はじめに

プログラムを学ぶにあたって、多くの人はその難しさに根を上げるだろう。プログラミング言語は数多くあり、Python・Java・C 言語など、プログラムを学ぶ初歩の人にとっては、何を学んでいいかということ自体分からないものだ。

だから、私のような初心者の為に、プログラム自体の分かり易い捉え方というものが必要である。特定の言語には依存せずに、プログラムの原理やら考え方やら、そのようなものを学ぶことが出来ればプログラムというものが分かってくると考えている。

初心者の人の為のものと言っても、そこにはプログラムの「本質」というものが現れているはずだ。なぜならば、ある一つのプログラミング言語を理解したならば、他のプログラミング言語を理解するのは、かなり楽になるはずだ。

プログラミングの考え方というものは、Python・Java・C 言語の各言語は、ほぼ共通しているものであり、単にその考え方を表現する表記法が異なっているだけだからだ。そのことを説明する為に、Python・Java・C 言語の各プログラミング言語の初心者向けのテキストから問題を抜粋した。その問題を Python で解き、プログラミングに必要な考えや制御構造を理解したつもりだ。その考えや制御構造が後に説明する変数であったり、データとの関係性・条件分岐・繰り返し処理(ループ)である。どの問題を解いていても変数は出てくるし、その変数に関連して、関数・条件分岐・繰り返し処理(ループ)が各言語の問題から出て来た。その大まかな原理を理解する為に、各言語の問題を集め、解いていったのだ。だから、以下に出てくるプログラミング構造の考えに関しては、Python からの考えが主になっている。

プログラミングに共通する手法というものが理解出来ていれば、別の言語を勉強するときにも、プログラミングに関しては全くの初心者ではない。理解した言語で同じ事を新しい言語ではどのように書くのか、という表現の違いを学んでいけばいいのである。

私はプログラミングを理解する為に何故 Python で学んだのか。その理由は、Python は他のプログラミング言語と違い、文法規則が少なく、理解し易い形式であるからだ。他のプログラミング言語のように、ごちゃごちゃとしていない。初歩の入門教材の中では、学び易いものだったからである。

第 1 章　プログラムとは何か

1-1　プログラム

コンピュータは、プログラムに書かれていることを忠実に実行するものだ。自分がして欲しいことをプログラムに書いておくと、コンピュータに実行させることが出来る。このようなプログラムは、私たちが自分で作ることが出来るのだ。

プログラムを作るには、コンピュータとプログラミング言語があればいい。これは、現在のプログラムが、コンピュータ内部に登録されるスタイルだからである。

一般にプログラムを作ると聞いたら難しく考えるのではないか。私はプログラムを作ると聞くと、パソコンの画面に訳が分からない単語を打ち込む、そのような難しいものだと考えていた。しかし、分かり易く考えれば、プログラムを作るということは、コンピュータに、ある作業をするように「命令を書き込む」ということだろう。一言で言えば、命令するのだ。そうすれば、コンピュータは忠実にその命令を実行する。プ

ログラムを作る、それはコンピュータに命令を与え、その命令を自動的に実行するように仕組んでおくものだと考えてみれば、分かり易いのではないか。しかし、プログラムを作るときは、コンピュータに実行させる処理の流れをどのように命令文に置き換えるかということが問題なのである。

1-2　プログラムの構成要素

　コンピュータを使って何らかの作業をするには、その作業に合うプログラムが必要である。私達が普段使う言葉に日本語・英語と様々あるように、プログラムする際に、使われるプログラミング言語にも種類がある。例えば、Python・Java・C 言語・Perl などである。

　プログラミング言語も「言語」なので、私達が普段使う言葉と構成が似ている。日本語などの言語は、文字が集まって単語になり、単語が集まって句になり、句が集まって文になり、文が集まって段落になり、段落が集まって一つの文章になる。各プログラミング言語の構成もこれと似ている。下記の通りである。

