2.9.9　インスタンス生成に関する制御： __new__

Python では，クラスのインスタンスが生成された直後に __init__ メソッドが実行され，当該インスタンスの初期化処理が行われるが，インスタンスの生成処理自体は特殊メソッド __new__ によって行われる．通常の場合，インスタンス生成の直前に自動的に __new__ メソッドが実行され，この処理を意識することはない．また，__new__ メソッドは，クラス階層の最上位のクラスである object クラスのものを起点として，全てのクラスはこれを継承している．

__new__ メソッドはユーザが定義するクラスでオーバーライドすることも可能であり，インスタンス生成に関するユーザ独自の制御が可能である．次に示す例は，コンストラクタが常に同じオブジェクトを返すクラスの定義である．

例. クラス Identical

```
>>> class Identical:  Enter      ←クラス定義の開始
...     __inst = ('Identical',)  Enter
...     def __new__(cls):  Enter
...         return cls.__inst  Enter
...  Enter      ←クラス定義の記述の終了
```

このように __new__ メソッドの最初の仮引数には当該クラスを受け取る．ただし，通常のクラスメソッドのようにデコレータ @classmethod を伴わない．上の例では，このクラスのコンストラクタが呼び出されると，クラス変数である s の値が無条件に返される．すなわち，オブジェクトの新たな生成は行わず，常にこの変数の値が返される．(次の例)

例. MySingleton コンストラクタで得られるオブジェクト（先の例の続き）

```
>>> x = Identical()  Enter      ←コンストラクタでオブジェクトを取得 (1)
>>> x  Enter                    ←内容確認
('Identical',)
>>> y = Identical()  Enter      ←コンストラクタでオブジェクトを取得 (2)
>>> y  Enter                    ←内容確認
('Identical',)
>>> x is y  Enter      ← x と y は同一のオブジェクト
True                   ←である．
```

コンストラクタで x と y にオブジェクトを得ているが，それらが同一のオブジェクトであることがわかる．もちろんそれらの値は，事後に作った ('MySingleton',) とは別のものである．(次の例)

例. x，y の同一性の調査（先の例の続き）

```
>>> z = ('Identical',)  Enter      ← x，y と同じ内容の別オブジェクト
>>> x is z  Enter      ← x と z は同一のオブジェクトか？
False                  ←違う
>>> print(id(x),id(y),id(z),sep='¥n')  Enter      ← x，y，z の識別値を調べる
2360142629040          ← x と
2360142629040          ← y は同じだが，
2360142630672          ← z は違う
```

以上のことから Identical は，コンストラクタが常に同じオブジェクトを返すクラスであることがわかる．(上の例の識別値は実行時に自動的に決められる)

注意）上に示した Identical クラスの設計パターンは実際のプログラミングにおいては推奨されない．
__new__ メソッドはインスタンスを生成するためのものであり，次に解説するように，基底クラスの __new__ メソッドを呼び出す形で設計するべきである．

2.9.9.1　新規インスタンス生成の仕組み

Python で新規クラスを定義する場合，基底クラス（スーパークラス）の __new__ メソッドが暗黙のうちに継承される．これにより，当該新規クラスのコンストラクタを実行すると，そのクラスの __new__ メソッドが実行される．そしてその中に記述されている

```
return super().__new__( cls, 引数並び… )
```

が実行され，新規オブジェクトが生成されて返される．この処理は最上位の object クラスまで連鎖的に実行される．

すなわち，object クラスの __new__ が実際に新たなオブジェクトを生成し，それがその拡張クラスのインスタンスとなる．
「引数並び」にはコンストラクタに与えた引数が渡される．

2.9.9.2 シングルトン

インスタンスを 1 回だけ生成し，以降はその同じインスタンスを返すようなクラスを**シングルトン**であるという．シングルトンクラスの例を次に示す．

例. シングルトンクラス MySingleton

```
>>> class MySingleton:  Enter      ←クラス定義の開始
...       __inst = None   Enter
...       def __new__(cls,*a,**ka):   Enter
...           if cls.__inst:   Enter
...               return cls.__inst    Enter
...           else:   Enter
...               cls.__inst = super().__new__(cls)    Enter
...               return cls.__inst    Enter
...       def __init__(self,v):   Enter
...           self.value = v    Enter
...   Enter      ←クラス定義の記述の終了
```

このクラス MySingleton は，隠蔽されたクラス変数 __inst を持ち，初期状態では None となっている．このクラスでは，コンストラクタが最初に呼び出された際に，super().__new__(cls) が実行されて新規オブジェクトが 1 つ生成され，続いて __init__ が呼び出されて初期化処理が施された後，それがこのクラスのインスタンスとして返される．しかし，2 回目以降のコンストラクタの呼び出しでは，初回に生成されたインスタンスに対して __init__ を実行して返す．

MySingleton クラスのインスタンス生成の例を次に示す．

例. MySingleton のコンストラクタ（先の例の続き）

```
>>> x = MySingleton(7)   Enter      ←初回のインスタンス生成
>>> x.value   Enter      ←値（value 属性）の確認
7                    ←コンストラクタの引数に与えた値
>>> y = MySingleton(9)   Enter      ← 2 回目のインスタンス生成
>>> y.value   Enter      ←値（value 属性）の確認
9                    ←コンストラクタの引数に与えた値
```

この例ではインスタンスが x，y にえられているが，これらが同一のインスタンス（初回生成時のもの）であることを確認する．（次の例）

例. x，y が同一のものであることの確認（先の例の続き）

```
>>> x.value = 11   Enter      ←初回生成時のインスタンスの value 属性に値を設定
>>> y.value   Enter      ← 2 回目生成時のインスタンスの value 属性の値を確認
11                   ←初回生成時のインスタンスのものと同じ
>>> print(id(x),id(y),sep='¥n')   Enter      ← x，y の識別値を調べる
2208385239760        ←どちらも同じ
2208385239760        ←識別値である
```

需要）上に示したクラス設計のパターンは，データベース接続，ファイルアクセス，通信といった局面で，
同一の処理対象に継続的に処理を実行する場合に応用できる．