内包表現

Pythonの弱点

- 実行速度が遅い
- リストの性質上，繰り返し処理をfor文で書いたり，行列・ベクトル演算にリストを使うと処理が非常に遅くなる

弱点克服に向けて

- 内包表記 ➡️ 繰り返し処理の強化
- Numpy ➡️ 行列・ベクトル演算の強化

内包表記

- 一般に map や filter と呼ばれる処理を実現する書き方
- Pythonの処理速度を上げるためには必須

map処理

- 例
  - `x = [1, 2, 3, 4, 5]` の各要素を2乗したリスト `y = [1, 4, 9, 16, 25]` を作成する

```
x = [1, 2, 3, 4, 5]
y = []
for v in x:
  y.append(v**2)
print(y)
```

- 前のループの結果が次のループに影響を与えない（順番が関係ない）
  ➡️ 本質的に繰り返し手続きで書く必要がない
- 各要素に同じ操作を適用する ➡️ map処理

例は「自身を２乗する」という操作のmap処理で書ける

```
## 内包表記
y = [v**2 for v in x]
```

「`x` の各要素 `v` に対して `v**2` を実行した**リストを作る**」

filter処理

- 例
  - `x = [1, 2, 3, 4, 5]` の各要素をチェックして偶数だけからなるリスト `y = [2, 4]` を作成する

```
x = [1, 2, 3, 4, 5]
y = []
for v in x:
  if v % 2 == 0:
    y.append(v)
print(y)
```

- 前のループの結果が次のループに影響を与えない（順番が関係ない）
  ➡️ 本質的に繰り返し手続きで書く必要がない
- 各要素をチェックして条件にあうものだけを取り出す ➡️ filter処理

例は「偶数（2で割った余りが0）」という条件によるfilter処理で書ける

```
## 内包表記
y = [v for v in x if v % 2 == 0]
```

「`x` の各要素 `v` に対して `v % 2 == 0` を満たすものだけ取り出した**リストを作る**」

その他注意

- mapとfilterを組み合わせた処理も書ける

```
## 内包表記
y = [v**2 for v in x if v % 2 == 0]
# y = [4, 16] となる
```

- 内包表記は「順番が関係ない」ことがわかるので内部で特別な処理を呼び出せる
  - インタプリタ言語の繰り返し処理は遅い
  - map/filter処理は一般的に速い