：普通の単語・・・文字→単語→句→段落→章

：プログラミング言語・・・単語→式→文→ブロック→プログラム

日本語などの言語の単語には、様々な品詞があり、句の作り方にも様々あり、文にも種類がある。同じように、プログラミング言語の構成要素にも種類がある。下記の通りである。

：単語

　：変数・・・データに付けられた名前。

　：定数・・・直接のデータ。数値・文字列など。

　：演算子・・・計算の為の四則演算子や、代入の為の代入演算子、比

較の為の比較演算子など。

：関数名・・・関数を呼び出す為の名前。必ず後ろに括弧を付ける。

：複文を作る構文・・・if 文や for 文・while 文など。

：式・・・変数などを演算子で結び付けたもの。あるいは関数の呼び出し。

：文

：単文(一行の文)

：宣言文・・・変数の名前を宣言する文。

：初期化の為の代入文・・・変数を最初に使うときに、変数宣言と代入を一緒に行う文。

：代入文・・・右辺で演算子を使って計算したり、関数に計算させたりした結果の値を左辺に代入する文。

：print 文・・・画面に表示する文。

：return 文・・・元の所に戻って、値を返す文。

：break 文・・・繰り返しを打ち切る文。

：import 文・・・別のプログラムを呼び込む為の文。

：複文(複数の行からの文)

：条件分岐

：繰り返し処理

：ブロック・・・複文を作るときや、関数の定義で、複数の文を一まとめにして括弧でくくったもの。

：プログラム・・・ブロックも一つの文とみなされているので、プログラムも、文の連なりから出来ている。

しかし、何もないプログラムの世界には、ごく単純な種類の「物」しかない。それが「文字列」と「数値」である。これらのプログラムの世

界に存在する「物」をオブジェクトと呼ぶ。それが「文字列オブジェクト」と「数値オブジェクト」であり、これらを組み合わせ、単文が作られている。基本的にプログラムにはこの単文の積み重ねで、命令を与えるのだ。

1-3　オブジェクト指向

　オブジェクトというものは、プログラムの世界に存在する「物」と上で述べた。では、オブジェクト指向とは何なのか。普段から、オブジェクトという言葉もオブジェクト指向という言葉も耳にしない言葉だろう。オブジェクトとは、プログラムの世界に存在する「物」のことであり、指向とは、「決まった方向に向かうこと」という意味だ。つまり、オブジェクト指向とは、「何々」よりも「物」の方に近づきたいという考え方である。「何々」というのは、プログラミングにおいては、「処理の流れ」のことだ。したがって、パソコンプログラミング入門以前(p152)によれば、プログラミングにおけるオブジェクト指向とは、
：「処理の流れ」よりも「物」の方に近づきたいとする考え方
となっている。

　私達の社会をオブジェクト指向の考え方で例えると、一人一人の人間をオブジェクトと捉えることが出来る。独立した個々の人間が集まって社会を築く。私達はそれぞれが自分の役割を持っていて、一人で出来ない事は会話を通してコミュニケーションを行い、仲間と協力して成し遂げようとするものである。

　つまり、一つ一つのオブジェクトは独立して存在していて、それぞれが固有の役割を持っている。オブジェクト指向のプログラムは、関連するデータ(会社の例で言えば、仕事に使うもの)と、そのデータ(一人で出

来ない事は、仲間と協力するという誰もが行う共通手順)に対する手続きがパックになって、オブジェクトとして管理される。そして、オブジェクトがいくつか集まってプログラムを構成しているのである。これらをまとめると、パソコンプログラミング入門以前(p153)によれば、オブジェクトは、

：オブジェクトに関連するデータと、そのデータに対する手続きで成り立っている

である。

　また、オブジェクト指向の考え方に沿ってプログラムを作ることを、オブジェクト指向プログラミングと呼ぶ。このプログラミングは構造化プログラミング(プログラムを機能ごとに分けて部品を作り、その部品を組み立てて、一つのプログラムを作るというやり方)をもとに発展させたプログラミング手法である。

