第2回 Python基礎

松尾・岩澤研究室
MATSUO-IWASAWA LAB UTOKYO

2024/10/15

今回の目的と目標

- 目的
  - データサイエンスとPythonの関係を理解し、Pythonを学ぶ意義を把握する。
  - GCIを受講する上で必要なGoogle Colaboratoryの操作を習得する。
  - Pythonの基本文法を理解し、今後の講義内容に対応できるようにする。

- 目標
  - Google Colaboratoryの基本操作ができるようになる。
  - 演算子、変数、コレクションを使って簡単な計算処理を記述できるようになる。
  - 条件分岐や繰り返し処理を使って、効率的なプログラムを作成できるようになる。
  - 関数を定義して、特定の処理をまとめられるようになる。

本講義では時間の関係上、全ての文法項目を説明することはできません。今後のGCIの講義で出てくるもっとも重要な事項について、できるだけ丁寧に説明を行います
詳細については、事前学習資料や動画などを活用してください。

目次

- なぜPythonを学ぶのか
  - Pythonの特徴とデータサイエンスとの関係
  - Pythonの学びかた
  - Google Colaboratryを触ってみよう

- Python文法I (演算子、変数、データ型)
- Python文法II (コレクション)
- Python文法III (条件分岐、繰り返し)
- Python文法IV (関数、ライブラリ)
- 発展事項

- まとめ
- 今週の宿題

Pythonの特徴とデータサイエンスとの関係

- 使いやすさと読みやすさ
  - Pythonはシンプルで読みやすい構文を持つため、プログラミング初心者でも比較的容易に習得できる
- 豊富なライブラリ
  - データサイエンスやAIで必要なライブラリが豊富に揃っている
- 学術研究での利用
  - 多くの学術論文や研究プロジェクトで使われており、最新の技術・アルゴリズムを利用しやすい。
- 求人市場での需要
  - Pythonはデータサイエンス、AI分野で最も求められるスキルの一つであり、とで就職やキャリアアップに直結する。

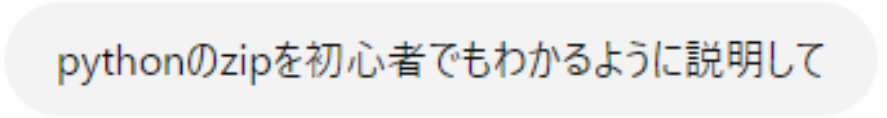
Python文法の学びかた

- 一度にすべてを完璧にしようとしない
  - 少しずつ、使いながら覚えていけば大丈夫です
- わからないときは、WebやChatGPTで調べる（slackで質問もGOOD！）
  - 公式サイトも見てみる
    - https://docs.python.org/ja/3/reference/introduction.html
- プログラムを自分でタイピング（写経）するのも効果的

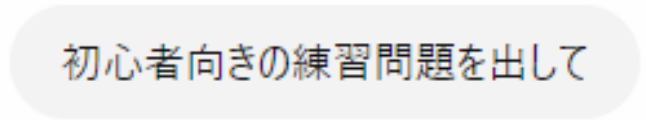
GCIでは、宿題やコンペ、最終課題などでPythonを使う機会がたくさんあります。
特にコンペで順位を上げていくためには、コードを書いたり、何度も書き直したり、することになります

生成AIをプログラミング学習に活用する例

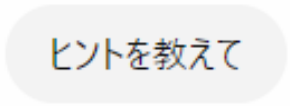
- 分からない用語を聞いてみる
  - プロンプト例：



- 問題を出してもらう
  - プロンプト例：



- ヒントを教えてもらう
  - プロンプト例：



出力されたエラーメッセージを入力するだけでも、色々教えてくれる時もあります

Google Colaboratryを触ってみよう

- 実際にGoogle Colaboratoryを立ち上げて、
- 簡単なプログラムを動かしてみます

実習

- 事前配布資料：Google Colabの使い方

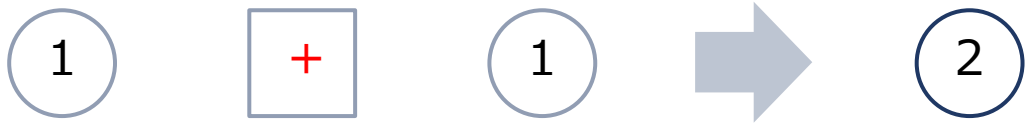
- 用語：
- ・Jupyter notebook (単にノートブックと呼ぶことが多い)
- ・コードセル、テキストセル
- 操作方法：
- 新規ノートブックの作成
- ノートブックの保存
- セルの実行の仕方

目次

演算子

プログラミングでは、計算処理を指示する**演算子**によって新たな値に**評価**されます
特に、足し算や引き算などの**算術演算子**は、数学のように数値間に置いて計算します

(例) 算術演算子：足し算



コード例

```
In  [1]: print(1 + 1)
Out [1]: 2
In  [2]: print(1 - 1)
Out [2]: 0
```

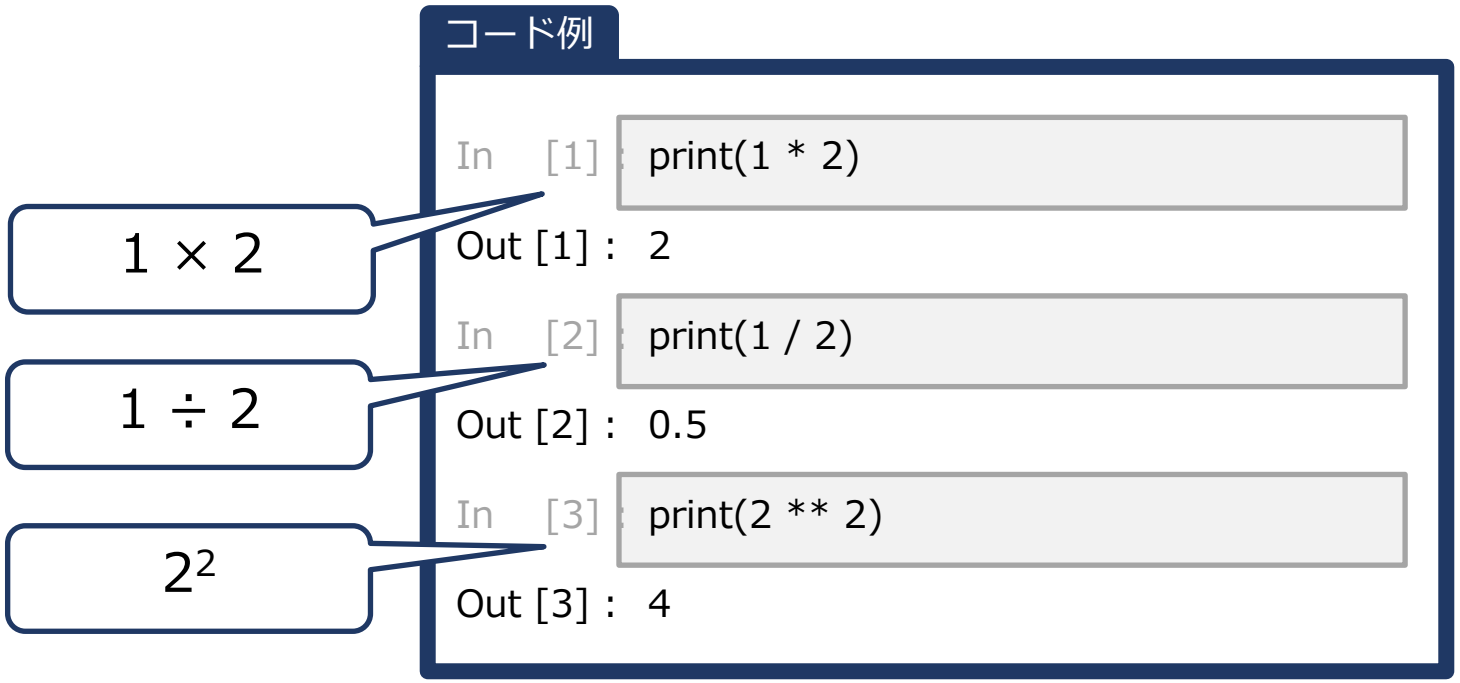
算術演算子以外の演算子に比較演算子や論理演算子があります（Python文法Ⅱ）。

