【Python】Python に参照渡しは存在しない話

CREFIL
2023年8月14日 18:39

はじめに

こんにちは。CREFIL の Manager の廣澤です。
私が今 JOIN している案件で初めて Python を触っているのですが、学習中にネット上の誤情報に騙されてしまったことがありました。
少しでも正しい情報を増やしたいと思い、その内容を記事にしようと思います。

Python はミュータブルかどうかで値渡しか参照渡しが変わる？

どうもネット上の記事を眺めていると、見出しに書いたような勘違いが広まっているようです。
どういう主張かと言えば、以下のように主張しているのです。

- 関数の引数に渡すのがイミュータブルな型（intなど）であれば、呼び出し元の変数は影響を受けない（値渡し）
- 関数の引数に渡すのがミュータブルな型（リストなど）であれば、呼び出し元の変数は影響を受ける（参照渡し）

これは部分的には正しいのですが、そもそも参照渡しを勘違いしていることによる誤りです。
Python には「(参照の) 値渡し」しか存在しません。

そもそも値渡し・参照渡しとは

値渡し、参照渡しとは、関数の引数への値の渡し方の種類です。
コードで示すと以下のような挙動の違いになります。

```
def func(arg):
    arg = arg * 2

a = 1
func(a)
print(a)
>>> 1  ## 値渡し　の場合の結果
>>> 2  ## 参照渡しの場合の結果
```

Python において、参照渡しの挙動をすることはありません。

参照の値渡しとは

Python は参照の値渡しを行います。
ここでいう「参照の値」とは、そのデータが存在するメモリのアドレス（番号）のことです。
Python はこれを id() 関数で見ることができるので、サンプルコードで確認してみます。

```
a = 1
b = 1
id(a)
>>> 140709427798824
id(b)
>>> 140709427798824

c = []
d = []
id(c)
>>> 2516224995008
id(d)
>>> 2516224977920
```

id() で出力した数字が、その変数のデータのアドレスです。
a と b の id() の結果が同じなのは、int 型で同じ値のデータについては複数オブジェクトを生成せず、メモリを節約するように Python の中で最適化しているためだと思われます。
c と d の id() の結果が異なるのは、変数に入れる時点でそれぞれ新しいオブジェクトを生成しているためです。
上のサンプルに続き、以下のコードを記述します。

```
def printId(arg):
    print(id(arg))

id(a)
>>> 140709427798824
printId(a)
>>> 140709427798824
id(b)
>>> 140709427798824
printId(b)
>>> 140709427798824
id(c)
>>> 2516224995008
printId(c)
>>> 2516224995008
id(d)
>>> 2516224977920
printId(d)
>>> 2516224977920
```

id() の結果と printId() 関数で出力される id(arg) の結果が同じです。
つまり、printId() の引数である args には呼び出し元の変数が保持しているデータと同じものが渡されている分かりました。
具体的な挙動としては、呼び出し元の変数（a~d）が保持しているデータのアドレス（id()の結果）がコピーされて引数に渡され、arg はそのデータを参照しています。
これを参照の値渡しと呼びます。

なぜ Python に参照渡しがあると思われるのか？

勘違いしているサイトでは以下のようなコードで、Python に参照渡しがあると主張しているようです。

```
def func(arg):
    arg.append(2)
```

```
a = [1]
func(a)
print(a)
>>> [1, 2]
```

これは呼び出し元の引数に影響を与えている挙動をしているため、参照渡しに見えるようです。

しかしこれは、呼び出し元の引数（a）と呼び出し先の引数（arg）は同じオブジェクトを指し示しているために起こっています。
呼び出し元の引数（a）は依然として同じオブジェクトを参照しています。

さて、では次の関数はどうでしょうか？

```
def add(arg1, arg2):
    arg1 = arg1 + arg2

a = 1
b = 2
c = [1]
d = [2]
add(a, b)
print(a)
>>> 1
add(c, d)
print(c)
>>> [1]
```

int 型の場合もリスト型の場合も同じ挙動となります。
もし Python のリスト型が参照渡しであるならば、c は [1, 2] となるはずです。

なぜこの場合は呼び出し元の変数は影響を受けないのでしょうか？
少し変更して id() を見てみましょう。

```
def add(arg1, arg2):
    print(id(arg1))
    arg1 = arg1 + arg2
    print(id(arg1))

c = [1]
d = [2]
print(id(c))
>>> 2516224996224
```

```
    add(c, d)
    >>> 2516224996224
    >>> 2516224977920
    print(id(c))
    >>> 2516224996224
```

arg1 = arg1 + arg2 において、arg1 のメモリアドレスが変わっています。
つまり、arg1 + arg2 の結果となる新しいオブジェクトが生成され、arg1 に再代入されています。
しかし arg1 と c はそれぞれ独立した変数であるため、add() 関数内での arg1 の変更は呼び出し元である c には反映されません。
これが参照の値渡しの挙動です。

まとめ

値渡し、参照渡し、参照の値渡しの挙動は以下のようになります。

- 値渡し
  - 呼び出し先で再代入した場合
    - 呼び出し元に影響なし
  - 呼び出し先でオブジェクトの操作をした場合
    - 呼び出し元に影響なし
- 参照渡し
  - 呼び出し先で再代入した場合
    - 呼び出し元変数の参照先も変わる
  - 呼び出し先でオブジェクトの操作をした場合
    - 呼び出し元変数が参照しているオブジェクトも変わる
- 参照の値渡し
  - 呼び出し先で再代入した場合
    - 呼び出し元変数の参照先は変わらず、影響を受けなくなる
  - 呼び出し先でオブジェクトの操作をした場合
    - 呼び出し元変数が参照しているオブジェクトも変わる

Python はこのうち、「参照の値渡し」しか持ちません。
最初から用意されている int 型や str 型はイミュータブルでオブジェクトの操作ができず再代入しかできないために値渡しのような挙動をしますが、あくまでも参照の値渡しをしています。

細かいことですが、これを理解していないと上のサンプルで示した add() 関数で呼び出し元も影響を受けるように勘違いしてしまいます。

この記事にて、少しでも初学者の混乱が減れば幸いです。