第 2 章　プログラムの基本

2-1　データ

　プログラムに限らず、コンピュータを使っているとデータという言葉が頻繁に使われるものだ。データという言葉は、誰もが使っているにも関わらず、実は本当の意味を理解している人は非常に少ない用語である。元々データという単語は、「資料や事実」という意味がある。一方、コンピュータの分野でデータという言葉が使われるときは、コンピュータが処理する情報を指す。Python で学ぶプログラム作法(p40)では、データの定義とは下記のようになっている。

：そこから結論を導き出すことが出来る何らかの事実または数値・情報

この定義だけではデータが何かということは、大して理解できないと思

う。しかし、データを説明する土台にはなっているはずだ。もう少し分かり易くする為に、プログラミング用語としてのデータがどのように使われるかを説明する。データとは、プログラムが扱う材料である。なので、材料であるデータが無い状態ではプログラムは何も役立つ処理を行うことはできない。

データをプログラムで上手く処理する為には、いろいろな工夫が必要である。プログラムでは様々な方法でデータを扱うが、その方法は多くの場合、データの種類(データ型)に依存する。全ての情報を 0 と 1 だけで扱わなければならないコンピュータにとって、データ型は重要な「ものさし」なのである。各データ型には、それに対して実行することの出来るいくつもの操作が関連付けられている。例えば、数値型のデータであれば、加算したり、減算したりの計算が出来る。加算や減算が数値型のデータに実行出来る操作なのである。

データには様々な型が存在する。私が集めた問題(Python でプログラミングを理解する為に、いろいろな初心者向けのテキストから問題を抜き取った)では、主に数値と文字列がそうだった。

数値は、整数と、少数を含めた実数に分かれる。整数では int を使い、実数では float を使う。文字列は、引用符で囲んで示さなければならない。引用符で囲んでいれば、”あ”のような一文字でも文字列と呼ぶ。文字列を示す引用符には、単一引用符(')と二重引用符(“)がある。これはどちらを使ってもよい。

データを定義することを通じてプログラマは、そのデータを何の為に使うかということを改めて認識でき、より適切なデータ型を選ぶことが出来るのである。

2-2　プログラムを学ぶ為のプログラム

それではまず、プログラムを学ぶ為(慣れる為)に極々小さなプログラムを作って動かしてみる。プログラムを学ぶにあたって大事な事は「作って動かすこと」だと思う。

それでは、簡単な計算プログラムを作る。「(4 + 6) ÷ 2」という簡単な計算をし、その結果を表示する。この計算式は、プログラムの中でも同じように書くことが出来る。ただし、半角の記号を使い「(4 + 6) / 2」と表示する。四則演算のルールは算数と同じであり、「4 + 6 / 2」ならば「6 / 2」が先に計算され、「(4 + 6) / 2」なら「(4 + 6)」が先に計算される。このように簡単な電卓としても使用出来るのだ。

コンピュータは、簡単な数値の計算だけでなく、様々な計算を行う。これらをまとめて演算という。コンピュータで行われる処理は、極端にいうと全てが演算である。だからと言って、全てが計算というわけではない。

演算に使う記号を演算子と言う。一般に演算子と聞くと、「+」や「-」のような記号を思い浮かべると思うが、プログラミング言語によっては「add」のようなキーワードを使って演算させることも出来る。つまり、「A + B」とも書くことも出来るし、「add B to A」とも書くことが出来るのだ。このように演算子には、記号だけでなく言葉(キーワード)も含まれるのである。

一連の命令を順序通りに実行するプログラムが、最も基本的なプログラムである。それが順次実行のプログラムだ。その中でも最も単純なものが、一つのコマンドだけからなるプログラムである。1-2 で出て来た単文がそうだ。このような順次実行の中では print 文が最も簡単なものだ。下記は print 文のサンプルコードである。

print “Hello”

