

Python プログラミング基礎 教材 4

九州大学 数理・データサイエンス教育研究センター

全体の構成

- 教材 1
 - インTRODクシヨン～プログラムってなんだ？
 - プログラミング言語 Python ～無料で遊べるのに凄い
 - 【準備】Gogole Colaboratoryへ行こう！
 - 文字を表示してみよう ～ 記念すべき最初のプログラム
- 教材 2
 - 計算させてみよう ～ たす, ひく, かける, わる
 - 変数 ～ 電卓を越える！
 - If文 ～ ますますプログラムっぽく
 - 便利なコメントアウト ～ ちょっと休憩
 - 数当てゲームを作ってみよう ～ ゲームの基本中の基本
- 教材 3
 - For文 ～ 人間には面倒な繰り返しを簡単に
 - 【寄り道】コンピュータの計算精度 ～ 意外と正確じゃない
 - 再びゲームを作ってみよう ～ 工夫次第でどんどん楽しく
- 教材 4
 - List ～ 複数の変数を固めたもの
 - 関数とライブラリ ～ ひみつ道具で遊ぼう
- 付録
 - if文のちょっと進んだ使い方
 - AIを用いたプログラミング
 - Google Driveとの連携

list

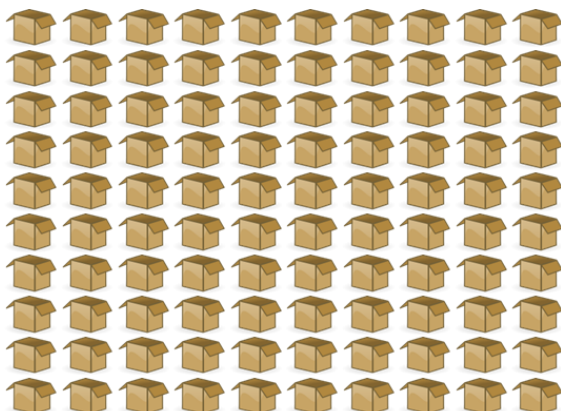
複数の変数を固めたもの

複数の人の身長データを扱おうとすると...

- 10人だと、10変数を準備して、それぞれ別の名前を付ける必要

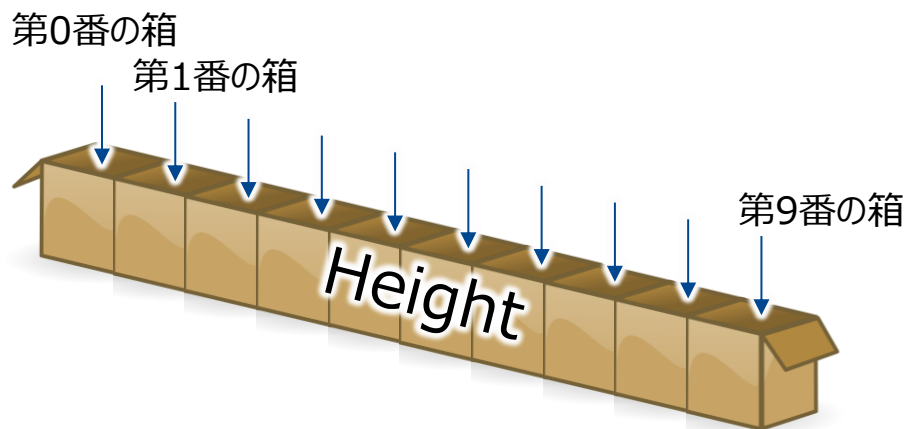


- 100人, 1000人だと, 名前を付けるだけで大変なことに



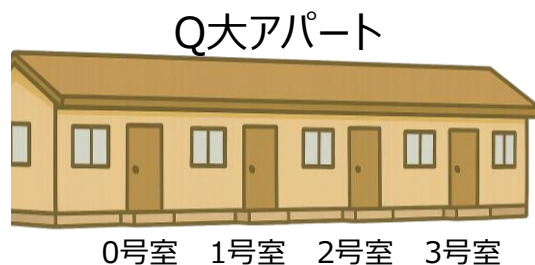
複数の箱を1つにまとめたら？

- 変数(名)の準備や管理が楽になるかも!?



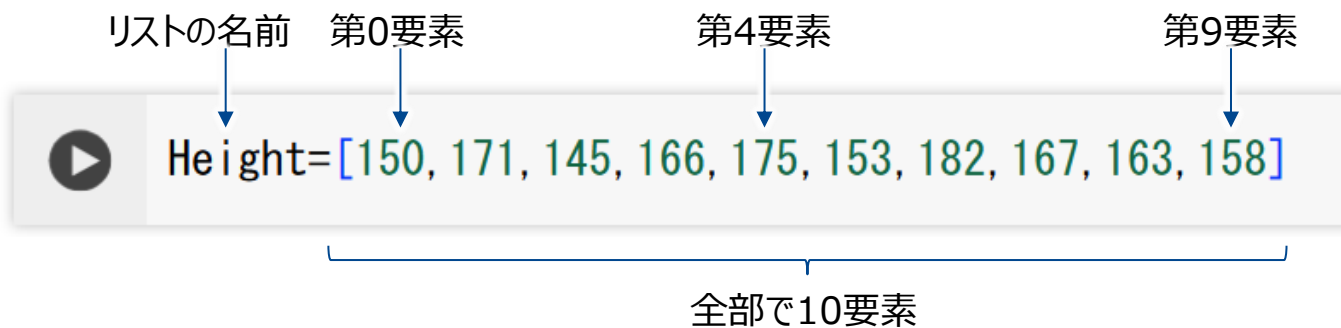
※なんで0から始めるかは、pythonのルールなのです。
range(10)と同じで、0から9.

