【ＤＲニュース・027】：プログラムはなぜ動くのか？どんな環境で動くのか？基礎知識

2016 年 12 月 05 日発信

今回は、プログラムの基礎知識が適切に説明されている日経ＢＰ社のプログラミング誌・日経ソフトウェアから、プログラムはなぜ動くのか？どんな環境が必要なのか？など、ポイントを抜粋してみました。また、プログラム言語の違い（できること・特徴・難易度）を参考にし、言語を学ぶ糧にしてください。

１．プログラムの基本

（１） 本来の仕組み

「プログラムのアイコンをダブルクリックすることでプログラムが動作する」などと、漠然と考えているようではダメです。

・・・**「メモリーにロードされたマシン語のプログラムが、ＣＰＵによって解釈・実行され、それによってコンピュータというシステム全体の制御やデータの演算が行われる」という**。

（２） プログラムとは　（本題に入る前のウォーミングアップ）

① **プログラムとは**、何ですか？・・・コンピュータが行うべき処理を順序立てて記述したもの
② プログラムの中には、何が含まれていますか？・・・**命令とデータ**
③ **マシン語**とは、何ですか？・・・コンピュータが解釈できる電気信号となったプログラム
④ 実行時のプログラムは、どこに格納されていますか？・・・**メモリー**（メイン・メモリー）
⑤ メモリーの**アドレス**とは？・・・メモリー上で命令やデータが格納されている場所を示す値
⑥ コンピュータの構成の中で、プログラムを解釈・実行する装置は何ですか？・・・**ＣＰＵ**

２．プログラマにとってのＣＰＵとは何か？

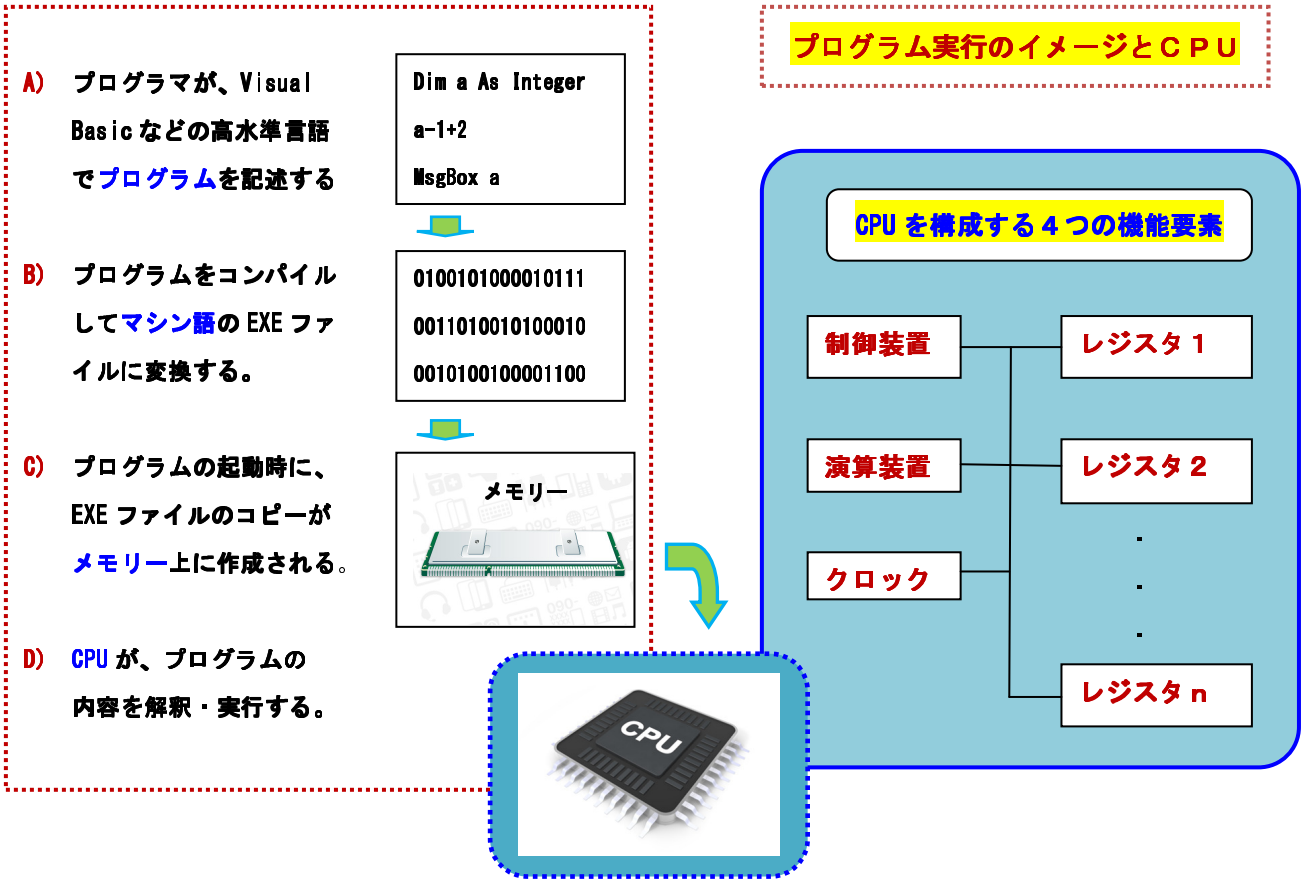
プログラムを解釈・実行する装置である**ＣＰＵの説明**から始めましょう。

ＣＰＵとは、Central Processing Unit（中央処理装置）の略称であり、コンピュータの頭脳となる装置です。**ＣＰＵの内部が数百万個～数億個のトランジスタで構成されています**。

しかし、このような知識では役に立ちません。**CPU がどんな働きをするかを理解することが必要です**。

（１） ＣＰＵの中身

① **CPU やメモリーの実体は、多くのトランジスタから構成された IC(Integrated Circuit)集積回路と呼ばれる電子部品です**。メイン・メモリー（**主記憶**）は読み書き可能なメモリー素子で構成され、１バイトずつにアドレス（番地）と呼ぶ番号がついて、**CPU はこのアドレスを使って、主記憶に格納された命令やデータを取り出したり、逆にデータを書き込んだりします**。



② **CPU の構造がわかると、プログラムが実行される仕組みが何となく見えてきませんか？**

プログラムが動き出すと、**クロック信号**に合わせて、**制御装置**がメモリーから**命令やデータ**を読み出します。**演算装置**でデータが演算され、その結果に応じて**制御装置**がコンピュータ全体を制御します。

③ **CPU はレジスタの集合体**

レジスタは、処理対象とする**命令やデータ**を格納する領域で、一種のメモリーのようなものです。種類によって異なるが、**１つの CPU の内部に２０～１００個ほどのレジスタがある。**

CPU の４つの要素の中で、プログラマが意識しなければならないものは、**レジスタだけです**。それは、**プログラムがレジスタを使ってデータを処理しているからです**。

レジスタに格納される値は、「**命令**」を表している場合と、「**データ**」を表している場合が、あります。**データには、演算に使われる値**と、**メモリーのアドレスを表す値**の２種類があります。値の種類によって、それを格納するレジスタの種類も異なります。

・・・・・・・・・ **さまざまな役割を持ったレジスタの集合体がＣＰＵなのです**。

④ レジスタの種類とその役割

レジスタの種類	個数	レジスタの役割
アキュムレータ	1	演算を行うデータを格納する。（記憶された値と別の値の間で演算が行われ、結果がまたそこに記憶されるもの。多様な演算命令に対応）
フラグ・レジスタ	1	演算処理後のＣＰＵの状態（ステータス）を格納する。（これらのフラグは条件分岐に主に用いられる）
プログラム・カウンタ	1	次に実行する命令が格納されたアドレスを格納する
ベース・レジスタ	n	データ用のメモリー領域の先頭アドレスを格納する
インデックス・レジスタ	n	ベース・レジスタからの相対アドレスを格納する
汎用レジスタ	n	任意のデータを格納する
命令レジスタ	n	命令そのものを格納する。（プログラマがプログラム・コードでこのレジスタの値を読み書きするのではなく、ＣＰＵが内部的に使用する）

（２） プログラムの流れを決めるプログラム・カウンタ

今度は、プログラムが記述された通りに、実行する（プログラムが流れる）仕組みの説明を行います。

① プログラムのメモリーへのコピーと実行の開始

Windows などの OS は、プログラムの起動が支持されると、ハードディスクに保存されているプログラムをメモリーにコピーします。下記の②の例ですと、アドレスの**０１００番地**が、このプログラムの開始位置です。起動されるとレジスタの一つの**プログラム・カウンタ**に**０１００を設定**します。CPU が一つの命令を実行するとプログラム・カウンタの値が自動的に一つ増加されて、**０１０１**となります。

CPU の制御装置は、プログラム・カウンタの値を参照して、メモリーから命令を読み出して実行します。すなわち、プログラム・カウンタがプログラムの流れを決めています。

② メモリーに配置されたプログラムの例　（加算結果を表示する）

アドレス	格納・値	メモリーの内容
０１００	命令	0105 番地の値をアキュムレータに格納する
０１０１	命令	0106 番地の値を汎用レジスタ１に格納する
０１０２	命令	アキュムレータの値と汎用レジスタ１の値を加算する
０１０３	命令	アキュムレータの値をディスプレイに表示する
０１０４	命令	プログラムを終了し、OS に戻る
０１０５	データ	１２３
０１０６	データ	４５６

（３） 条件分岐と繰り返しの仕組み

プログラムの流れには、「順次進行」、「条件分岐」、および「繰り返し」の３種類があります。

順次進行の場合は、プログラム・カウンタの値が一つずつ増加していくだけですが、プログラムの中に**条件分岐**や**繰り返し**がある場合は、それらのマシン語命令が、プログラム・カウンタの値を任意のアドレス（＋1 でない値）に設定することになります。