算術演算子の種類

Pythonでは、足し算や引き算は数学の記号と共通しますが、
掛け算や割り算、べき乗は特有の記号となります

算術演算子の記号

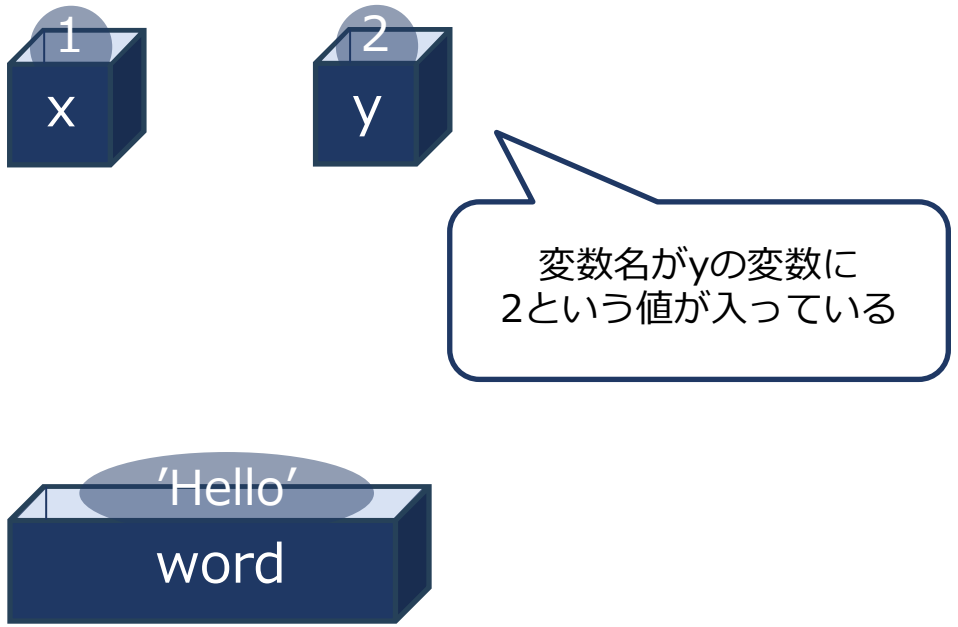
算術演算	演算子記号
足し算	+
引き算	-
掛け算	*
割り算	/
割り算（整数部）	//
割り算（余り部）	%
べき乗	**



「割り算（余り部）」は、あるデータがちょうど割り切れるか調べるのに便利。例）「% 2」 ➤ 偶数なら0
➤ 奇数なら1

変数

データを格納するための箱を変数と言い、計算結果を保持します
変数名=値とすることで、左辺の変数に右辺のデータを代入します



コード例

```
In [1]: x = 1
        y = 2
        print(x+y)
```

Out [1]： 3

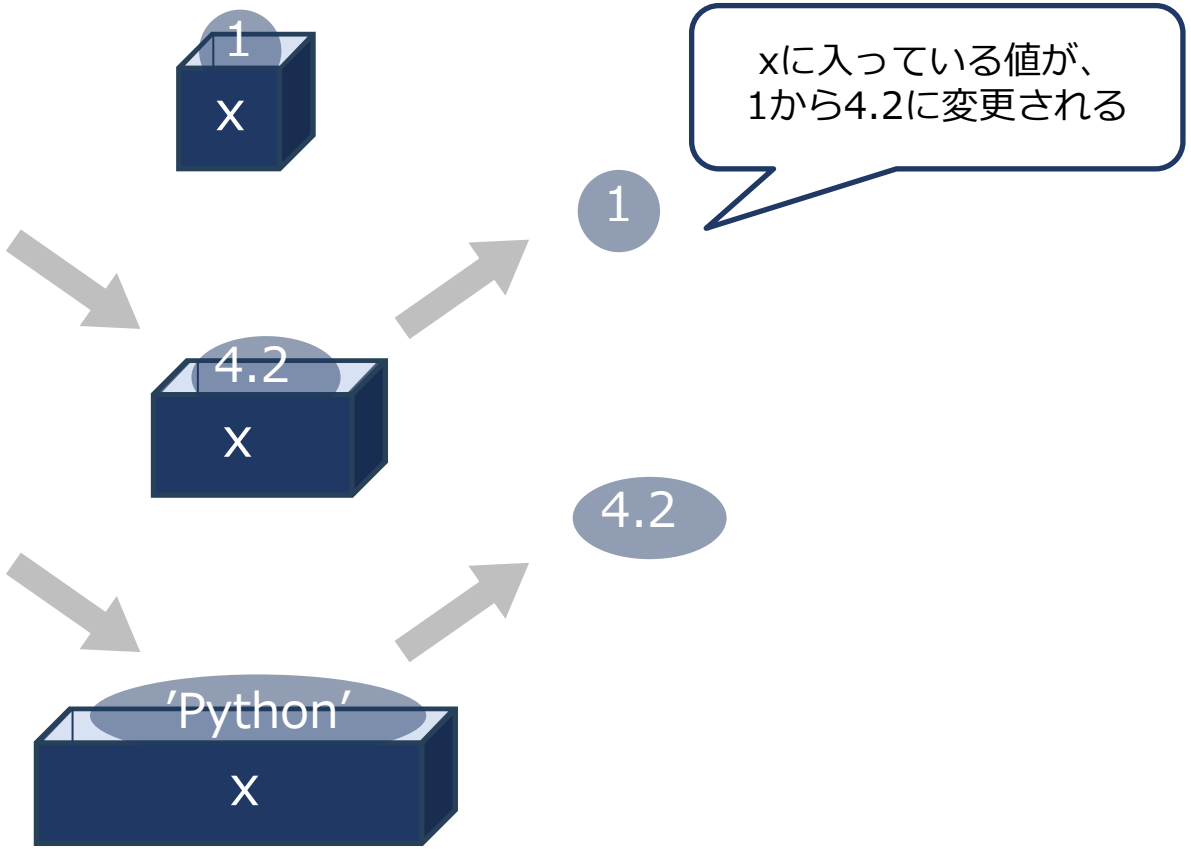
```
In [2]: word = 'Hello'
        print(word)
```

Out [2]： Hello

変数には様々なデータ型の値を代入できる（Python文法Ⅰ：データ型）

変数への再代入

同じ変数に再代入することで、格納されている値が変更されます



コード例

```
In [1]: x = 1
        print(x)
Out [1]: 1

In [2]: x = 4.2
        print(x)
Out [2]: 4.2

In [3]: x = 'Python'
        print(x)
Out [3]: Python
```

データ型

プログラムで扱う値の種類を**データ型**と言います
変数に代入した値のデータ型は、type関数で調べられます

主なデータ型

データ型	内容	例
int	整数	1
float	小数	3.14
str	文字列	‘Python’
bool	真偽値	True、False

(例）string1 = ‘文字列1’
　　string2 = “文字列2”
「‘」または「 “” 」で挟むと文字列
　　代入時に変数の型がstr型に

type関数：
かっこ内に変数入力
➤ データ型出力

コード例

```
In  [1]: num = 1
         print(type(num))
Out [1]: int

In  [2]: pi = 3.1
         print(type(pi))
Out [2]: float
```

