Pythonプログラミング～後半～

会津大学

講習会内容

- Pythonプログラミング
  - Pythonの紹介
  - 演算子，標準入出力，データ型，変数，プログラム実行
  - 条件分岐
  - 繰り返し処理
  - リスト
  - ライブラリの利用
    - 乱数の生成
    - グラフの作成
  - 関数

リスト構造

- 数値や文字列などを並べて格納できるデータ型
  - ロッカーや棚に値が格納されているイメージ
  - それぞれの要素をコンマ(,)で区切って，
    全体を角カッコ[]で囲む
  - 例えば，12, 24, 36の3つの数値を格納するリストは
    - [12，24，36]となる

12	24	36

- 様々なデータ型の値を格納したリストを作成できる

例: [123，'abc'，-4.56] # 整数，文字列，小数のリスト

123	'abc'	-4.56

リスト構造

- リスト構造リストの格納されている値を要素とよび，
  インデックスとよばれる数値を指定して各値を取得できる
  - **最初の要素（一番左の要素）のインデックス: 0**
  - **最後の要素（一番右の要素）のインデックス: 要素数-1**

```
>>> x = [1, 2, 3, 'x', 'y', 'z']↵ # 変数xのリストを定義する
>>> x[0]↵ # 最初の要素のインデックスは0となる
1

>>> x[1]↵ # 2番目の要素のインデックスは1となる
2

>>> x[5]↵ # 最後の要素のインデックスは「要素数-1」となる
'z'
```

リスト構造に関する操作

- **len(リスト名)**
  - リストの要素数を求めることができる
- **リスト名.append(値)**
  - カッコの中にある値をリストの最後に追加する
- **リスト名.insert(値1, 値2)**
  - 値2を値1のインデックスの箇所に追加する

- 要素数が0のリスト（空リスト）を作成する場合,

  変数名 = [] と書けばよい

リスト構造に関する操作

```
>>> x = [5, 6, 7, 8, 9, 10]↵ # リストの初期化
>>> len(x)↵ # 要素数
6

>>> x.append(11)↵ # 最後の要素に11を追加する
>>> x↵
[5, 6, 7, 8, 9, 10, 11]

>>> x.insert(0, 4)↵ # インデックス0番に4を追加する
>>> x↵
[4, 5, 6, 7, 8, 9, 10, 11]
```

リスト構造に関する操作

```
>>> a = []↵ # 要素数0の空リストを作成する
>>> len(a)↵ # 要素数
0

>>> a.append(999)↵ # 最後の要素に999を追加する
>>> a↵
[999]

>>> a.insert(0, -111)↵ # インデックス0番に-111を追加する
>>> a↵
[-111, 999]
```

リスト構造と繰り返し処理

- 数値や文字列などを並べて格納しているため，
  繰り返し処理と相性がよく，以下の方法で1つずつ値を
  順番に取り出すことができる

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for i in x:
  print(i) # iの値がx[0]の値，x[1]の値，･･･，x[9]の値となる
```

ここまでのまとめ演習4

1. 要素数0の空リストを作成し，appendを利用して，
   先頭から，1, 3, 5, 7, 9, 11, …, 93, 95, 97, 99を格納する
   リストを作成せよ

2. この作成したリストに対して，合計値，平均値を求める
   プログラムを作成せよ

3. この作成したリストに対して，
   99, 97, 95, 93, …, 11, 9, 7, 5, 3, 1と逆順になるような
   プログラムを作成せよ

ライブラリ

- Pythonにはライブラリとよばれる「プログラムの集まり」が用意されている
  - このライブラリを利用することで，様々な機能を持つプログラムを簡単に作成できるようになる

1. 標準ライブラリ
   - Pythonをインストールした際に付属しているライブラリ

     例: `math`（数学機能），`random`（乱数機能）
2. 外部ライブラリ
   - 新たにダウンロードしてインストールが必要なライブラリ

     例: `matplotlib`（グラフ機能），`pytorch`（深層学習）

ライブラリの使用例

- ライブラリを導入する場合，最初に**import ライブラリ名**と書く
- ライブラリに含まれる機能を使用するには，ドット(**.**)を打ち，機能名を続けることで使用することができる

- 累乗と平方根を求める**math**ライブラリの**pow()**，**sqrt()**の使用例

