Python基礎研修
Python Basic Training

4.制御構文

4.1 条件分岐

4.2 反復

マッチスル

研修概要

マッチスル

研修概要

- ✓ 本研修は、これからPythonを使ったプログラミングを始めたい方を対象としています。
- ✓ Pythonによる基本的な構文に加え、関数やモジュールの使い方なども学習します。

前提知識

- ✓ プログラミングの基礎知識を有していること。

研修目標

- ✓ Pythonによる、条件分岐や反復などの基本的な構文について理解すること。
- ✓ Pythonによる、関数の定義や利用、モジュールの利用について理解すること。

目次

マッチスル

4.1 条件分岐
Conditional branch

章の目次

- ✓ 制御構文
- ✓ 複合文
- ✓ 条件分岐
- ✓ 条件分岐の構文
  - ✓ ifステートメント
  - ✓ elseステートメント
  - ✓ elifステートメント
- ✓ コラム：ネストされた条件分岐
- ✓ 練習問題

制御構文

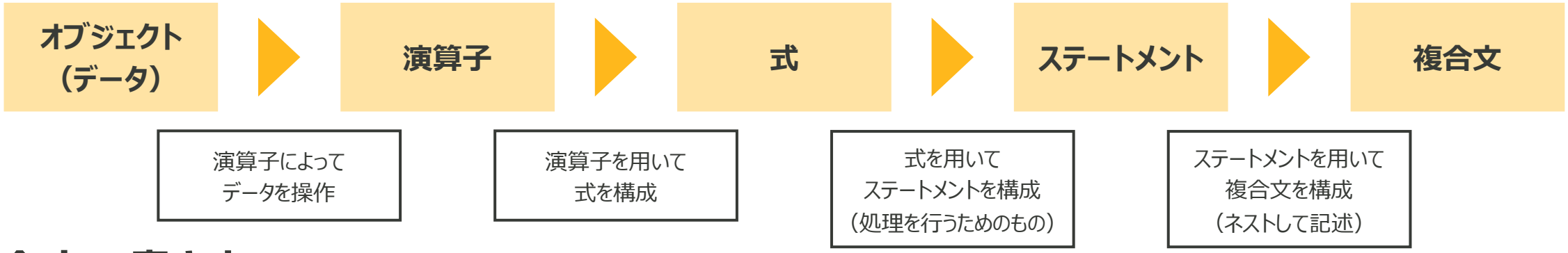
マッチスル

- 複雑なプログラムの記述
- これらは 制御構文 を用いて記述
- 最も基本的な制御構文：
  - 繰り返し (for, while)
  - 条件分岐 (if)
- Python の制御構文は、ヘッダ (header) と ブロック (block) の 2 つの部分で構成
- これらを合わせて 複合文 (compound statement) と呼ぶ

マッチスル

複合文

- ✓ 複合文とは、複数のステートメントによって構成されたもの。



複合文の書き方

- ✓ ヘッダ行と、インデントを空けてネストしたコードブロックによって構成される。

```
if 条件式:
    実行したい処理
    実行したい処理
```

