第 2 章 変数とデータ

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．

2.1 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，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．

2.2 データ型

1 型

では，変数の型と聞いたら何を思いうかべるだろうか．
「アルファベットとか数字とか？」と思った方，ご名答．言語によって少々異なるが，考えかたとしてはほぼ正解である．

型 (Type)とは，**プログラム上で取り扱われる値が，どのようなもので，どのような範囲で変動し，どの程度のメモリを確保しなければならないかを示すもの**である．大抵の言語に存在する有名な型をいくつか列挙する．

型	値	言語等
int	整数値	C/C++, C#, Java, Go など
float	小数値	同上
bool / boolean	真偽値(True or False)	上記 + TypeScript, Rust など
number	数値	TypeScript など
char	文字	C/C++, Java, C# など
string / str	文字列	TypeScript, C#, Rust, Go など
date	日付	TypeScript, C++, Go など
time	時刻，時間	同上

また，変数を宣言する際に，型を併記することを**型注釈 (Type Annotation)**という．

2 静的/動的型付け言語

さて，この「型」だが，内部的にはすべての言語がもつものであるが，一部の言語ではプログラム中に型を書かない．このような言語は，実行時に値の型が動的に決まることから**動的型付け言語** (Dynamic Typed Language)という．逆に，プログラム中に型注釈などの形で型をひとつひとつ定義する形の言語のことを**静的型付け言語** (Static Typed Language)という．

静的型付け言語と動的型付け言語のどちらがよいかという論争は，エディタ戦争と同等またはそれ以上に深刻なテーマであるため，本書ではその結論を示さない．ただし，一般に，静的型付け言語のほうが安全なソフトウェアが構築しやすいといわれている．

なお，昨今のトレンドとして，動的型付け言語に静的型付けを部分的に取りいれたスーパーセットを作成しようというものがあり，その代表格が TypeScript である．**スーパーセット (SuperSet)とは，ある言語をベースに，すべての構成要素を保持しつつ拡張したもの**のことである．このため，TypeScript には，型を指定せず，どのような値でも動的にメモリを確保して収容できる型「any」が存在する．ただし，any は本質的には素の JavaScript の変数と同じため，静的型付けによるエラーの防止ができず，any を多用するならば TypeScript ではなく JavaScript を使うべきである，という意見も散見される．ちなみに，TypeScript においては，型を明示しなければ，型を推論するのではなく，エラーを出すわけでもなく，any 型として扱う．このことを「暗黙の any」という．JavaScript の代表的なリンターである「ESLint」や「Biome」では，この「暗黙の any」をエラーにするオプションがあるほどである．

では，ここで，Python (動的型付け)，Rust，C++，TypeScript (静的型付け) によるコードを比較してみよう．
Rust はデフォルトでは変数への再代入はできないことに注意されたい．

なお，実行結果はコメントアウトで示してある．
Python におけるコメントアウトは #，その他 3 言語におけるコメントアウトは//である．

Python

```python
a = 2
b = 4
print(a + b) # 6
c = a + b
c = c + 4
print(c) # 10
```

Rust

```rust
let a: i8 = 2;
let b: i8 = 4;
println!("{}", a + b); // 6
let mut c: i8 = a + b;
c = c + 4;
println!("{}", c); // 10
```

C++

```cpp
int a = 2;
int b = 4;
cout << a + b << endl; // 6
int c = a + b;
c = c + 4;
cout << c << endl; // 10
```

TypeScript

```typescript
let a: number = 2;
let b: number = 4;
console.log(a + b); // 6
let c: number = a + b;
c = c + 4;
console.log(c); // 10
```