_platformsをそれぞれ縦に結合する.
- ② ①でできたデータを内部結合(inner join)する.
- ③ ②でできたデータに以下の処理をする.
  - 欠損値(NA)がある行を削除
  - Netflix, Hulu, Prime\_Video, Disneyに対し, 0, 1以外の行を削除
  - IMDb列を10倍
  - Rotten\_Tomatoes列の%を取って, 数値型(float型)に変換
- ④ ③でできたデータをcsvにして保存する.

※以降の問題は, 1.で作成したデータを使用してください.

# 問題

2. Directorsごとの映画の作成数を算出し, 作成数の降順(大きい順)にソートしましょう.

※ Directorsが複数人の場合でも1人とみなしてください.

3. プラットフォームの配信パターンごとにIMDbとRotten\_Tomatoesの平均値を算出しましょう.

4. Country列からそれぞれの映画が何か国で上映されたか算出しましょう.



# 問題

5. 各プラットフォームで放映されている作品に対し、推奨年齢の割合を算出しましょう.

1 : 放映されている

0 : 放映されていない

6. IMDb列の大きさにより以下の条件で分類し、それぞれについて、Runtimeのヒストグラムと密度曲線を作成しましょう.

▪ 70以上 : high

▪ 50以上70未満 : middle

▪ 50未満 : low

7. AgeごとにIMDb, Rotten\_Tomatoes, runtimeの散布図行列を作成しましょう.

# 注意事項

・各問題について、説明のレクチャーと解説のレクチャーに分かれています。  
以下の流れで進めることをオススメします。

- ① 説明のレクチャーをみて、どのような前処理をするか把握する。
- ② ①を踏まえ、自分で考えてコーディングする。
- ③ 解説のレクチャーをみて、②のコーディング内容と照らし合わせる。

※上記の流れはあくまでオススメなので、自分の進めやすいように進めてOKです。

・これらの問題は、前回までのレクチャーを参考にしたり、ググったりしてもOKです。

・一部の問題については、前回までのレクチャーで学習していない部分もあります。そのような問題でもググって自分の答えを探してみましょう。

# 問題:1

1. 以下を実施し、データを整理しましょう。

- ① info\_movies, info\_platformsをそれぞれ縦に結合する.
  - ② ①でできたデータを内部結合(inner join)する.
  - ③ ②でできたデータに以下の処理をする.
    - 欠損値がある行を削除
    - Netflix, Hulu, Prime\_Video, Disneyに対し, 0, 1以外の行を削除
    - IMDb列を10倍
    - Rotten\_Tomatoes列の%を取って, 数値型(float型)に変換
  - ④ ③でできたデータをcsvにして保存する.
- ※以降の問題は, 1.で作成したデータを使用してください.

# 解説: 1

#	パッケージ	関数・メソッド	用途
1	pathlib	Path	パスオブジェクトの作成
2	pathlib	glob	パスの取得
3	pandas	concat	縦結合
4	pandas	merge	キー結合 (inner_join)
5	pandas	dropna	欠損値行削除
6	pandas	query	行の抽出
7	pandas	assign	列の追加・更新
8	pandas	str.replace	マッチの単数置換
9	pandas	to_csv	csvの出力

## 問題:2

2. Directorsごとの映画の作成数を算出し, 作成数の降順(大きい順)にソートしましょう.

※ Directorsが複数人の場合でも1人とみなしてください.

## 解説: 2

2. Directorsごとの映画の作成数を算出し, 作成数の降順(大きい順)にソートしましょう.

※ Directorsが複数人の場合でも1人とみなしてください.

#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	pandas	groupby	グルーピング
3	pandas	count	集約メソッド
4	pandas	sort_values	ソート

## 問題:3

3. プラットフォームの配信パターンごとにIMDbとRotten\_Tomatoesの平均値を算出しましょう.

## 解説: 3

3. プラットフォームの配信パターンごとにIMDbとRotten\_Tomatoesの平均値を算出しましょう.