ヘッダ行（見出し行）

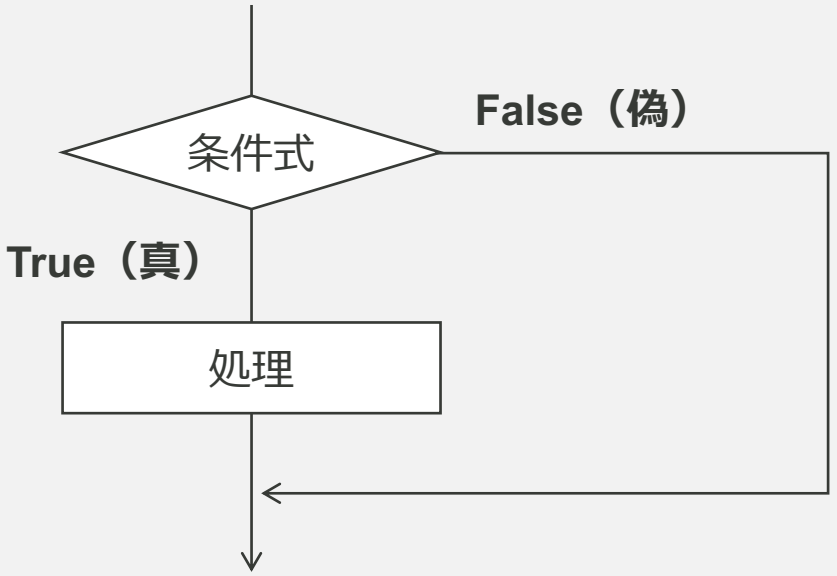
ネストしたコードブロック

インデント（半角スペース4つ または タブ）

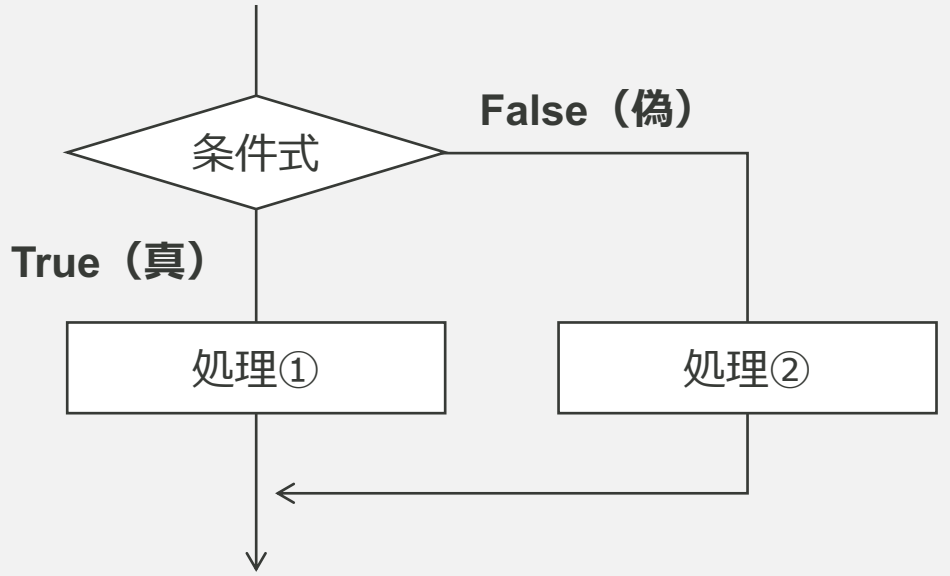
条件分岐とは

- ✓ 背景：複雑な処理の記述には、条件を設定し、条件によって、処理を変えるようにする。
- ✓ 条件分岐とは、条件式を満たす場合と、満たさない場合とで処理を分岐させる構文のこと。
- ✓ 組み合わせ次第では、真と偽の二分岐だけでなく三分岐以上の複雑な分岐も実現可能。

１）条件を満たす場合のみ処理を行う
分岐処理のフローチャート



２）条件を満たさない場合は異なる処理を行う
分岐処理のフローチャート



条件分岐の構文

branch_if.py

```
01: apple = 90
02:
03: if apple < 100:
04:     # appleが100g未満の場合
05:     print("This apple is small.")
06:
07: # appleの重さに関わらず出力
08: print("This is apple.")
```