- 「アパート名は一つで、各部屋は『〇号室』と扱う」と似てる？

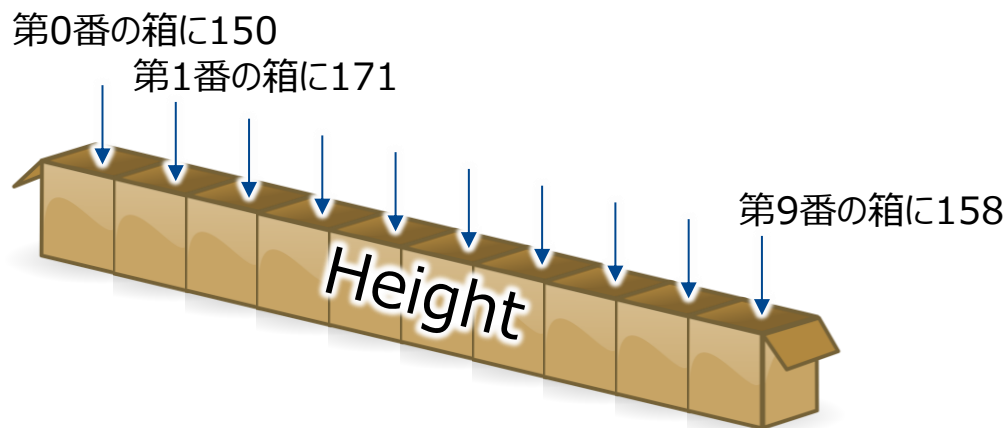


変数をまとめたもの = list (リスト)

- 数字のリスト：[]の中に数字をコンマ区切りで記載



- イメージ



Printで中身を呼び出してみる

```
Height=[150, 171, 145, 166, 175, 153, 182, 167, 163, 158]
```

```
print(Height)
```

Heightをprint

```
[150, 171, 145, 166, 175, 153, 182, 167, 163, 158]
```

全要素が表示された

```
print(Height[3])
```

Heightの第3要素(0,1,2,3なので4番め)をprint

```
166
```

第3要素である166が表示された

```
print(Height[0]+Height[1])
```

Heightの第0要素と第1要素を合計してprint

```
321
```

150と171の合計である321が表示された

for文を使ってリストの全要素を表示させる



```
Height=[150, 171, 145, 166, 175, 153, 182, 167, 163, 158]
```

```
for i in range(10):  
    print(Height[i])
```

iを0から9まで変える

第i要素を表示



```
150  
171  
145  
166  
175  
153  
182  
167  
163  
158
```

第0要素(1番め)である150が表示された

第4要素(5番め)である175が表示された

第9要素(10番め)である158が表示された

リストの中身を書き換える

▶ Height=[150, 171, 145, 166, 175, 153, 182, 167, 163, 158]

Height[1]=173

第1要素（2番め）の中身を173にしてみる

```
for i in range(10):  
    print(Height[i])
```



150
173
145
166
175
153
182
167
163
158

（元々の値である171ではなく）173が表示された

現時点では参考：後述する「関数」と共に使うと、 リストのありがたさがよくわかる！

```

▶ Height=[150, 171, 145, 166, 175, 153, 182, 167, 163, 158]

count = len(Height) # 要素数を教えてくれる関数 len
total = sum(Height) # 合計値を教えてくれる関数 sum
avg = total / count # 上の二つがあれば平均も出せる
maximum = max(Height) # 最大値を教えてくれる関数 max
minimum = min(Height) # 最小値を教えてくれる関数 min

print(f"要素数: {count}")
print(f"合計: {total}")
print(f"平均: {avg}")
print(f"最大: {maximum}")
print(f"最小: {minimum}")

```

```

⇒ 要素数: 10
合計: 1630
平均: 163.0
最大: 182
最小: 145

```

リスト全体(Height)を
「一括して」「関数」に
渡すだけで、
合計したり最大値を
見つけたりしてくれる

便利そう!
要素が10個よりずっと多くても、
全く同じプログラムで
合計とか出せるし!



