関数の定義と活用

Python プログラミング入門

処理をまとめて再利用── コードを「部品」にする技術を視覚的に学ぶ

関数の定義

引数と戻り値

再利用性

この知識があると何ができる？

関数は、プログラムを「部品化」して効率よく開発するための基礎技術です

同じ処理の繰り返しを削減

税込計算、割引処理など
1回書くだけで何度でも使える

修正が1箇所で済む

税率変更、仕様変更など
関数内を直すだけで全体に反映

コードが読みやすくなる

処理に名前がつくので
何をしているか一目でわかる

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

関数とは？

処理をまとめて「名前」をつけたもの

日常生活のたとえ

「コーヒーを淹れて」と言えば

1. 豆を挽く
2. お湯を沸かす
3. ドリップする

細かい手順を説明しなくても伝わる！

CODE

```
# 関数を定義する
def greet():
    print("こんにちは！")

# 関数を呼び出す
greet()
→ こんにちは！
```

print()、len() などの組み込み関数は使ってきた → 今回は「自分で作る」方法を学ぶ

関数の定義

def文の書き方

```
def 関数名():
    処理1
    処理2
```

- def の後に関数名を書く
- 関数名の後に () をつける
- 最後に : (コロン)をつける
- 処理はインデントする

関数名のコツ

```
# 何をするか動詞で始める
calculate_total()
show_menu()
get_user_name()
```

```
# 推奨されない関数名
f()                    何をする関数かわからない
MyFunction()
```

引数と戻り値

引数（ひきすう）

関数に情報を渡す仕組み

CODE

```
def greet(name):
    print("こんにちは、{}さん！"
          .format(name))

greet("田中")
→ こんにちは、田中さん！
```

戻り値（もどりち）

関数から結果を受け取る仕組み（ return文）

CODE

```
def add(a, b):
    return a + b

total = add(10, 20)
→ 30
```

値を渡す

引数

→

関数で処理

def内の処理

→

結果を返す

return

関数を使う3つのメリット

再利用性

一度作れば何度でも使える

```
def fmt(price):
    return "{:,}円"
      .format(price)

fmt(1500)
→ 1,500円
```

保守性

修正は1箇所だけでOK

```
def calc_tax(price):
    rate = 0.10
    return price*(1+rate)
# ↑ここだけ直せば全体に反映
```


可読性

処理の意図が明確になる

```
# 意図が明確
grade = get_grade(85)
→ A
```

引数のデフォルト値

引数を省略できるようにする仕組み

CODE

```
def greet(name, greeting="こんにちは"):
    print("{}、{}さん！"
          .format(greeting, name))

# 値を指定
greet("田中", "おはよう")
→ おはよう、田中さん！

# デフォルト値を使う（省略）
greet("佐藤")
→ こんにちは、佐藤さん！
```

正しい順序

```
def create_user(name, age, role="一般"):
```

デフォルト値なし→ デフォルト値ありの順番

エラーになる例

```
def create_user(name, role="一般", age):
```

デフォルト値を活用すると、よく使う設定は省略でき、必要なときだけ指定できる柔軟な関数になる

実用シーン

関数なし（同じ処理を3回書く）

```
# 商品A
price_a = 1000
tax_a = price_a * 0.1
total_a = price_a + tax_a

# 商品B（同じ計算を繰り返す...）
price_b = 1500
tax_b = price_b * 0.1
total_b = price_b + tax_b

# 商品C（さらに繰り返す...）
price_c = 800
tax_c = price_c * 0.1
total_c = price_c + tax_c
```

→

関数あり（1回定義、何度でも使う）

```
def calc_tax(price):
    tax = price * 0.1
    return price + tax

# 関数を使って計算
print(calc_tax(1000))
→ 1100.0
print(calc_tax(1500))
→ 1650.0
print(calc_tax(800))
→ 880.0
```

まとめ

関数の基本

処理をまとめて名前をつけたもの

def 関数名(): で定義する

関数名() で呼び出す

引数と戻り値

引数: 関数に情報を渡す

戻り値: return で結果を返す

デフォルト値で省略可能に

3つのメリット

再利用性: 何度でも使える

保守性: 修正は1箇所だけ

可読性: 処理の意図が明確

次のステップ → リスト、辞書、ループと組み合わせた総合演習