条件分岐の構文

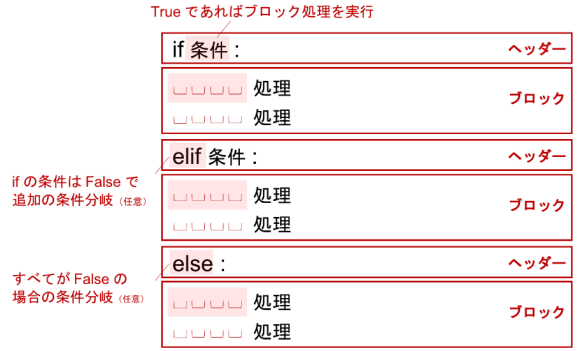
branch_else.py

```
01: apple = 100
02: if apple < 100:
03:     # appleが100g未満の場合
04:     print("This apple is small.")
05: else:
06:     # appleが100g以上の場合
07:     print("This apple is large.")
08: # appleの重さに関わらず出力
09: print("This is apple.")
```

条件分岐の構文

elifステートメント（1/2）

- ✓ 条件が複数ある分岐を行う場合には、<elif> ステートメントを使用する。
- ✓ elif ステートメントは複数記述することができ、三分岐以上の分岐も可能。




```
if 条件式①:
    実行したい処理①
elif 条件式②:
    実行したい処理②
```

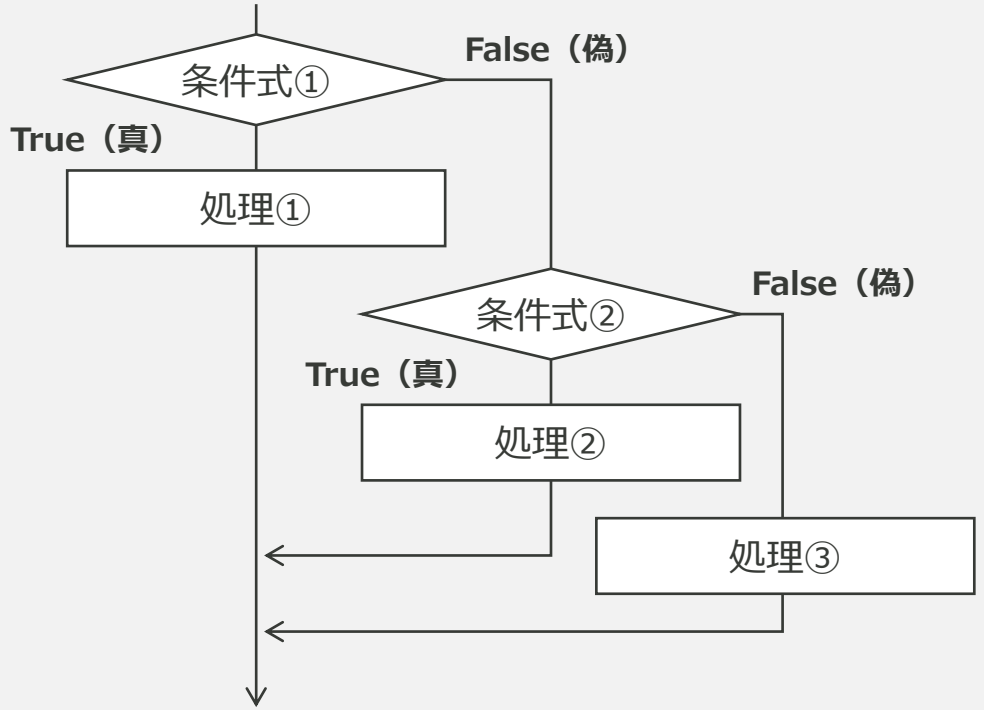
```
elif 条件式③:
    実行したい処理③
elif 条件式④:
    実行したい処理④
```

elifステートメントは、複数記述可能

```
else:
    実行したい処理⑤
```

elseステートメントは、省略可能

フローチャート



条件分岐の構文

マッチスル

elifステートメント（2/2）

branch_elif.py

```
01: apple = int(input("Please put the weight of apple. --> "))
02:
03: if apple < 0:
04:     # appleが0g未満の場合
05:     print("This is not an apple.")
06: elif apple < 100:
07:     # appleが100g未満の場合
08:     print("This apple is small.")
09: elif apple < 150:
10:     # appleが150g未満の場合
11:     print("This apple is middle.")
12: elif apple < 200:
13:     # appleが200g未満の場合
14:     print("This apple is large.")
15: else:
16:     # appleが200g以上の場合
17:     print("This apple is very large.")
18:
19: # appleの重さに関わらず出力
20: print("This is apple.")
```

input関数によってキーボードから入力されたデータ（戻り値）は、Python内部では数字であってもすべて文字列として認識される。そのため、int関数によって文字列から数値に変換する必要がある。