※ bool型は「はい」と「いいえ」のような2値で条件確認に利用できます（Python文法Ⅱ）

文法Iの練習問題

- 「GCI2024W_第2回_Python基礎.ipynb」を開く
- 練習問題を解く

実習

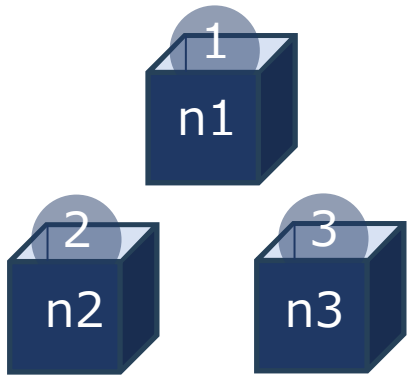
目次

- 文法II
  - コレクション
  - List
  - Tuple
  - Dict
  - 練習問題

コレクション

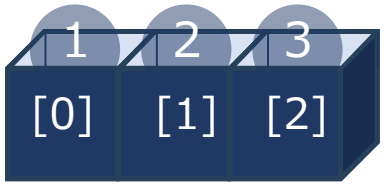
コレクションでは値をまとめて管理できます
リストやタプル、辞書などの種類があります

変数ごとに値代入

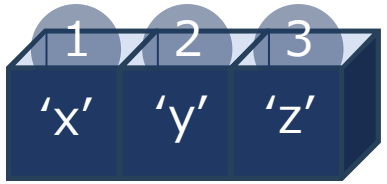


まとめる

コレクションに値代入



順番で管理



キーワードで管理

（例）

・リスト
・タプル

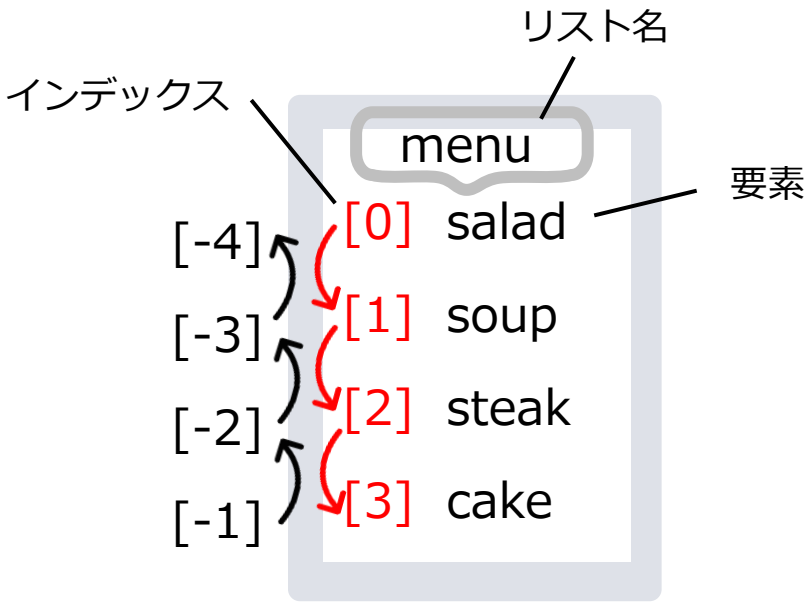
・辞書

コレクションでは変数名は1つになります

リスト

リストは角かっこ[]を使って値（**要素**）を順番（**インデックス**）で管理します
リストからインデックスを指定して要素の抽出・変更(置換)が可能

リストの要素とインデックス



メニューの
2番目は？

コード例

```
In [1]: menu = ['salad', 'soup', 'steak', 'cake']
        print(menu[1])
```

Out [1]： soup

最後プリン
に変更

```
In [2]: menu[-1] = 'pudding'
        print(menu)
```

Out [2]： ['salad', 'soup', 'steak', 'pudding']

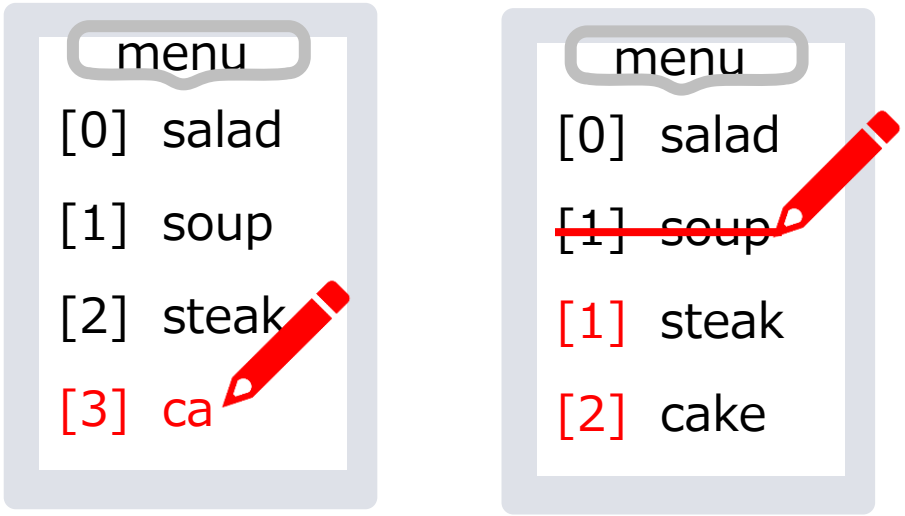
インデックスは0から、後ろからは-1から数えます
リストの要素はstr型以外にint型やfloat型などのデータ型も可能

リストの追加・削除

リストは**要素の追加・削除**などの関数を持ちます
リストの変数名の後ろに.(ドット）を付けて利用できます

- **要素の追加**
  リスト変数名.append(末尾追加要素)

- **要素の削除**
  リスト変数名.remove(削除要素)



コード例

```
In [1]: menu = ['salad', 'soup', 'steak']
        menu.append('cake')
        print(menu[-1])
Out [1]: cake

In [2]: menu.remove('soup')
        print(menu)
Out [2]: ['salad', 'steak', 'cake']
```

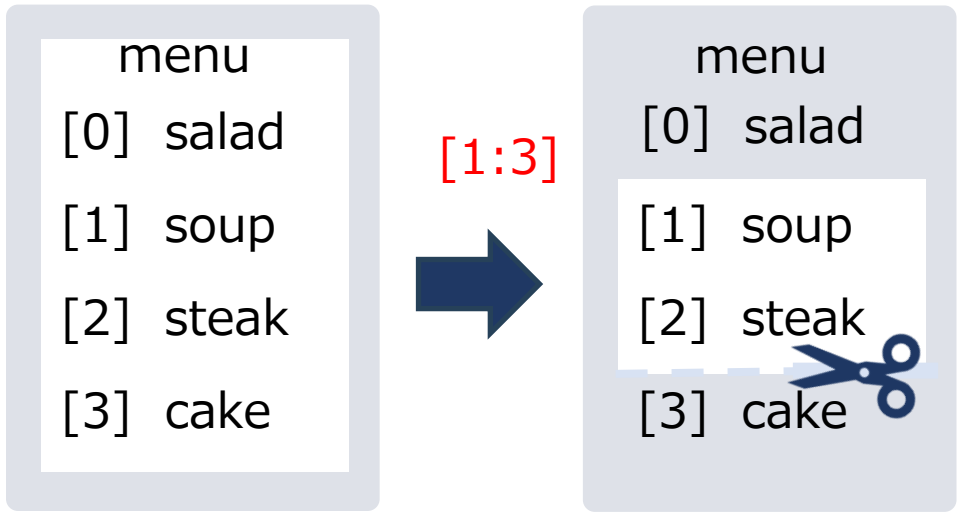
removeで削除される要素がリスト内に重複する場合、該当する最初の要素が削除されます

