第0章

プログラミングとは

この本は，プログラミングについて学ぶ本だ。次章以降で，プログラミングの具体的なやり方を説明するが，まず，その準備をかねて，そもそも「プログラミング」とは何かを考えてみよう。

[脚注] 第0章から始まったことに違和感を覚えた方も多いと思うが，その理由はこの章の後半で説明する。

プログラミングはプログラムを作ること

「プログラミング」は英語から来た言葉で，英語で綴ればprogrammingあるいはprogramingとなる。programmingは「プログラムを作る」という意味の動詞programのing形だ。前にbe動詞を伴わずに使われているので，このing形は「動名詞」となり，programmingで「プログラムを作ること」という意味になる。

programには「プログラムを作る」という動詞のほかに名詞の意味もある。普通の世界で「プログラム」というと，たとえば，コンサートの「プログラム」などが思い浮かぶ。コンサートの「プログラム」は，「コンサートでどんな曲を演奏するか，その曲名と順番を書いたもの」だ。演奏者はプログラムに従って，順番にコンサートを進行する。

コンピュータの「プログラム」もこれと似ていて，「（演奏者ではなく）コンピュータが何をどんな順序でやるかを書いたもの」だ。見方を少し変えると「（何らかの仕事をするために）コンピュータにやってもらう一連の操作を記述した文書」ということができる。

プログラミング言語と自然言語

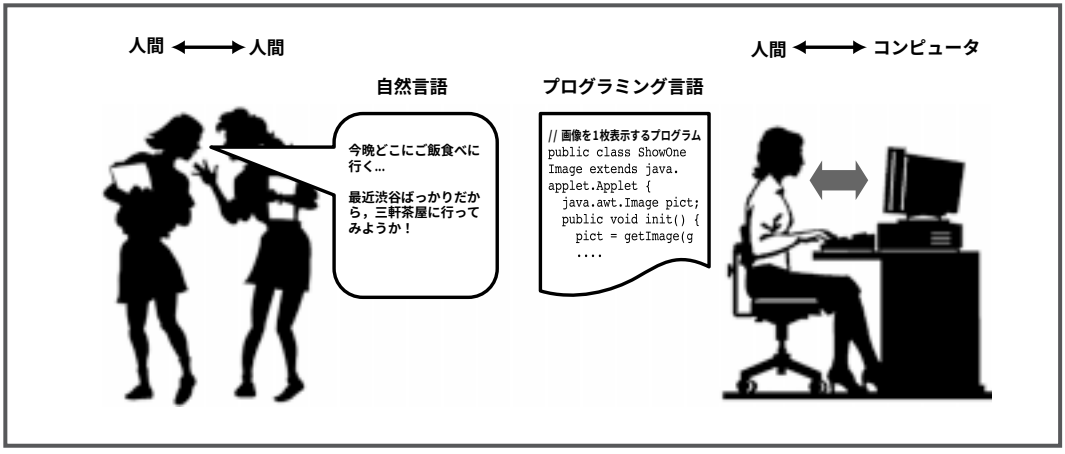
コンサートのプログラムは，普通の言葉，「日本語」とか「韓国語」とか「英語」とか，人間が人間とコミュニケーションをとるための言葉，を使って書かれる。これに対して，コンピュータの「プログラム」を書くときには，それ専用の言語である「プログラミング言語」を使う。

プログラミング言語は，人間がコンピュータとやりとりをするために，（ひとりあるいは何人かの人が集まって決めて作った）人工的な言語「人工言語」である。これに対して，「日本語」や「英語」などは，遠い昔に源を発し，徐々に徐々に変化してきて，今も少しずつ変化しているもので，使われているうちに自然に広まった「自然言語」である。

日本語や英語などの自然言語を使ってコンピュータ用のプログラムを書くことができれば，新しい言葉をあらためて覚える必要がなくて便利だ。しかし，残念ながら「自然言語」を，そのままプログラムを書くこと（つまり，プログラミング）に使うのは少なくとも今のところは無理だ。

なぜ，自然言語をそのままプログラミング言語として使えないのだろうか？　それを考えるために，コンピュータがどのような処理をしているか，いくつか例をあげて見てみよう。