```
>>> import math↵ # mathライブラリの導入
>>> math.pow(2, 5)↵ # 2の5乗
32.0

>>> math.sqrt(2)↵ # 2の平方根（ルート2）
1.4142135623730951
```

ライブラリの使用例

- **as**を使用することで，ライブラリに別名をつけて使用できる
- **math**ライブラリを**m**という別名をつけた場合の，
  累乗と平方根を求める**pow()**，**sqrt()**の使用例

```
>>> import math as m↵ # mathライブラリをmとして導入
>>> m.pow(2, 5)↵ # 2の5乗
32.0

>>> m.sqrt(2)↵ # 2の平方根（ルート2）
1.4142135623730951
```

randomライブラリ

- Pythonでランダムな処理が必要な場合に使用する
  - ランダムな数字のことを乱数とよぶ
    - サイコロをふるということをプログラムで扱う場合,
      このような処理が必要となる

- **random.randint(a, b)**
  - a ≤ N ≤ b であるようなランダムな整数Nを1つ返す

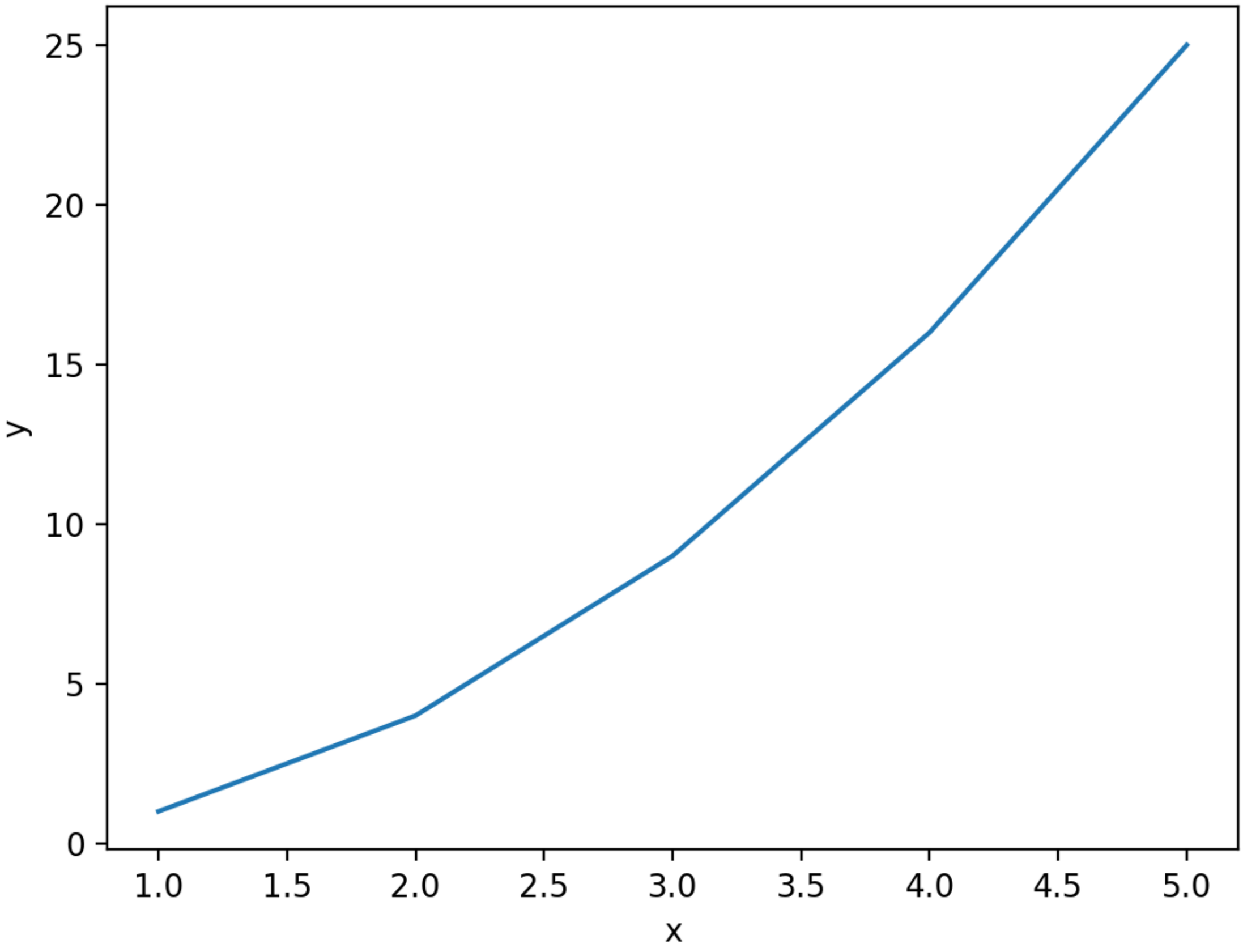
```
import random