よって、前のアドレスに戻って同じ命令を繰り返したり、任意のアドレスにジャンプして分岐することが出来るのです。

CPU が何らかの演算を行うたびに、その結果に応じて、フラグ・レジスタの値が自動的に設定されます。**条件分岐では、ジャンプ命令の前に何らかの比較演算が行われます**。ジャンプ命令を実行するかどうかは、**CPU がフラグ・レジスタの値を参照することで、判断がされます**。

フラグ・レジスタは、直前に実行した演算の結果として、アキュムレータの値が、**負**、**ゼロ**、**正**のいずれかになったかを記録する役割を持つものです。**比較のための実行が行われると**、**CPU の演算装置は**、**内部で（こっそりと）「XXX－YYY」という減算を行うのです**。
・・・・**この演算の結果が正、ゼロ、負のいずれになったかが、フラグ・レジスタに記録されます**。
（**正**は XXX が YYY より**大きい**、**ゼロ**は**等しい**、**負**は XXX が YYY より**小さい**ことを表します）

（４） 関数の呼び出しの仕組み

関数の呼び出しは、関数内部の処理が完了したら、関数の呼び出し元（関数を呼び出した命令の次のアドレス）に処理の流れが戻ってこなければなりません。

この問題を解決するのが、「**コール命令**」と「**リターン命令**」です。関数の呼び出しは、コール命令によって行われます。

コール命令は、関数の入口のアドレスをプログラム・カウンタに設定する前に関数呼び出しの次に実行すべき命令のアドレスを「**スタック**」と呼ばれるメイン・メモリー上の領域に保存します。関数の処理が終了したら、関数の最後（出口）でリターン命令を実行します。**リターン命令は**、スタックに保存されたアドレスをプログラム・カウンタに設定する機能を持っています。

（５） ベースとインデックスで配列を実現

ベース・レジスタとインデックス・レジスタをペアで使うことで、メイン・メモリー上の特定のメモリー領域を区切って配列のように使うことが出来ます。

インデックス・レジスタの値は、高水準言語で記述されたプログラムにおける配列の添字（インデックス）に相当するものとなります。

配列のように特定のメモリー領域を区切って連続的に参照するためには、２つのレジスタを使った方が便利なのです。

実際のアドレス	＝	ベース・レジスタ	＋	インデックス・レジスタ
１０００００００		１０００００００		００００００００
１０００００００１		１０００００００		００００００００１
１０００００００２		１０００００００		００００００００２
・ ・		・ ・		・ ・
１０００ＦＦＦＦ		１０００００００		００００ＦＦＦＦ

配列の添字（ｲﾝﾃﾞｯｸｽ）に相当する値となる

（６） ＣＰＵに出来ることは、いたって単純！！

ＣＰＵが実行できるマシン語命令の主な種類と機能を挙げてみます。実際行っている機能は単純です。

マシン語命令の種類	機能
データ転送命令	レジスタとメモリー、メモリーとメモリー、レジスタと周辺装置の間でデータの読み書きをする
演算命令	アキュムレータで算術演算、論理演算、比較演算、及びシフト演算を行う
ジャンプ命令	条件分岐、繰り返し、及び無条件のジャンプを行う
コール／リターン命令	関数を呼び出す／呼び出し元に戻る

（７） 高水準言語＆マシン語とは？

高水準言語とは、人間の言葉（主に英語）に近い構文で記述できるプログラミング言語の総称です。
ＢＡＳＩＣ、Ｃ言語、Ｃ++、Ｊａｖａ、Ｐａｓｃａｌ、ＦＯＲＴＲＡＮ、ＣＯＢＯＬ、ＰＬ１などは、高水準言語です。

高水準言語のプログラムは、コンパイルしてマシン語に変換してから実行します。
マシン語は、ＣＰＵが直接解釈し実行できる電気信号となったプログラムのことです。
人間にわかるように、電気信号のＯＮ／ＯＦＦを、２進数の０／１で表すこともあります。

（８） クロック信号の周波数（しゅうはすう）

メガは１０の６乗；凄いです

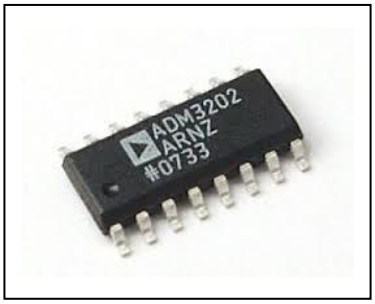
Pentium５００ＭＨｚならクロック信号の周波数が５００ＭＨｚ（ＭＨｚ＝１００万回／秒）です。
・・・・・・・・・ **周波数が大きいほどＣＰＵの動作は速くなります。**

３．データをＯＮ／ＯＦＦの２進数でイメージしよう

C 言語や BASIC などの高水準言語で書かれたプログラム・コードの中に示された数値、文字列、およびグラフィックなどの情報は、コンピュータの内部で、すべて２進数の値として取り扱われます。

（１） 情報を２進数で取り扱う理由

コンピュータの内部は、IC（Integrated Circuit（集積回路）の略称）と呼ばれる電子部品で構成されています。CPU(マイクロプロセッサ)やメモリーも IC の一種です。



IC は、黒いボディーの両側に数本～数百本のピンが付いたムカデ（百足）のような形状やボディーの裏にピンが並んだ剣山のような形状をしています。

IC が持つすべてのピンは、直流電圧０Ｖか、＋５Ｖのいずれかの状態になっています。　（**０Ｖ：ＯＦＦ、＋５Ｖ：ＯＮ**）
つまり、IC のピン１本では、２つの状態しか表せないのです。

（２） ２進数の桁数とバイト

２進数の桁数は、８桁、１６桁、３２桁、・・・のように**８の倍数とする**のが一般的です。
これは、**コンピュータで取り扱う情報の単位が、８桁の２進数を基本としているからです**。

８桁の２進数のことを「バイト（byte)」と呼びます。バイトは情報の基本単位となります。ビットが最小単位で、バイトが基本単位です。メモリーやディスクには、**バイト単位でデータが格納され、バイト単位でデータが読み書きされます**。ビット単位では、読み書きできません。

８桁（１バイト）で表現するときは「**００１００１１１**」とし、１６桁（２バイト）で表現するときは「**００００００００　００１００１１１**」とします。

Pentium などの３２ビット・マクロプロセッサは、情報を入出力するピンを３２本持っています。Pentium は、一度に３２桁（３２ビット＝４バイト）の２進数の情報を処理できるのです。

２進数の演算には、シフト演算、算術演算、論理演算や**補数**などをマスターする必要があります。
好奇心や向上心のある方は、別途、演算の基本を調査して、マスターしてください。

CPU と２進数を説明するには～コンピュータに１＋２という計算をさせたいなら、足し算命令と、１と２という数字を与える必要があります⇒IC のピンに電気で命令や数値を与えるのです。

４．コンピュータが小数計算を間違える理由

プログラムの実行結果として正しい値が得られないこともあります。その一例が少数点の計算です。コンピュータが小数点数を取り扱う仕組みを基礎知識としてつかんでください。

（１） 少数点数を２進数で表すには

① ウォーミングアップとして、1011.0011 と小数点を持った２進数を 10 進数に変換してみよう

１０１１．００１１　　（２進数の少数点数から１０進数へ変換）

整数部　　少数部

１×２の３乗＝１×８＝８　　０×２の－１乗＝０×０．５＝０

０×２の２乗＝０×４＝０　　０×２の－２乗＝０×０．２５＝０

１×２の１乗＝１×２＝２　　１×２の－３乗＝１×０．１２５＝０．１２５

１×２の０乗＝１×１＝１．　　１×２の－４乗＝１×０．０６２５＝０．０６２５

重みを掛けて足すと　　１１．１８７５

２進数の表現	対応する１０進数
０	０
０．０００１	０．０６２５
０．００１	０．１２５
０．００１１	０．１８７５
０．０１	０．２５
０．０１０１	０．３１２５
０．０１１	０．３７５
０．０１１１	０．４３７５
０．１	０．５
・・	・・
０．１１１１	０．９３７５

② 小数点以下４桁の２進数で表せる値

2 進数としては連続しているが、10 進数では飛び飛びの値になります。10 進数の 0（ゼロ）の次が 0.0625 になっています。この間にある小数点は、小数点以下４桁の 2 進数では表せません。
2 進数の小数点以下の桁数を増やせば、対応する 10 進数の個数を増やすことが出来ますが、いくら桁数を増やしても **0.1** にはできないのです。

10 進数の 0.1 を 2 進数に変換すると、0.00011001100 ・・・・・（以下 1100 の繰り返し）という「**循環小数**（じゅんかんしょうすう）」になります。　**したがって、0.1 を 100 回加えても 10 になりません。**

③ 浮動少数点数（ふどうしょうすうてんすう）とは

コンピューターの数値演算方式のひとつ。浮動小数点とは、コンピューター内部での小数を扱う方法のひとつで、**数値を仮数部（かすうぶ）と指数部（しすうぶ）と呼ばれる部分に分けて表します**。コンピューターでは、データの種類ごとにデータのサイズが固定されているため、計算結果が決められた桁数を超えてしまうと表現できなくなます。**そこで、桁数の大きな数値を扱うために浮動小数点方式が用いられてます**。（IEEE754 規格；浮動少数点数の代表的な規格、制度の違いを考慮して、32 バージョン（単制度）と 64 バージョン（倍制度）を用意している）
・・・符号部（ふごうぶ）、指数部、仮数部からなる浮動小数点数には、正規表現やイクセス表現がある。

