リストとループの活用

Python プログラミング入門

複数データの管理から効率的な処理まで、視覚的に学ぶ

リストの基本

リスト操作

ループ活用

enumerate

スライス

内包表記

この知識があると何ができる？

リストとループは、「使えるプログラム」を作るための必須スキルです

大量データの一括管理

100人分の点数を
1つの変数で管理
いちいち変数を
作る必要なし

データの自動集計

合計・平均・最大値を
自動で計算
手作業のミスが
なくなる

条件に合うデータ抽出

80点以上だけを
自動で抽出
フィルタリングが
一瞬で完了

生成AIにコードを書かせる場合でも、リストとループを理解していれば「指示の精度」が格段に上がります

なぜリストが必要？

1つの変数＝1つの値 → 大量データの管理が非効率

リストを使わない場合

```
# 生徒ごとに変数が必要...
student1 = 85
student2 = 92
student3 = 78
student4 = 88

# 平均計算も大変...
avg = (s1+s2+s3+s4) / 4
```

リストを使う場合

```
# まとめて管理！
scores = [85, 92, 78, 88]

# 平均も簡単！
avg = sum(scores)/len(scores)

→ 85.75
```

リストの基本: インデックス

各要素には0から始まる番号(インデックス)が振られる

```
fruits = ["りんご", "バナナ", "オレンジ", "ぶどう"]
```

CODE

```
# インデックスで要素にアクセス
print(fruits[0])
→ りんご
print(fruits[2])
→ オレンジ
```

CODE

```
# 負のインデックス(後ろから)
print(fruits[-1])
→ ぶどう(最後の要素)
print(fruits[-2])
→ オレンジ
```

リスト操作メソッド

append()

末尾に1つ追加

```
fruits = ["りんご"]
fruits.append("バナナ")
→ ["りんご", "バナナ"]
```

insert()

指定位置に挿入

```
fruits = ["A", "C"]
fruits.insert(1, "B")
→ ["A", "B", "C"]
```

remove()

指定値を削除

```
fruits = ["A", "B", "C"]
fruits.remove("B")
→ ["A", "C"]
```

extend()

複数要素を追加

```
a = [1, 2]
a.extend([3, 4])
→ [1, 2, 3, 4]
```

append と extend の違い

混同しやすいポイントを視覚的に理解しよう

append([4, 5])

リストごと1つの要素として追加される

CODE

```
list1 = [1, 2, 3]
list1.append([4, 5])

→ [1, 2, 3, [4, 5]]
```

extend([4, 5])

要素が個別に追加される

CODE

```
list2 = [1, 2, 3]
list2.extend([4, 5])

→ [1, 2, 3, 4, 5]
```

リスト + for文

全要素に同じ処理を自動実行する

基本パターン

CODE

```
fruits = ["りんご", "バナナ", "オレンジ"]

for fruit in fruits:
    print("{}が好き".format(fruit))
```

→ りんごが好き
→ バナナが好き
→ オレンジが好き

実用例: 点数の集計

CODE

```
scores = [85, 92, 78, 88, 95]

total = 0
for score in scores:
    total += score

avg = total / len(scores)
```

→ 平均: 87.6点

条件に合うデータだけを抽出

for文 + if文で、必要なデータだけを新しいリストに集める

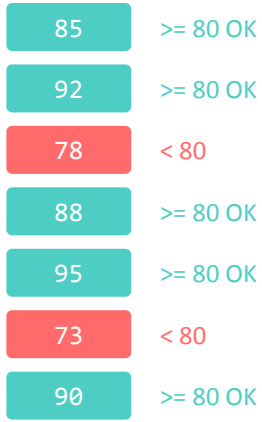
CODE

```
scores = [85, 92, 78, 88, 95, 73, 90]

# 80点以上だけを集める
high_scores = []
for score in scores:
    if score >= 80:
        high_scores.append(score)

→ [85, 92, 88, 95, 90]
```

処理のイメージ



この「フィルタリング」パターンは実務で非常によく使います

enumerate 関数

インデックス（番号）と要素を同時に取得する

基本の使い方

```
fruits = ["りんご", "バナナ", "オレンジ"]
for i, fruit in enumerate(fruits):
    print("{}番目: {}".format(i, fruit))
→ 0番目: りんご
→ 1番目: バナナ
```

実用例：成績順位表示

```
names = ["田中", "佐藤", "鈴木"]
scores = [95, 92, 88]
for rank, (name, score) in enumerate(zip(names, scores), 1):
    print("{}位: {} ({}点)".format(rank, name, score))
```

リストのスライス

リストの一部分を取り出す[開始:終了] / [開始:終了:ステップ]

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

範囲指定

```
numbers[2:5]
→ [2, 3, 4]

numbers[:4]
→ [0, 1, 2, 3]

numbers[7:]
→ [7, 8, 9]
```

ステップ指定

```
numbers[::2]
→ [0, 2, 4, 6, 8]

numbers[1:8:2]
→ [1, 3, 5, 7]
```

逆順

```
numbers[::-1]
→ [9, 8, 7, ..., 0]

# 全要素を逆順に！
```

リスト内包表記

for文でリストを作る処理を1行 で書ける

通常のfor文（4行）

```
squares = []
for i in range(5):
    squares.append(i ** 2)
→ [0, 1, 4, 9, 16]
```

内包表記（1行）

```
squares = [i**2 for i in range(5)]

→ [0, 1, 4, 9, 16]
```

書き方の構造

```
[式 for 変数 in リストやrange()]
```

条件付きリスト内包表記

if文を追加して、条件に合う要素だけを集める

フィルタリング

```
scores = [85, 92, 78, 88, 95, 73]
high = [s for s in scores if s>=80]
→ [85, 92, 88, 95]
```

変換＋フィルタ

```
nums = [1,2,3,4,5,6,7,8,9,10]
r = [n*2 for n in nums if n%2==0]
→ [4, 8, 12, 16, 20]
```

実用例：税込価格

```
prices = [1200, 800, 1500]
tax = [int(p*1.1) for p in prices]
→ [1320, 880, 1650]
```

💡 使い分けの目安

1行で理解できる → 内包表記

処理が複雑 → 普通のfor文

迷ったら → 普通のfor文

まとめ: リストとループ

リストの基本

複数データを1つにまとめて管理
インデックスは 0 から
len() で要素数取得

リスト操作

append: 末尾に追加
insert: 指定位置に挿入
remove: 削除 / extend: 複数追加

ループ活用

for文で全要素を処理
if文で条件フィルタリング
新しいリストの作成

enumerate

番号と要素を同時取得
開始番号を指定可能
zip()との組み合わせ

スライス

[開始:終了] で部分取得
[::ステップ] で間隔指定
[::-1] で逆順

内包表記

for文の処理を1行で
if条件でフィルタ可能
シンプルな処理向き