# 【初心者向け】Python基本の8つのデータ型完全ガイド

Pythonプログラミングで必ず押さえておくべき8つの基本データ型を徹底解説します。

## 目次



---

## int（整数型）

整数を扱うデータ型です。Pythonのintは任意精度なので、桁数の制限がありません。

### 基本的な使い方

```python
# 基本
age = 25
count = -10
big_number = 1_000_000  # アンダースコアで読みやすく（実際の値は1000000）

# 演算
result = 10 + 5  # 15（加算）
result = 10 - 3  # 7（減算）
result = 10 * 2  # 20（乗算）
result = 10 // 3  # 3（整数除算）
result = 10 % 3  # 1（剰余）
result = 2 ** 8  # 256（べき乗）
```

### よく使う操作

```python
```

```python
# 絶対値
abs(-10)  # 10

# 累乗
pow(2, 3)  # 8

# 最大値・最小値
max(1, 5, 3)  # 5
min(1, 5, 3)  # 1

# 型チェック
isinstance(10, int)  # True
type(10)  # <class 'int'>
```

---

## float（浮動小数点数型）

小数を扱うデータ型です。

### 基本的な使い方

```python
# 基本
price = 99.99
temperature = -15.5
pi = 3.14159

# 科学的記法
small = 1.5e-3  # 0.0015
large = 2.5e6   # 2500000.0

# 演算
result = 10.5 + 2.3  # 12.8
result = 10.0 / 3.0  # 3.3333...
```

### よく使う操作

```python
```

```python
# 四捨五入
round(3.14159, 2)  # 3.14

# 切り上げ・切り捨て
import math
math.ceil(3.2)   # 4
math.floor(3.8)  # 3

# 絶対値
abs(-10.5)  # 10.5

# 平方根
math.sqrt(16)  # 4.0
```

### 注意点：浮動小数点数の誤差

```python
# ⚠️ 浮動小数点数の誤差
0.1 + 0.2  # 0.30000000000000004

# ✅ 正確な比較が必要な場合
from decimal import Decimal
Decimal('0.1') + Decimal('0.2')  # Decimal('0.3')
```

---

## str（文字列型）

テキストデータを扱うデータ型です。イミュータブル（変更不可）です。

### 基本的な使い方

```python
```

```python
# 基本
name = "太郎"
message = 'Hello, World!'

# 複数行
multiline = """これは
複数行の
文字列です"""

# 文字列の連結
greeting = "Hello, " + "World!"
full_name = "山田" + " " + "太郎"

# 繰り返し
repeat = "ABC" * 3  # "ABCABCABC"
```

## f-string（フォーマット済み文字列）

python

```python
# f-string（Python 3.6+、推奨）
name = "太郎"
age = 25
message = f"{name}は{age}歳です"
message = f"{name}は来年{age + 1}歳になります"

# 小数点の桁数指定
price = 1234.5
f"価格: ¥{price:,.2f}"  # "価格: ¥1,234.50"
```

## よく使う操作

python

```python
text = "Hello World"

# 長さ
len(text)  # 11

# 大文字・小文字
text.upper()  # "HELLO WORLD"
text.lower()  # "hello world"
text.capitalize()  # "Hello world"

# 検索
text.find("World")  # 6（インデックス）
"World" in text  # True
text.count("o")  # 2

# 置換
text.replace("World", "Python")  # "Hello Python"

# 分割・結合
words = text.split()  # ['Hello', 'World']
"-".join(words)  # "Hello-World"

# 前後の空白削除
"  Hello  ".strip()  # "Hello"
```

## インデックスとスライス

```python
text = "Python"

# インデックス
text[0]  # 'P'
text[-1]  # 'n'（最後）

# スライス [start:end:step]
text[0:3]  # 'Pyt'
text[:3]  # 'Pyt'（先頭から）
text[3:]  # 'hon'（最後まで）
text[::2]  # 'Pto'（2文字おき）
text[::-1]  # 'nohtyP'（反転）
```

---

## list（リスト型）

順序を持つ、変更可能なコレクション型です。Pythonで最もよく使われるデータ型の一つです。

### 基本的な作成

```python
# 基本
fruits = ["りんご", "バナナ", "みかん"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "text", True, 3.14, None]  # 異なる型も混在可

# 空のリスト
empty = []
empty = list()

# 範囲からリストを作成
numbers = list(range(5))  # [0, 1, 2, 3, 4]
numbers = list(range(1, 6))  # [1, 2, 3, 4, 5]
```