数字以外でもリストにできる！

- 例 1 : 文字列のリスト

```
▶ Names=["山田", "山本", "山下", "山口"]  
print (Names[1])
```

```
⇒ 山本
```

Printの時と同様、文字列は" "で囲む

- 例 2 : 文字列と数字が混在するリスト

```
▶ BodyData=["山田", 172, 62]  
print (BodyData[2])
```

```
⇒ 62
```

文字列は" "で囲むが、数値は不要

ここまでできるなら、
複数の人のBodyDataを
入れたリストが欲しいなあ…

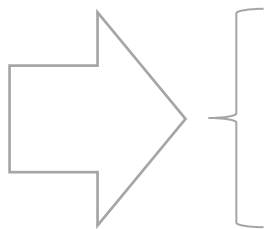


リストの話, どこかで聞いたかも…

- すばらしい. 実はここでこっそり登場

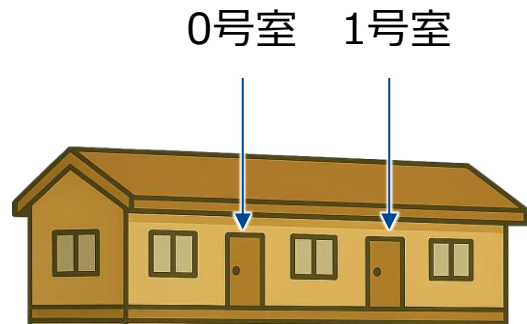
range(100)以外の繰り返しの指定法

- 範囲指定タイプ
 - `range(100)` → 0, 1, 2, ..., 98, 99
 - `range(50, 100)` → 50, 51, ..., 98, 99 #開始番号指定
 - `range(0, 100, 5)` → 0, 5, 10, 15, ..., 90, 95 #開始と刻み幅
 - `range(10, 0, -1)` → 10, 9, 8, ..., 2, 1 #負の刻み幅による逆順
- リストタイプ (`[]`内に書かれたものが書かれた順番通りに)
 - `[1, 2, 3, 5, 7, 11]` → 1, 2, 3, 5, 7, 11
 - `[5, 1, 2, -10, 6]` → 5, 1, 2, -10, 6
 - `["みかん", "りんご", "バナナ"]` → "みかん", "りんご", "バナナ"
 - `["みかん", 5, "りんご", 3, "バナナ", 1]` → "みかん", 5, "りんご", 3, "バナナ", 1
- なので, `range(5)`と`[0,1,2,3,4]`は同じ
 - 前者のほうが書くのは楽ですね

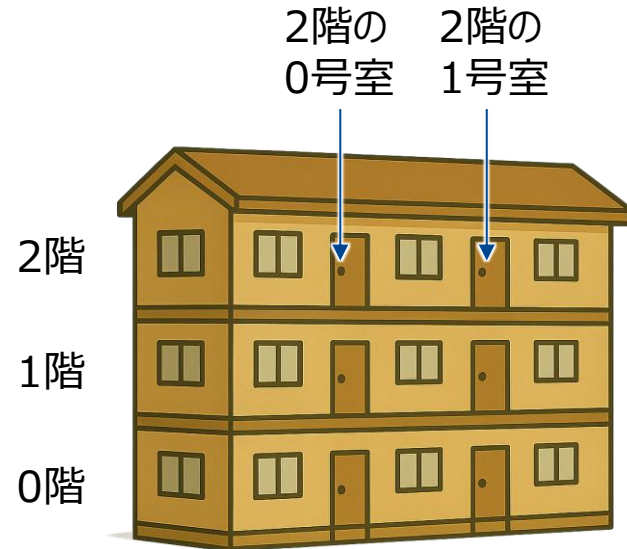


リストのリスト (1/3)

- アパートを再び考える



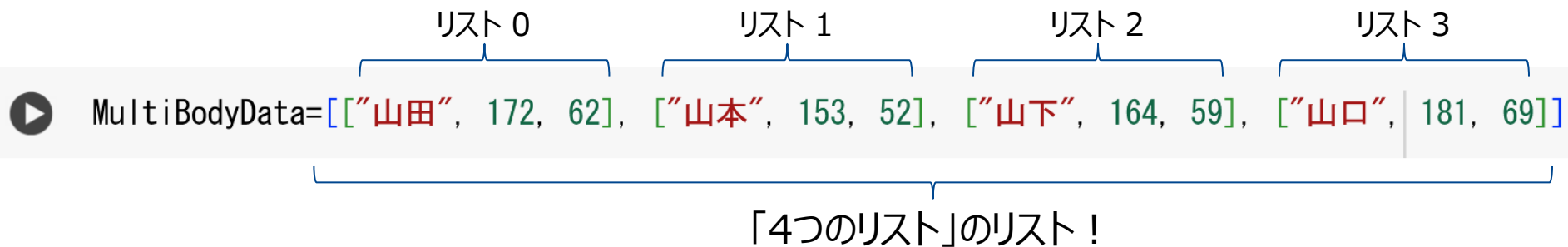
2要素を含むリスト



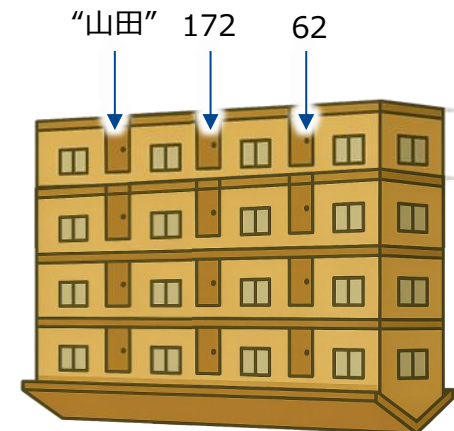
「2要素を含むリスト」を
3つ含むリスト
(リストのリスト！)