リストの切り出し

リストからインデックスの範囲を指定してリストを切り出せます

リストの切り出し：スライス

リスト変数名[始点：終点]



1stと2nd
ディッシュ

コード例

```
In [1]: menu = ['salad', 'soup', 'steak', 'cake']
        print(menu[1:3])
Out [1]: ['soup', 'steak']
In [2]: menu2 = menu[:-1]
        print(menu2)
Out [2]: ['salad', 'soup', 'steak']
```

始点を省略

スライス表記の[x:y]はx以上y未満を表し、xやyを省略した場合は残る範囲まで選択されます

2次元リスト

2重の角かっこ[[]]を使って2次元の表データなどを管理します
行や列番号を指定して要素やリストを抽出できます

2次元のリスト

(例) 2行3列の2次元リスト

	[0]列目	[1]列目	[2]列目
[0]行目	[['24/3' ,	'24/4' ,	'24/5'],
[1]行目	[80,	60 ,	90]]

24/4

0行1列の要素
[0][1]

コード例

```
In  [1] : table = [['24/3', '24/4', '24/5'], [80, 60, 90]]
          print(table[0][1])
Out [1] : 24/4
In  [2] : print(table[1][2])
Out [2] : 90
```

1行2列の要素
[1][2]

3つ以上の多次元のリストも作成可能です

タプル

リストの中身を変更できないものがタプルです
丸かっこ(,)を使い、データを管理します

タプルは一度定義すると中身を変更できないので、特に順番やデータ型が変更されると困る際に使用します

それ以外は基本的にリストと同様です

コード例

```
In  [1]: color = ('red', 'green', 'blue')
         print(color[0])

Out [1]: red

In  [2]: menu = ('rice', 200)
         print(menu[1])

Out [2]: 200
```

データ型の違う要素を管理することも多い

辞書

辞書は{}を使い、キー(key)とそれに対応する値(value)のペアでデータを保持します
要素には順番がなく、各キーを一意(ユニーク)にすることで、ペアの値にアクセスします

辞書は
{キー：値, キー：値, …}の形を取る
例　{‘apple’：100, ‘banana’：80, …}
　　キー　　値

キーには数値、文字列、タプルなどが入れられる

値の追加、更新関数

辞書変数名[キー名] = 値

キー名のキーが辞書内に存在しない場合は
　新しい値が追加される
すでに存在している場合は
　そのキーに対応する値が更新される

コード例

```
In  [1] : menu = {‘apple’ : 100, ‘banana’ : 80, ‘orange’ : 120}
          print(menu[‘apple’])
Out [1] : 100

In  [2] : menu[‘grape’] = 150
          print(menu)
Out [2] : {‘apple’ : 100, ‘banana’ : 80, ‘orange’ : 120, ‘grape’ : 150}

In  [3] : menu[‘banana’] = 60
          print(menu)
Out [3] : {‘apple’ : 100, ‘banana’ : 60, ‘orange’ : 120, ‘grape’ : 150}
```

キーが‘grape’で値が150の要素を追加

キーが’banana’の値を更新

辞書

del 変数名[キー名]で要素を削除でき、複数要素を同時に処理できます

要素の削除関数

del 辞書変数名[キー名] (, [キー名]…)

キーを指定

コード例

```
In  [1] : menu = {'apple' : 100, 'banana' : 80, 'orange' : 120}
          del menu['apple']
          print(menu)
Out [1] : {'banana' : 80, 'orange' : 120}

In  [2] : menu = {'apple' : 100, 'banana' : 80, 'orange' : 120}
          del menu['apple'], menu['orange']
          print(menu)
Out [2] : {'banana' : 80}
```

複数指定して削除も可能

文法IIの練習問題

- 「GCI2024W_第2回_Python基礎.ipynb」を開く
- 練習問題を解く

実習

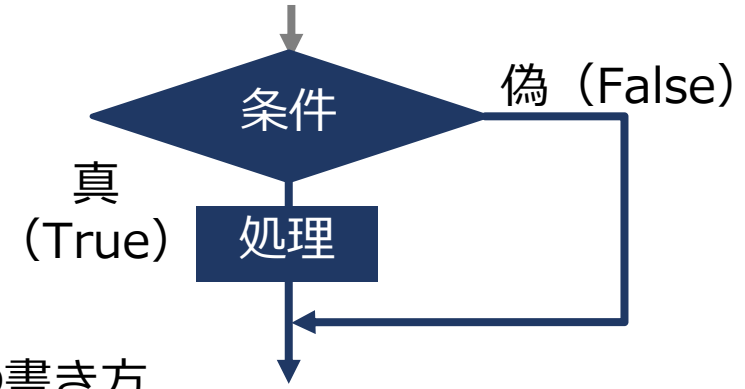
目次

- 文法III
  - 条件分岐、繰り返し
  - 条件分岐(if文)
  - 繰り返し(for文)
  - 練習問題

条件分岐（if文）

条件分岐(if文)は条件の真偽値（TrueかFalse）によって**処理の流れを変更**するコードの記載方法のことを意味します

if文（もし～条件なら～する）



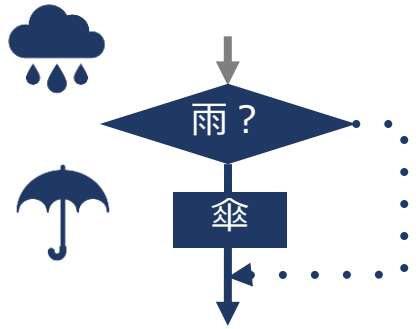
コードの書き方

```
if 条件の真偽値:
    真のときの処理
```

（インデント）

真偽値が真の場合のみ:（コロン）下のインデント(スペース4つ)以降の処理が実行され、偽の場合は実行されません

(例) 雨なら
傘を持つ



コード例

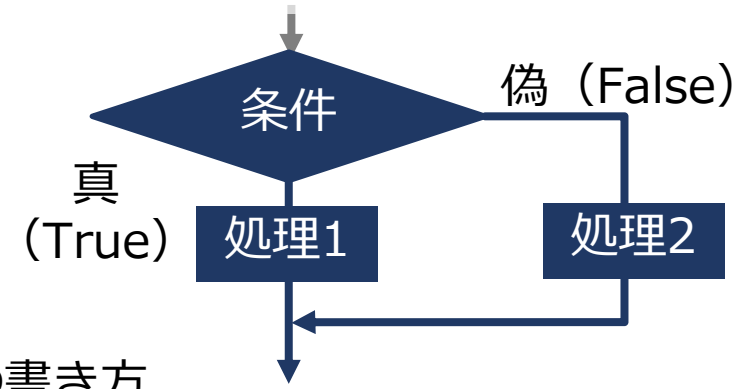
```
In [1]: is_rain = True
        if is_rain:
            print('傘を持つ')

Out [1]: 傘を持つ
```