（２） コンピュータの計算間違いを回避するには

回避策は、「小数点数を整数に置き換えて計算する」ことです。

少数点数の計算では間違えることがあるコンピュータでも、整数の計算では（扱える値の範囲を超えなければ）決して間違えることはありません。そこで計算するときだけ一時的に整数を使い、計算結果を小数点数で表示します。

たとえば、0.1 を 100 回加える計算なら、0.1 を 10 倍して、 1 を 100 回加える計算に置き換えて、その結果を 10 で割って示せば良いことになります。

（３） 2 進数と 16 進数

2 進数は、ビット単位でデータを表す場合に便利ですが、桁数が非常に多くなってしまうので、見た目にわかりづらいのが難点です。そのため実際のプログラミングでは、**2 進数の代わりに 16 進数がよく使われます**。

Visual Basic のプログラムでは、数値の先頭に「**&H**」を付けることで 16 進数を表せます。
C 言語のプログラムでは、数値の先頭に「**0x**」を付けます。

2 進数の 4 桁は、ちょうど 16 進数の 1 桁に相当します。
16 進数を使うことで、2 進数の数値の桁数を１／４ にすることが出来ます。

５． メモリーを工夫して使う

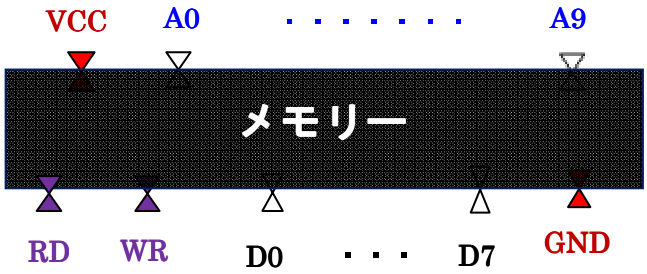
コンピュータは、データを処理する機械であり、その処理手順やデータ構造を示したものがプログラムにほかなりません。処理の対象となるデータは、メモリーやディスクに格納されています。

したがて、プログラマたるもの、メモリーやディスクを自由自在に使いこなせなければなりません。そのために、メモリーやディスクの構造を物理的（ハードウェア的）にも論理的（ソフトウェア的）にもイメージできるようになることです。

（１） メモリーの物理的な仕組みはシンプルです

メモリーの実体は、メモリーIC と呼ばれる電子部品です。メモリーIC には、DRAM, SRAM, ROM など、さまざまな種類のものがありますが、外部から見た基本的な仕組みはどれも同じです。
（RAM は、読み書き可能なメモリーで、記憶保持動作が常に必要な DRAM と不必要な SRAM に大別されます。　ROM は、読み出し専用のメモリーです）

メモリーICには、電源、アドレス信号、データ信号、および制御信号を入出力するための多くのピン（ICの足）があり、アドレス（番地）を指定して、データを読み書きするようになっています。



【メモリーICのピン配置】
- VCC、GND：電源
- A0~A9　：アドレス信号
- D0~D7　：データ信号
- RD、WR　：制御信号

このメモリーICの中に、どれだけのデータが格納できるのだろうか？、データ信号ピンがD0～D7の８本あるので、一度に８ビット（＝１バイト）のデータを入出力できることがわかります。

アドレス信号ピンがA0～A9の１０本あるので、0000000000～1111111111＝１０２４通りの指定可。１０２４＝１Ｋだから、１ＫＢの容量を持ったメモリーICです。

（２） メモリーの論理的なイメージはビルディング

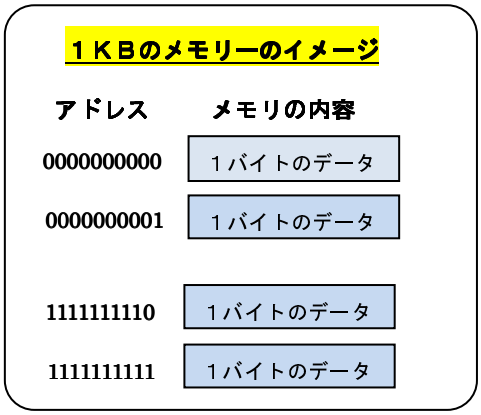
１フロアに１バイトのデータが格納された１０２４階建てのビルディング。フロアの番号がアドレスを示す。

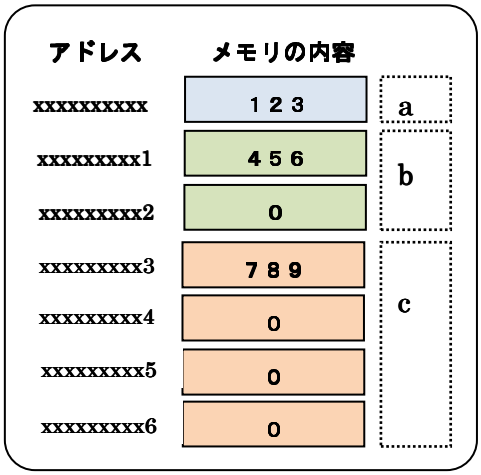
なお、プログラマから見たメモリーには、物理的なメモリに存在しない概念があります。それは「型」です。

３つの変数の型は、１バイト型を表すchar，２バイト型を表すshort，４バイト型を表すlongとなっている。

【３つの変数にデータ書き込みのC言語：例】

```
//  変数の宣言          // データの書き込み
char    a;              a =123;
short   b;              b =456 ;
long    c;              c =789 ;
```





変数の「型」により、占有するメモリーのサイズが異なります。

（３） メモリを工夫して使うための基本は配列(はいれつ)です

配列を使うことで、効率的なプログラミングが可能となります。繰り返しを行なうループ処理の中で配列を使えば、短いコードで、配列の要素を順番に読み出したり、順番に書き込んだりすることが出来ます。

多くのプログラムでは、**さまざまな工夫を凝らして配列を使います**。
「**C 言語でのポインタ**」、「**スタック**」、「**キュー**」、「**リスト**」、および「**２分探索木（バイナリー・サーチ・ツリー）**」という配列の変形手法などがあります。

- **ポインタとは**、データの値そのものではなく、データが格納されているメモリーのアドレスを持つ変数のことで、ポインタを使うことで任意のアドレスを指定してデータの読み書きが可能となります。（**C 言語**の大きな特徴であり、難解なものだとも言われています）
- **スタックとキューは**、どちらもアドレスやインデックスを指定せずに配列の要素を読み書きできるものです。**スタックでは、「LIFO（後入れ先だし）方式」、キューでは、「FIFO(先入れ先だし)方式」**によって、書き込み読み出しの順序を決めておくことで、アドレスやインデックスの使用を不要としています。
  さらに、**四角いメモリを丸く（リング状態で）使う形態のリング・バッファもあります**。
- **リストは**、要素の追加や削除が容易です。
- **２分検索木(にぶんけんさくき)は**、リストをさらに工夫し、配列に要素を追加するときに、その大小関係を考慮して左右二つの方向に分岐するリストとしたもので、効率的にデータを探せます。

６． メモリーとディスクの親密(しんみつ)な関係(かんけい)

メモリーとディスクの機能は、プログラムの命令やデータを記憶するという点で、同じだと言えます。
コンピュータの５大要素の中でも、メモリーとディスクは、どちらも記憶装置に分類されます。
ただし、電気的(でんきてき)な記憶を行なうメモリーと、磁気的(じきてき)な記憶を行うディスクには、違いがあります。

同じ記憶容量で比べれば、メモリーは高速だが高価であり、ディスクは低速だが安価です。
メモリーはメイン・メモリー（CPU で実行されるプログラムの命令やデータを記憶するメモリー）を指し、ディスクは主にハード・ディスクを指します。

（１） 本題に入る前の出題　（ウォーミングアップ）

ハンガリー出身のアメリカの数学者、
ノイマンが論理面を担当！！

① ノイマン型コンピュータを考案した人は、誰でしょう：**フォン・ノイマン**
数学者のフォン・ノイマンは、記憶装置にプログラムを格納し、逐次実行するという仕組みは、「**ストアド・プログラム方式（プログラム内蔵方式）**」と呼ばれ、それが考案された時代には実に画期的なことでした。それ以前は、コンピュータの配線を変えることで動作していたからです。**現在でも基本的にはフォン・ノイマンが考案した、この方式を採用しています**。

② **メモリーを使ってディスクのアクセス速度を向上させる仕組みは？**：**ディスク・キャッシュ**
ディスク･キャッシュは、一度ディスクから読み出されたデータをメモリーに保存しておき、再度同じデータが読み出されるときに、ディスクではなくメモリーから高速に読み出せるようにするものです。

③ **ディスクの一部を仮想的にメモリーとして使う仕組みを何と呼ぶか？**：**仮想記憶**
仮想記憶によって、メモリー容量の少ないコンピュータであっても、サイズの大きなプログラムを実行できます。

（2） プログラムは、メモリーに読み出さないと実行出来ない

ディスクに記憶されたプログラムが、メモリーにロードされたから実行されます。ディスクに記憶されたままのプログラムを、そのまま実行することはできません。

なぜなら、プログラムの内容を解釈して実行する CPU は、内部にあるプログラム・カウンタでメモリーのアドレスを指定し、そこからプログラムを読み出すようになっているからです。

（3） ディスクをメモリーの一部として使う「仮想記憶装置（かそうきおくそうち）」

仮想記憶とは、ディスクの一部を仮想的にメモリーとして使うものです。
ディスク・キャッシュが仮想的なディスク（実体はメモリー）であったのに対し、仮想記憶は仮想的なメモリー（実体はディスク）です。