リストのリスト (2/3)

- 例：入れたリスト複数の人のBodyDataを入れたリスト



- あえて改行を入れると、「(変数の) アパート感」が出る



リストのリスト (3/3)

```
MultiBodyData=[["山田", 172, 62], ["山本", 153, 52], ["山下", 164, 59], ["山口", 181, 69]]
```

- 第1要素をprintで表示

```
print(MultiBodyData[1])
```

```
['山本', 153, 52]
```

第1要素(2番め)の「リスト」が出力

- 第1要素の第2要素をprintで表示

```
print(MultiBodyData[1][2])
```

```
52
```

「第1要素(2番め)の『リスト』」の第2要素(3番め)が出力

練習：リストのリスト (あまり面白くないですが…)

▶ MultiBodyData=[["山田", 172, 62], ["山本", 153, 52], ["山下", 164, 59], ["山口", 181, 69]]

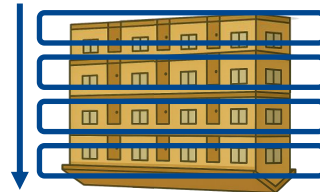
```
{ for i in range(4):
    print(f"{i} 番めの人のデータ: {MultiBodyData[i]}")
```

```
{ for i in range(3):
    print(f"0番目の人の{i}番めのデータ: {MultiBodyData[0][i]}")
```

```
{ for i in range(4):
    print(f"{i} 番めの人の0番目のデータ: {MultiBodyData[i][0]}")
```

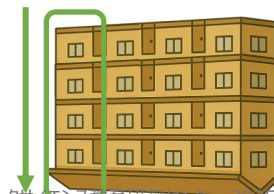
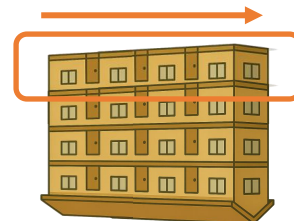
⇒

```
{ 0番めの人のデータ: ['山田', 172, 62]
  1番めの人のデータ: ['山本', 153, 52]
  2番めの人のデータ: ['山下', 164, 59]
  3番めの人のデータ: ['山口', 181, 69]
```



```
{ 0番目の人の0番めのデータ: 山田
  0番目の人の1番めのデータ: 172
  0番目の人の2番めのデータ: 62
```

```
{ 0番めの人の0番目のデータ: 山田
  1番めの人の0番目のデータ: 山本
  2番めの人の0番目のデータ: 山下
  3番めの人の0番目のデータ: 山口
```



参考：appendという技で 徐々にリストを大きくしていくこともできる

▶ `Height=[]`
`print(Height)` ← 空っぽのリスト

⇒ `[]` ← Heightは空っぽであることがわかる

▶ `Height.append(150)`
`print(Height)` ← 数字150を追加

⇒ `[150]` ← Heightは150という要素1つだけのリスト

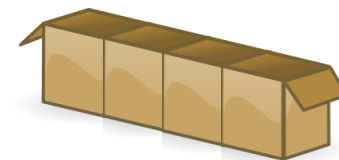
▶ `Height.append(171)`
`print(Height)` ← さらに数字171を追加

⇒ `[150, 171]` ← Heightは150と171という要素2つを含むリスト

参考：リストとその親戚たち (1/2)

● リスト: ex. $L = [1, 4, 5]$

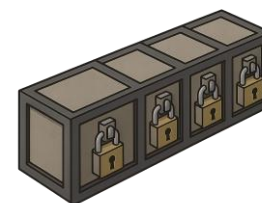
- 個別の値を参照できる…`print(L[0])`とすれば1と表示
- 順序大事… $[1, 4, 5]$ と $[4, 1, 5]$ は違う
- 後から要素に別の値を代入可能…`L[0]=2`とすれば $[2, 4, 5]$ に
- 後から要素を追加(`append`)も可能 …`L.append(2)`とすれば $[1, 4, 5, 2]$ に
- 色々なタイプの要素の混在が可能… $L = [1, 4, 5, \text{"apple"}]$ もOK



自由度が高い

● タプル: ex. $G = (1, 4, 5)$

- 個別の値を参照できる…`print(G[0])`とすれば1と表示
- 順序大事… $(1, 4, 5)$ と $(4, 1, 5)$ は違う
- 後から要素に別の値を代入できない…`G[0]=2`とするとエラーが
- 後から要素を追加(`append`)はできない …`G.append(2)`とするとエラーが
- 色々なタイプの要素の混在が可能… $G(1, 4, 5, \text{"apple"})$ もOK



一度入れたら
変えられない

参考：リストとその親戚たち (2/2)

- 範囲: ex. $R1 = \text{range}(10)$ や $R2 = \text{range}(5, 20)$
 - 個別の値を参照できる…`print(R1[0])`とすれば0と表示
 - 要素は連続順に並んでいる… $R1$ なら0,1,2…9. $R2$ なら5,6,…,19
 - 後から要素に別の値を代入できない…`R1[0]=2`とするとエラーが
 - 後から要素を追加(`append`)はできない…`R1.append(10)`とするとエラーが
 - 色々なタイプの要素の混在できない… 入るのは整数だけ

一定間隔も可能. 例えば
 $R3 = (0, 11, 3)$ とすると,
 3間隔で0,3,6,9

他にもいろいろ

- 集合: ex. $S = \{1, 4, 5\}$.
 - 個別値参照NG. 順序無関係. 代入NG, 追加OK (`add`を利用), 混在OK



- ライブラリ (後述) `array`で使えるarray (配列)
- ライブラリ `numpy`で使える `array` (`ndarray`とも. 先述の「リストのリスト」のようなことも可能)

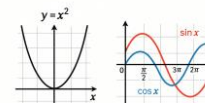
関数とライブラリ

ひみつ道具で遊ぼう

関数!?

