辞書の基本

Python プログラミング入門

「名前」でデータを引き出す、Pythonの便利なデータ構造

辞書とは

リストとの違い

作成と操作

安全なアクセス

メソッド

for文との活用

この知識があると何ができる？

辞書は「名前でデータを管理する」ための必須スキルです

名前で即座にデータ取得

「佐藤さんの電話番号」を
一発で取得できる
リストのように
検索ループ不要

ユーザー情報の管理

名前・年齢・メールなど
を1つにまとめる
Webアプリの
データ管理に必須

在庫や集計の自動化

商品コードで在庫管理
投票のカウント集計
実務でよく使う
パターンが身につく

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

なぜ辞書が必要？

リストだと「名前で検索」するのが面倒 → 辞書なら一発

⊗ リストで管理する場合

```
# 2つのリストで管理...
names = ["田中", "佐藤", "鈴木"]
phones = ["090-...", "080-...", ...]

# 佐藤さんを探すには...
for i, name in enumerate(names):
    if name == "佐藤":
        print(phones[i])
        break
```

✓ 辞書で管理する場合

```
# 辞書なら一発！
phone_book = {
    "田中": "090-1111-2222",
    "佐藤": "080-3333-4444",
    "鈴木": "090-5555-6666"
}

print(phone_book["佐藤"])
→ 080-3333-4444
```

辞書の基本概念：キーと値

キー（見出し語）で値（意味）を引き出す仕組み

実世界の辞書

りんご → バラ科の落葉高木の果実

Python → プログラミング言語

辞書 → キーと値のペアでデータ管理

キー（key）　　値（value）

Pythonの辞書

CODE

```
# 基本的な辞書の構造
user = {
    "name": "田中太郎",
    "age": 25,
    "city": "東京"
}
```

キーは重複不可 / 値は何でもOK（文字列, 数値, リスト, 辞書...）

リストと辞書の違い

「番号で管理」か「名前で管理」か

リスト：番号（インデックス）でアクセス


```
students = ["田中", "佐藤", "鈴木"]
print(students[1])
→ 佐藤
```

辞書：キー（名前）でアクセス


```
scores = {"田中": 85, "佐藤": 92, ...}
print(scores["佐藤"])
→ 92
```

どちらを使うべき？

リスト → 順番が意味を持つ場合

```
曜日: ["月", "火", "水", ...]
成績の履歴: [80, 85, 90, ...]
```

辞書 → 名前でデータを引きたい場合

```
ユーザー情報: {"user001": "田中太郎"}
商品在庫: {"A001": 50, "B002": 30}
```

辞書の作成とアクセス

波括弧 { } で作成し、キーで値を取り出す

辞書の作成

CODE

```
# 空の辞書
empty = {}

# キーと値のペアを持つ辞書
user = {
    "name": "田中太郎",
    "age": 25,
    "city": "東京"
}
```

構文ポイント

キーと値は : で区切る

複数のペアは , で区切る

キー: 文字列や数値 / 値: 何でもOK

要素へのアクセス

CODE

```
# キーで値を取得
print(user["name"])
→ 田中太郎

print(user["age"])
→ 25

print(user["city"])
→ 東京
```

リストとの違い

リスト: list[0] → 番号でアクセス

辞書: dict["キー"] → 名前でアクセス

安全なアクセス：get() と in

存在しないキーへのアクセスでエラーを防ぐ方法

存在しないキーでエラー

```
user = {"name": "田中", "age": 25}
print(user["email"])
→ KeyError: 'email'
```

get()メソッドで安全に取得

```
# キーがなければ None を返す
email = user.get("email")
→ None

# デフォルト値を指定できる
email = user.get("email", "未登録")
→ 未登録
```

in演算子で存在確認

CODE

```
user = {"name": "田中", "age": 25}

if "email" in user:
    print(user["email"])
else:
    print("メール未登録")

→ メール未登録
```