このように、print 文では書く。この例の場合は、H・e・l・l・o という一連の文字を表示する。この print 文を使えば、以前に紹介した計算式も簡単に表示が出来る。以降に書くサンプルコードでは、Python プロンプト(>>>)に対して入力するテキストを示す。

```
>>> print (4 + 6) / 2
```

この結果 5 と表示(Python では数字は数値と認識され、それに演算子を続けると計算が実行)される。

前にも述べた通り、上のサンプルコードを括弧無しで入力すると、上の例とは全く違う 7 という計算結果が得られてしまう。これは、コンピュータが加算と減算よりも乗算と除算を優先して計算したからだ。数学の世界では、これは当たり前のことなのだが、実際にプログラミングすると、このような結果になるとは予想していないかもしれない。

どのプログラミング言語にも、算術演算を評価する際の評価順序を決定するその言語独自の規則(「演算子の優先順位」という)が定められている。しかし、演算子の優先順位などは、うっかり間違える場合があるかもしれないし、他の人がプログラムを見た場合に勘違いすることもあるかもしれない。こうしたプログラムの書き間違いや読み間違いを防ぐには、括弧を使って演算の優先度を明らかに示すことが大切なのである。

また、算術演算を行う場合、普通の数学と同じように、0 で割るゼロ除算などの「してはいけない計算」はしてはならない。自分ではそうしなかったつもりでも、結果的にしてしまうという場合がある。例えば「/ 0」と書かなくても、除数(割る数)に指定した変数に、たまたま 0 が入っていればゼロ除算が起こってしまう。プログラムは、そういう偶然を含め、いろいろな状況を想定して作る必要があるのだ。

2-3　変数とデータ

私が解いてきた問題でも、変数を使わない問題は無い、というくらい変数はよく使われるものだ。最も重要な単語であると言っても、過言ではない。

プログラムとは、変数をどう使うかによって展開していくものである。データを担っているものが変数であり、それが無ければ有意義なものには成り得ない。つまり、変数の扱いに慣れていないとプログラムを作るのは難しいと思うし、逆に変数を中心にプログラムを作っていくと、そこまで難しくはないだろう。

変数とは、計算プログラムでも使用した 4 や 6(数値の他に”Hello”などの文字列)といったデータを保存しておく格納庫の役割を果たすものである。プログラムの中で使うデータは、変数という形でプログラマが自分で定義するのだ。変数とは文字通り「変わる数」であり、variable つまり「変化するもの」なのである。

プログラムの中で変数が必要になったら、必ず定義しなければならない。これは、値を代入したり、計算を行ったりする前に行う。変数の定義とは、変数の名前・データ型・サイズを宣言(プログラミング言語によっては、データ型を指定すると必然的にサイズが決まるものもある)することだ。

変数の使い方を簡単なサンプルコードを用いて説明する。

```
>>> x = 4・・・①
    y = 6・・・②
    Average = (x + y) / 2・・・③
    print = Average・・・④
```

まず、①の「x = 4」で、変数 x に 4 という値をセットする。このことを

「代入」と言う。以降、新たな値が代入されるまで、変数 x は 4 という値を持つことになる。同じように、②の「y = 6」で、変数 y に 6 という値が代入される。③では、変数 x と変数 y を使った計算結果を、変数 Average に代入している。変数 x には 4、変数 y には 6 が代入されているので、変数 Average には「(4 + 6) / 2」の計算結果の 5 が代入されているというわけである。そして、④で、その結果を表示しているのだ。

また、変数の名前は、何に使われる変数かというのを、想像し易い名前にしたほうが分かり易い。サンプルコードでは x、y、Average という名前を使っているが、Average とは「平均」を意味している。サンプルコードは平均値を求めるプログラムなので、これが適切だろう。

私が問題を解いていて思ったことは、変数宣言文と、変数の初期化の為の代入文との違いだった。この二つの文の形式は、

① 変数宣言文・・・型　変数;