### 要素の追加

```python
fruits = ["りんご", "バナナ"]

# 末尾に1つ追加
fruits.append("みかん")
# ['りんご', 'バナナ', 'みかん']

# 指定位置に追加
fruits.insert(0, "いちご")  # 先頭に追加
fruits.insert(2, "メロン")  # インデックス2に追加

# 複数追加
fruits.extend(["ぶどう", "桃"])
fruits += ["梨", "柿"]

# リスト同士の結合
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined = list1 + list2  # [1, 2, 3, 4, 5, 6]
```

### 要素の削除

```python
```

```python
fruits = ["りんご", "バナナ", "みかん", "いちご"]

# 値で削除（最初に見つかったものを削除）
fruits.remove("バナナ")

# インデックスで削除
del fruits[0]

# 末尾を削除して取得
last = fruits.pop()  # "いちご"を削除して返す
second = fruits.pop(1)  # インデックス1を削除して返す

# すべて削除
fruits.clear()
```

## 要素へのアクセス

python

```python
fruits = ["りんご", "バナナ", "みかん", "いちご", "ぶどう"]

# インデックスでアクセス
first = fruits[0]  # "りんご"
last = fruits[-1]  # "ぶどう"

# スライス [start:end:step]
subset = fruits[1:3]  # ['バナナ', 'みかん']
first_three = fruits[:3]  # ['りんご', 'バナナ', 'みかん']
every_other = fruits[::2]  # ['りんご', 'みかん', 'ぶどう']

# 要素の変更
fruits[0] = "メロン"
fruits[1:3] = ["桃", "梨"]  # 複数要素を一度に変更
```

## よく使う操作

python

```python
```

```python
numbers = [3, 1, 4, 1, 5, 9, 2, 6]

# 長さ
len(numbers)  # 8

# 出現回数
numbers.count(1)  # 2

# インデックス取得
numbers.index(5)  # 4

# 存在確認
4 in numbers  # True

# 最大値・最小値・合計
max(numbers)  # 9
min(numbers)  # 1
sum(numbers)  # 31

# ソート
numbers.sort()  # リストを直接変更
sorted_nums = sorted(numbers)  # 新しいリストを返す

# 降順ソート
numbers.sort(reverse=True)

# 反転
numbers.reverse()  # リストを直接変更
```

## リスト内包表記

python

```python
# 基本
squares = [x**2 for x in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# 条件付き
evens = [x for x in range(20) if x % 2 == 0]
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

# if-else
labels = ["正" if x > 0 else "負" for x in [-1, 2, -3, 4]]
# ['負', '正', '負', '正']

# ネストしたループ
pairs = [(x, y) for x in range(3) for y in range(3)]
# [(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)]
```

## ループ処理

python

```python
fruits = ["りんご", "バナナ", "みかん"]

# 基本的なループ
for fruit in fruits:
    print(fruit)

# インデックス付きループ
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")

# 複数リストを同時にループ
names = ["太郎", "花子", "次郎"]
ages = [25, 23, 30]
for name, age in zip(names, ages):
    print(f"{name}は{age}歳")
```

## 注意点

python

```python
# ⚠️ コピーの罠
list1 = [1, 2, 3]
list2 = list1  # 参照のコピー
list2.append(4)
print(list1)  # [1, 2, 3, 4] - list1も変わる！

# ✅ 正しいコピー
list2 = list1.copy()
list2 = list1[:]
list2 = list(list1)

# ⚠️ ループ中の削除は危険
numbers = [1, 2, 3, 4, 5]
# ❌ 間違い
for num in numbers:
    if num % 2 == 0:
        numbers.remove(num)

# ✅ リスト内包表記を使う
numbers = [num for num in numbers if num % 2 != 0]
```

---

## tuple（タプル型）

順序を持つ、**変更不可能な**コレクション型です。リストと似ていますが、一度作成したら変更できません。

### 基本的な作成

python

```python
# 基本
coordinates = (10, 20)
rgb = (255, 0, 0)
person = ("太郎", 25, "東京")

# 1要素のタプル（カンマが必要！）
single = (1,)  # ✅ タプル
not_tuple = (1)  # ❌ これは整数の1

# 空のタプル
empty = ()
empty = tuple()

# カッコなしでも作成可能
point = 10, 20  # (10, 20)

# リストからタプルを作成
numbers = tuple([1, 2, 3, 4, 5])
```