# ランダムに1以上6以下の整数を100回出力する
for i in range(100):
    print(random.randint(1, 6))
```

matplotlibライブラリ

- グラフ作成に特化した外部ライブラリ
- リスト構造に格納された値をグラフとして可視化する例
  - インデックスに対応した関係のグラフとなる
  - 様々なグラフを出力することができるが，
    デフォルトの設定では，折れ線グラフとなる
  - **plt.show()**をしないとグラフとして表示されないので注意

```
>>> import matplotlib.pyplot as plt # ライブラリの導入
>>> x = [1, 2, 3, 4, 5]            # x軸の値
>>> y = [1, 4, 9, 16, 25]          # y軸の値
>>> plt.xlabel('x')                # x軸の名前の設定
>>> plt.ylabel('y')                # y軸の名前の設定
>>> plt.plot(x, y)                 # 折れ線グラフの作成
>>> plt.show()                     # 作成したグラフの表示
```


25
20
15
10
5
0
y
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
x

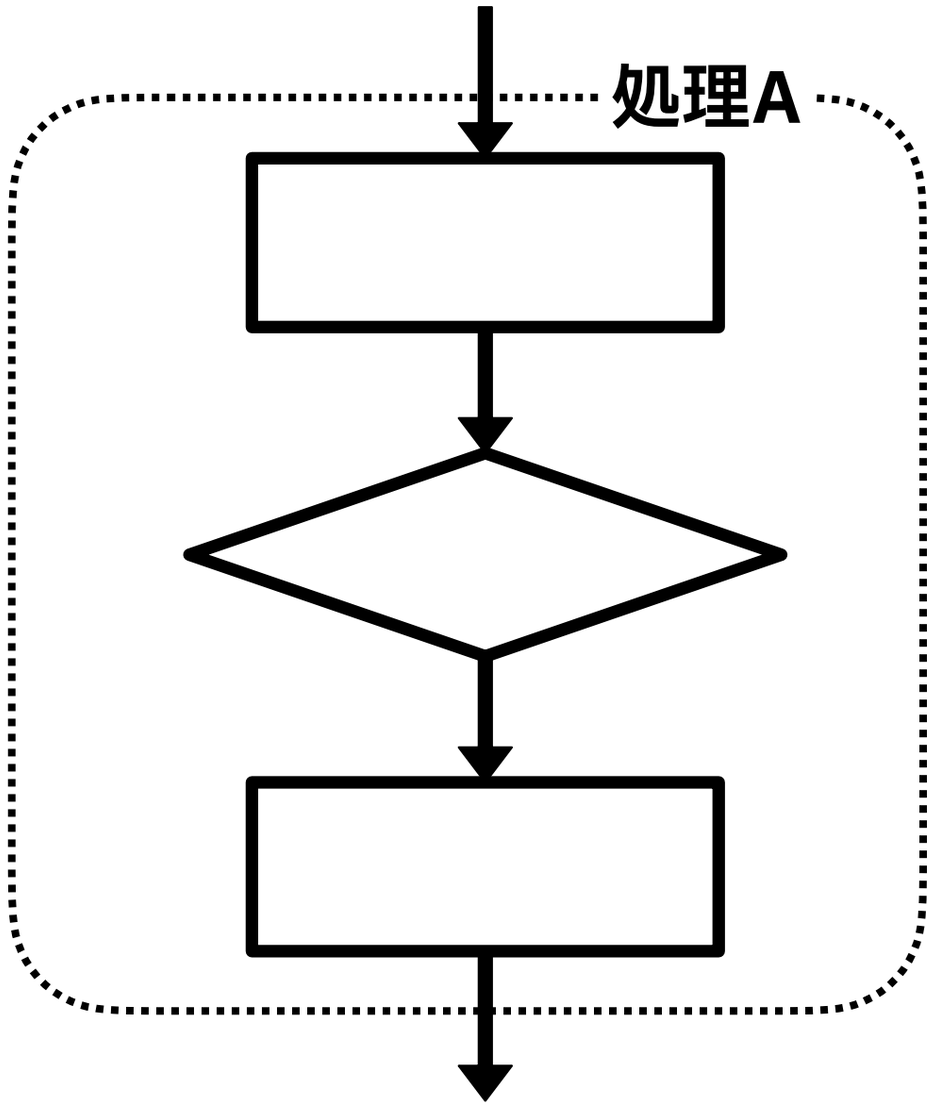
ここまでのまとめ演習5

1. スライドに書かれているプログラムを利用して，
   乱数を利用して，プログラムで10000回サイコロを振ったときに，
   各目がそれぞれ何回出たのかを出力するプログラムを作成せよ

2. じゃんけんにおけるグーを0，チョキを1，パーを2として扱うと，
   プログラムでじゃんけんを行うことができる，
   この説明を利用して簡単なじゃんけんゲームを各自で考え，
   プログラムで実装せよ

関数

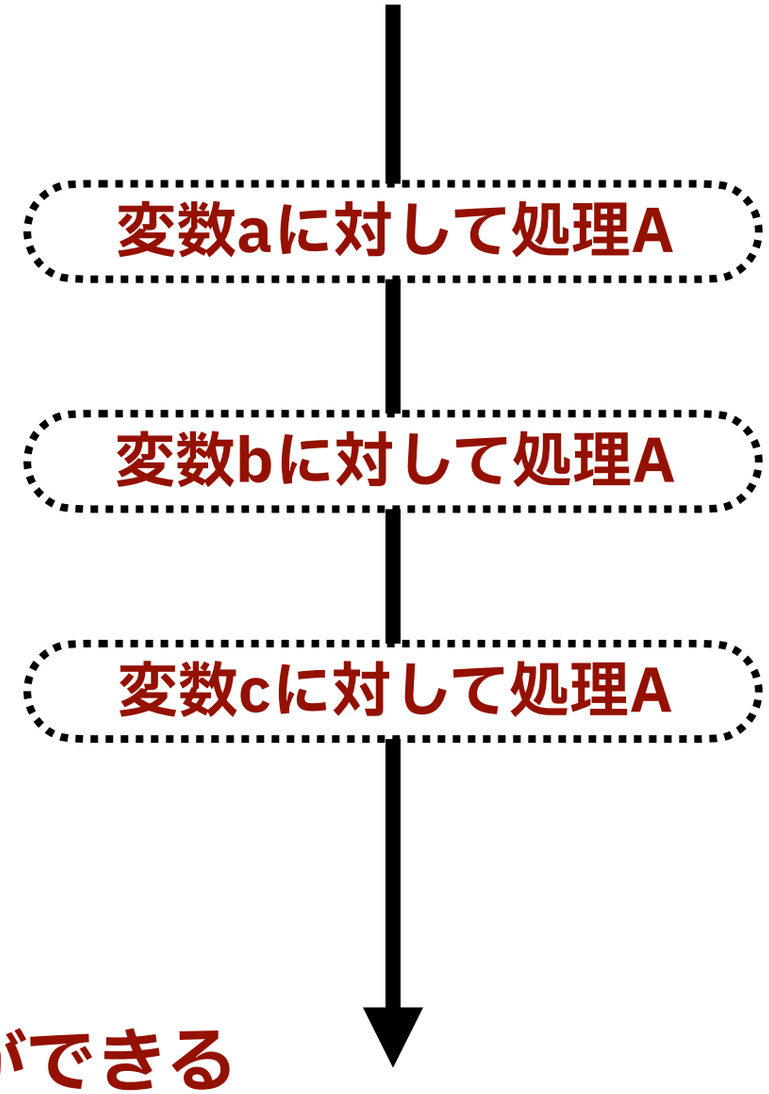
❖ ひとまとまりの処理を行うプログラムに名前を付けたもの



処理Aを関数化

プログラムを
見やすくする

処理Aに
変数を渡すことで，
異なる変数に対して，
同じ処理Aを行うことができる



関数のメリット

- 同じ処理を何回も書く必要が無くなる
  - 1つの処理を定義すれば，その関数を実行することで，同じ処理を行うことができる
  - プログラムが見やすくする

- 同じ処理を別のプログラムで再利用できる
  - 関数を上手く作成することで，別のプログラムでそのまま使用できる

関数の定義

- Pythonでは**def**を使用して関数を定義できる
- 一般的には以下のように関数を定義する