コラム：ネストされた条件分岐

ネストされた条件分岐（1/2）

- ✓ コードブロックの中に、さらにifステートメントを記述することも可能（ネストと呼ぶ）
- ✓ ネストする場合、さらにコードブロックが必要となるので、インデントの数に注意が必要。

```
if 条件式①:
    実行したい処理①
    if 条件式②:
        実行したい処理②
    else:
        実行したい処理③
else
    実行したい処理④
    if 条件式②:
        実行したい処理⑤
    else:
        実行したい処理⑥
```

ifステートメント内の if文（ステートメント）

elseステートメント内の if文（ステートメント）

マッチスル

ネストされた条件分岐（2/2）

branch_nest.py

```
01: apple = int(input("Please put the weight of apple. --> "))
02: color = input("Please put the color of apple.(Red/Green) --> ")
03:
04: # 赤いリンゴの場合
05: if (color == "Red" or  color == "red"):
06:     if apple < 100:
07:         print("This is a red and small apple.")
08:     else:
09:         print("This is a red and large apple.")
10: # 緑のりんごの場合
11: elif (color == "Green" or color == "green"):
12:     if apple < 100:
13:         print("This is a green and small apple.")
14:     else:
15:         print("This is a green and large apple.")
16: else:
17:     print("This is not an apple.")
18:
19: # appleの重さに関わらず出力
20: print("This is apple.")
```

Pythonはどこで役に立つか？複雑な計算が得意

マッチスル

うるう年かどうかの判定：確率１で決まる。
うるう年かどうかは以下のような条件を基に判定される。
その年が4で割り切れるなら、基本的にその年はうるう年
ただし、その年が100で割り切れるときには、その年はうるう年ではない
ただし、その年が400で割り切れるときには、その年はうるう年

```
year = input("年を入力してください: ")
try:
    year = int(year)
    if (year<0):
        print(year, "は年ではありません。")
    elif (year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)):
        print(year, " はうるう年です。")
    else:
        print(year, " はうるう年ではありません。")
except ValueError:
    print(year, "は数値ではありません。")
```

```
年を入力してください:  2024
2024  はうるう年です。
```

```
import math

print(math.sqrt(2))
print(math.log(2))
print(math.log2(2))
print(math.factorial(3))
print(math.comb(3, 1))
```

```
1.4142135623730951
0.6931471805599453
1.0
6
3
```