#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	pandas	groupby	グルーピング
3	pandas	mean	集約メソッド



## 問題:4

4. Country列からそれぞれの映画が何か国で上映されたか算出しましょう.

## 解説: 4

4. Country列からそれぞれの映画が何か国で上映されたか算出しましょう.

#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	pandas	assign	列の追加・更新
3	pandas	str.split	マッチの分割
4	pandas	map	シリーズの行処理

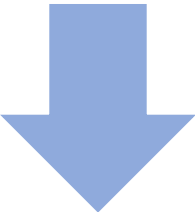
## 問題:5

5. 各プラットフォームで放映されている作品に対し、推奨年齢の割合を算出しましょう.

1 : 放映されている

0 : 放映されていない

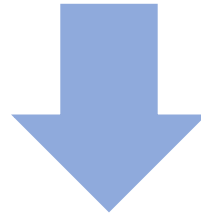
Hulu



platform	13+	16+	18+
Hulu	0.2	0.2	0.6

**プラットフォーム** : Disney, Hulu, Netflix, Prime\_Video  
**推奨年齢** : 7+, 13+, 16+, 18+, all

※数値は正確ではない



platform	7+	13+	16+	18+	all
Disney	0.1	0.1	0.2	0.3	0.3
Hulu	0.2	0.5	0.1	0.1	0.1
Netflix	0.3	0.2	0.1	0.2	0.2
Prime_Video	0.1	0.1	0.1	0.1	0.6

## 解説:5

5. 各プラットフォームで放映されている作品に対し、推奨年齢の割合を算出しましょう。      1 : 放映されている    0 : 放映されていない

#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	pandas	melt	縦変換
3	pandas	query	行の抽出
4	pandas	groupby	グルーピング
5	pandas	count	集約メソッド
6	pandas	sum	集約メソッド
7	pandas	assign	列の追加・更新
8	pandas	merge	キー結合(left_join)
9	pandas	pivot	横変換

## 問題:6

6. IMDb列の大きさにより以下の条件で分類し, それぞれについて, Runtimeのヒストグラムと密度曲線を作成しましょう.

- 70以上 : high
- 50以上70未満 : middle
- 50未満 : low

## 解説: 6

6. IMDb列の大きさにより以下の条件で分類し, それぞれについて, Runtimeのヒストグラムと密度曲線を作成しましょう.

- 70以上 : high
- 50以上70未満 : middle
- 50未満 : low

#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	pandas	assign	列の追加・更新
3	pandas	map	シリーズの行処理
4	plotnine	geom_histogram	ヒストグラム
5	plotnine	facet_wrap	カテゴリごとに出力(複数)
6	plotnine	geom_density	密度曲線



## 問題:7

7. AgeごとにIMDb, Rotten\_Tomatoes, runtimeの散布図行列を作成しましょう.

## 解説: 7

7. AgeごとにIMDb, Rotten\_Tomatoes, runtimeの散布図行列を作成しましょう.

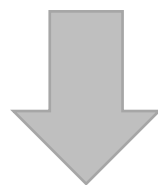
#	パッケージ	関数・メソッド	用途
1	pandas	filter	列の抽出
2	seaborn	pairplot	散布図行列の作成

# 全体のまとめ

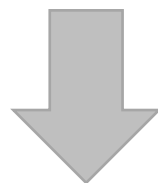
#	パート	内容
①	環境構築	Google Colaboratory
②	Python	<ul style="list-style-type: none"><li>・データ型</li><li>・リスト</li><li>・辞書</li><li>・条件分岐処理</li><li>・繰り返し処理</li><li>・関数</li><li>・クラス</li></ul>
③	パッケージ	<ul style="list-style-type: none"><li>・numpy</li><li>・pandas</li><li>・plotnine</li></ul>
④	総合演習	映画データの前処理

# Pythonの難しいところ

コーディング方法が様々！



あまり良くないコーディング方法 ⇒ bad  
オススメのコーディング方法 ⇒ good



個人的な主観

自分自身のgoodとbadの確立！！

# PythonとR



**統一的なコーディング！！**

**PythonとR, 両方必要！！**