※真偽値を代入する変数名には「〇〇かどうか」という意味で「is_〇〇」をよく使います

条件分岐（if-else文）

if文に**else**を使用すると、条件が偽(False)となる場合の処理を付け足せます

if-else文（もし～なら～する、
そうでないなら～する）

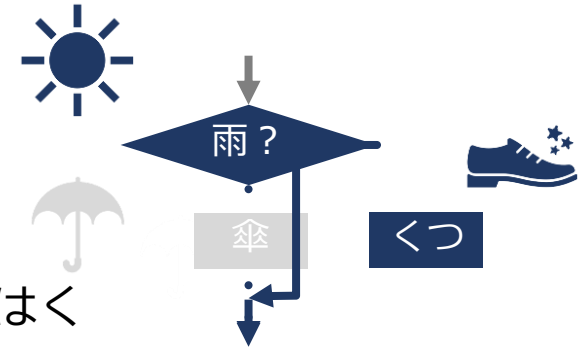


コードの書き方

if 条件の真偽値:
　インデント　真のときの処理1
else:
　インデント　偽のときの処理2

真偽値が真の場合、処理1が実行され、偽の場合、処理２が実行

(例) 雨なら
　傘を持つ
そうでないなら
　新しいくつをはく



コード例

```
In [1]: is_rain = False
        if is_rain:
            print('傘を持つ')
        else:
            print('新しいくつをはく')
```

Out [1]： 新しいくつをはく

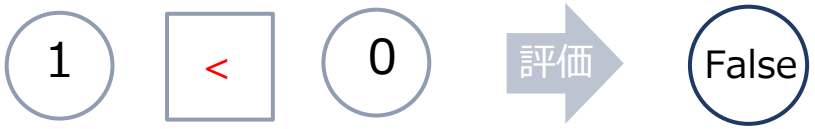
比較演算子

比較演算子は左右の値と評価されて真偽値を表わすものです。if文での条件に利用できます

比較演算子の記号

演算子記号	意味
==	左右の値が同じ
!=	左右の値が同じでない
>	左が右の値より大きい
<	左が右の値より小さい
>=	左が右の値以上
<=	左が右の値以下

(例) 比較演算子：<（小なり）



コード例

```
In  [1]: print(1 == 1)
Out [1]: True

In  [2]: print(1 < 0)
Out [2]: False

In  [3]: age = 18
         if age < 20:
            print('No Drink')
Out [3]: No Drink
```

True

論理演算子

論理演算子は複数の演算の真偽値を評価するものです
特に、if文で比較演算子と組み合わせて使用することが多いです

論理演算子の記号と優先度

演算子記号	意味
and	かつ
or	または
not	でなければ

演算子	優先度
算術	高
比較	中
論理	低

(例) and演算子

(例) not演算子

コード例

```
In  [1]: print(True and False)
Out [1]: False
In  [2]: print(True or False)
Out [2]: True
In  [3]: BMI = 20
         if BMI >= 18 and BMI < 25:
            print('normal weight')
Out [3]: normal weight
```

True and True ➤ True

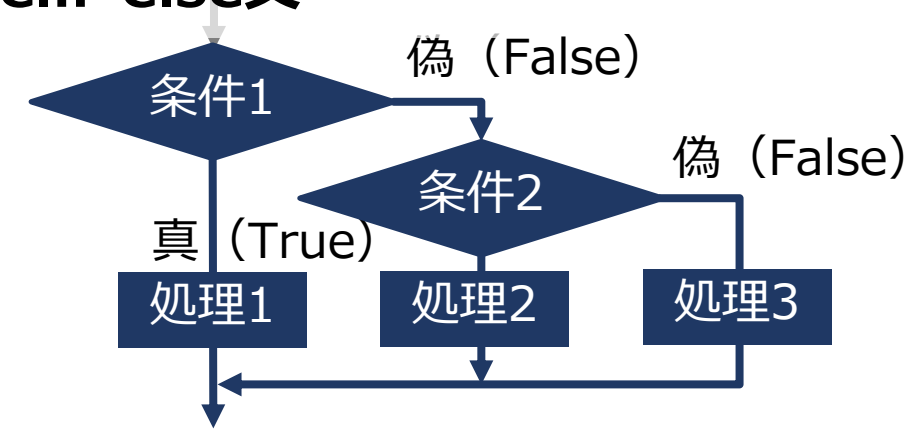
※BMIは体重と身長から算出される肥満度指標

andとorは左右の真偽値とともに評価され、notは右の真偽値のみと評価されるので注意

条件分岐（if-elif-else文）

条件分岐を3つ以上にしたい場合に**elif**を追加します
比較・論理演算子と組み合わせてより複雑な条件分岐が可能です

if-elif-else文



コードの書き方

```
if 条件1:
    インデント 条件1が真のときの処理1
elif 条件2:
    インデント 条件1が偽で条件2が真のときの処理2
else:
    インデント 条件1と2が偽のときの処理3
```

elifはelse ifの略です。elifを複数使用する場合、最初に真となる条件だけ処理が実行されます

コード例

```
In [1]: BMI = 20

if BMI >= 25:
    print('over weight')
elif BMI >= 18 and BMI < 25:
    print('normal weight')
else:
    print('under weight')

Out [1]: normal weight
```

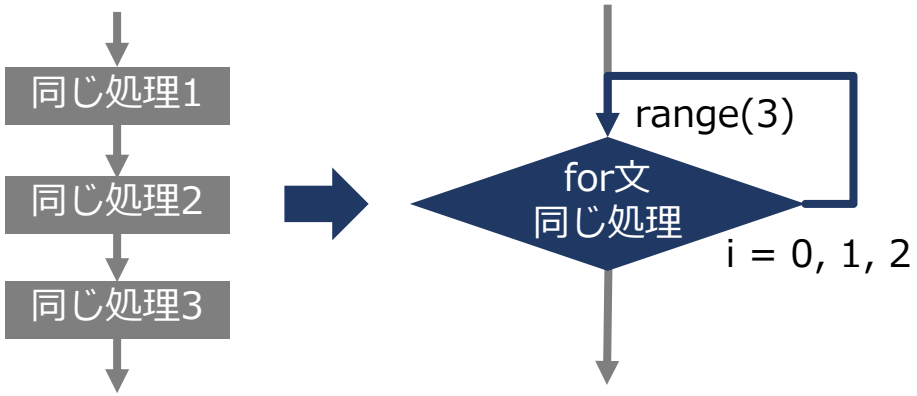
False

True

※「and BMI < 25」の部分は省略しても同じ動作

繰り返し（for文）

for文は**同じ処理を何度も書かずに繰り返したい**場合に使用します
範囲をrange()で指定して繰り返せます



コードの書き方

0, 1, 2…

```
for 変数 in range(繰り返す回数):
    繰り返し処理
```

インデント

指定回数が終わるまで:（コロン）下のインデント(スペース4つ)
以降の処理が続きます。変数は0から処理ごとに1ずつ増えます

コード例

カンマを挟むと
スペースが空く

```
In  [1]: for i in range(5) :
             print('Python文法', i)

Out [1] : Python文法 0
          Python文法 1
          Python文法 2
          Python文法 3
          Python文法 4
```

※range(5)はrange(0,5)の省略形。range(x, y)を
使用するとxから始まりy未満までと指定できます

※for文内で定義した変数に特別な意味がない数であれば、
変数名にiやjなどの文字がよく使われます

for文とリスト

for文に**リスト**を組み合わせると**各要素を取り出し**て同じ処理を繰り返せます

コードの書き方

リスト[0]、リスト[1]、…

```
for 変数 in リスト:
    インデント 繰り返し処理
```

リストの要素が変数に順に代入され、要素数分処理が繰り返されます

menu
[0] salad
[1] soup
[2] steak
[3] cake

menu順に出される

コード例

```
In [1]: menu = ['salad', 'soup', 'steak', 'cake']

        for food in menu:
            print(food)

Out [1]: salad
         soup
         steak
         cake
```

