# 第4回
# 制御構文（ブロックの概念、if文、for文、while文）

## 目次

# プログラムの処理の流れ

- Pythonで記述した命令（処理）は、どのような順番で実行されるのでしょうか？
  - 原則として命令は上から下に順番に実行されます（逐次実行）

処理の順番

```
taijyu = 60
print(taijyu)
shincho = 1.6
bmi = taijyu / shincho / shincho
print(bmi)
```

  - 同じ命令を繰り返し実行したい場合や、条件によって実行する命令を切り替えるにはどうしたらよい？

  - 制御構文を使う



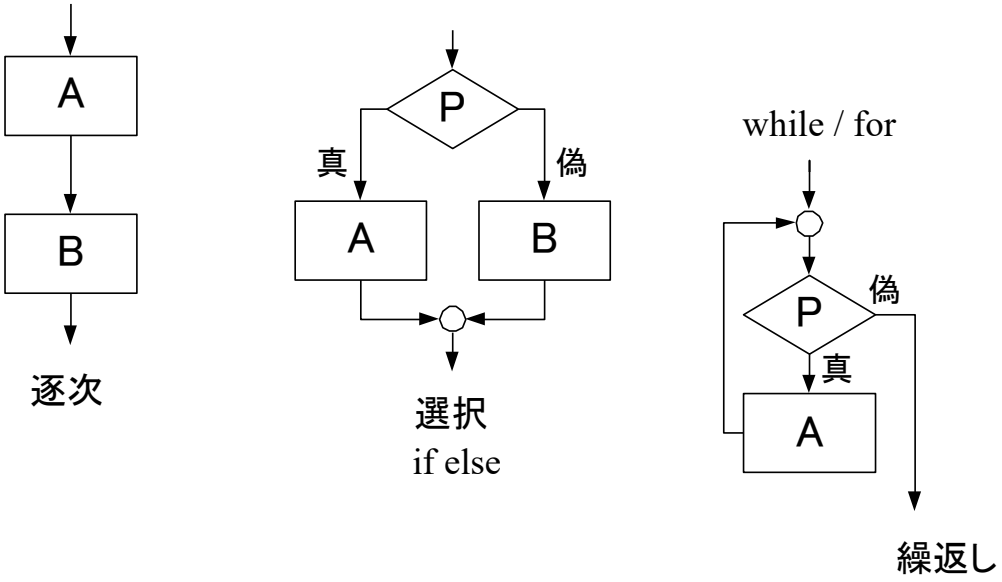
# 構造化プログラミング

- 1960年代・・・「ソフトウェア危機」が叫ばれ始めた

- エズガー・ダイクストラ（E. W. Dijkstra）
  - 主著：「構造化プログラミング」/1968年
  - 構造化定理
    - プログラムの流れを3つの基本制御構造（逐次、選択、繰返し）の組み合わせで表現
    - GOTOレス（やたらめったらジャンプしない）
    - プログラムは、制御の流れに対して、1つの入り口と出口を持つ
  - 逐次(sequence)：
    - 命令A,Bを順番に実行
  - 選択(selection, if-then-else)：
    - 条件Pの真偽により、命令A,Bの実行を選択
  - 繰返し(iteration, do-while)：
    - 条件Pが真である間、命令Aを繰り返し実行

# 制御構文：逐次、選択、繰り返し

- Pythonにおいても3つの制御構文を組み合わせてプログラムの流れ（フロー）を記述する。








# 選択（if文とelse文）

- if文により、条件によって実行する処理を切り替えることができます。
  - 構文

```
if 条件式 :
    条件式が成り立つ場合に実行する処理1
    条件式が成り立つ場合に実行する処理2
    ・・・
else :
    条件式が成り立たない場合に実行する処理1
    条件式が成り立たない場合に実行する処理2
    ・・・
```

成り立つ場合に実行されるブロック

成り立たない場合に実行されるブロック

ブロックは行頭を等しく字下げします（インデント）

  - 条件によって実行したい処理の塊を「ブロック」と呼びます。
  - ブロックはカッコなどで囲んで範囲を表すのではなく、行頭を等しく字下げすることで範囲を表します。
  - 成り立たない場合の処理はelse文のブロックに書きます。成り立たない場合の処理がなければelse文とそのブロックは不要です。