仮想記憶によって､メモリーが不足している状態でもプログラムの実行が可能となります｡しかし、CPU はメモリーにロードされたプログラムだけしか実行ができません。

したがって、仮想記憶を実現するためには、実際のメモリーの内容とディスク上の仮想メモリーの内容を部分的に置き換えながら（スワップしながら）プログラムを実行しなければなりません。
（ゲーム中に「Loading…」と出てくるとき、CD に記憶されたソフトを部分的にメモリーに再配置）

仮想記憶の手法には、「**ページング方式**」と「**セグメント方式**」があります。

- **ページング方式**
  Windows が採用しているのは、ページング方式です。実行されるプログラムを、その構造に関係なく一定の大きさの「ページ」に分割し、**ページ単位**でメモリーとディスク間の置き換えを行なう方式です。
- **セグメント方式**
  実行されるプログラムを、関数やデータの集合など意味のある単位にまとめたセグメントに分割し、**セグメント単位**でメモリーとディスク間の置き換えを行なう方式です。

７．プログラムはどんな環境で動くのか

たとえば、Windows 用のプログラムを、そのまま Macintosh で動作させることは、基本的にはできないのです。　ＯＳが違うから～～ではＯＳの違いとは何でしょうか？

（１）　動作環境　：　ＯＳ　＆　ハードウェア

一般的に、アプリケーションの動作環境は、**Windows（ＯＳ）、AT互換機（ハードウェア）**のように、**ＯＳ（オペレーティング・システム）とコンピュータ本体（ハードウェア）の種類で示されます**。

ＯＳは、ハードウェアの種類ごとに専用のものが必要です。また、同じＯＳであっても**マイクロプロセッサの種類が異なれば、同じアプリケーションは動作しません**。

（２）　ＡＰＩはＯＳごとに違う

今度はＯＳの種類に目を向けてみましょう。多くのハードウェアでは、インストールできるＯＳの種類にいくつかの選択肢があります。たとえば、ＡＴ互換機なら、Windows だけでなく、UNIX、BTRON など、様々なＯＳが利用できます。
もちろん、アプリケーションは、ＯＳの種類ごとに専用のものを作らなければなりません。

アプリケーションから OS への命令の仕方を定めたものが、「ＡＰＩ（Application Programming Interface」です。

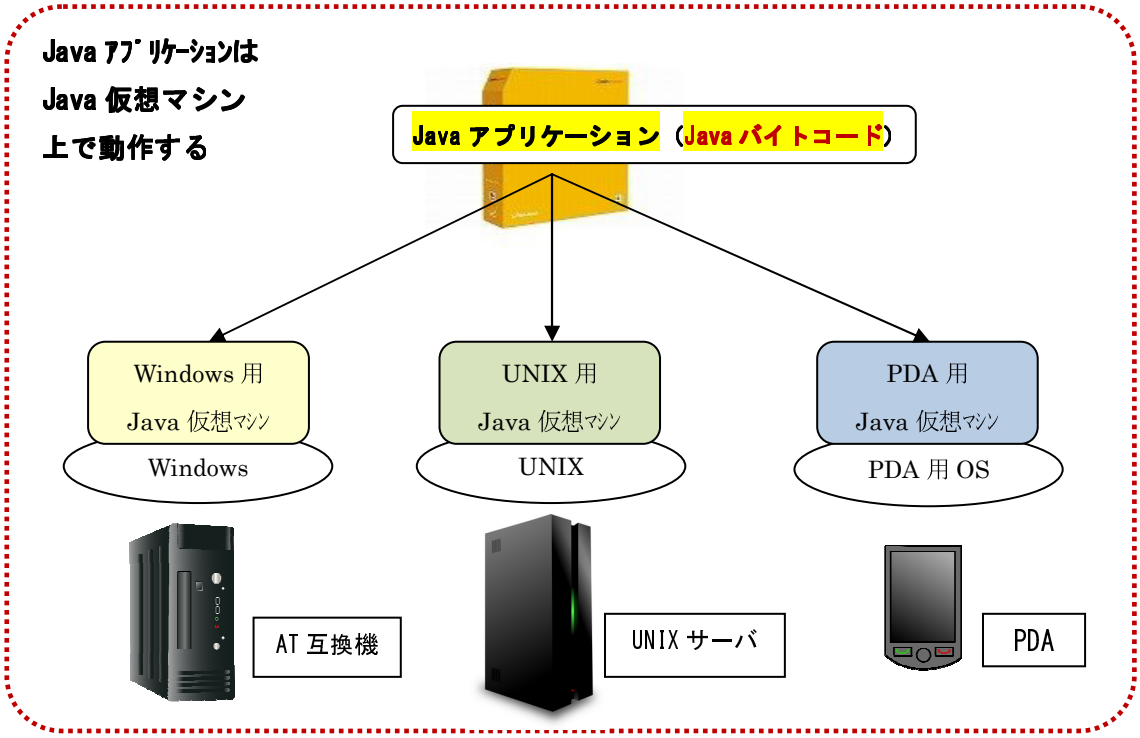
ＯＳごとにＡＰＩが異なるので、同じアプリケーションを他のＯＳ用に作り直す場合には、アプリケーションが API を利用している部分を書換えなければなりません。API として提供されるのは、キー入力、マウス入力、画面出力、ファイル入出力などの周辺装置と入出力を行う機能です。

（３）　どこでも同じ実行環境を提供する Java 仮想マシン

Java には、２つの側面があります。１つはプログラミング言語としての Java であり、もう一つはプログラムの実行環境としての Java です。

Java は、他のプログラミング言語と同様に、Java の文法で記述されたソースコードをコンパイルしたものを実行します。ただし、コンパイル後に生成されるのは、特定のマイクロプロセッサ用の「ネイティブ・コード」ではなく、「バイトコード」と呼ばれるものです。

バイトコードの実行環境を「Java 仮想マシン（JavaＶＭ）」と呼びます。
Java 仮想マシンは、Java バイトコードを逐次ネイティブ・コードに変換しながら実行します。



たとえば、サン・マイクロシステムズが公開している AT 互換機用の Java 開発ツールと Java 仮想マシンを使った場合、ユーザーが作成した**ソースコード**（Sample.Java）を**コンパイラ**（Javac.exe）が**バイトコード**（Sample.class）に**変換します**。**バイトコード**を **JavaVM**（Java.exe）が **x86 マイクロプロセッサ用の**ネガティブ・コードに**変換し**、**x86 マイクロプロセッサが処理を実行します**。

- **バイトコードとは**、プログラムのデータ形式・実行形式の一つ。
  プログラムを実行するランタイム（インタープリタ）での実行速度を向上するために、数値や命令のバイナリ化を行ったもの。
  バイトコードにすることで、**一つのバイナリ（ファイル）でアーキテクチャの異なるコンピュータでも実行できる。**個人がスクリプト言語を作る際に、インタープリタと比べ高速な実行が可能で、機械語コンパイラほどの手間をかけずに済むという利点があります。

- **ネイティブ・コードとは**、コンピュータに理解できる言語（マシン語）で記述されたプログラム。数値の羅列として表現されるため、そのままの形で人間が理解するのは困難。
  通常は、人間がプログラミング言語を使って作成したソースコードを、コンパイラなどの変換ソフトウェアを使ってネイティブ・コードに変換する。
  マイクロプロッセサが解釈・実行できる形式のプログラムがネイティブ・コードである。

- **JavaVM**（Java Virtual Machine ; **Java 仮想マシン**）とは、Java バイトコードとして定義された命令セットを、Java バイトコードが実行された OS や CPU アーキテクチャーといった実行環境に応じたネイティブ・コードに変換して実行する仕組み、またはそのソフトウェアのことである。

Java 仮想マシンは、スタック型の仮想マシンであり、JavaAPI や、複数のツールと一緒に Java Runtime Environment（JRE）として配布されている。ガベージコレクションや Hotspot などは仮想マシンによって実行される。

- **ガーベジコレクションとは**、アプリケーションソフトを実行するために確保されたメモリ領域のうち、そのアプリケーションの終了後も確保されたままになっている領域をクリアして、ほかのアプリケーションソフトで使えるようにすることである。

- **HotSpot（ホットスポット）とは**、オラクル（サン・マイクロシステムズ）が提供している Java 仮想マシンで使われている高速化のための技術の名称。デスクトップ向け・サーバ向け・携帯電話向け(Java ME)がある。性能を改善するためにジャストインタイムコンパイル方式、Adaptive optimization（適応性のある最適化）などの技術を使っている。

8．ソース・ファイルから実行可能ファイルができるまで

プログラムを書いたら、コンパイルを行って実行可能ファイルを作成します。それを行ってくれるのがコンパイラです。ここでは、コンパイラの働きに注目して、プログラムの作成から実行までの流れを追ってみましょう。

（1） ソースコードを翻訳するのがコンパイラ

C 言語などの高水準言語で記述されたソースコードをネイティブ・コードに翻訳する機能を持ったプログラムのことを「**コンパイラ**」と呼びます。**コンパイラは**、ソースコードを記述するプログラミング言語の種類に応じて、専用のものが必要になります。

C 言語のコンパイラを C コンパイラと呼び、BASIC なら BASIC コンパイラです。

コンパイラの中に、ソースコードとネイティブ・コードの対応表があるようなイメージを持ってください。ただし、実際には対応表だけではネイティブ・コードを生成できません。

読み込んだソースコードの字句解析、構文解析、意味解析などを経て、ネイティブ・コードを生成するコンパイラがほとんどです。

コンパイラ自体もプログラムの一種なので、動作環境というものがあります。
たとえば、Windows 用の C コンパイラや UNIX 用の C コンパイラなどがあります。