## 要素へのアクセス

python

```python
point = (10, 20, 30)

# インデックスでアクセス
x = point[0]  # 10
z = point[-1]  # 30

# スライス
subset = point[0:2]  # (10, 20)
reversed_point = point[::-1]  # (30, 20, 10)

# アンパッキング（展開）
x, y, z = point
print(x, y, z)  # 10 20 30

# 一部だけ取得
first, *rest = (1, 2, 3, 4, 5)
# first = 1, rest = [2, 3, 4, 5]

*beginning, last = (1, 2, 3, 4, 5)
# beginning = [1, 2, 3, 4], last = 5

first, *middle, last = (1, 2, 3, 4, 5)
# first = 1, middle = [2, 3, 4], last = 5
```

## よく使う操作

python

```python
numbers = (1, 2, 3, 2, 1, 4)

# 長さ
len(numbers)  # 6

# 出現回数
numbers.count(2)  # 2

# インデックス取得
numbers.index(3)  # 2

# 存在確認
2 in numbers  # True

# 最大値・最小値・合計
max(numbers)  # 4
min(numbers)  # 1
sum(numbers)  # 13
```

### タプルの結合と繰り返し

```python
# 結合
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
combined = tuple1 + tuple2  # (1, 2, 3, 4, 5, 6)

# 繰り返し
repeated = (1, 2) * 3  # (1, 2, 1, 2, 1, 2)
```

### タプルとリストの使い分け

```python
```

```python
# ✅ タプルを使うべき場面

# 1. 変更されるべきでないデータ
DATE_OF_BIRTH = (1990, 5, 15)  # 生年月日

# 2. 座標や色などの固定データ
position = (35.6762, 139.6503)  # 緯度経度
color = (255, 128, 0)  # RGB

# 3. 関数から複数の値を返す
def get_user_info():
    return "太郎", 25, "東京"

name, age, city = get_user_info()

# 4. 辞書のキーとして使用
locations = {
    (35.6762, 139.6503): "東京",
    (34.6937, 135.5023): "大阪"
}

# ✅ リストを使うべき場面

# 1. 要素を追加・削除する必要がある
tasks = ["買い物", "掃除"]
tasks.append("料理")

# 2. 要素を変更する必要がある
scores = [85, 90, 78]
scores[0] = 88

# 3. ソートが必要
numbers = [3, 1, 4, 1, 5]
numbers.sort()
```

## 注意点

python

```python
# ⚠️ タプルは変更不可
point = (10, 20, 30)
# point[0] = 15  # ❌ TypeError

# ⚠️ 1要素のタプルの落とし穴
not_tuple = (1)    # int型の1
is_tuple = (1,)    # tuple型の(1,)

# ⚠️ タプルのアンパッキング時の要素数
x, y = (1, 2, 3)  # ❌ ValueError: too many values

# ✅ *を使って残りをまとめる
x, *rest = (1, 2, 3)  # x=1, rest=[2, 3]
```

---

## dict（辞書型）

キーと値のペアでデータを管理する型です。高速な検索が特徴です。

### 基本的な使い方

```python
# 基本
student = {
    "name": "田中",
    "age": 20,
    "grade": "A"
}

# 別の作成方法
student = dict(name="田中", age=20, grade="A")

# 空の辞書
empty = {}
empty = dict()
```

### 追加・更新・削除

```python
```

```python
student = {"name": "田中", "age": 20}

# 追加・更新
student["grade"] = "A"
student["age"] = 21

# 複数追加・更新
student.update({"city": "東京", "club": "テニス"})

# 削除
del student["city"]
age = student.pop("age")

# すべて削除
student.clear()
```

### 取得

```python
student = {"name": "田中", "age": 20, "grade": "A"}

# 直接アクセス
name = student["name"]  # "田中"

# getメソッド（推奨）
grade = student.get("grade")  # "A"
city = student.get("city")  # None
city = student.get("city", "未設定")  # デフォルト値

# キーの存在確認
if "age" in student:
    print(f"年齢: {student['age']}")

# すべてのキー・値・ペア
keys = student.keys()
values = student.values()
items = student.items()
```

### 辞書内包表記

```python
```

```python
# 基本
squares = {x: x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# 条件付き
even_squares = {x: x**2 for x in range(10) if x % 2 == 0}

# 2つのリストから辞書を作成
keys = ['name', 'age', 'city']
values = ['田中', 25, '東京']
person = dict(zip(keys, values))
```