```
def 関数名(引数リスト):
    関数で行う処理
```

- 関数名は変数と同様に名前をつけることができる
- defに続くコードブロックに書かれているプログラムが関数で実行される処理となる

関数の定義

```
def 関数名(引数リスト):
    関数で行う処理
```

- **引数(ひきすう)**: 関数を実行させる条件
  - 関数内部でその引数を使用することができる
  - 値を引数に設定することを『引数を渡す』と呼ぶ
  - 引数がない場合もある
- **戻り値**: 関数を実行した後に，結果として呼び出し元に戻る値
  - 戻り値がない場合もある

関数の定義

- 引数リスト: 引数の個数を規定
  - 引数が複数ある場合，コンマで区切る

- 戻り値がある場合: **return文**を使用して値を返す
  - **return 定数** や **return 式**
  - return文は関数中に多数存在しても問題ない

- 戻り値がない場合，以下の1文を書くこと（ただし省略可能）
  - **return**

関数の使用

- 関数を使用する（呼び出す）場合，**関数名()**と書く
- プログラムの実行が関数の呼び出しに到達すると，その関数の先頭行に移動して，関数の中身のプログラムを実行する
- 関数の末尾まで実行すると，関数を呼び出したところに戻り，続きのプログラムを実行する

引数あり関数の例

❖ **def**を使用して関数を作成する際に，丸カッコ内に
引数リストを書くことで，引数あり関数を作成できる

```
# printHello関数の定義
def printHello(name):
  print('Hello, ' + name)
# printHello関数ここまで

# ここからプログラムがスタート
n = input() # 文字列を入力

printHello(n) # 関数の呼び出し
# nの値がprintHello関数のnameに代入される
```

戻り値あり関数の例

❖ 関数呼び出しによって得られた値をプログラムで使用したい場合，

return文を使用する

```
# getAdd2関数の定義
def getAdd2(num1, num2):
  return num1 + num2 # この結果が返される
# getAdd2関数ここまで

# ここからスタート
a = int(input())
b = int(input())
print(getAdd2(a, b)) # 関数の呼び出し
# aの値がgetAdd2関数のnum1に，bの値がgetAdd2関数のnum2に代入される
```

ここまでのまとめ演習6

1. 自然数numを引数にとり，Hello をnum回出力する関数を作成せよ

   ❖ 関数名: `print_Hello(num)`

2. 身長と体重を引数にとり，BMIの値を返す関数を作成せよ

   ❖ 関数名: `calc_BMI(height, weight)`

   ❖ 引数リストのheightは身長[m]の値，
   weightは体重[kg]の値を示す

   ❖ BMIの値は体重を身長の2乗値で割ることで求められる

   ❖ `float(input())`とすることで小数値を入力できる