# if文の簡単な例

- if文を使って、BMIが25.0以上の場合に太っていることを表示し、さらにダイエットを促すメッセージを表示します。
  - 字下げ(インデント)を入れるには、Tabキーが便利です。

ifのブロックはこの2行です。

elseのブロックはこの2行です。

```
taijyu = 60
print(taijyu)
shincho = 1.6
bmi = taijyu / shincho / shincho
print(bmi)
if bmi >= 25.0:
    print("あなたは太っています。")
    print("ダイエットを始めましょう。")
else:
    print("あなたは太っていません。")
    print("ダイエットは不要です。")
```

```
60
23.4375
あなたは太っていません。
ダイエットは不要です。
```

  - taijyuやshinchoの値を変化させて、if文が機能していることを確かめてみましょう。



# 条件式の書き方(1)

- 条件式は比較演算子と論理演算子を組み合わせて記述します。
  - 比較演算子
    - 2つの変数・定数同士の大小関係を比較する演算子です。

| 比較演算子 | 読み方 | 使い方 |
|---|---|---|
| == | 等しい | a == b |
| < | 小なり | a < b |
| > | 大なり | a > b |
| <= | 以下 | a <= b |
| >= | 以上 | a >= b |
| != | 等しくない | a != b |

      - 補足:「=」は代入を表す代入演算子なので、等しいかどうかを判定するための演算子は「=」を2つ重ねて記述します。

    - Python独特の記法も
      - Pythonでは、3つの変数・定数同士の大小関係を比較することも可能となっています。
      - 例:　a < b <= c など

# 条件式の書き方(2)

- 論理演算子
  - 複数の条件式を組み合わせて、より複雑な条件を表すときに使う演算子です。

| 論理演算子 | 働き | 使い方の例 | 意味 |
|---|---|---|---|
| and | かつ | a >= 10 and a != 50 | aは10以上かつ50でない |
| or | または | a == 1 or b == 1 | aは1またはbは1 |
| not | でない | not a == 100 | aは100以外 |

  - orよりもandが先に評価されるので、orを先に評価したい場合は丸括弧()で囲みます。
    - 例1・・・if a == 1 or b == 1 and c == 1:
      aが1か、bが1かつcが1ならば成り立つ
    - 例2・・・ if (a == 1 or b == 1) and c == 1:
      aが1またはbが1であり、かつcが1ならば成り立つ

- 条件式と真偽(bool)型
  - 実は、条件式を記述すると、その条件式が成り立つ場合は真偽型のTrueが、成り立たない場合はFalseが返ってきます。if文では、そのTrueやFalseをもとに実行するか否かを判断しています。

```
print(bmi >= 25.0)
```

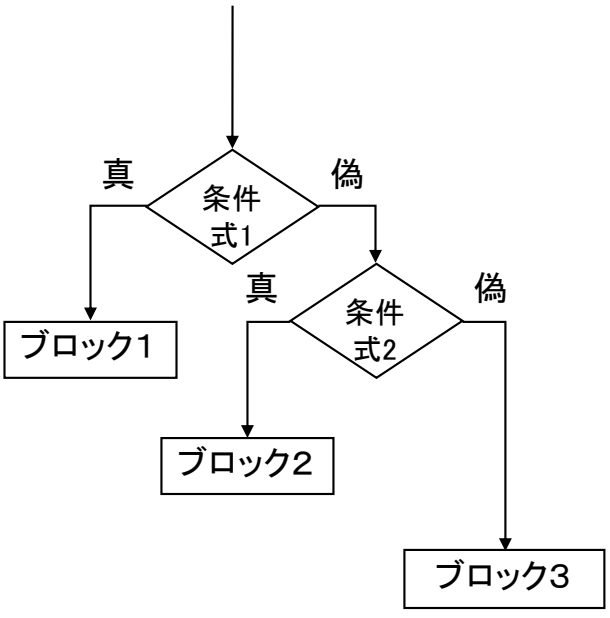
True または False



# if文はelif文で連結できる

- elif文を使うことで、if文が成り立たなかった後で、追加の条件式を評価することができます。

```
例:
if 条件式1:
   ブロック1
elif 条件式2:
   ブロック2
else:
   ブロック3
```



# 繰り返し(while文)

- while文により、繰り返し処理(ループ処理)を記述することができます。
  - ある条件式が成り立っている間だけ、ブロックに書かれた処理を繰り返し実行します。
  - 構文