---

## set（集合型）

重複のない要素の集まりを扱う型です。順序は保持されません。

### 基本的な使い方

python

```python
# 基本
numbers = {1, 2, 3, 4, 5}
fruits = {"りんご", "バナナ", "みかん"}

# 重複は自動的に削除される
unique = {1, 2, 2, 3, 3, 3}  # {1, 2, 3}

# リストから作成
numbers = set([1, 2, 3, 2, 1])  # {1, 2, 3}

# 空のセット
empty = set()
```

### 追加・削除

python

```python
```

```python
fruits = {"りんご", "バナナ"}

# 追加
fruits.add("みかん")

# 複数追加
fruits.update(["いちご", "ぶどう"])

# 削除
fruits.remove("バナナ")  # 存在しないとKeyError
fruits.discard("バナナ")  # 存在しなくてもエラーなし

# すべて削除
fruits.clear()
```

## 集合演算

```python
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

# 和集合（どちらかに含まれる）
union = a | b  # {1, 2, 3, 4, 5, 6}

# 積集合（両方に含まれる）
intersection = a & b  # {3, 4}

# 差集合（aにあってbにない）
difference = a - b  # {1, 2}

# 対称差（どちらか一方にだけ含まれる）
sym_diff = a ^ b  # {1, 2, 5, 6}
```

## 実用例

```python
```

```python
# 重複削除
numbers = [1, 2, 2, 3, 3, 3, 4, 5]
unique = list(set(numbers))  # [1, 2, 3, 4, 5]

# 共通要素の検索
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
common = set(list1) & set(list2)  # {3, 4}
```

---

## bool（ブール型）

真偽値を扱う型です。`True`か`False`の2値のみです。

### 基本的な使い方

```python
# 基本
is_active = True
is_finished = False

# 比較演算の結果
is_adult = age >= 18
is_equal = name == "田中"

# 論理演算
result = True and False  # False
result = True or False   # True
result = not True        # False
```

### 比較演算子

```python
```

```python
# 等価・不等価
10 == 10  # True
10 != 5   # True

# 大小比較
10 > 5    # True
10 >= 10  # True

# 複数条件
x = 15
10 < x < 20  # True
```

### 真偽値への変換

```python
# 数値
bool(0)    # False
bool(1)    # True

# 文字列
bool("")    # False（空文字列）
bool("Hello") # True

# コレクション
bool([])    # False（空のリスト）
bool([1, 2]) # True
bool({})    # False（空の辞書）

# None
bool(None)  # False
```

---

## 型変換と注意点

### 基本的な型変換

```python
```

```python
# 文字列 → 整数
age = int("25")  # 25

# 文字列 → 浮動小数点数
price = float("99.99")  # 99.99

# 数値 → 文字列
text = str(123)  # "123"

# リスト → セット（重複削除）
numbers = [1, 2, 2, 3]
unique = set(numbers)  # {1, 2, 3}

# 文字列 → リスト
chars = list("abc")  # ['a', 'b', 'c']

# リスト → 文字列
text = "".join(['a', 'b', 'c'])  # "abc"
```

### 型チェック

```python
# isinstance関数（推奨）
isinstance(10, int)  # True
isinstance(10, (int, float))  # True
isinstance("text", str)  # True
```

---

## まとめ：データ型選択ガイド

### 用途別の選び方

| データ型 | 使用場面 | 変更可能 | 順序 | 重複 |
|---|---|---|---|---|
| **int** | カウント、ID | - | - | - |
| **float** | 価格、距離 | - | - | - |
| **str** | テキスト | ❌ | ✅ | ✅ |
| **list** | 動的なコレクション | ✅ | ✅ | ✅ |
| **tuple** | 固定データ | ❌ | ✅ | ✅ |
| **dict** | キーと値のペア | ✅ | ✅* | キー:❌ |
| **set** | 重複排除 | ✅ | ❌ | ❌ |
| **bool** | フラグ、条件 | - | - | - |

*Python 3.7+では順序を保持

### ベストプラクティス

```
python

# ✅ 推奨
- f-stringを使う
- dict.get()でデフォルト値を設定
- is演算子でNoneをチェック
- リスト内包表記を活用
- 変更不要ならtupleを使う

# ❌ 非推奨
- 古いフォーマット方法
- KeyErrorを無視
- ==でNoneをチェック
- ループ中にリストを変更
```

実際にコードを書いて、データ型の特性を体感してみてください！

---

### 参考リンク

- Python公式ドキュメント - 組み込み型