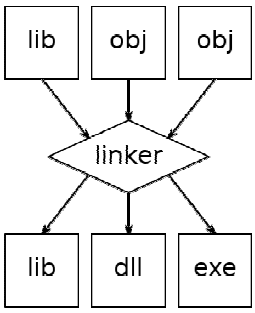
（２） コンパイルだけでは実行可能ファイルが得られない

ソースコードの翻訳結果としてコンパイラが生成するのは、ネイティブ・コードのファイルです。ただし、このファイルを、そのままでは実行できません。

実行可能な EXE ファイルを得るためには、コンパイルに続けて「リンク」という処理が必要です。

複数のオブジェクト･ファイルを結合して１つの EXE ファイルを生成する処理が「リンク」であり、リンクを行うプログラムのことをリンカ（リンケージ・エディッタや結合プログラムとも呼ばれています）と呼びます。　（ｵﾌﾞｼﾞｪｸﾄ・ﾌｧｲﾙのオブジェクトは、コンパイラの生成物という意味）、

- **リンケージエディタ**（リンカ(linker)、連係編集ﾌﾟﾛｸﾞﾗﾑとも）**とは、**機械語のプログラムの断片を結合し実行可能なプログラムを作成するプログラムのことである。
  例として、C 言語では、ソースファイルをコンパイルするとオブジェクトファイルが生成される。それに他のオブジェクトファイルやライブラリを結合して１つのプログラムが完成する。この結合（リンクという）の際にリンケージエディタが使われる。



- **リンクの処理の概要　（静的リンク）**
  - ① **ファイルの連結**・・・分割コンパイルによってオブジェクトファイルが複数ある場合、連結して単体のファイルとする。
  - ② **ﾗｲﾌﾞﾗﾘ内の連結**・・・ライブラリの関数を呼び出している場合は、ライブラリ内のオブジェクトファイルも連結される。
  - ③ **OS 固有ｺｰﾄﾞの追加**・・OS によって決められたスタートアップコードやメタ情報を追加する。
  - ④ **アドレスの解決**・・・ソース内で使用した関数や変数の名前は、最終的にはすべてメモリアドレスに置き換えられる。
    リロケータブルなプログラムでは、これはプログラム先頭等からの相対アドレスとなるが、プログラム全体のサイズがわかるまで相対アドレスを決定する事はできないため、**この処理はリンク時に実施される**。

（３） 実行可能ファイルの実行に必要なことは？

今度は、**EXE** ファイルが実行される仕組みは、**EXE** ファイルは、単独のファイルとしてハード・ディスクに記録されています。　（**EXE** ファイルの EXE とは、Executabule（実行可能）を表します）

エクスプローラで **EXE** ファイルをダブルクリックすれば、**EXE** ファイルの内容がメモリーにロードされて実行されます。

９．ＯＳとアプリケーションの関係

プログラムを実行してコンピュータを使う目的の多くは、業務の効率化です。
Word のようなワープロ・ソフトや Excel などの表計算ソフトのように、特定の業務を効率化してくれるプログラムのことを「アプリケーション」と総称します。

ＯＳの上で動作するプログラムのことをアプリケーションまたは応用プログラムと言います。

アプリケーションの実行環境となるＯＳは、市販されているものをそのまま使えば良いのですが、ＯＳを無視してアプリケーションを作成することはできません。

（１）ＯＳの存在を意識しよう

アプリケーションを作成するプログラマに意識して欲しいのは、**ハードウェアではなく、ＯＳの機能を利用するアプリケーションを作成しているということです**。

プログラマには、ハードウェアの基本的な知識を持つことも必要ですが、**ＯＳが存在する以上**、**ハードウェアを直接制御するプログラム・コードを記述することは滅多にありません**。

ＯＳによって、プログラマがハードウェアを意識しなくて済むようになったので、プログラマの人口というものも増えました。ただし、**一人前のプログラマになるためには、基本的なハードウェアのことが解っていて、それがＯＳによって抽象化されているため、効率的にプログラミングができるという事実を知っておくべきです**。

（２）ＯＳと高水準言語がハードウェアを抽象化してくれる

ＯＳが動作している環境で実行されるアプリケーションは、ハードウェアを直接制御せずに、ＯＳを介して間接的にハードウェアを制御するからです。

さらに、高水準言語を使うことによって、ハードウェアを完全に抽象化出来るのです。

Open はファイルをオープンする関数、**Print** はファイルに文字列を書き込む関数、**Close** はファイルをクローズする関数です。

ファイルをオープンすることで、OS が自動的にファイルの読み書きを管理するためのメモリー領域を確保します。その管理番号がファイル管理番号となります。プログラマはファイルの読み書きを管理するためのメモリー領域の内容や、その実態がどこにあるのかを、意識する必要はありません。

１０．ハードウェアを制御する方法

マイクロプロセッサ（CPU）を制御するためには、コンパイラやアセンブラが生成するネイティブ・コードをメイン・メモリーにロードして実行すれば良いことがわかりましたが、マクロプロセッサやメイン・メモリー以外のハードウェアを制御するには、どうすれば良いのだろうか？

（１） アプリケーションはハードウェアと無関係か？

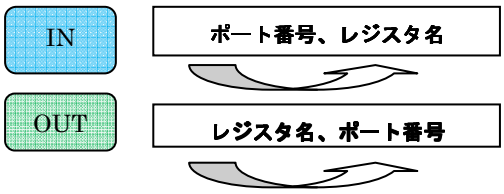
Visual Basic や **Visual C++**などの高水準言語を使って、Windows アプリケーションを作成する場合に、ハードウェアを直接制御する命令は滅多にありません。ハードウェアの制御は、Windows(OS)が一手に引き受けているからです。
ただし、間接的に制御する手段は、OS が提供するシステム・コールを使えば良いのです。

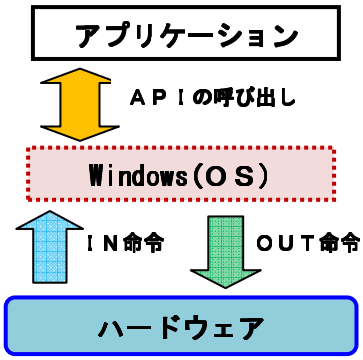
Windows の場合は、システム・コールのことを **API**(Application Proguramming Interface)と呼びます。個々の **API** は、アプリケーションから呼び出し関数になっています。

（２） ハードウェアとの入出力を支えるのは IN 命令と OUT 命令

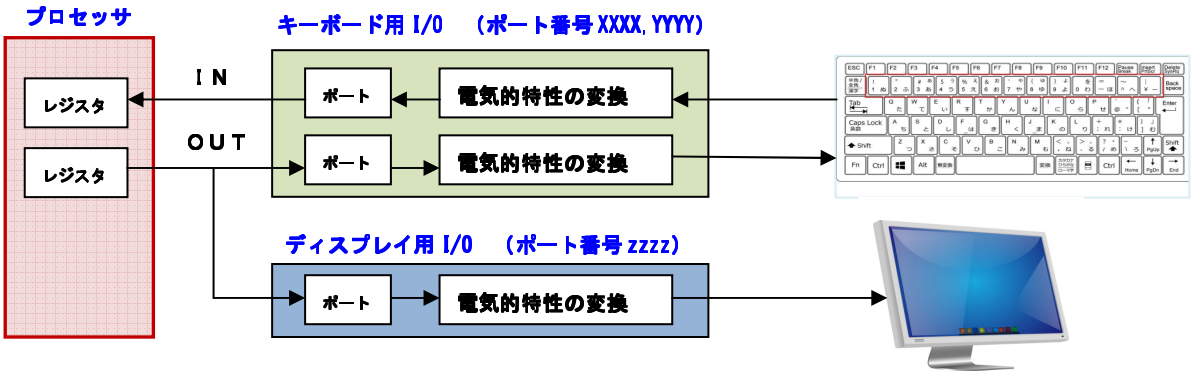
Windows（OS）がハードウェアを制御するために使うのは、入出力命令です

IN 命令は、指定したポート番号のポートからデータを入力し、それをプロセッサ内部のレジスタに格納。**OUT 命令は**、レジスタに格納されているデータを指定したポート番号のポートに出力。





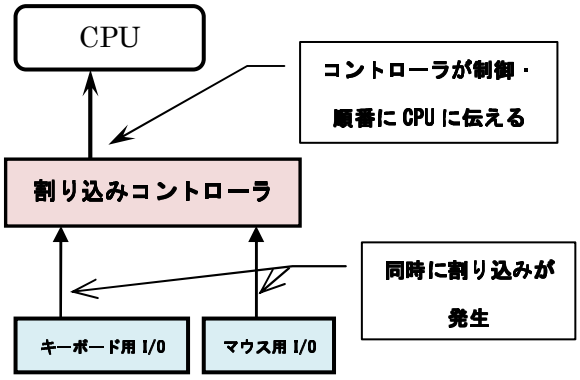
ポートを橋渡しにして、プロセッサと周辺装置がデータの受け渡しを行います。



（３） 周辺装置が割り込みを要求する

IQR(Interrupt Request アイアールキュー)とは、割り込みを要求という意味です。

IRQ は、現在実行中のプログラムを一旦停止して他のプログラムに実行を移すために必要な仕組みです。割り込み処理はハードウェアの制御で重要な役割を担っています。割り込み処理がないと、処理が円滑に進まない場合があるからです。



これは、オフィスで種類作成をしているときに、電話がかかってきた状況に似ています。
もしも、割り込み機能がないと種類作成が終わるまで電話応答ができません。これでは不便ですね。
だから、割り込み処理に価値があるのです。電話対応後に元の書類作成に戻るように制御します。