マッチスル

条件分岐

条件分岐とは

- ✓ 条件式を満たす場合と、満たさない場合とで処理を分岐させる構文のこと。
- ✓ 真と偽の二分岐のほか、三分岐以上の複雑な分岐も構成することが可能。

ifステートメント

- ✓ 条件式を満たす場合の処理を記述するステートメント。
- ✓ インデントを解除すると、ステートメントは終了する。

elseステートメント

- ✓ 条件式を満たさない場合の処理を記述するステートメント。
- ✓ インデントを解除すると、ステートメントは終了する。

elifステートメント

- ✓ 条件が複数ある分岐を行う場合に使用するステートメント。
- ✓ 複数記述することができ、三分岐以上の分岐も構成することが可能。

マッチスル

4.2 反復

Loop

章のまとめ

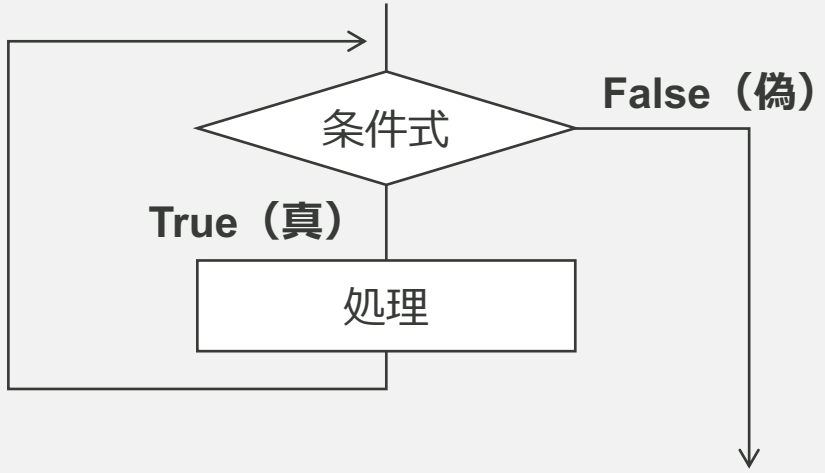
反復（繰り返し）

- ✓ **反復とは**
- ✓ **反復の種類**
- ✓ **反復の構文**
  - ✓ whileループ①
  - ✓ whileループ②
  - ✓ breakステートメント
  - ✓ continueステートメント
  - ✓ forループ①
  - ✓ forループ②
- ✓ **コラム**
- ✓ **コラム①：ステートメントと複合文**
- ✓ **コラム②：リスト内包表記**
- ✓ **練習問題**

反復とは

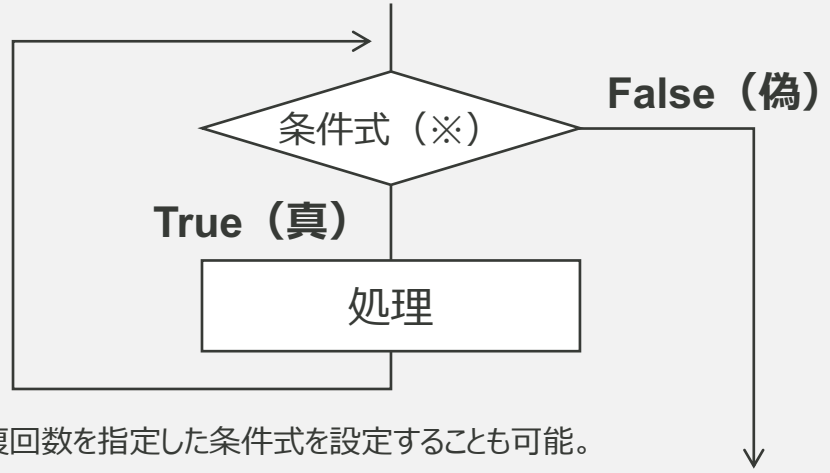
- ✓ 背景：複雑な処理の記述として、条件を満たしている間に繰り返し処理を行う方法がある。
- ✓ 反復とは、条件式を満たしている間は同じ処理を繰り返す構文のこと。
- ✓ 指定した回数のみ繰り返す方法もあり、以下の二種類の構文が存在。
  - ✓ **whileループ**：whileステートメントを使用した構文
  - ✓ **forループ**　：forステートメントを使用した構文

