この記事は最終更新日から1年以上が経過しています。

@rdon_key (rdon_key)

Pythonの関数の引数は参照渡しなのか？参照の値渡し

Python 言語

最終更新日 2023年03月04日 投稿日 2023年03月03日

多くの言語の関数やメソッドの引数は、値渡し、参照渡し、参照の値渡しの３種類に分類される。この記事では、引数の3種類を改めて整理しPythonにおける引数が参照の値渡しであることを説明する。

引数について

関数やメソッドにおいて実行時に使用される変数や値のことを引数と呼ぶ。次のadd_numbers関数では、xおよびyが引数である。

```
def add_numbers(x, y):
    return x + y
```

値渡し、参照渡し、参照の値渡し

値渡しでは、引数に値を変数にコピーして渡す。完全にコピーしたものを渡すので、呼び出し側は影響を受けない。単純でわかりやすいが、この方

法はオブジェクトや構造体のように大きい値の場合大量のメモリを消費してしまう。

参照渡しでは、引数に値ではなく値が格納されているキー（メモリアドレスや固有のidなど）を渡す。呼び出し側と同じ変数を操作することになるので、書き換え操作を行うと呼び出し元の値も影響を受ける。この方法では、キーだけを直接渡すのでメモリ消費がない。（実際のメモリ消費量は実装による。）

参照の値渡しでは、引数に値のキーを変数にコピーして渡す。参照渡しと同じように書き換え操作を行うと呼び出し元の変数も影響を受ける。しかし、引数は変数なので別の値のキーを上書きすることができる。キーをコピーした変数分だけメモリを消費する。

違いを次のように整理する。

	呼び出し元の値の書き換え	**引数の保存場所**	**メモリ消費**
値渡し	できない	関数内に新規に値をコピー	大（値と同じ）
参照渡し	できる	呼び出し元と同一	なし（実装による）
参照の値渡し	できる	関数内に新規にキーをコピー	小（キーを保存する分）

C言語では値渡しのみしか存在せず、参照の値渡しをポインタ型変数の値渡しとして実現した。この影響かもしれないが、参照の値渡しのことを参照渡しと呼ぶこともある。だが、C言語より後に開発されたC++言語等では参照渡しが別途設定されていることもあるので区別したほうが良い。Pythonでは参照の値渡しのみを採用しているが、あたかも値渡しのように振舞うことがある。次節からはそれらのことについて述べる。

Pythonの引数

まず、参照の値渡しであることを示す。Pythonではすべての変数がオブジェクトである。値のまま関数に渡すのはメモリ効率が悪いので、すべての

引き数に対して参照の値渡しを行っている。具体的なコードを示す。id関数はオブジェクトのキーとなる値を返す組み込み関数である。

```
def add_one(in_list):
    print (f'in: {id(in_list)}')
    in_list.append(1)

out_list = [1, 2, 3]
print (f'out: {id(out_list)}')
print(out_list)
add_one(out_list)
print (f'out: {id(out_list)}')
print(out_list)
```

実行結果

```
out: 139862611002496
[1, 2, 3]
in: 139862611002496
out: 139862611002496
[1, 2, 3, 1]
```

全てのオブジェクトのIDが同一で、add_one関数の呼び出し後に元のオブジェクトが変化して、`[1, 2, 3, 1]`になったことがわかる。これは参照渡しもしくは参照の値渡しの動作である。

このコードを少し書き換える。

```
def add_one(in_list):
    print (f'in: {id(in_list)}')
    in_list=[4,5,6]
    print ('in_list=[4,5,6]')
    print (f'in: {id(in_list)}')
    print(in_list)


out_list = [1, 2, 3]
print (f'out: {id(out_list)}')
print(out_list)
add_one(out_list)
print (f'out: {id(out_list)}')
print(out_list)
```

実行結果

```
out: 140347445396544
[1, 2, 3]
in: 140347445396544
in_list=[4,5,6]
in: 140347443969664
[4, 5, 6]
out: 140347445396544
[1, 2, 3]
```