- 二次関数とか三角関数とか，昔数学で習ったアレ？

- 数学が苦手だった人は，辛い思い出が…!?



- もっとゆるく考えれば，**関数＝「何かを入れたら，それに応じて何かしてくれる」モノ**

- 確かに，二次関数 x^2 も「 x に2を入れたら」「4を返してくれる」ヤツだ

- 日常の色々なモノも「関数」として見てみると…

- 鍋 → 「野菜」と「熱」を入れたら，煮てくれる
- 車 → 「アクセルを踏む力」を入れたら，動いてくれる
- 仕事する人 → 「給料」を入れたら，〇〇の仕事をしてくれる
- サイコロ → 「転がす力」を入れたら，数字の1から6のどれかを示してくれる
- 自動販売機 → 「お金」（と「ボタン選択」）を入れたら，商品を出してくれる
- 植物 → 「水や日光や二酸化炭素」を入れたら，花を咲かせてくれる

少しは関数が
フレンドリーに



関数 in python

- これまでのプログラムの中で、 **皆さん既に関数を使っています**
 - `print("hello")` → 関数`print`は、"hello"を入れたら、それを画面表示してくれる
 - `input("please")` → 関数`input`は、"please"を入れたら、それを画面表示しつつ、ユーザーに入力を促すボックスを表示し、入力されたら、それを文字列として返してくれる
 - `random.randint(1, 6)` → 関数`random.randint`は、1と6を入れたら、1から6までのどれか一つをランダムに返してくれる
- Pythonには、様々な関数が大量に準備されていて、すぐに使える
 - 自分でプログラムしなくてもいいので、めちゃめちゃ便利！
 - 中にはめちゃくちゃ高機能なものも
 - それも無料で使えることがほとんど！ なんて素敵な（オープンな）世の中…
- 自分で関数を作ることも可能！（後述）

「標準的な関数」の例

- `print()` : 画面に表示
 - `print("hello")` → 画面にhelloと表示される
- `sum()` : 合計
 - `sum([2,5,-1])` → 6を返す
- `abs()` : 絶対値
 - `abs(-1)` → 1を返す
- `max(), min()` : 最大値, 最小値
 - `max([2,5,-1])` → 5を返す
- `pow()` : べき乗
 - `pow(2, 3)` → 2^3 を返す
- `int()` : 文字列を整数値に変換する
 - `int("50")` → 整数50を返す
 - 注意: "50"は"hello"と同様の文字列. "5"と"0"という「文字」がならんだだけ. なので"50"+"60"は"5060"という文字列になる(110)にはならない. 一方, `int("50")+int("60")`とすれば, それぞれ50, 60という数になるので110になる.

練習：標準的な関数を使ってみる

- 以下のプログラムを実際に動かしてみよう



```
MyList = [-3, 2, 5, 6]
```

```
print(sum(MyList))
```

```
print(max(MyList))
```

```
print(abs(MyList[0]))
```

実はリストを変数に入れることもできる

リスト内の要素を合計して表示

リスト内の要素の最大値表示

リスト内の最初の要素を絶対値表示



```
10
```

```
6
```

```
3
```

使える関数をライブラリ※でパワーアップ！

- 以前 “import random” と書けば、乱数（サイコロ）が使えるのを覚えてますか？
 - このrandomは「ライブラリと呼ばれる道具箱」の一つ！
 - random内には、乱数関係の様々な道具（関数）が



- 実はrandom以外にも、膨大な数のライブラリがpythonにはあり、**多くは無料**で使える！
 - 使わない手はない！



よく使うライブラリ (1/2)



名称	説明	用途の例
math	数学の基本的な計算	平方根、三角関数など
random	乱数の生成	くじ引き、シャッフルなど
datetime	日付や時刻の操作	今日の日付、経過時間
time	時間の測定・遅延処理	実行時間の測定、待機
os	ファイル・フォルダ操作	ファイル一覧、パス結合
sys	システム情報取得	終了処理、コマンド引数
matplotlib.pyplot	グラフ描画用	折れ線、棒、円グラフなど
statistics	基本統計計算	平均、分散、中央値
csv	CSV形式の読み書き	表形式データの保存・読み込み
json	JSON形式の処理	設定ファイルやAPIとの連携

よく使うライブラリ (2/2)



名称	説明	用途の例
numpy	配列計算と数値処理の定番	大量データ処理、高速な数学演算
pandas	表形式データの操作	データの読み書き、加工、集計
scikit-learn	機械学習モデルの学習・評価	回帰、分類、クラスタリングなど
pytorch, tensorflow	深層学習のフレームワーク	ニューラルネットの学習と推論
seaborn	見やすく美しい統計グラフ	相関関係、ヒートマップ、 箱ひげ図

