

## 第3章 変数とデータのしくみ

本章では、プログラミングにおいて多用する「変数」と、データの取り扱いについて、Python, C++, Rust, TypeScript を用いて学ぶ。  
また、同時に、プログラミングとは切っても切りはなせない「型」や「値」についても取り扱う。

## 3.1 変数と定数

### 1 変数

みなさんは、変数と聞いて何を思いうかべるだろうか。

多くの人は、やはり、数学で散々苦しめられた用いられる  $x$  や  $y$  を思いうかべるだろう。

しかし、プログラムにおける 変数 (Variable) とは、値を保持しておく箱のようなものである。この箱の中身は、(一般に) 必要に応じて自由に取り出したり入れかえたりすることができる。

変数は、通常、宣言 (declare) と初期化 (initialize) をする必要がある。宣言は、値を入れる箱を新たに用意して名前をつけることであり、初期化はその箱の中に初期値を入れることである。一部の言語では初期化を省略できるが、その場合には内部的には「空である」という状態 (null) で保存される。(null はいろいろと問題があるが、これは後ほど取り扱う)

また、変数に値を入れることを代入 (Assignment) といい、もともと値が代入されている変数に再度値を代入することを再代入 (Re-Assignment) という。

### 2 定数

一方で、変数の「値を名前をつけた箱に入れる」という性質は、「よくつかう値に名前をつける」こととほぼ同義であり、このために変数が使われる (すなわち、再代入をはじめとする書き換えを行わない) ことがある。

このようなケースでは、途中で意図せず値が書き換わることがあり、十分注意しなければならない。

そのような場合に、書き換えのできない、ただ単純に値に名前をつけるものがあり、これを定数 (Constant) という。

特に重要な定数については、`CONSTANT_CASE` のように、定数名すべてを大文字として、複合語はアンダーバーで仕切る形で命名することがある。このような形式を `CONSTANT_CASE` といい、このような命名のルールを命名規則という。このほかの命名規則については後述する。

### 3 スコープ

では、変数や定数は、どこで定義してもよいのだろうか。

プログラムは複数ファイルに分割することができるが、そのファイルに書いたすべての変数が有効だと、どこかで名前が重複しそうなものである。（これで実際に重複することを名前衝突 (Name Conflict または Name Collision) という。）ただし、これはまた別の問題である。

名前衝突そのものの解決策は、C/C++やC#の名前空間 (Namespace) などがある。名前空間は、変数などを区切って、別々の名前空間では同じ名前を用いても衝突しないというものである。

どちらかというところ、問題は、一度使った変数がいつまでも残りつづけて、意図せず変更されて繰り返し処理の中で処理がおかしくなったり、メモリ使用量が増大したりするリスクが考えられることである。

では、これにはどのように対処するのだろうか。

変数は、プログラム全体で使えるものと、その関数の中でのみ使えるものがある。それぞれグローバル変数 (Global Variable)、ローカル変数 (Local Variable) という。また、その変数が使える範囲をスコープ (Scope) といい、これにもまたグローバルとローカルがある。

また、言語によっては、「{ }」で囲まれている範囲をスコープといい、変数が有効な範囲をライフタイム (Life Time) という。

このしくみをうまく使って、グローバル変数を可能な限り減らすことで、先述のようなリスクを抑えることができる。

また、言語によっては、グローバル変数が定義できない場合がある。この場合は、関数の呼び出し時に、変数を引数に渡して、関数の戻り値を変数に代入することによって対処する。