② 変数の初期化の為の代入文・・・型　変数　=　直接のデータ、式

である。②の式には、いろいろな種類があり、関数の呼び出しであったり、演算子を使った式であったりする。

しかし、Python では、変数の型は書かなくてもいい。これは、Python に型がないというのではなく、値を代入することによって自動的に型が決まるのだ。下記のように、

```
>>> x = 1
```

と書けば、初期化の為の代入文を行うことによって、意識しなくとも変数宣言がなされているのだ。

しかし、Python 以外のプログラミング言語では、変数宣言と変数の初期化の為の代入文は別々のものだ。例えば C 言語では、

```
>>> int x;
```

これで変数 x の型を宣言しなければならない。Python にはこの手間がないのである。しかし、Python では意識しなくとも変数宣言がなされていると言っても、変数を宣言するというのは重要な事であるので、変数宣言を意識するというのは大切だ。このことは、Python で学んでいれば忘れがちなことである。

変数に値をセットすることを「代入」と上で述べた(それ以降も「代入」という言葉は使ってきた)が、プログラミングでは、代入も一つの演算と考える。代入に使用する演算子を代入演算子と言う。代入演算子は、ほとんどの言語で、「=」を使うものである。他に、代入する為の命令語として、「move」などがあるが、実際のプログラムでは「=」をよく使う。そして、私も最初はよく混同していたのだが、「=」を等号(イコール)と混同することには注意する。

```
>>> x = x + 1
```

と書かれていれば、「x」と「x + 1」がイコールなのではなく、変数 x の値に 1 を足した結果を、改めて変数 x に代入するという意味なのである。

プログラムは、

①演算子を使って計算する

②関数を呼び出す

③関数の呼び出しを使って演算子で計算して、その結果をもう一度変数に代入する

という代入文によって進展していく。

①の演算子を使って計算するというのは、前に述べたことである。②の関数を呼び出すというのは、名前の後に括弧がついたものである。括弧の中には、関数に引き渡す引数を書く。そして、その結果は何らかの変数に代入するか、直接 print 文で表示する。ここで、関数の呼び出しの

簡単なサンプルコードを書く。私が問題を解いていたときに、よく出て来たものであり、これはキーボードから直接打ち込むものである。

>>> name = raw_input(“あなたの名前を入力して下さい。”¥n)

と書くと、「あなたの名前を入力して下さい。」との表示があり、自分の名前を入力すると、その結果が変数 name に文字列として代入されるのだ。この後は、print 文で表示すればいい。また、この raw_input には文字列だけではなく数値も書くことが出来る。その際の方法は、

>>> old = int(raw_input(“年齢を入力して下さい。”¥n))

という関数呼び出しを使う。このように raw_input を int で囲むのである。int で囲めば文字列を数値に直して使うことが出来る。この他に、小数点数に直す float がある。この int や float を使えば、上の変数 old を使い、

>>> year = 2005 – old

のように、年を求める計算にも使えるのである。raw_input などの関数は、プログラミング言語自体が既に持っているものであり、それを使うことでゼロから自分で考えるよりも、効率的にプログラムを作ることが出来る。

　関数については、

①使う

②定義する

という二つの側面がある。

まず、関数を「使う」ということである。

：データ→関数→データ

というように、何らかのデータを関数に渡して、関数は何らかの処理をし、その結果を返す、という流れになっているのが基本形である。渡す

データを「引数」と言い、返されるデータを「戻り値」と言う。Python では、この「使う」という形は、

：戻り値を受け取る変数 = 関数(引数, 引数,・・・)

であり、上の raw_input がそうだ。

次に、関数を「定義する」ということである。関数にはプログラミング言語自体が既に持っているものだけではなく、自分で作ることが出来る。関数を作るときに考える注意点は、

①何をする為の関数か

②目的を達成する為に何が必要か

③結果として何を返すか

である。これらの注意点は関数を利用する(呼び出す)ときにも注意するべきことだ。目的に合う関数を使い、決められたものを関数側にきちんと引き渡し、結果を正しく受け取るということなのである。この「作る」という形は下記の通りだ。