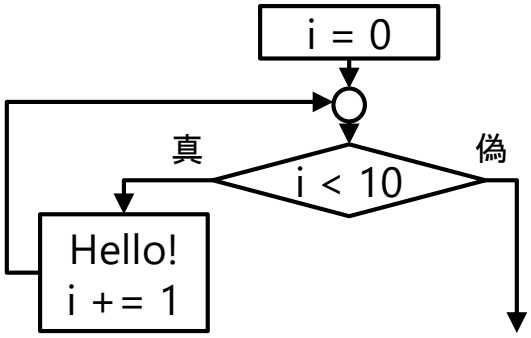
成り立つ場合に繰り返し実行されるブロック

```
while 条件式 :
    条件式が成り立つ場合に実行する処理1
    条件式が成り立つ場合に実行する処理2
    ・・・
```

ブロックは行頭を等しく字下げします(インデント)

  - 例:Helloを10回表示する

```
i = 0
while i < 10:
    print("Hello! " + str(i))
    i += 1
```






# while文による無限ループ

- while文では条件式がTrueの場合に処理を繰り返します。従って、条件式にTrueを指定すると、処理が常に繰り返されます(無限ループ)。
  - 無限ループはプログラムが終了せず暴走することになるため注意が必要です。

```
while True:
    print("Hello!")
```

- break文
  - 無限ループであっても、適切にbreak文を使用することで無限ループから抜け出すことができます。
  - if文とbreak文を併用することで、複雑な繰り返し条件の指定が可能になります。
- continue文
  - continue文はそれ以降の処理を省略してループ処理の先頭に戻す文です。
  - break文と同様に、複雑な繰り返し処理の記述が可能になります。

```
while True:
    n = input("0なら終了")
    if n == "0":
        break
    print("Hello! " + n)
print("終了しました!")
```

breakの例

```
while True:
    n = input("0なら終了、1なら何もしない")
    if n == "0":
        break
    if n == "1":
        continue
    print("Hello! " + n)
print("終了しました!")
```

continueの例

# 繰り返し(for文)

- for文では、シーケンスから要素を1つずつ取り出して、要素がなくなるまで処理を実行します。
  - シーケンスとは？
    - 複数の要素を持ちインデックスで要素の指定ができるデータ型の総称で、文字列、リスト、タプルが該当します。
  - 構文

```
for 要素を取り出す変数 in シーケンス :
    要素ごとに実行する処理1
    要素ごとに実行する処理2
    ・・・
```

ブロックは行頭を等しく字下げします（インデント）

シーケンスの要素が無くなるまで繰り返し実行されるブロック

  - 例：

```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

結果はどちらも同じ

```
for i in (1, 2, 3, 4, 5):
    print(i)
```

```
1
2
3
4
5
```



# シーケンスとrange関数

- for文を使って0から100まで繰り返したい場合、シーケンスをリストで記述するのは効率的ではありません。
例：[0, 1, 2, ・・・, 100]と長々と記述するのはナンセンス

- range関数を使うと、シーケンスを楽に記述できます。
構文：range([開始],終了,[ステップ])
  - ここで、[]は省略可能ですが、終了はシーケンスで実行したい末尾の数+1を設定します。
  - ステップは、幾つおきに増加させるかの設定です。
  - 例：
    - range(100) → [0, 1, 2,・・・, 99]
    - range(0, 100, 2) → [0, 2, 4,・・・, 98]
    - range(0, 101, 5) → [0, 5, 10,・・・, 100]
    - range(1, 101, 5) → [1, 6, 11,・・・, 96]

# 課題4-1

- if文の簡単な例（p7）について、下記の表示となるように修正してください。
  - bmiが25.0以上ならばp7と同様に「あなたは太っています。」「ダイエットを始めましょう。」と表示します。
  - それ以外において、bmiが18.5以上ならば「あなたは普通です。」「ダイエットは不要です。」と表示します。
  - さらにそれ以外ならば「あなたは痩せています。」「もっと栄養を摂りましょう。」と表示します。



# 課題4-2

- P11に示したwhile文による「Helloを10回表示する」をfor文で書き変えてください。

```
i = 0
while i < 10:
    print("Hello! " + str(i))
    i += 1
```

for文では？

# 課題4–3

- 下記のシーケンスによる繰り返しを、range関数を使って記述してみましょう。

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print("Hello! " + str(i))
```

range関数を使うと？