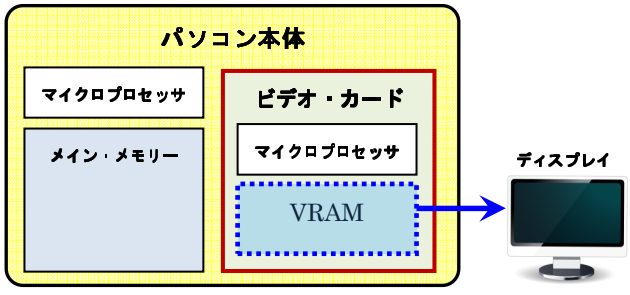
（４） 割り込みでリアルタイムな処理が実現する

メイン・プログラムの実行中に、どの程度の頻度で割り込みが発生するのだろうか？　実はほとんどの周辺装置（キーボード・マウス・ディスプレイ・プリンタなど）から頻繁に発します。
その理由は、周辺装置から入力されたデータをリアルタイムで処理してもらうためです。

割り込み処理が無いと、メイン・プログラムの中で、周辺装置からのデータの入力があったかどうかを、すべての周辺装置に対して、繰り返し調べ続けなければなりません。ロスタイムが大きい。

（５） 文字やグラフィックスが表示される仕組み

ディスプレイに表示される情報を常に記録しているメモリーが存在しています。
このメモリーのことを「**VRAM (VideoRAM)**」と呼びます。プログラムで **VRAM** にデータを書き込めば、それがディスプレイに表示されます。



現在のパソコンでは、ビデオ・カードやグラフィックス・ボードと呼ばれる専用のハードウェア上に、メイン・メモリーと独立した専用の **VRAM** と**専用のマイクロプロセッサ**が装備されているのが一般的です。これは、色数の多いグラフィックスを描画するのが当たり前の Windows で、数ＭＢの VRAM が必要とされ、高速に描画するためには、専用のマイクロプロセッサが必要になるからです。

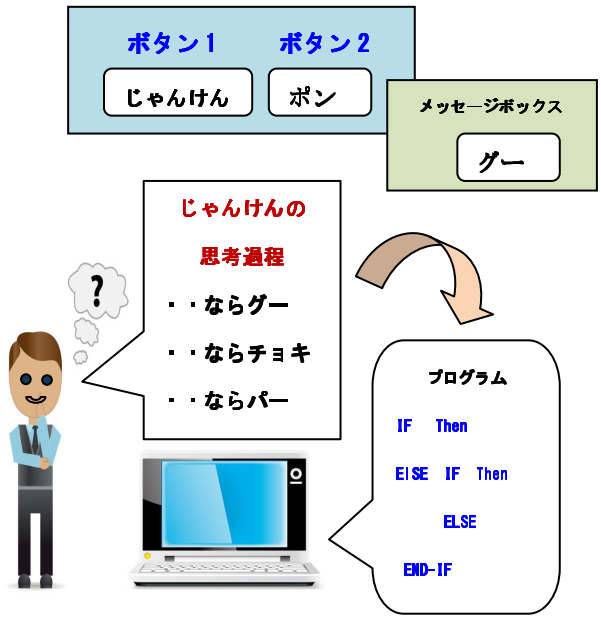
１１．コンピュータに「考え」させるためには

（１） 人間の考え方をプログラムで表す

コンピュータに物を考えさせるには、
どうしたらよいのだろか？

人間は、「じゃんけん、ポン」といった掛け声を挙げながらじゃんけんをします。

「**じゃんけん**」と言ったタイミングでは、グー、チョキ、パーのどれを出すか決めていなく、「**ポン**」と言ったタイミングで手を決めているでしょう～これをプログラムで表すことになります。



「**じゃんけん**」**ボタン**をクリックするとタイマーが起動し、0.1 秒間隔で、グー、チョキ、パーのどれを出すかが順番に変化していきます。「**ポン**」**ボタン**をクリックしたタイミングで、その時に考えていた**手をメッセージボックスに表示**します。

（２） 思考パターンをプログラムで表す

勘や思い付き、癖もありますが、人間には、思考パターンというものがあります。
思考パターンは、考え方のリズムであると考えられ、これは、「グー、グー、パー、チョキ」や「チョキ、グー、パー、パー」のように、短い音楽のフレーズのようなパターンが心の中にあって、それに基づいて手を決める場合です。

このような思考パターンをプログラムで表すには、「**リスト**」を使うのが一般的です。
リストとは、複数のデータが列のように前後につながったデータ構造です。
単純なリストは、連続した配列として簡単に表現できます。

パターンを**ランダム**に選んで、それぞれの**リスト**に格納された 4 つの手を、前後の勝敗にかかわらずに順番にだします。

【リストの配列の例】

パターン１	パターン２
P(1, 1)=" グー "	P(2, 1)=" チョキ "
P(1, 2)=" グー "	P(2, 2)=" グー "
P(1, 3)=" パー "	P(2, 3)=" パー "
P(1, 4)=" チョキ "	P(2, 4)=" パー "

人間の判断や推論といった能力は記号処理（きごうしょり）に依存していると考えられており、このような記号処理を研究する分野を、「人工知能（AI）の始まり」と呼びます。
現在人工知能は、さらに進んで機械学習やディープラーニングなど AI の認識が進化しています。

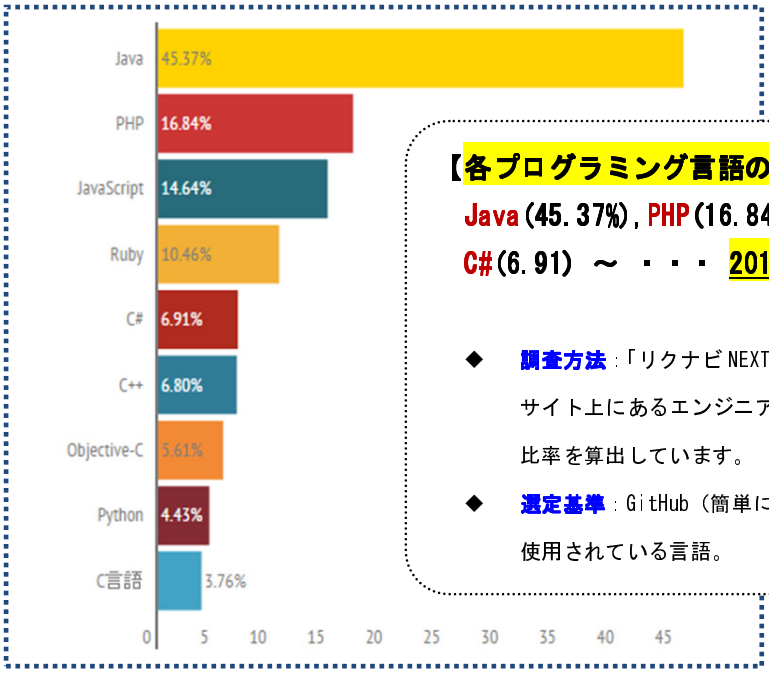
１２．プログラミング言語の違い

プログラムの本質は、今後コンピュータが進化発展を続けても、決して変わることはありません。
でも、世の中には「プログラミング言語」と呼ばれるものは実は何百種類（200 以上）もあります。

・・・・・　では、プログラミング言語の違いってなんなのだろう？

（１）プログラミング言語の求人ニーズ

プログラミングの学習を始めるにあたっての悩みが「一体どのプログラミング言語を学べばよいか」ということです。そこで今回はプロのエンジニア 9 名がおすすめする言語をご紹介させていただきます。求人ニーズの多い順に並べてみました。



【各プログラミング言語の求人数比率】
Java（45.37%）, PHP（16.84%）, JavaScript（14.64%）, Ruby（10.46）
C#（6.91）～ ・・・ 2015 年 10 月 20 日現在 の調査資料です。

- **調査方法**：「リクナビ NEXT」、「マイナビ転職」、「DODA」、「Green」の４つのサイト上にあるエンジニア求人 8308 件中、各プログラミング言語の求人数比率を算出しています。
- **選定基準**：GitHub（簡単に言えばソース共有できる Web サービス）上で良く使用されている言語。　※その他、求人数で多いのは、COBOL です。

（２）プログラミング言語の特徴と比較　＜できること＞、＜特徴＞、＜難易度＞

次に、プロのエンジニア 9 名がおすすめする言語の上位６言語に対して、「**できること**」「**特徴**」「**難易度**」をプロのエンジニアに解説してもらいました。

① Ｊａｖａ

A）＜できること＞

- 一言でいえば「なんでもできる」といった感じ。
- 自作のゲームだったり、ツールだったり、Web ページから android アプリまで制作可能。
- エミュレータや実機ですぐに動作確認ができるので感覚的に理解ができる。

B) ＜特徴＞

- オブジェクト指向と呼ばれる技術の代表で、色んな説明がされますが、実際に触ってみると「あれ、こんなもんでいいんだ」と意外とあっさり受け入れられると思います。
  しかもオブジェクト指向のメリットを最大限活用しなくとも、Java でのプログラミングは結構何でもできちゃいます。
- 汎用性は高い、というか歴史の長い言語なのでこんな使い方したいなぁと思ったら Web でちょっと調べるだけで方法が載っています。
- **型**の宣言等はしっかりやらないといけないので、自由度は PHP や Ruby 等に比べ低いですが、チームや会社で仕事するときにはこの考え方ができていないと仕事にならないです。

C) ＜難易度＞

- 難易度は厳密すぎず緩すぎずといったオーソドックス印象です。
- リファレンスが膨大にあるので、標準的な難易度ながらスムーズに習得できると思います。
  逆に、初心者向けのリファレンスが多すぎて迷ってしまうくらいです。