図0-1　プログラミング言語と自然言語



コンピュータによる「世界」の表現――色

自然言語をそのままプログラミング言語として使えない理由として，コンピュータで扱えることが，人間が扱うこと（考えること）のごく一部の世界でしかないということがあげられる。コンピュータが扱えるのは「数」だけで，これを使った加減乗除などの「演算」しかできない。「文」や「絵」や「音」を扱うこともできるが，実はコンピュータの中では文字や数字や音を，それぞれ数字に対応させて，これを並べたり，入れ替えたりしているにすぎない。

コンピュータがそのまま直（じか）に扱えるのは「数」だけ，突き詰めれば0と1だけだ。たとえてみれば，何億個とか何十億個とかの，電球がずらーっと並んでいて，電球がついたり消えたりしているだけの話なのだ［脚注1］。

画面上に色を出す場合を考えてみよう。コンピュータの内部にある何千万個，何億個かの電球のうちの何個か（1千万個から2千万個ぐらい）は，画面の表示を司（つかさど）るものだ。たとえば，1千万個の電球全部が消えれば真っ黒な画面が表示されるし，全部がつけば真っ白な画面になるような仕組みになっている。

虫眼鏡を使って画面を拡大して，画面上の1点［脚注2］に注

［脚注1］この電球の役目をするのが「メモリ」と呼ばれるものである。電球の数が多ければ多いほど，一度にたくさんのものを覚えられるわけだ。

［脚注2］この1点のことを「ピクセル」とか「ドット」とか呼ぶ。

目してみよう。この1点を、最近の多くのパソコンでは24個の電球を使って表す。24個全部の電球がつくと白い点になる。全部が消えると黒点だ。この24個の電球を，8個ずつ3つのグループに分けて，そのうちの第1のグループだけが点灯して，他の2グループが全部消えた場合は，点の色が「赤」になるようになっている。また，第2グループだけが点灯すると「緑」，第3グループだけが点灯した場合には「青」になる。

電球がついている状態（オンの状態）と消えている状態（オフの状態）を1と0に対応させれば，数で表すことができる。たとえば，赤い色は第1グループが全部オンで，第2，第3グループが全部オフなので，次のような1または0の並びで表現できるわけだ。

```
11111111 00000000 00000000 ➡ 赤
```

同様に緑や青は次のようになる。

```
00000000 11111111 00000000 ➡ 緑
00000000 00000000 11111111 ➡ 青
```

この手法で他の色も表すことができて，下のように第1グループの4個，第2グループの6個，それに第3グループの6個がオンになった場合は「水色」になる。

```
1100011 11101101 11101101 ➡ 水色
```

さて，赤を表す第1グループの電球のオン，オフのパターンは全部でいくつあるだろうか？　8個の電球がそれぞれ独立に（ほかとは関係なく）オンになったりオフになったりするわけだから，2×2×2×2×2×2×2×2＝2^8＝256通りのパターンができる。だから，8個の電球のグループを使えば，256個のものに対応させることができるわけだ。8個の電球で256の状態を表すことができる。

コンピュータでは，この256個を1からではなく，0から始めて，1，2，3，……といって，255までの数字に対応させることに決まってる。この考え方を使うと，電球のオンオフではなく，0から255の数字3つを使って色を表すこともできることになる。(255, 0, 0) で赤，(0, 255, 0) で緑，(0, 0, 255) で青，そして (99, 237, 237) で「水色」といった具合だ（図0-2）。

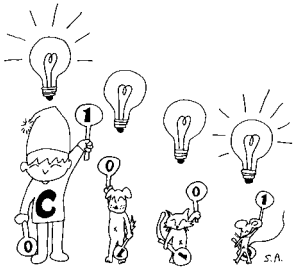
8＋8＋8個ある電球のある特定の8個の電球がついているにすぎないのだが，この状態に意味を付与してこれを「赤」と呼ぶ。そのような状態のときには，画面上の対応する1点（ピクセル）が赤くなるように，コンピュータは作られているのである。

横1,024個，縦に768個の点がある画面だとすると，全部で1,024×768=786,432個の点があり，それぞれの点に24個の電球が用意されているわけだ[脚注]。つまり，この大きさの1画面分を表現するのに，786,432×24＝18,874,368個（約1,900万個）の電球が用意されていると考えることができる。

図0-2　RGBという方式の色の指定法。8×8×8個の電球のオンオフを指定していることに相当する。(99, 237, 237) で「水色」を表す（Windowsの例）