- このスライドのライブラリは、特にAI・機械学習でよく使う
 - AI・機械学習系のプログラムを自分でゼロから作らなくても、これらライブラリの関数を使えばOK！
 - この効率化により、AI・機械学習系の技術が世界中で爆発的に進化！



余談：巨人の肩に乗る

- 「先人の積み重ね」に基づいて何かを発見すること
 - 研究分野でよく言われるフレーズ
 - 「過去の膨大な研究 = 巨人」の肩に乗ることで、新しい発見が可能になる
 - 逆に言えば、「巨人の肩に乗らなければ、遠くを見通せず、新しい発見をすることは難しい」
- Python のライブラリを使うことも同じ！
 - 巨人（先人エンジニア）の努力によるライブラリを使うことで、短時間で高度なプログラムを実現できる
 - 積極的に使いましょう！
 - そして専門家になった暁には、後輩のためにライブラリを残そう！



ライブラリ “math” に準備された関数の例： “import math”により以下のような算術関数ができる

- 頭に “math.”を付けるのを忘れないように
- `sqrt()` : 平方根
 - `math.sqrt(2)` → 1.4142135623730951を返す
- `sin()`, `cos()`, `tan()` : 三角関数
 - `math.cos(0)` → 1.0を返す
- `log()`, `log2`, `log10()` : 対数関数
 - `math.log10(1000)` → 3.0を返す
- `floor()`, `ceil()` : 切り捨て, 切り上げ
 - `math.floor(1.3)` → 1を返す
 - `math.ceil(1.3)` → 2を返す
- `gcd()`, `lcm()` : 最大公約数, 最小公倍数
 - `math.gcd(4,16,8,10)` → 2を返す
 - `math.lcm(4,16,8,10)` → 80を返す
- `comb`, `perm` : 組み合わせ, 順列
 - `math.comb(4,2)` → 6を返す (4つから2つ選ぶ組み合わせの数)

練習：ライブラリ math に準備された関数を使ってみる

- 以下のプログラムを実際に動かしてみよう

```
import math
print(math.sqrt(2))
print(math.cos(0))
print(math.floor(1.3))
print(math.ceil(1.3))
print(math.gcd(4, 16, 8, 10))
print(math.lcm(4, 16, 8, 10))
print(math.comb(4, 2))
print(math.comb(1000, 50))
```

ライブラリを使うときは、これを忘れないように！

ライブラリの各関数の前には"math."が必要

1000個から50個を選ぶ組み合わせの数は…

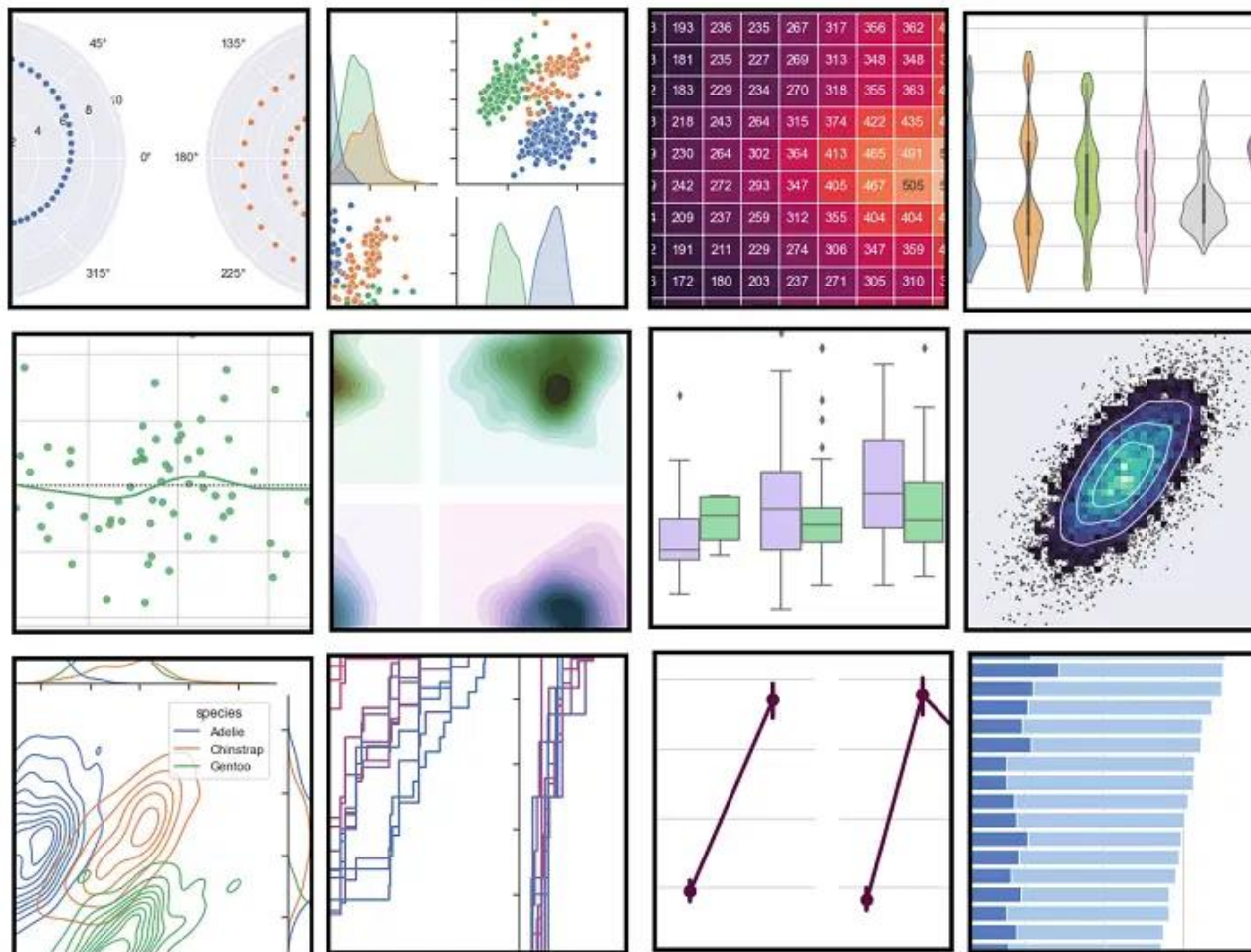
```
1. 4142135623730951
1.0
1
2
2
80
6
```