② ＰＨＰ

A) ＜できること＞

- 簡単な Web ページの作成からブラウザゲームまで幅広く作成が可能。
- 開発に要する時間が比較的短いので、Web 系のサービス開発に向いており、メディア系、EC サイト、ソーシャルゲームのバックグラウンド問わず、様々なサイトが開発できる。
- データベースや API との連携、cookie へのアクセス駆使することで幅広いコンテンツや Web システムの作成が可能。
- JavaScript 等の言語を併用することができ、動きのあるリッチなコンテンツや Web ページに仕上げることができる。
- 動的なウェブコンテンツを含むウェブサイトの構築に適した、オープンソースのソフトウェア群である「LAMP」のうち、「P」として知られています。
  （・Linux **OS** ・Apache, Web **サーバ** ・**M**aria**DB**/**M**ySQL ・**P** for PHP/Perl/**P**ython の**頭文字**）

B) ＜特徴＞

- WEB サービスの開発実績が多く、Web 業界では人気があります。WordPress などオープンソースの製品が多いことも大きな特徴の一つです。
- 全てオープンソースであり、無料で使えるワンセットの言語担当として普及しています。CakePHP や ZendFramework、FuelPHP、Symfony など、フレームワークが多いのが魅力で、ライブラリも豊富です。ただし、CGI よりも実行速度が速くはありますが、基本的に呼ばれてから処理されるので速くありません。
- また、Web 系の開発以外で使用されることはないので汎用性が高くありません。HTML や JavaScript の習得が必須となるでしょう。
- 昨今の PHP 人気からか勉強会の数も多く、リファレンスや学習書も豊富にあります。

C) ＜難易度＞

- PHP は一般に浸透している言語のため、書籍や勉強会、Web 上にはリファレンスやサイトに使える処理が数多くそろっているため学習は比較的簡単です。
- C/C++や Java に比べれば比較的学習はしやすく、個人でも簡単なシステムから Web サービスまで作りやすいといえます。
- PHP 自体は良くも悪くも**型**の宣言等がなく少ないコード量で多くのことができ、また他の言語よりもルールが少ないので覚えやすいです。
  そのため勉強したばかりの初心者でも簡単なホームページは作成できます。
- また、**型**の宣言等をせずにある程度直感的に進めて動作を確認することができるので、実際にコードを書いて実装をするといった感覚が他の言語に比べて容易に得られます。
- 反面、**PHP** は宣言等がなくルールも少ないので、**型**がしっかりとしている C 言語や Java に取り組もうとした際に苦戦するかと思います。また、**HTML** と **JavaScript** を組み合わせた開発が主体となるため、**HTML** の知識が多少は必要でしょう。
- それでも初心者に適している理由は、自分の PC に **XAMPP** などのローカル開発環境を簡単に構築できるからです。

③ ＪａｖａＳｃｒｉｐｔ

A) ＜できること＞

- オブジェクト指向型、関数型、命令型のプログラミングへの対応が可能。
- ブラウザ側で動作する言語で、Web サービスでの高機能な画面動作の実現に必須となる。
- Unity というゲームエンジン上で JavaScript を使ってゲーム開発ができる。
- HTML にさまざまな動きや機能をあたえることができる。
- 開発環境なしに、テキストエディタを使ってプログラミングができる。
- 最近だと 3DCG をレンダリングしたりもできるらしい。

B) ＜特徴＞

- Web 開発において幅広く利用されており、初心者からヘビーユーザーまで愛用しています。
- node.js 等のサーバーサイドでの利用も可能で、バックエンド／フロントエンド両方が実現可能であり、Unity での開発にも用いられることから最近注目度が高まっている言語です。
- ライブラリも jQuery や node.js、AngularJS といったフレームワークまであり、楽に実装ができるようになってきました。
- デメリットとしては、ブラウザにより動作に差が出る事があるので動作確認が大変、プログラミングの自由度が高いために不具合がでやすい、といったところでしょうか。
- 同じスクリプト言語の PHP と記述形式が似ているので合わせて学習していくとよいと思います。

C) ＜難易度＞

- コーディングの基本的な部分では、ほかの言語よりもルールが少なく、かつ学習サイトや技術書も多くあるため学習は比較的簡単にできるでしょう。
- ライブラリを使えば、何百行も記述する必要のあった JavaScript をたったの数行で用いることができるので、簡単におしゃれな動作を実現でき、さらに取っ付き易くなりました。
- 自身の PC 内にブラウザを用意できれば、あら簡単、すぐに開発ができてしまいます。
- ただ、JavaScript オンリーの知識ではプログラマとしては不十分で、基本的に HTML や PHP などフロント側とセットに覚えて、いわゆるマークアップエンジニアや、フロントエンジニアといった称号になることが多いです。
- オブジェクト指向プログラミングの場合は、プロトタイプなどの理解は必要になってくるので、本格的に使う場合は深い学習が必要になってくるかと。

④ Ｒｕｂｙ（ルビー）

日本で開発されたオブジェクト指向スクリプト言語（比較的単純なプログラムを記述するための、簡易的なプログラミング言語全般をいう）で、国際規格に認証された言語です。

A) ＜できること＞

- 簡単な Web サービスから大規模なものまで開発ができます。Twitter やクックパッドから、自治体業務システムまで運用されている。
- 手早く書けてすぐ動かせたりするので、バグの対応やコーディングのサイクルが早い
- 組込のクラスライブラリが高性能なので、重要でないところは自分で記述する必要がない

B) ＜特徴＞

- 他の言語に比べて柔軟性が高く、比較的習得しやすい言語として評判が高いです。文法に縛られることなく自由度が高い言語です。
- インタプリタ言語なので実行速度が遅くなりますが、実行時間で比べるとその差は少ない。
- やはりわずかなソースコードで多くの機能が実現できることが最大の魅力でしょう。
- シンプルで美しい文法は、解析などがスピーディーに行われ、保守性の面でも好まれます。
- テキスト処理機能が強力で、システム管理系において効果的に活用できます。
- ただ、自由度が高いため、チームで開発をするときなどはある程度ルールを決めて記述をしていかなければ、処理がかみ合わなくなってしまう可能性もあります。

C) ＜難易度＞

- 覚えることも C/C++や Java などに比べ少なく、初心者でも比較的容易に学習することができます。
- また、オブジェクト指向前提で作られているので、初心者がオブジェクト指向（ソフトウェア開発技法のひとつ。あるデータの処理をオブジェクト（モノ）にまとめて部品として扱い、部品の組み合わせでシステム全体を構築）を学ぶにはもってこいの言語です。

⑤ C＃

A) ＜できること＞

- パソコン、サーバー用のプログラムによく利用されているうえ、ゲーム、アプリ開発などにも幅広く使用されている。
- 企業向けの業務システムで利用されることも多い。
- 最近流行りの「Unity」というゲームエンジンでは C#で記述することもできるので、個人でゲームを作ってしまうことが可能。

B) ＜特徴＞

- C 言語を拡張して C++なら､C#は C++を拡張したものだと思いきや C 言語の名を冠した Java でした。Java と文法が似ているため、Java エンジニアが習得しやすい言語です。
- 最近出てきた比較的新しく注目度が上がっている言語で､.NET 環境で主に利用されている言語です。
- C++で問題となったメモリ管理の問題が解決されている点が魅力ですが､Windows 環境でのみの利用が可能という制限があります。

C) ＜難易度＞

- C や C++ほど覚えることもなく、学習に関しては難しくはありません。
- Visual Studio を使うと恐ろしいほどのサポートが付きます。
- 最近では Unity などでも利用されるようになったため、関連する技術書や学習サイトが増えてきたので、個人での学習においても困りません。

⑥ C＋＋

A) ＜できること＞

- C 言語の拡張版言語で、C 言語でできることは C++でもできる。
- コンシューマーゲームから生産性ソフトウェア、ロボット工学から電化製品のような組込みシステムまで手広く開発することが可能。
- 基本的に C 言語と同じ作法の為、大抵の言語で通用する。

B) ＜特徴＞

- C 言語の発展系であるため、PHP や Ruby、Java といった言語よりも難易度が高いです。ただし C 言語の拡張版言語であるため C 言語を使用したことがあれば勉強がしやすいです。例えば、型の宣言などは PHP などに比べて厳密になります。
- 古くから利用されている言語のため、リファレンスや参考書などは多くありますが、実装できることの幅が広く、覚えることも多いので、初めて C++言語を学習する場合は取捨選択をして学習を進めないと学習にかける時間が膨大になってしまいます。

C) ＜難易度＞

- C 言語の発展系であるため、PHP や Ruby、Java といった言語よりも難易度が高いです。
- ただし C 言語の拡張版言語であるため C 言語を使用したことがあれば勉強がしやすいです。
- 例えば、型の宣言などは PHP などに比べて厳密になります。
- 古くから利用されている言語のため、リファレンスや参考書などは多くありますが、実装できることの幅が広く、覚えることも多いので、初めて C++言語を学習する場合は取捨選択をして学習を進めないと学習にかける時間が膨大になってしまいます。

⑦ Object-C

A) ＜できること＞

- iOS のアプリ開発のために作られた言語で、iPhone/iPad アプリや Mac の PC アプリを開発することが可能。
- 実機を所持していれば実装を進めることが可能。

B) ＜特徴＞

- C 言語の拡張版というよりも、C 言語で記述されたものを制御しやすくするための言語。
- ソース自体は一見 C 言語とかけ離れたように見えますが、実際には C 言語の上位互換的な存在で、基本的な構文などは C 言語になぞらえています。
- 扱える場所が限られているためこの言語を選択する際には注意が必要です。

C) ＜難易度＞

- C/C++をある程度理解できれば、そこまで困難ではありません。逆に言えば、C/C++の基礎を学んでいないと学習には時間がかかってしまうかと思います。
- 独特な記載が必要な箇所はリファレンスや参考書などで勉強する必要があります。
- また、実機を使用しての実装が行えるため、動作確認やメモリ管理などを意識して作成する必要があり、処理の流れなど注意して作成、学習を進めることが重要です。