関数内の`in_list=[4,5,6]`でin_listを代入する。Pythonでは代入は新しいオブジェクトを生成する。これによりin_listのidが`in: 140347443969664`に変化している。しかし、関数を終了し呼び出し元に戻ってくると、`out: 140347445396544`に戻る。これは参照の値渡しの動作である。

イミュータブルとミュータブル

イミュータブルとは書き込みができないオブジェクトのことで、通常の書き込みができるオブジェクトのことをミュータブルと呼ぶ。Pythonではイミュータブルなオブジェクトを引数で渡したとき、参照の値渡しではなく値渡しのように振舞うと思うかもしれない。イミュータブルなオブジェクトであるcomplexの例を見ていこう。

```python
def update_complex(z):
    print(f"in1)id of z:{id(z)}")
    print(z);
    z += 1j
    print("z += 1j")
    print(f"in2)id of z:{id(z)}")
    print(z);

a = complex(1, 2)
print(f"out1)id of a:{id(a)}")
print(a)
update_complex(a)
print(f"out2)id of a:{id(a)}")
print(a)
```

結果

```
out1)id of a:140567753204272
(1+2j)
in1)id of z:140567753204272
```

```
(1+2j)
z += 1j
in2)id of z:140567753206352
(1+3j)
out2)id of a:140567753204272
(1+2j)
```

関数の呼び出し前と後で値は変わっていない。これは値渡しの性質である。一方、関数内では、`z += 1j`の実行前後で、オブジェクトのキーとなるidの値が変わっている事にお気づきだろうか。イミュータブルなオブジェクトに変更操作が行われると、コピーしたオブジェクトを生成しそのオブジェクトに変更操作を行う。これは一見すると値渡しの振舞に近い。しかし、実のところ`+=`演算子が、加算した新しいオブジェクトを生成しているにすぎない。

```
>>> a = complex(1, 2)
>>> id(a)
140708241902576
>>> a += 1j
>>> id(a)
140708241902864
```

イミュターブルであっても書き込みができる場合がある。それはオブジェクトに含まれるオブジェクトまでは影響を及ぼさないからだ。tupleの例をあげる。

```
def update_tuple(in_tuple):
    print (f'in: {id(in_tuple)}')
    in_tuple[0]['name']='alice'
    print ("in_tuple[0]['name']='alice'")
    print (f'in: {id(in_tuple)}')
    print(in_tuple)

out_tuple = ({'name': 'mike'},'hello')
print (f'out: {id(out_tuple)}')
print(out_tuple)
update_tuple(out_tuple)
print (f'out: {id(out_tuple)}')
print(out_tuple)
```

結果

```
out: 140594714782464
({'name': 'mike'}, 'hello')
in: 140594714782464
in_tuple[0]['name']='alice'
in: 140594714782464
({'name': 'alice'}, 'hello')
out: 140594714782464
({'name': 'alice'}, 'hello'
```

呼び出し元のout_tupleが書き換わっており、idも変わらないのでミュータブルのように見えるが、tuple型はイミュータブルであり、破壊的な関数は用意されていない。

Python公式サイトの注意

これまでPythonにおいてイミュターブルとミュータブルの違いが重要であることを説明した。公式ドキュメントにも記載があるが、わかりにくい表記も見られる。公式サイトでイミュターブルとミュータブルかを確認する上での注意点を述べる。

date型の説明へのリンクを次に示す。https://docs.python.org/ja/3/library/datetime.html#datetime.date

date型はイミュータブルであるが、このリンクから読み進めると記載されておらず見すごす。datetime型の説明の途中に記述がある。

> これらの型のオブジェクトは変更不可能 (immutable) です。

datetime --- 基本的な日付型および時間型

ソースコード: Lib/datetime.py

---

datetime モジュールは、日付や時刻を操作するためのクラスを提供しています。

日付や時刻に対する算術がサポートされている一方、実装では出力のフォーマットや操作のための効率的な属性の抽出に重点を置いています。

参考:

calendar モジュール
汎用のカレンダー関連関数。

time モジュール
時刻へのアクセスと変換。

zoneinfo モジュール
Concrete time zones representing the IANA time zone database.