こんな数！

9460461017585217846063722277728044918729694001668654064793569321343252697198115263280

可視化関係のライブラリ “matplotlib” (1/3)

- 様々なデータの可視化が可能！



可視化関係のライブラリ “matplotlib” (2/3)

折れ線グラフを書いてみる

▶ `import matplotlib.pyplot`

matplotlib.pyplot を読み込む

データ (x軸とy軸)

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

`y = [10, 20, 15, 25, 30]`

適当に作ったデータ(例えば, x=1のときy=10)

折れ線グラフの描画

`matplotlib.pyplot.plot(x, y)`

「書くのはx,yを使った折れ線グラフです」

タイトルとラベル

`matplotlib.pyplot.title("Sample Line Graph")`

`matplotlib.pyplot.xlabel("X Axis")`

`matplotlib.pyplot.ylabel("Y Axis")`

「タイトル (軸ラベル) は. . . です」

表示

`matplotlib.pyplot.show()`

「では, 実際に書いて表示してください」

- `matplotlib.pyplot.XXX`は, すべて`matplotlib.pyplot`に入っている関数
- 上のプログラムを実行すると…

可視化関係のライブラリ “matplotlib” (3/3)

折れ線グラフを書いてみる

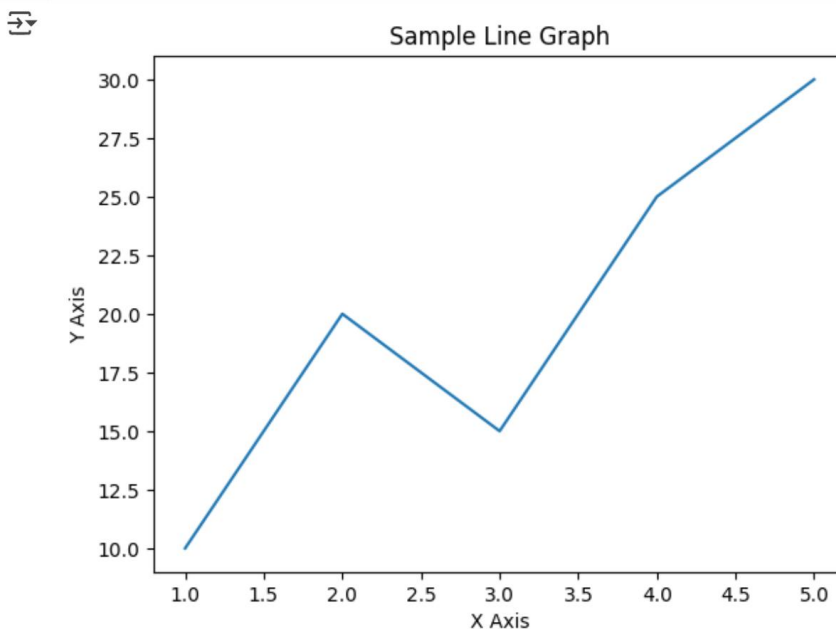
```
▶ import matplotlib.pyplot

# データ (x軸とy軸)
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

# 折れ線グラフの描画
matplotlib.pyplot.plot(x, y)

# タイトルとラベル
matplotlib.pyplot.title("Sample Line Graph")
matplotlib.pyplot.xlabel("X Axis")
matplotlib.pyplot.ylabel("Y Axis")

# 表示
matplotlib.pyplot.show()
```



できた！

参考：ライブラリ名で長くなった関数名を 何度も書くのは面倒…

- as を使うと省略形を定義できます

「matplotlib.pyplotをpltと書くよ」と
定義(pltの代わりにappleでもpotatoでもOK)

```
▶ import matplotlib.pyplot

# データ (x軸とy軸)
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

# 折れ線グラフの描画
matplotlib.pyplot.plot(x, y)

# タイトルとラベル
matplotlib.pyplot.title("Sample Line Graph")
matplotlib.pyplot.xlabel("X Axis")
matplotlib.pyplot.ylabel("Y Axis")

# 表示
matplotlib.pyplot.show()
```