：def 関数名(引数名, 引数名,・・・):

　　引数を使った一連の処理。

　　同じ高さにインデントする。

　　処理結果を返す場合は、return 値を書く。

インデントが def と同じ高さになったところで関数定義は終了する。

である。インデントで同じ高さに合わすというのは、Python だけであり、Java などは{}で囲む。これを使うときは、

：変数名 = 関数名(引数, 引数)

とする。では、def 関数を使ったサンプルコードを書く。

```
>>> x = int(sys.argv[1])・・・変数宣言
    y = int(sys.argv[2])
```

```
def kasan(a, b):・・・関数の定義
    return a + b
def gensan(a, b):
    return a – b
def jousan(a, b):
    return a * b
def josan(a, b):
    return a / b
print kasan(x, y)・・・プログラム本体
print gensan(x, y)
print jousan(x, y)
print josan(x, y)
```

これは、「変数名 = 関数名(引数, 引数)」なので、変数宣言をした変数 x と変数 y が引数 a と引数 b になっている。関数の定義で引数を使った一連の処理(この場合は、加算・減算・乗算・除算)を書き、return 値で処理結果を返している。後は、プログラム本体の print 文で、それぞれを表示しているのだ。また、Python では関数は使う前に定義しておく必要があり、Java などとは違うところである。

第 3 章　プログラムの基本制御構造

3-1　条件分岐

プログラムの基本制御構造は、順次実行・条件分岐・繰り返しである。順次実行とは、1-2 で紹介した単文がそうであり、一つの命令が終われば次の命令を実行するというものだ。

条件分岐とは、仮に 100 キロの荷物を載せているトラックが一本道を

走っている。そして、分かれ道に差し掛かる。そこで、荷物が 100 キロ以上なら右の道へ、100 キロ以下なら左の道へと、「もし、何々だったら何々する」というように、条件によって処理内容を決めることである。条件分岐には、どのプログラミング言語も if 文を使うことが多い。条件分岐の形は下記の通りだ。

```
>>> if　条件 1:
    条件 1 が成り立っているときの処理
    elif　条件 2:
    条件 2 が成り立っているときの処理
    else:
    いずれの条件も成り立たなかったときの処理
```

これは、if 文で条件を示し、その if 文の結果が真の場合はその結果が表示される。If 文の後ろには、真か偽のいずれかに評価される項目であれば、何を置いてもかまわない。偽の場合は elif 文の処理に入り、結果が if 文・elif 文のどちらでもないときは else の処理を実行するのだ。Java は条件は括弧で囲むが、Python は囲まない。インデントすればいい。

解いてきた条件分岐の問題では、raw_input でユーザーに質問を出し、それを条件によって答えてもらうという形で出て来ることが多かった。その形のサンプルコード下記の通りだ。条件分岐は真と偽の認識がきちんと確かめることが出来ていれば、何とかなるだろう。

```
>>> x = raw_input(“どの面積を求めますか。1.三角形 2.台形 3.円”¥n)
    if x == “1”:
        teihen = int(raw_input(“底辺を入力して下さい。”¥n))
        takasa = int(raw_input(“高さを入力して下さい。¥n))
        mennseki = teihen * takasa / 2
```

```
elif x == “2”:
    joutei = int(raw_input(“上底を入力して下さい。”¥n))
    katei = int(raw_input(“下底を入力して下さい。”¥n))
    takasa = int(raw_input(“高さを入力して下さい。”¥n))
    menseki = (joutei + katei) * takasa / 2
elif x == “3”:
    hanei = int(raw_input(“半径を入力して下さい。”¥n))
    menseki = hankei * hankei * 3.14
else:
    print (1～3 の番号を入力して下さい。)
print “面積は” menseki “です。”
```