使い分けガイド

dict["key"] → キーがある確信がある時
dict.get("key") → キーがないかもしれない時
"key" in dict → 分岐処理に使う時

辞書の操作: 追加・変更・削除

追加

```
# 新しいキーに代入
user["email"] = "t@ex.com"

# emailが追加される
```

変更

```
# 既存のキーに再代入
user["age"] = 26

# 25 → 26 に変わる
```

削除

```
# delで削除
del user["temp"]

# tempが消える
```

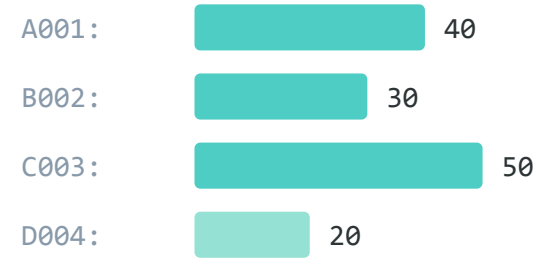
実用例: 在庫管理

CODE

```
inventory = {"A001": 50, "B002": 30, "C003": 0}

inventory["D004"] = 20    # 新商品追加
inventory["A001"] -= 10   # 10個売れた
inventory["C003"] += 50   # 50個入荷
```

実行後の在庫



辞書のメソッド

keys()

すべてのキーを取得

```
scores = {"田中": 85, "佐藤": 92}

print(scores.keys())
→ dict_keys(['田中', '佐藤'])
```

values()

すべての値を取得

```
print(scores.values())
→ dict_values([85, 92])

# 合計や平均にも使える
sum(scores.values())
→ 177
```

items()

キーと値のペアを取得

```
print(scores.items())
→ [('田中',85), ('佐藤',92)]

# for文と相性抜群！
for name, score in scores.items():
    print(f"{name}: {score}点")
```

辞書とfor文の組み合わせ

辞書のデータを順番に処理する方法

キーでループ

```
scores = {"田中": 85, "佐藤": 92, "鈴木": 78}

for name in scores:
    print(f"{name}: {scores[name]}点")

→ 田中: 85点
→ 佐藤: 92点
→ 鈴木: 78点
```

items()でループ（推奨）

```
# キーと値を同時に取得できる
for name, score in scores.items():
    print(f"{name}: {score}点")

→ 田中: 85点
→ 佐藤: 92点
→ 鈴木: 78点
```

辞書の内包表記

CODE

```
# 1から5の2乗を辞書にする
squares = {n: n**2 for n in range(1, 6)}
→ {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

実用例: 成績判定システム

辞書 + for文 + 条件分岐で実践的なプログラムを作る

CODE

```
scores = {
    "田中": 85, "佐藤": 92,
    "鈴木": 78, "高橋": 65, "伊藤": 58
}

for name, score in scores.items():
    if score >= 80:
        grade = "優"
    elif score >= 70:
        grade = "良"
    elif score >= 60:
        grade = "可"
    else:
        grade = "不可"
    print(f"{name}: {score}点 ({grade})")
```

実行結果

田中	85点	優
佐藤	92点	優
鈴木	78点	良
高橋	65点	可
伊藤	58点	不可

合格者(60点以上): 4人 / 5人

辞書 + for文 + if文 = 実務で使えるデータ処理パターン

まとめ: 辞書の基本

辞書の基本

キーと値のペアでデータを管理
{キー: 値} の形式で作成
リストは「順番」、辞書は「名前」

辞書の操作

dict[キー] = 値 で追加・変更
del dict[キー] で削除
get() で安全にアクセス

辞書のメソッド

keys(): すべてのキー取得
values(): すべての値を取得
items(): ペアで取得

辞書とループ

for文でキーを順番に処理
items()でキーと値を同時取得
内包表記で辞書を簡潔に生成

次回: 関数の定義と活用 → リスト・辞書・ループを組み合わせて実践的なプログラムへ