長い名前を何度も
書くのは面倒だな



```
▶ import matplotlib.pyplot as plt

# データ (x軸とy軸)
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

# 折れ線グラフの描画
plt.plot(x, y)

# タイトルとラベル
plt.title("Sample Line Graph")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

# 表示
plt.show()
```

短くできた！



自分オリジナルの関数を作ってみる！

例：「入力された数値を2倍して返す」関数

関数
doubleを
定義
(define)



```
def double(x):
    y = 2 * x
    return y
```

xを入力とするdoubleという名前の関数を定義するよ

入力 x を 2倍したものを変数yに入れるよ

yに入っている値を返すよ

上のセルを
実行すれば、
どこでも関数
doubleが
使えるように



```
double(30)
```



```
60
```



```
double(15)+double(10)
```



```
50
```



```
double(double(10))
```



```
40
```

なぜ40になるか、わかるかな？



```
bai = double(100)
print(bai)
```



```
200
```



```
num=1
for i in range(10):
    num = double(num)
print(num)
```



```
1024
```

練習：BMIを計算する関数BMIを定義して使ってみよう！



```
def BMI (w, h) :  
    ans=w/ (h*h)  
    return ans
```

BMI定義部

```
print (BMI (60, 1.5))
```

wに60, hに1.5を指定してBMI計算, ansを表示

```
print (BMI (40, 1.5))
```

wに40, hに1.5を指定してBMI計算, ansを表示



```
26.666666666666668  
17.777777777777778
```

練習：関数BMIをもう少し凝ってみよう

```

▶ def BMI (w, h) :
    ans=w/(h*h)
    if ans>=25:
        print("BMIが適正範囲を超えています")
    elif ans<=18.5:
        print("BMIが適正範囲を下回っています")
    else:
        print("適正範囲です")
    return ans

```

もう少し凝った
BMI定義部

```

weight=60
height=1.5
print(BMI (weight, height))

```



```

BMIが適正範囲を超えています
26.666666666666668

```

アドバイスも表示できるようになった

参考：「定義の中の変数」と「定義の外の変数」： 変数のスコープ(1/3)



```
def BMI (w, h) :
    ans=w/ (h*h)
    return ans
```

BMI定義部
(先ほどの練習と同じ)

```
weight=60
```

```
height=1.5
```

```
print (BMI (weight, height))
```

変数weightとheightの中身をBMIに渡している



```
26.666666666666668
```

wとhではなく
weightとheightで
値を渡している点に注目！

どういうコト？



実は、wとhは「定義部の中
だけ」で有効な変数！

参考：「定義の中の変数」と「定義の外の変数」： 変数のスコープ(2/3)

▶ `def BMI(w, h):`
 `ans=w/(h*h)`
 `return ans` } BMI定義部
 (先ほどと同じ)

```
weight=60
height=1.5
print(BMI(weight, height))
print(w)
```

wの中身を印刷しようとするエラーが！

⇔ 26. 66666666666666668

```
NameError                                Traceback
<ipython-input-5-cd563db4c9c6> in <cell line: 0>()
      6 height=1.5
      7 print(BMI(weight, height))
----> 8 print(w)
```

NameError: name 'w' is not defined

「wなんて知らない！」というエラー

参考：「定義の中の変数」と「定義の外の変数」： 変数のスコープ(3/3)

- なぜエラーが？ → 変数には見える「範囲（スコープ）」がある

「BMIの定義の外」の世界

weight=60

変数weightの
中身をこの穴に

height=1.5

変数heightの
中身をこの穴に



外の世界からは
wやhの存在すらわからない
→だから使おうとするとエラーに

print

この穴から
出てきたものを
表示

外の世界からは見えない
「BMIの定義の中」の世界

ここから入ってくるものを
変数wに代入

ここから入ってくるものを
変数hに代入

$$\text{ans} = w / (h * h)$$

ここから変数ansの
中身を戻す(return)



参考：どんな関数が教えてくれる help 関数(1/2)

説明

▶ `help(print)`

⇒ Help on built-in function print in module builtins:

```
print(*args, sep=' ', end='¶n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
        string inserted between values, default a space.
    end
        string appended after the last value, default a newline.
    file
        a file-like object (stream); defaults to the current sys.stdout.
    flush
        whether to forcibly flush the stream.
```

説明

▶ `help(random.randint)`

⇒ Help on method randint in module random:

```
randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

参考：どんな関数が教えてくれる help 関数(2/2)

自作の関数の場合

- 関数の説明をあらかじめ'''xxxx'''で囲って記載
 - 「3重」引用符である点に注意

```

▶ def BMI(w, h):
    '''This function calculates BMI'''
    ans=w/(h*h)
    return ans

help(BMI)
⇒ Help on function BMI in module __main__:

   BMI(w, h)
      This function calculates BMI

▶ print(BMI(60, 1.5))
⇒ 26.666666666666668
  
```

help関数で呼び出された場合に表示する説明

help関数で呼び出した

説明内容が表示された

もちろん, BMI計算もそのまま可能