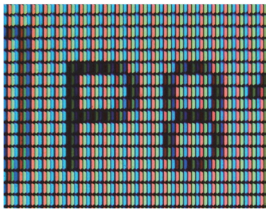
繰り返し処理が嬉しい理由

- データは**配列（シーケンス）**の形式で存在することが多い

※ **複数の値が順に並んでいるデータ**、例えば、数値ベクトル、文字列、Pythonのリスト、画像のピクセル配置、は全て配列とみなされる

➡ 「**複数のデータを連続的に処理する**」ための効率的な仕組みが不可欠



1
2
3
4
5
⋮

y	X1	X2	X3	...	XM
0					
1					
0					
...					
0					

繰り返し処理を使う・使わないの例

効率の良い、読みやすい、バグが起きにくい ような書き方が重要

例： ‘Hello Python’ の中の文字を 1 個ずつ出力する処理

```
message = 'Hello Python'

print(message[0])
print(message[1])
print(message[2])
print(message[3])
print(message[4])
print(message[5])
print(message[6])
print(message[7])
print(message[8])
print(message[9])
print(message[10])
print(message[11])
```

VS

繰り返し処理（Forループ）を用いた場合

```
message = 'Hello Python'

for ch in message:
    print(ch)
```

同じ文を**何度も書く代わりに**、
ループの中に処理を書くことで
その処理を**決められた回数繰り返せる**

For文の活用①

- 代表的な繰り返し処理の記述法は **For文**
  予め決まった回数だけ配列に対して処理を繰り返す場合に使用

■ For文の書き方
配列の各要素をループ毎に頭から取り出して、
ブロック中で処理に使う

for 変数 in シーケンス:
(indent) **繰り返し実行したい処理**

条件分岐と同様に
ブロック内でインデント

つまり、**インデントが繰り返し処理の開始と終了の印**となる

For文の活用②

■ For文の書き方
配列の各要素をループ毎に頭から取り出して、
ブロック中で処理に使う

for 変数 in シーケンス:
(indent) **繰り返し実行したい処理**

```
for x in [1,2,3,4,5,6]:
    print(x*2)
```

```
2
4
6
8
10
12
```

インデント

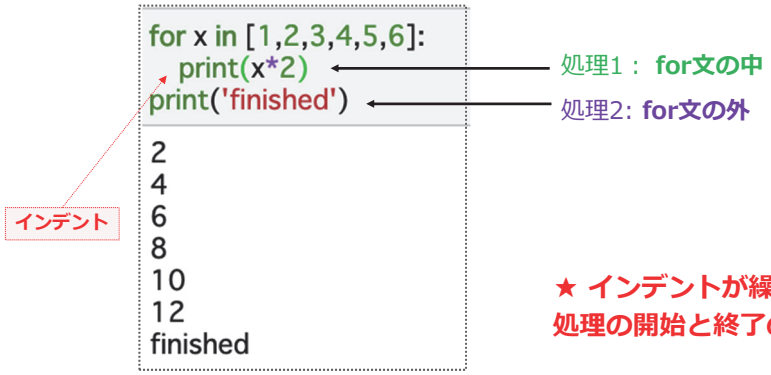
配列 [1,2,3,4,5,6] の各要素を順に取り出し、2を乗じた値を順に出力

配列の全要素を「見終わる」まで処理を繰り返す

For文の活用③

繰り返し処理の終了を明確に意識しないとバグを引き起こす

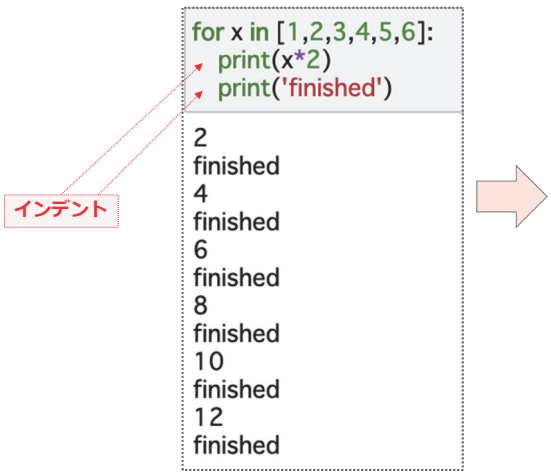
下の例では、
処理1: print(x*2) は**for文の中**、処理2: print('finished') は**for文の外**なので、処理1が指定回数繰り返された後に、処理2は**1回だけ**実行される



★ インデントが繰り返し処理の開始と終了の印！

For文の活用④

（間違った処理の例）
外で処理したかった print('finished') をForループの中に入れてしまった場合



余分な処理がされてしまう

処理が重いコードの場合、プログラムがクラッシュすることも …