Question: menuとfoodの型は何ですか？

for文とリスト

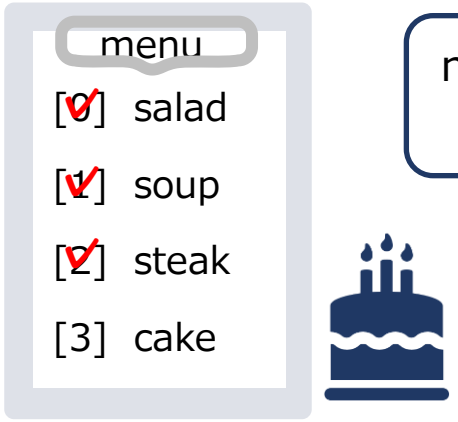
for文に**リスト**を組み合わせると**各要素を取り出し**て同じ処理を繰り返せます

コードの書き方

リスト[0]、リスト[1]、…

```
for 変数 in リスト:
インデント 繰り返し処理
```

リストの要素が変数に順に代入され、要素数分処理が繰り返されます



menu順に出される

コード例

```
In [1]: menu = ['salad', 'soup', 'steak', 'cake']

        for food in menu:
            print(food)

Out [1]: salad
         soup
         steak
         cake
```

Question: menuとfoodの型は何ですか？
Answer：
・menuは、文字列を要素とするリスト
・foodは、文字列（単一の文字列変数でfor文の中で値が変わっていく）

for文とリスト

リストとfor文の組み合わせでインデックスと要素の両方を
利用したい場合、**enumerate()**が便利です

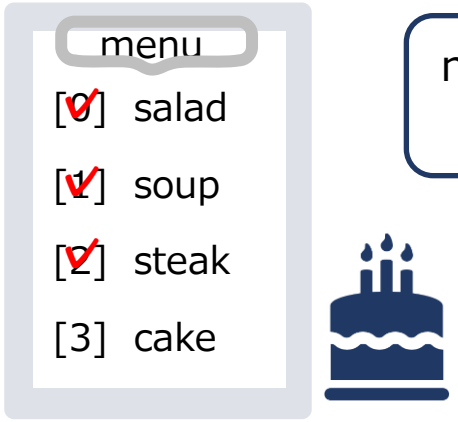
コードの書き方

0、1、…　　リスト[0]、リスト[1]、…

```
for 変数1, 変数2 in enumerate(リスト):
    繰り返し処理
```

（インデント）

変数1にリストのインデックスが、
変数2にリストの要素が順に代入され、処理が繰り返されます



menu順に
出される

コード例

```
In [1]: menu = ['salad', 'soup', 'steak', 'cake']

        for i, food in enumerate(menu):
            print(i, food)

Out [1]: 0 salad
         1 soup
         2 steak
         3 cake
```

enumerateは数える上げるという意味

文法IIIの練習問題

- 「GCI2024W_第2回_Python基礎.ipynb」を開く
- 練習問題を解く

実習

目次

- 文法IV
  - 関数
  - モジュール
  - 練習問題

関数とは

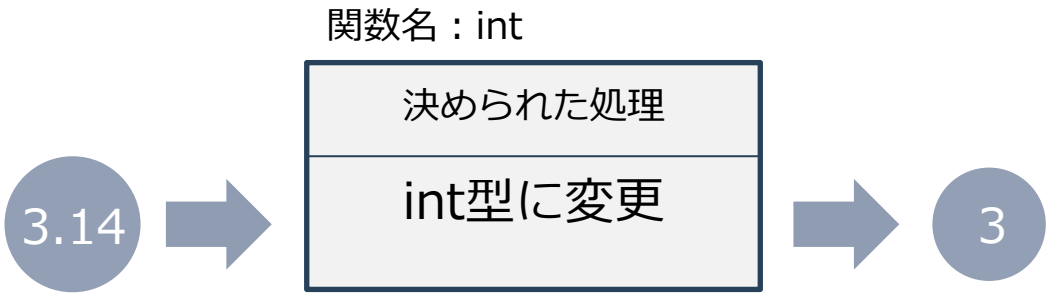
関数は決められた処理を実行するコードをひとくくりに管理するものです
Pythonには事前に定義されている関数が多くあります

これまでに使用したprint()やtype()も関数です

Pythonで事前に作られている関数の例

関数	処理内容
print()	値を表示
type()	データ型を出力
int()	整数のint型に変更
str()	文字列のstr型に変更
sum()	合計値を出力
len()	要素数を出力

（例）int関数のイメージ



関数の定義方法と使い方

開発者はオリジナルの関数を作成できます
何度も同じような処理を書かずに済み、メンテナンスも容易です

関数の定義と使い方

def 関数名():
インデント 決められた処理を行うコード

関数名()と書くと定義した処理が実行さます

（オリジナル関数例）

関数名：greet

決められた処理
“こんにちは”を出力

greet()は、インデントが戻っているので、関数定義の外です

コード例

```
In [1]: def greet():
            print(“こんにちは”)

        greet()
```

1回 ”こんにちは” を出力

```
Out [1]： こんにちは
```

```
In [2]: greet()
        greet()
```

2回 ”こんにちは” を出力

```
Out [2]： こんにちは
          こんにちは
```

defは、define（定義する）の略称です。定義のかっこ、カンマ、インデント（スペース4つ）は省略できないので注意

関数の引数

関数に変数の値を引き渡したい場合に**引数**を定義します

引数のある関数の定義と使い方

def 関数名(引数):
インデント 決められた処理を行うコード

関数名(引数)と書くと定義した処理が実行さます

（オリジナル関数例）

引数はnameです

コード例

```
In [1]: def greet(name):
            print(name+“さん、こんにちは”)

        greet(‘松尾’)
```

Out [1]： 松尾さん、こんにちは

```
In [2]: greet(‘岩澤’)
```

Out [2]： 岩澤さん、こんにちは

引数は関数定義のかっこ内にカンマを挟んで複数定義できます。例 ➤ def(引数1, 引数2, …):

関数の戻り値

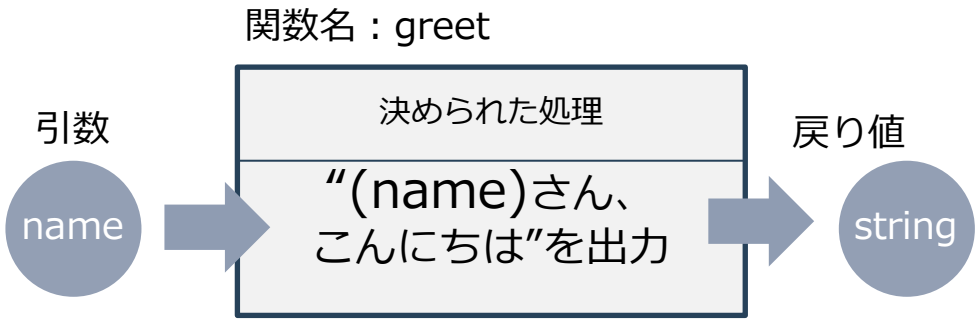
関数から処理結果を受け取りたい場合に**戻り値(返り値)**を定義します

戻り値のある関数の定義と使い方

def 関数名(引数):
インデント 決められた処理を行うコード
インデント return 戻り値

関数名(引数)で呼び出し、戻り値を変数に代入できます

（オリジナル関数例）



コード例

```
In [1]: def greet(name):
            string = name+“さん、こんにちは”
            return string

        string = greet(‘松尾’)
        print(string)
```

