4. Pythonの復習

(再掲) プログラムは「文」をならべる

- やりたいことを並べればプログラムが完成
- プログラムは上から下に順に実行される
- 例:
  - ```
    name = input('君の名は? ')
    print('こんにちは', name, 'さん')
    ```

上から順に
1行ずつ実行
    ```

(再掲) プログラムを部品としてまとめる

- **def 関数名(引数リスト):** の下に、まとめたい処理をインデント(字下げ)して書くと「関数」という部品を作れる

- さっきのプログラムを部品にすると？

```
def hello():
    name = input('君の名は? ')
    print('こんにちは', name, 'さん')

hello()
hello()
```

部品は一度作ってしまえば、
何回でも使うことができる！

(再掲) 引数と戻り値を入れてもOK

- 関数に渡したい情報を引数として渡せる
- 関数の結果(戻り値)を **return** 文で返せる
  - 戻り値を決定し、その関数の実行を終了
    (それ以降の関数内の命令は実行されない)

- 例:

```
def name(first, last):
    return first + ' ' + last

print(name('Ken', 'Sakamura'))
print(name('Taro', 'Pico'))
```

2つの仮引数

return の後が
戻り値

実引数を
渡して実行

(再掲) 補足: 戻り値のない関数

- 戻り値のない関数を作ることもできます
  - ```
    def hello():
        name = input('君の名は? ')
        print('こんにちは', name, 'さん')
    ```

- return 文がない場合、戻り値はありません
  - ```
    a = hello()
    君の名は? Enryo
    こんにちは Enryo さん
    ```
  - ```
    print(a)
    ```

hello() の戻り値を入れた変数 a には何も入らない！
    ```

(再掲) 注意: return と print をきちんと区別しよう

- 以下の2つの関数の違いは分かりますか？

```
def hello_01():
  name = input('君の名は? ')
  print('こんにちは', name, 'さん')
```

```
def hello_02():
  name = input('君の名は? ')
  return 'こんにちは' + name + 'さん'
```

- printに値を返す、return に値を表示するという機能はない！
  - どうして以下のような挙動の違いになるのかを説明できるようにしておこう

```
a = hello_01()
君の名は? Enryo
こんにちは Enryo さん
print(a)
```

```
a = hello_02()
君の名は? Enryo
print(a)
こんにちはEnryoさん
```

(再掲) 条件と繰り返し

- プログラムは基本的には上から下に順に実行される
  - ```
    name = input('あなたの名前は:')
    print('こんにちは', name, 'さん')
    ```

上から順に
1行ずつ実行

- if/while/forを使うことで、実行に条件をつけたり、繰り返して実行することができる
  - if … (elif … else …)文：もし～であれば～を実行
  - while文：～が成立する間、～を実行
  - for文：一つ一つの要素について～を実行
    ```

(再掲) if文

- **if** 条件1:
  条件1が成り立つ
  場合の処理
  elif 条件2:
  条件1が成り立たず、
  条件2が成り立つ場合
  の処理
  elif 条件3:
  条件1,2が成り立たず、
  条件3が成り立つ場合
  …
  else:
  いずれの条件も成り
  立たない場合の処理

- 例:

```
temperature = 25.0

if temperature > 25.0:
  print('暑いですね')
elif temperature > 15.0:
  print('快適ですね')
else:
  print('寒いですね')
```

(再掲) while文

- **while** 条件:
  条件が成り立つ間
  実行される処理

- ループの実行中に、処理を中断することもできる
  - break
    - whileの処理を中断し、ループを抜ける
  - continue
    - whileの処理を中断し、条件のチェックからやり直し

- 例: 1～100の合計を計算・表示

```
i = 1
sum = 0

while i <= 100:
    sum += i
    i += 1

print(i)
```

(再掲) for文

- **for** 変数 **in** コレクション:
  各要素に対して実行
  する処理

- whileと同様に break, continue も
  使える

- 例: すべての果物をジュースに

```
fruits = [
  'apple',
  'banana',
  'orange',
]

for v in fruits:
  print(v + ' juice')
```

(再掲) ネスト(入れ子構造)もOK

- if/while/for 文の中に、if/while/for 文を入れることができる

- 例: 対戦相手のすべての組み合わせを表示
  - ```
    team = [
      'carp', 'tigers', 'giants',
      'baystars', 'swallows', 'dragons'
    ]

    for t1 in team:
      for t2 in team:
        if t1 == t2:
          continue  # 同じチーム同士は対戦しない
        print(t1, 'vs', t2)
    ```
    ```

(再掲) 数を列挙するには range を

- 数のリスト [1, 2, 3, … , n] のようなものを作りたい場合は range を使おう
  - **range**(n, m): n以上m未満の整数
  - 「未満」の部分に注意

- 例: 1～100の合計を計算・表示

```
total = 0
for i in range(1, 101):
  total += i

print(total)
```