このサンプルコードは、if 文で三角形の面積計算を行い、elif 文で台形の面積計算を行い、もう一つの elif 文で円の面積計算を行い、if 文・elif 文のどちらでもない場合は else の処理を行う。ここで使っている面積計算は、変数を使った順次実行である。そして、このサンプルコードは最初にユーザーに入力してもらうときに int で数値に直しているが、変数 menseki の計算のときに数値に直してもよい。また、このサンプルコードでは「==」が出て来た。これはイコール(等号)であり、これを比較演算子という。比較演算子は他に下記のものがある。

：a == b・・・a と b は等しい

：a > b・・・a は b より大きい

：a < b・・・a は b より小さい

：a >= b・・・a は b 以上

：a <= b・・・a は b 以下

比較演算子は、一定の条件に合うかどうか、二つ以上の変数の大小関係

はどうか、二つの値は等しいか等しくないか、などを比べて判定するときに使うものである。また、多くの言語でイコール(等号)は「==」だが、イコール(等号)に「=」を使う言語もあるので注意する。イコール(等号)の他に、比較演算子で使う演算子のサンプルコードは、

```
>>> if 12 > 10:
    print “12 は 10 より大きい。”
```

となる。この場合は、12 が 10 より大きいので print 文で表示されるが、この条件が成り立っていないときは勿論エラーになる。

3-2　繰り返し処理(ループ)

繰り返し処理は、条件分岐よりは難しいと思う。私が分からなかった事は、繰り返しというのは、「どこからどこまでが繰り返しなんだ、繰り返しの処理とはどこまで続いているんだ」と考えている内に訳が分からなくなるところだった。繰り返しの問題は、最大値を求める為・合計値を求める為に多く出て来た。場合によっては、条件分岐と混ぜて繰り返しを使う問題もあり、条件の中で繰り返すという場面もある。繰り返しにしても条件分岐にしても、変数と関数の関係が強い。Python で解いたからというわけではなく、各言語が共通している。

繰り返し処理とは、特定の変数の値を少しずつ増加させながら、同じコードを繰り返し実行するものである。一回の繰り返しごとに変数の値が変化し、それによって実行される内容や繰り返しのカウントが変わっていくのだ。

繰り返し処理には二つの構文がある。for 文・while 文である。どちらも繰り返しの処理ではあるのだが、その処理の作り方に違いがある。繰り返し処理を理解するにあたっては、全ての制御処理が最初の一行に集

約されているので、for 文を反復して覚えるのがいい。

まずは for 文の説明をする。while 文との大きな違いは、for 文は繰り返しの回数があらかじめ分かっていなければ使えないという制限があることである。また、Python の for 文は Java の for 文とは大きく異なる。Python での for 文の形は下記の通りだ。

：for 変数 in リスト:

　　繰り返しの処理。

この for 文はリストの中の一つ一つの項目を順番に処理していく為に使われている。ここで出て来たリストとは、同じ種類のデータが一列に繋がっているものとみなす構造のことである。複数の項目を丸括弧で囲ったものだ。上の形を使った簡単なサンプルコードを書いてみる。

```
>>> for i in range(5):
        print i
```

これを動かすとその結果は、

```
0
1
2
3
4
```

と表示される。繰り返しは最初の for で繰り返しの条件(いくつからいくつまで、変数の値を動かすか)を設定する。変数 i は、そのとき一つの値に固定される。range(回数)というのは、0 から回数分の番号を返す関数である。よって、range(5)は(0.1.2.3.4)というリストになる。それを 0 から順番に変数 i に入れていき、print 文で表示すると上の結果になるのだ。

リストは range という元々ある関数のものだけでなく、変数で作るこ

とも出来る。例えば、

```
>>> cities = (“東京”, “大阪”, “京都”, “神戸”, “横浜”)・・・①
    for i in cities:・・・②
        print i・・・③
```