Out [1]： 松尾さん、こんにちは

モジュール

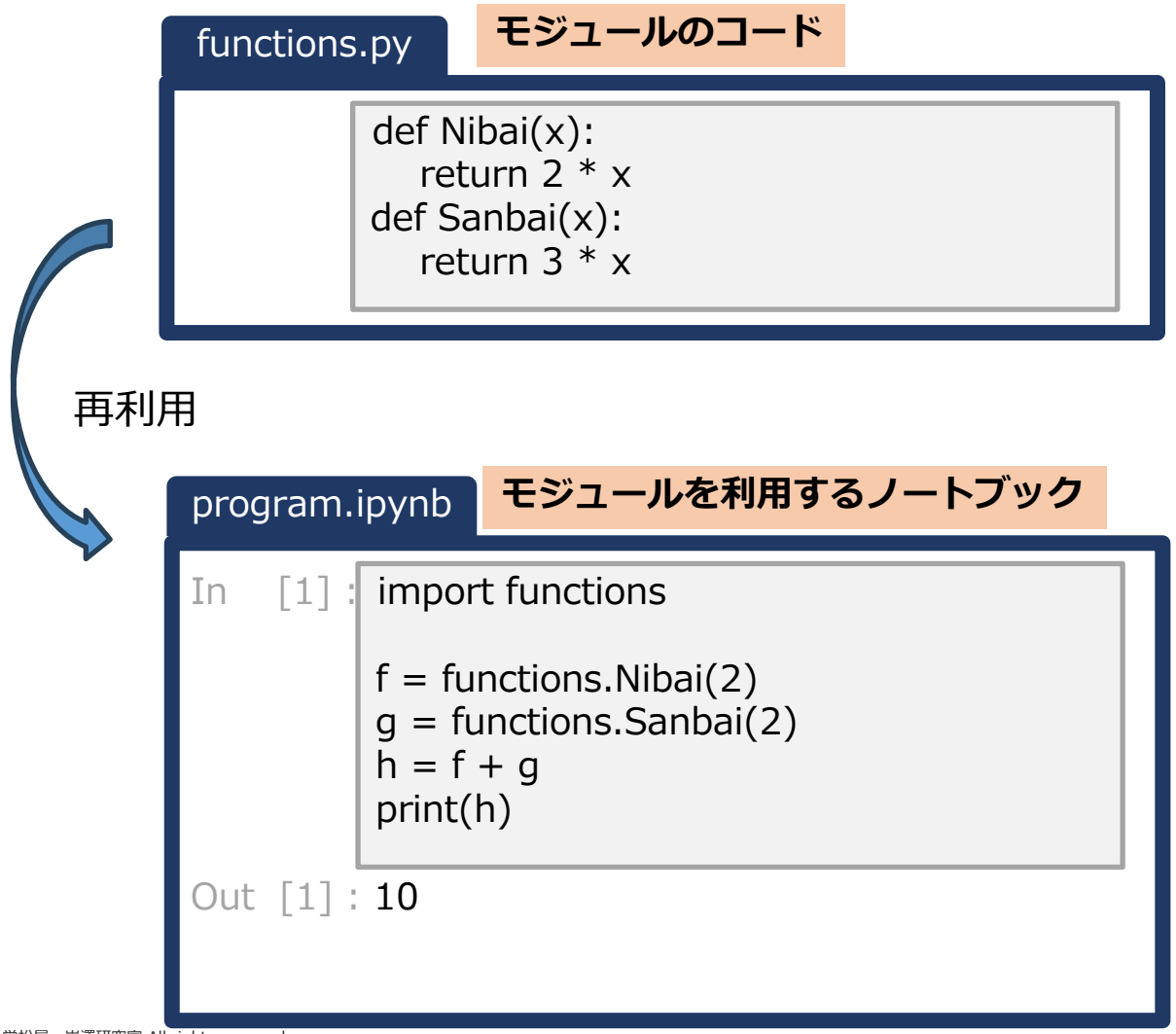
モジュールとは、プログラムを再利用するためのコードをファイルとしてまとめたものです

- 大きなプログラムを複数の小さなプログラムに分割して管理できる
- 他人が開発したプログラムを簡単に利用できる

importの書式：
Import モジュール名

※モジュール名はファイル名から.pyを除いたファイル名

Importされた関数の呼び出し方法：
モジュール名.関数名()



ライブラリ

ライブラリとは、モジュールを再利用可能な形でまとめたもの

- Pythonはデータ分析・機械学習のために開発されているライブラリが多数ある
- 仕様を知るだけで便利な道具が使い放題になる！
- この講座では様々なライブラリの使い方を学習する
  (Numpy, Pandas, Matplotlib, Scikit-learn, ……)