⑧ Python （パイソン）

A) ＜できること＞

- 日本では珍しいが海外の Web サービスでは広く使用されている。
- Web アプリケーション、デスクトップアプリケーションなどの開発が可能。
- システム用のスクリプト、各種自動処理、理工学・統計解析のツールとしても使用可能。

B) ＜特徴＞

- **C** や **C++**と仲が良く、何でもそれなりにこなすことができ、**C 言語**や **C++**のライブラリも利用可能。**Ruby** がライバル的存在。
- データ解析や自然言語の解析など数字を扱うことも得意としていて、分析を任されやすい傾向にあります。

- 少ないコード量でかつ簡潔に記述ができるため、「読みやすく、書きやすい」との太鼓判が。
- 生産性、保守性が高く海外では頻繁に利用されている言語で、最近は需要が急速に高まってきている印象です。Google でも使用されているらしい。
- ただ日本においては、**Python** を愛用している人口が他言語と比べて圧倒的に少ない点があり、求人数もさほど多くはありません。

C) ＜難易度＞

- **Python** は古くから利用されている言語であるため、リファレンスや技術書も豊富にそろっているため、学習に困ることは少ないかと。
- 難易度も **C/C++**、**Java** 等に比べれば難しいものではありません。

⑨ C言語

A) ＜できること＞

- OS 開発の際に役に立ちます。機械に近い言語なので、ハードウェアに近い部分を触ることができます。また、汎用性が高く、非常に多くの分野で採用されています。
- 様々な言語の元になっており、元祖プログラミング言語とも呼べる言語です。開発が今なお続いている現役の言語です。
- C 言語で学習した内容は、大抵の言語で通用します。**C++**は **C 言語**の上位互換なので、相性は抜群です。**C++**でも **C 言語**で使用できる関数が使用可能なので **C 言語→C++**と取得するのも良いでしょう。

B) ＜特徴＞

- C 言語はすべて記述した通りに動きます。逆にいうなら、必要な処理はすべてコーディングする必要があります。つまり自由度がある文法なため、プログラマの技量が問われるということです。
- コンパイルすることで、機械語に翻訳されているため非常に速いのも特徴です。C++と同様「プログラムでパソコンを制御する」というのを学ぶには、良い言語といえます。
- 多くの言語にあるクラスという概念をサポートしていません。(大抵、構造体で代用します)
- 環境に依存するため、OS が変わるとソースコードの大改修が必要です。

C) ＜難易度＞

- 初心者が軽い気持ちで C 言語を学び始めると痛い目にあいます。が、C 言語をある程度使いこなせるようになれば、たいていの言語にも応用ができるので初めに学んでおいて損はありません。
- 多くのエンジニアがつまずいてきたポインタ管理などの理解が必要なため、使いこなせるようになるには時間が掛かります。
- ただし、古くから利用されている言語のため、学習書には困りません。
- 実装できることが多く、実装に必要な作法もあるので習得には時間がかかります。

（３） Web 系プログラミング言語の違い

世の中には「プログラミング言語」と呼ばれるものは実は何百種類もあります。しかし、初心者には「その違い」が分からず、何から学んでいいのか、何を学んでいいのか悩んでしまうでしょう。
ここでは、そのプログラミング言語の中でもメジャーで WEB 系の業界で最も使われている「**Java**」「**PHP**」「**JavaScript**」「**Ruby**」という４大言語について、その違いを簡単に見ていきたいと思います。

① 書き方の違い

プログラミング言語というのは、機械に命令をするための「**言葉**」です。

Web 系言語	**「あいうえお」とインターネットのブラウザ画面に表示する**
Java	System.out.print("あいうえお");
ＰＨＰ	echo 'あいうえお';
JavaScript	document.write("あいうえお");
Ruby	puts "あいうえお"
※PHP や Ruby の方が短く簡単に書けます。 ちなみに Ruby の場合は、他のプログラミング言語では、命令文の最後に「;（セミコロン）」をつけるお決まりのものが必要ありません。	

② 型の違い

どのプログラミング言語にも、「**変数**」という文字や数値などの情報を入れておく「**箱**」があります。その変数や値には「**型**」というものがあります。

Web 系言語	**静的型付け言語 or 動的型付け言語**
Java	同じ型の変数には同じ型の値しか入れられない、**静的型付け言語**
ＰＨＰ、JavaScript、Ruby	なんでも入れられる**動的型付け言語**
※「**動的型付け言語**」というのは、「型とか気にせずになんでも放り込んで好き放題自由にできる」よって言う言語のことです。 ※**Java** の場合は、「**静的型付け言語**」といって、忠実に「最初にこの型の値しかこの変数には入れない！」と決めたら一生守るというタイプです。 型が決っていることで何がいいかというと「その型のデータ（値）以外は箱に入れられない」ということです。そうすることで、変なものが入ってきそうになった場合には、エラーになるので、「あ、変なものが入ってきたな」とわかります。	

③ 翻訳の違い

JavaScript や **PHP**、**Ruby** といったプログラミング言語は、「**インタプリタ型**」といって、実際に動かす際にプログラムを１行１行機械語に翻訳しながら実行をしていくものです。

その場で翻訳しながら読んで内容を理解していくわけなので、若干処理が遅い（読むスピードが遅い）という**デメリット**はありますが、プログラムを修正したらすぐ実行できるという開発の速さもあるので「開発スピードを重視」する場合に有効です。

Java 言語も同じくインタプリタ型ですが、一旦「中間言語」という形に翻訳します。
機械語に翻訳しやすい形に一旦翻訳をするという感じです。それから、実行していきます。

なので、随時翻訳をするよりも翻訳が速い（処理スピードが速い）という**メリット**があります。**大規模でユーザー数も多いなど「早く処理して早く画面を表示して～」という場合に向いています**。

逆に一旦、中間言語に翻訳する必要があるので、「ちょっと修正したらまた中間言語に翻訳をして～」と手間がかかるので開発スピードは落ちるという**デメリット**はあります。

Web 系言語	コンパイル&インタプリタ型
Java	一旦、**中間言語にコンパイルし**、それからひとつひとつ翻訳しながら実行していく**インタプリタ型の変化番**
ＰＨＰ、JavaScript、Ruby	その場でひとつひとつ翻訳しながら実行していく**インタプリタ型**
※インタプリタ型は、実際に動かす際にプログラムを１行１行機械語に翻訳しながら実行する型式。	

④ 環境構築の違い

環境構築とは、実際に手持ちの PC の中でプログラミング言語を書いて動かす環境を作ることをいいます。

Web 系言語	環境構築・インストールなど
Java	Eclipse など IDE（パソコンとハードディスクを接続するインターフェースの規格のひとつ）を入れたり、環境変数の設定をしたりと～色々やる事が多く初心者にはハードルが高い。
ＰＨＰ	MAMP をインストールすれば終了。
JavaScript	**何もいらない**。適当にﾌｧｲﾙを作ってﾌﾟﾛｸﾞﾗﾑを書けばブラウザで動く。
Ruby	Xcode をインストールして、rbenv を入れて～とやることが少し多い。

⑤ まとめ

色々なプログラミング言語がありますが、出来ることはほとんど同じです。

（**iPhone アプリ**を作るには**Objective-C 言語**や**Swift 言語**でないと、という場合もあります。ちなみに、最近では**JavaScript** で iPhone アプリを作ることもできるようになりました）

ちなみに**HTML** や**CSS 言語**ってプログラミング言語じゃないの？と思われるかもしれませんが、正式には「**マークアップ言語**」といって「何かの処理をさせる」といったいわゆるプログラミング言語とは違い、視覚表現や文章構造などを記述するための**形式言語**である。
テキストファイルであることが多いが、バイナリデータによる形式もあります。

書き方の違いやメリット・デメリットはありますが、**根本的にやることは同じなので**、最初に学ぶ場合は「**環境構築が簡単**」で「**すぐに動かせる**」「**動いているのが分かる**」ものがいいでしょう。

⑥ プログラミング初心者がどの言語を学ぶべきか？

Web 開発に関わるならば **JavaScript** を学ばねばならないのは明らかです。バックエンドのウェブ開発をしたいのなら **SQL は必須**でしょう。

データ関連のウェブサイトやデータサイエンスに関心があるのなら、**Python が推奨**されます。企業で働くのなら、**Java が必要**です。システムの構築に関心がなく、単に労働機会が欲しいだけなら **SQL**（特に数学好きであれば）をおすすめします。

ＯＳ・ハードウェア・ＣＰＵ・メモリとアドレス・命令とデータ・マシン語等々とプログラムの基礎知識を学ぶことで、プログラムが動作する仕組みが、実にシンプルだと実感できたはずです。

プログラムの本質は、今後コンピュータが進化発展を続けても、決して変わることはありません。

昔は、学生のためにｲﾝﾀﾌﾟﾘﾀ型のﾌﾟﾛｸﾞﾗﾑ学習用・簡易言語を開発したり、FORTRUN でﾛｹｯﾄ追跡のﾌﾟﾛｸﾞﾗﾑを実習、ｱｾﾝﾌﾞﾗ言語で金融のｵﾝﾗｲﾝ制御を開発、COBOL で３次元ﾏﾄﾘｯｸｽの業務ﾌﾟﾛｸﾞﾗﾑを作り、
処理時間が大幅にかかってしまったりと、（実は，４次元？をｺﾝﾊﾟｲﾙしてｴﾗで失敗したことも）

少なくとも、一つの言語をとことん追求して、本質を理解しマスターすれば、自ずとプログラムの基礎知識が身に付き、高度なプログラミング技術を応用することが出来ます。

皆さんも、失敗を恐れることなく、いろいろな言語にチャレンジして、仕事を謳歌してください。