条件式を満たす間、反復するフローチャート　**whileループ**



- ✓ 条件式を満たす間、ずっと同じ処理を繰り返す。
- ✓ 条件式を満たさなくなった時点で、反復を終了。

指定回数のみ、反復するフローチャート　**forループ**



- ✓ 反復回数を指定した条件式を設定することも可能。
- ✓ 指定した回数に到達すると自動的に反復を終了し、後続の処理へ遷移する。

マッチスル

反復の種類

✓ ステートメントの組み合わせ次第で、以下の制御を実現することが可能。

制御の種類	構文の種類	主なステートメント	概要
前判定	whileループ	whileステートメント	最も基本的な反復の構文。 条件式を満たしている限りは同じ処理を繰り返し、満たさなくなれば反復を終了する。 条件式の設定次第では、無限ループと呼ばれる状態も作れてしまうため注意が必要。
後判定	whileループ	whileステートメント breakステートメント	Javaなどの他のプログラミング言語では、専用のステートメントが存在する構文。 pythonでは専用のステートメントは存在しないため、同様の制御を行いたい場合、 whileステートメントとbreakステートメントの組み合わせによって便宜的に実現する。
複合データの反復	forループ	forステートメント	複合データ（複数の要素を持つオブジェクト）を対象とした反復の構文。 含まれる要素を順に取り出して処理を行い、要素がなくなれば自動的に反復を終了する。 条件式の設定ミスによる反復漏れや無限ループを気にする必要が無いため、便利な構文。
指定回数のみ反復	forループ	forステートメント range関数	range関数を用いて疑似的に複合データを作ることにより、 便宜的に指定回数のみ同じ処理を繰り返すことを実現した構文。 複合データの反復と同様、無限ループなどを気にする必要がない。

マッチスル

whileループ①：前判定

- ✓ 条件式を満たしている限りは同じ処理を繰り返し、 満たさなくなれば反復を終了する。
- ✓ 繰り返しの度に条件が変化するように設定しなければ、無限ループに陥る可能性あり。

```
while 条件式:
    実行したい処理
```

loop_while.py

```
01: count = 1
02:
03: while count < 10:
04:     print("Current loop count : " + str(count))
05:
06:     # カウントアップ
07:     count += 1
08:
```

10未満の場合、"継続"

7行目の処理が無ければ、条件式は常に（ 1 < 10 ）となる。
これは無限ループと呼ばれ、whileループを抜けることができなくなり、
永遠にプログラムが終了しない危険な状態となるため、注意が必要。

マッチスル

whileループ②：後判定

- ✓ while ループ内に、**＜反復を終了するための条件＞**を設けた構文。
- ✓ 終了条件よりも前方にある処理は、必ず一度は実行する点が特徴。

```
while True:
    実行したい処理
    if 条件式:
        break
```

loop_while_post.py

```
01: count = 1
02:
03: while True:
04:     print("Current loop count : " + str(count))
05:     count += 1
06:
07:     if count > 10:
08:         break
```

10を超えた場合、"終了"

whileステートメントの条件式を真に固定して無限ループを構成し、
条件分岐とbreakステートメントを組み合わせることで反復を終了する。
7行目は、whileステートメントで用いる継続条件とは異なる点に注意。

反復の構文

breakステートメント

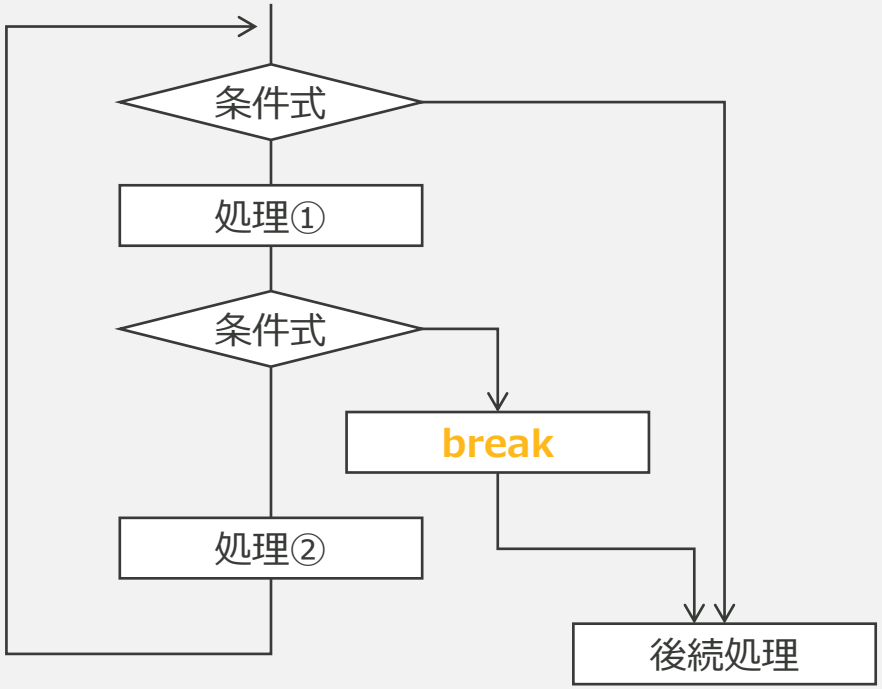
- ✓ 属する反復を終了し、後続の処理へ遷移する。
- ✓ breakステートメント以降に記述された、反復内のコードブロックは実行しない。

```
while 条件式:
    実行したい処理①

    if 条件式:
        break

    実行したい処理②

実行したい処理（後続処理）
```

