5–1 オブジェクト

これまで、いろいろな型の変数を使ってきました。Python は、すべてをオブジェクトとして扱うため、“変数ありき”ではなく、“オブジェクトありき”です。

オブジェクトとは

ここまでの学習は、『変数とは、値を格納する箱のようなものである。』という理解のもとで進めてきました。実は、これは、**真っ赤な嘘**なのです。次章以降に進む前に、より正確に学習する必要があります。

まず、変数を少し大まかに説明すると、次のようになります。

重要 **変数**は、**オブジェクト**を**参照**するものであって、オブジェクトに**結び付けられた**名前にすぎない。

この説明では、分からないでしょう。インタラクティブシェルで確認します。

例 5-1　オブジェクトの識別番号

```
>>> n = 17
>>> id(17)
140711199888704      ← 17の識別番号
>>> id(n)
140711199888704      ← n の識別番号（17の識別番号と同じ）
```

▶ 注意：表示される値は、環境などによって異なります。これ以降も同様です。

変数 n に 17 を代入した後で、組込み関数である **id 関数**を2回呼び出しています。その id 関数は、オブジェクトに固有の値である**識別番号**（*identity*）を返却する関数です。

▶ 異なるオブジェクトに同じ識別番号が与えられることはありません。

Python の代入では、**Fig.5-1** a のような、値のコピーは行われません。

図 b に示すように、まず値 17 の int 型オブジェクトが存在していて、そのオブジェクトを参照するように、n という名前を**結び付ける**（*bind*）だけです。

整数リテラル 17 の識別番号と、変数 n の識別番号が同一になる理由が分かりました。

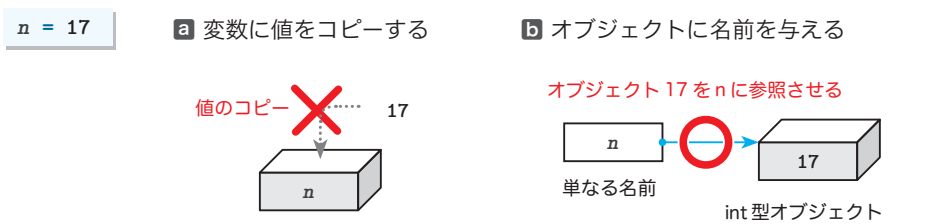

Fig.5-1　変数への値の代入

さて、変数には、格納ずみの値とは異なる型の値を代入できることを第 1 章で学習しました。どうなっているのか、確認しましょう。

例 5-2　変数に異なる型の値を代入

```
>>> n = 5
>>> id(n)
140711199888732
>>> n = 'ABC'
>>> id(n)
140711199888764    ← 識別番号が変化している
```

文字列の代入後に n の識別番号が変わっています。

Fig.5-2 に示すように、変数 n の参照先が、int 型の 5 から、str 型の 'ABC' へと更新されたのです。

当然ながら、int 型オブジェクト 5 自体は、型も値も変わっていません。

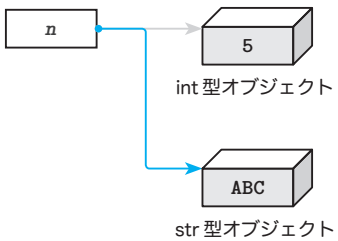

Fig.5-2　代入による参照先の更新

重要 変数に対する**代入**を行っても、値のコピーは行われない。参照先が代入されるだけである。

次は、p.70 で学習した《2 値の交換》を行います。

例 5-3　二つの変数の値を交換

```
>>> a = 5
>>> b = 7
>>> id(a), id(b)
(140711199888544, 140711199888608)
>>> a, b = b, a                    ← aとbを交換
>>> id(a), id(b)
(140711199888608, 140711199888544)
```

交換後に、変数 a と b の識別番号が入れかわっています。

Fig.5-3 に示すように、交換されているのは、変数の値ではなく、変数の参照先です。

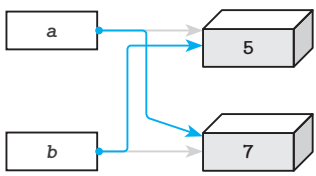

Fig.5-3　代入による参照先の交換

*

Python が“変数ありき”ではなくて、“オブジェクトありき”であることの意味が、少しずつ分かってきました。

さて、各オブジェクトは、**記憶域**（*storage*）すなわち**メモリ**を占有します。そのため、識別番号は、記憶域上の位置として表されるのが基本です。

重要 Python では、すべてが**オブジェクト**である。記憶域の一部を占有するオブジェクトは、**同一性＝識別番号**（*identity*）、**型**（*type*）、**値**（*value*）などの属性をもつ。

▶　識別番号は、カタカナで『アイデンティティ』です。「他のオブジェクトとは異なる識別番号をもっていて、他とは必ず区別できる」といったニュアンスです。

　オブジェクトの識別番号を調べるのは id 関数ですが、オブジェクトの**型**を調べるのは、第 1 章で学習した **type 関数**です。