コード例

```
In  [1]: import numpy

         x = numpy.array([1, 2])
```

文法IVの練習問題

- １．「GCI2024W_第2回_Python基礎.ipynb」を開く
- ２．練習問題を解く

実習

(発展事項)クラス

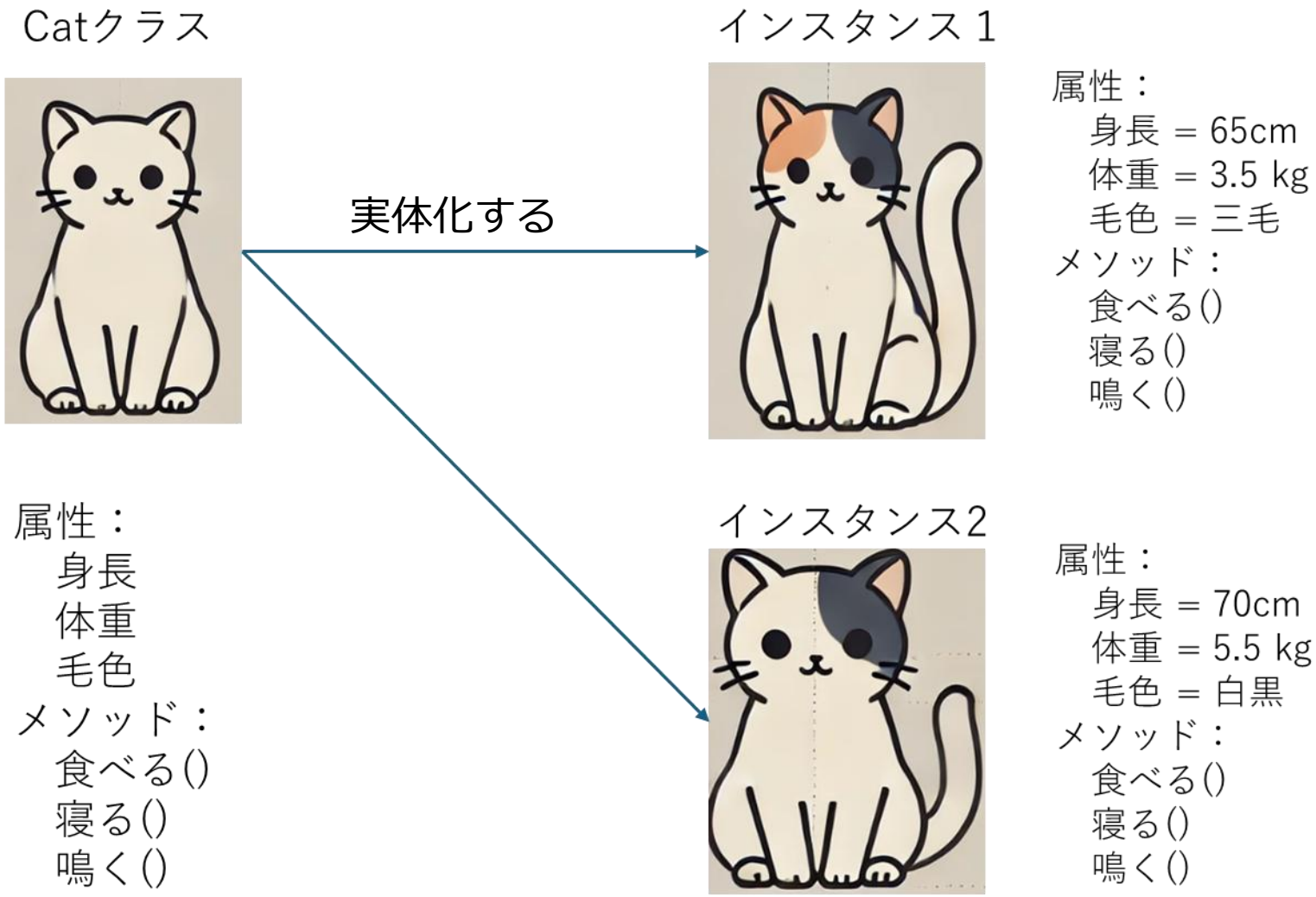
クラスとは**データと処理をまとめたもの**です

Pythonにはさまざまなクラスがあります

- **表データを管理するクラス**
- **データを可視化するクラス**
- **機械学習モデルのクラス**

(発展事項)クラスとインスタンス

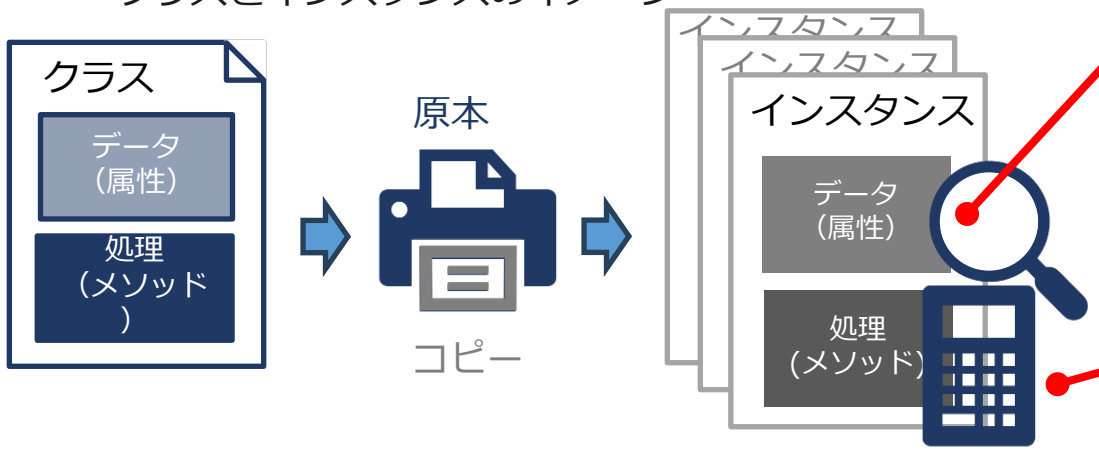
- クラスは、データと処理をまとめたひな型ですが、クラスを使うために、そのクラスのインスタンスを生成する必要があります



(発展事項)クラスの利用方法

インスタンスは、変数へ代入することで利用できます
インスタンスを代入した変数により、属性とメソッドを使うことができます

クラスとインスタンスのイメージ



属性の参照方法

クラスに所属する変数

クラスを代入した変数名**.属性名**

通常の変数の参照と異なり、
クラスを代入した変数名と.（ドット）が必要です

メソッドの使用方法

クラスを代入した変数名**.メソッド名()**

通常の関数の使用方法と異なり、
クラスを代入した変数名と.（ドット）が必要です

クラスに所属する関数

インスタンスの生成方法

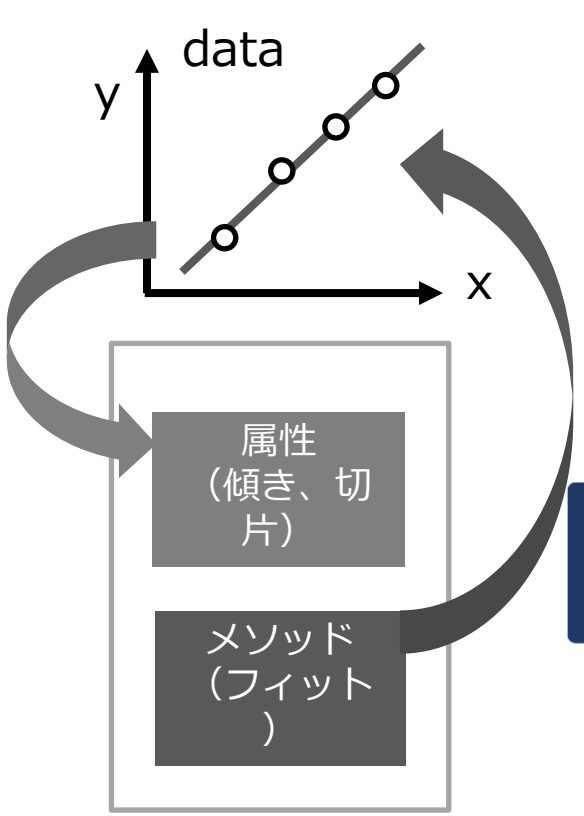
変数名 **= クラス名()**

関数を使用するときのようにかっこは必須です
かっこ内に値を入れる必要がある場合もあります

(発展事項) 機械学習におけるクラスの利用例

線形回帰のクラスでは傾き・切片(属性)を求めるメソッド(Fit)を活用します

（例）線形回帰モデル「y = ax + b」の例 (a:傾き、b:切片)



fit**メソッド**で
dataにフィット

コード例

```
In  [1]: from sklearn.linear_model import LinearRegression

         x_data = [[28], [30], [32], [34]]
         y_data = [10, 80, 100, 120]

         model = LinearRegression()

         model.fit(x_data, y_data)

         print('傾き : ', int(model.coef_[0]))
         print('切片 : ', int(model.intercept_))

Out [1]: 傾き :  17
         切片 :  -464
```

クラスの
インスタンス生成

傾き、切片
（**属性**）の参照

※coefficient（係数）、intercept（切片）

[発展事項]

- 今日の講義では扱わなかったが、今後の講義で出てくる文法事項
- 無名関数 (lambda式)
- map関数
- リスト内包表記
- 条件式 (三項演算子)
- イテレータ
- ※初学者は今日の内容を理解・定着することを優先してください

今回の目的と目標

- 目的
  - データサイエンスとPythonの関係を理解し、Pythonを学ぶ意義を把握する。
  - GCIを受講する上で必要なGoogle Colaboratoryの操作を習得する。
  - Pythonの基本文法を理解し、今後の講義内容に対応できるようにする。

- 目標
  - Google Colaboratoryの基本操作ができるようになる。
  - 演算子、変数、コレクションを使って簡単な計算処理を記述できるようになる。
  - 条件分岐や繰り返し処理を使って、効率的なプログラムを作成できるようになる。
  - 関数を定義して、特定の処理をまとめられるようになる。

不明点は、slackで積極的に質問しましょう
ＯＨでＴＡに聞いてみてもよいでしょう

今週の宿題

- 宿題提出方法の解説

- 今回の宿題は、以下のノートブックとなります
  - GCI2024W_第2回_Python基礎_宿題.ipynb

- Omnicampusの使い方
  - https://curved-rambutan-a71.notion.site/Omnicampus-b3315b99634c46c6b4a4d8e21a1e985e