breakステートメントに到達すると、反復は終了し、後続処理へ遷移

breakステートメントに到達すると、それ以降の処理は実行されない

フローチャート



マッチスル

continueステートメント

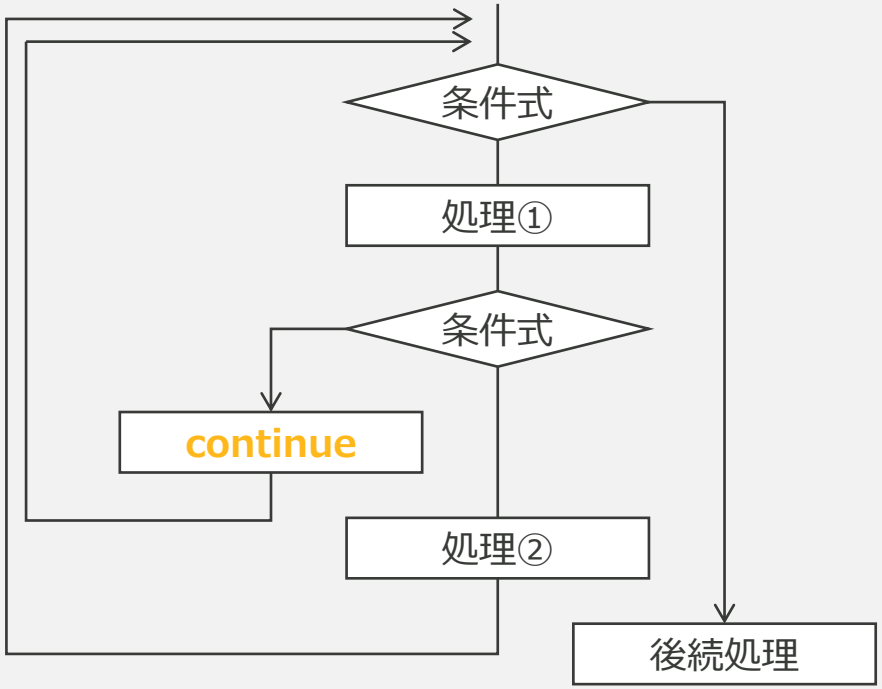
- ✓ 属する反復の先頭に戻り、引続き反復を継続する。
- ✓ continueステートメント以降に記述された、反復内のコードブロックは実行しない。

```
while 条件式:
    実行したい処理①

    if 条件式:
        continue

    実行したい処理②

実行したい処理（後続処理）
```

continueステートメントに到達すると、反復の先頭へ遷移

continueステートメントに到達すると、それ以降の処理は実行されない

フローチャート



反復の構文

マッチスル

forループ①：（複合データの反復）

- ✓ 複合データを対象とした反復では、<for>ステートメントを使用する。
- ✓ 含まれる要素を順に取り出して処理し、要素がなくなれば自動的に反復を終了する。

```
for 変数 in オブジェクト:
    実行したい処理
```