①で cities という文字列の変数リストを作っている。後の処理は同じことであり、その結果は、

東京
大阪
京都
神戸
横浜

となる。

while 文は for 文と違い、繰り返しの回数が分かっていなくても使うことが出来る。while 文の内容は「条件式が真である間、その処理を繰り返す」ということなのである。この為、指定された条件が最初から偽であった場合は、実行されることはないのである。while 文の形は、Python の for 文と違い、他の言語とあまり変わりはなく、

```
：while (条件が成立している間):
        (この処理を行う)
```

のという形になる。

```
>>> i = 1・・・①
    while i <= 12:・・・②
        print “%d*12=%d” % (i, i*12)・・・③
        i = i + 1・・・④
```

これは、①で変数の初期化を行っている。②で変数 i は 12 以下という条

件指定を行う。③では、書式化文字「%」を使って文字列を書式化している。%の後ろの d という文字は、そこに置くデータが「10 進数」であることを示している。この文字列を表示すると、%d の部分が文字列の後ろの%記号に続く括弧に入れた値で置き換えられる。この場合は、「1×12」から「12×12」までを計算している。④の「i = i + 1」というのは変数 i に 1 をインクリメント(加算)することによって、while 文を終了させている。この命令文がないと「無限ループ」になる。

繰り返し処理は一言で言えば、「どこからどこまで繰り返すか」ということが分かっていれば出来るものである。言葉では簡単に説明出来るのだが、実際に繰り返し処理のプログラムを書くとなると分からない。これはどの構文でも言えることなのだが、やはり、各構文ごとに反復して練習して理解するしかないのである。

　また、繰り返しで注意すべき事は、

①条件判定に使われる変数の初期化

②条件が判定されるタイミング

③条件判定に使われる変数の増減

などだ。これらの注意点はプログラムを書くときに怠らないようにしたい。例えば、条件判定に使われる変数の初期値が、最初から条件を満たしてしまう値なら、処理が行われない可能性も出て来る。そして、上のサンプルコードでも出て来たように、よく忘れてしまうのが、条件判定に使われる変数をインクリメント(加算)またはデクリメント(減算)する命令文を書き忘れることである。この命令文を書き忘れた為に、条件が成立することなく、繰り返しから抜け出せない「無限ループ」になることもあるのだ。

終わりに

この論文は、プログラミングに関しては初心者も同然の私が書いたものであり、どこまで私が考えていたところまで近づけたかどうかは分からないし、どこまで分かり易くできたか不安はある。しかし、この Python を中心としたプログラミングの説明で少しは、プログラミングの基本的なところは分かってもらえたのではないかと思う。はじめの部分で話したが、言語は違ってもプログラミングの基本的な手法は同じである。プログラミングとは、パズルみたいなものであり、紹介してきた変数や関数、それに基本的な制御構造は、そのパズルの 1 ピースである。試行錯誤の後、組み合わせの関係でパズルは一つ一つがつながっていく。プログラミングも同じで、組み合わせで一つ一つが動いていくのだ。

今日の晩ご飯のメニューをカレーライスに決めたときにカレーライスを作るまでの手順は、店に行く→材料を決める→買う→家に帰る→ご飯を炊く→材料を仕込む→調理に移る。こうした手順を決めるものだろう。この手順を決めたものをフローチャートと言う。そして、そのフローチャートにもと基づいて調理した結果、カレーライスが完成するのである。プログラミングもこれと同じことで、頭の中でその処理の仕方(フローチャート)を考え、後はその中にパズルのピースを当てはめていくだけである。そう考えれば、数学にも似ているかもしれない。

しかし、これは口で言うのはとても簡単なものであるが、実際に行ってみると、とても難しいものだ。それは、まがりなりにもプログラミングを勉強してきた私自身がよく理解している。だから、プログラムの基本を理解しなければならないのだ。それを理解する為には、何回もの練習が必要だ。根気よく問題に取り組み、プログラムが上手く実行できなくても、気にせず、諦めずに続けることが大切なのである。