dateutil パッケージ
拡張タイムゾーンと構文解析サポートのあるサードパーティーライブラリ。

Aware オブジェクトと Naive オブジェクト

日時のオブジェクトは、それらがタイムゾーンの情報を含んでいるかどうかによって "aware" あるいは "naive" に分類されます。

タイムゾーンや夏時間の情報のような、アルゴリズム的で政治的な適用可能な時間調節に関する知識を持っているため、 **aware** オブジェクトは他の aware オブジェクトとの相対関係を特定できます。 aware オブジェクトは解釈の余地のない特定の実時刻を表現します。 [1]

naive オブジェクトには他の日付時刻オブジェクトとの相対関係を把握するのに足る情報が含まれません。あるプログラム内の数字がメートルを表わしているのか、マイルなのか、それとも質量なのかがプログラムによって異なるように、naive オブジェクトが協定世界時 (UTC) なのか、現地時間なのか、それとも他のタイムゾーンなのかはそのプログラムに依存します。Naive オブジェクトはいくつかの現実的な側面を無視してしまうというコストを無視すれば、簡単に理解でき、うまく利用することができます。

aware オブジェクトを必要とするアプリケーションのために、 datetime と time オブジェクトは追加のタイムゾーン情報の属性 tzinfo を持ちます。 tzinfo には抽象クラス tzinfo のサブクラスのインスタンスを設定できます。 これらの tzinfo オブジェクトは UTC 時間からのオフセットやタイムゾーンの名前、夏時間が実施されるかどうかの情報を保持しています。

ただ一つの具象 tzinfo クラスである timezone クラスが datetime モジュールで提供されています。 timezone クラスは、UTCからのオフセットが固定である単純なタイムゾーン（例えばUTCそれ自体）、および北アメリカにおける東部標準時（EST）／東部夏時間（EDT）のような単純ではないタイムゾーンの両方を表現できます。より深く詳細までタイムゾーンをサポートするかはアプリケーションに依存します。世界中の時刻の調整を決めるルールは合理的というよりかは政治的なもので、頻繁に変わり、UTC を除くと都合のよい基準というものはありません。

定数

datetime モジュールでは以下の定数を公開しています:

datetime.**MINYEAR**
date や datetime オブジェクトで許されている、年を表現する最小の数字です。 MINYEAR は 1 です。

datetime.**MAXYEAR**
date や datetime オブジェクトで許されている、年を表現する最大の数字です。 MAXYEAR は 9999 です。

datetime.**UTC**
Alias for the UTC timezone singleton datetime.timezone.utc.

バージョン 3.11 で追加.

利用可能なデータ型

class datetime.**date**
理想的な naive な日付で、これまでもこれからも現在のグレゴリオ暦 (Gregorian calender) が有効であることを仮定しています。 属性は year, month, および day です。

class datetime.**time**
理想的な時刻で、特定の日から独立しており、毎日が厳密に 24*60*60 秒であると仮定しています。("うるう秒: leap seconds" の概念はありません。) 属性は hour, minute, second, microsecond, および tzinfo です。

class datetime.**datetime**
日付と時刻を組み合わせたものです。 属性は year, month, day, hour, minute, second, microsecond, および tzinfo です。

class datetime.**timedelta**
date, time, あるいは datetime クラスの二つのインスタンス間の時間差をマイクロ秒精度で表す経過時間値です。

class datetime.**tzinfo**
タイムゾーン情報オブジェクトの抽象基底クラスです。 datetime および time クラスで用いられ、カスタマイズ可能な時刻修正の概念 (たとえばタイムゾーンや夏時間の計算) を提供します。

class datetime.**timezone**
tzinfo 抽象基底クラスを UTC からの固定オフセットとして実装するクラスです。

バージョン 3.2 で追加.

これらの型のオブジェクトは変更不可能 (immutable) です。

サブクラスの関係は以下のようになります:

検索して必要な型だけ確認する際には注意が必要である。

まとめ

Pythonの関数の引き数はすべて参照の値渡しである。イミュータブルなオブジェクトの場合、値渡しのように見える場合もあるが実際は演算子や関数によって複製されている。イミュータブルなオブジェクトの書き込みができない性質は、含まれるオブジェクトまでは考慮されない。