loop_for.py

```
01: week = ["Sunday","Monday","Tuesday","Wednesday","Thuesday","Friday","Saturday"]
02:
03: for day in week:
04:     print(day)
05:
06:
07:
08:
```

forステートメントの変数は、繰り返しの度に次の要素に更新される。
この変数を活用すると、複合データの要素に対し順に処理を行える。

反復の構文

マッチスル

forループ②：（指定回数のみ反復）

- ✓ 整数リストを生成することができる汎用的な **<range関数>** を利用した構文。
- ✓ 無限ループに陥る懸念がないため、繰り返しの回数が決まっている場合に有用。

```
for 変数 in range関数:
    実行したい処理
```

loop_for_range.py

```
01: sum = 0
02:
03: for count in range(1, 10):
04:     print("Current loop count : " + str(count))
05:
06:     sum += count
07:     print("Current sum values : " + str(sum))
08:
```

繰り返しの回数をカウントする変数や条件式を考慮する必要がない。
そのため、反復漏れや無限ループに陥る懸念のないのが利点の一つ。
但し、回数に因らない反復の場合は利用できない。

コラム①：ステートメントと複合文

マッチスル

ステートメントと複合文

- ✓ **<whileステートメント>** や **<whileループ>** など、似たような用語が混用している。
- ✓ 下記の図を参考に、それぞれがどの部分を指す用語なのか誤らないよう気を付けたい。
  - ✓ **ステートメント**：個々の目的別の処理のかたまりを指す。句や節とも呼ばれる。
  - ✓ **複合文**　　：複数のステートメントで構成された、一連の構文全体を指す。

```
01: if apple < 0:
02:     print("This is not an apple.")
03: elif apple < 100:
04:     print("This apple is small.")
05: elif apple < 150:
06:     print("This apple is middle.")
07: else:
08:     print("This apple is very large.")
```

ifステートメント

elifステートメント

elifステートメント

elseステートメント

if文（複合文）

if文やwhileループ、forループなど慣例的に様々な呼称があるが、複合文全体を指していると思われる場合は、中に含まれるすべてのステートメントを含めた構文全体を指していると捉えて解釈すること。

コラム②：リスト内包表記

マッチスル

リスト内包表記

- ✓ リストの内部に条件分岐や反復を記述し、新しいリストを作成する構文。
- ✓ コンパクトに記述できる点や、処理速度が速い点が利点として挙げられる。
- ✓ その一方、ソースコードの可読性が損なわれる点が欠点として挙げられる。

loop_for_list.py

```
01: list_a = [x for x in range(1, 10)]
02: list_b = [x * 2 for x in [1, 10, 3, 8, -3]]
03: list_c = [x * 2 for x in [1, 10, 3, 8, -3] if x < 5]
04:
05: print(list_a)
06: print(list_b)
07: print(list_c)
08: print(range(1,10))
```

章のまとめ

マッチスル

反復

反復とは

- ✓ 条件式を満たしている間は同じ処理を繰り返す構文のこと。
- ✓ 複合データを対象にした構文や、指定回数のみ繰り返す構文も存在。

whileループ

- ✓ 条件式を満たしている間は同じ処理を繰り返し、満たさなくなれば反復を終了する。
- ✓ breakステートメントを組み合わせることで、便宜的に後判定を実現することも可能。

forループ

- ✓ 複合データを対象にした反復や、指定回数のみ繰り返す反復を行いたい場合に利用する。
- ✓ 条件式の設定ミスによる反復漏れや無限ループに陥る懸念がなく、使いやすいのが特徴的。

breakステートメント / continueステートメント

- ✓ breakステートメント　　：属する反復を終了し、後続の処理へ遷移したい場合に利用。
- ✓ continueステートメント：属する反復の先頭に戻り、引続き反復を継続したい場合に利用。