メモ

この1個の電球のことを「ビット（bit）」と呼ぶ。bitはbinary digit（「2進の数字」の意）の最初のbと最後のitを合わせたもの。つまり，1,024×768の大きさの画面では，約1,900万ビットを使って，画面の状態を表現している。



コンピュータによる「世界」の表現――文字

今度は文字について考えてみよう。ひとつの文字は，16個の電球で表す場合が多い（いろいろな表し方があるが，その多くは16個の電球，つまり16ビットを使って1文字を表す）。16個の電球を使えば，$2^{16}=65{,}536$字の文字を表現することができるわけだ。これだけあれば日本語に使われる文字を表現するのには，ほぼ十分だ。

どの数字にどの文字を対応させるかの方法――これを「コード系」という――には，いくつかの種類がある。これまで，一般のパソコンでよく使われてきたコード系は「Shift JIS」というものだ。シフトJISでは，33440で「あ」，33441で（小さい）「ぃ」，33442で（普通の）「い」，33478で「と」，36434で「山」，37100で「川」，35827で「空」を表すといった具合に，数値と漢字が対応付けられている。したがって，「山と川」という文字の並び（文字列）は，コンピュータの内部で

[脚注] 縦横の点の数はコンピュータによっていろいろだ。少し前には640×480（横に640個，縦に480個）という大きさが標準的だったが，その後だんだん大きな画面が普通になってきて，最近では1,024×768あるいはそれ以上というのが一般的になってきている。

は，シフトJISを使う場合「36434, 33478, 37100」という数字の列で表されている。

コンピュータは欧米を中心に発達してきた。欧米で使われているのは，小文字と大文字のAからZと，数字の0から9，それに@，&，%などいくつかの記号，そしてフランス語やスペイン語などで使われるéやñなどいくつかの特別な文字だけなので，7つか8つの電球（ビット），つまり128個か256個の数字があれば，すべての文字に対応させることができてしまう。このため，多くのプログラミング言語では，（7ビットでは切りが悪いので）8ビットを使って文字を表している。

この8ビット，つまり英数字などの1文字を表すのに十分なビットの集まりのことを「バイト（byte）」と呼ぶ。1バイトは8ビットで，欧米の多くの言語で使われる文字ひとつを表現するのに十分な量だ。

日本語を扱うときは，1文字を表すのに8ビットでは足りないので，16ビット，つまり2バイトを使う（あるいはこれ以上使うこともある）。1バイトで1文字を表すことになっている体系の中で，2バイトで1文字を表現しなければならないので，特殊な処理をしなければならないことが多い。このため，日本語など，文字数の多い自然言語で使われる文字を扱うのはなかなか大変だ。

最近のプログラミング言語，たとえばこの本で学ぶJava（ジャバ）やJavaScript（ジャバスクリプト）などでは，アジアなどで使われている文字数の多い言語のことも考えて，最初から文字は2バイト（あるいはそれ以上）で扱うとしているものも増えてきている。

ちなみに，上で見た色に関していえば，3つのグループに分けたそれぞれのグループを1バイト（8ビット）で表現できることになる。画面上の1点の色を扱うのに，最近の多くのパソコンでは3バイト使っているわけだ。画面上のすべての点の色を表現するのには，一般的なものでは (1,024×768×24)÷8＝2,359,296バイト必要なわけだ。

メモ

1,024バイトのことを1キロバイト（kilobyte，略してKバイトあるいはKB）といい，1,024Kバイトのことを1メガバイト（megabyte，略してMBあるいはMバイト）と呼ぶ。もともとkiloは1,000，megaは1,000,000（百万）を表す言葉だが，2^{10}が1,024で2進数が基本のコンピュータの世界には都合がよいので，1,024倍（つまり2^{10}倍）をキロ，1,024倍の1,024倍（つまり2^{20}倍）をメガと呼んでいる。画面にあるすべての点の色を表現するには，2Mバイト強のメモリが必要になるわけだ。

ちなみに，1,024メガバイトのことを1ギガバイト（giga byte，略してGBあるいはGバイト）と呼ぶ。さらに，1,024ギガバイトのことを1テラバイト（tera byte）という。「テラバイト」はまだあまり使われることは多くないので，略されることは多くないが「Tバイト」とか「TB」などと表記されることになるのだろう。

整数と小数

電球のオンオフで整数や色や文字を表すこと（整数，色，文字とビットのオンオフの並びを対応させること）ができるのはおわかりいただけただろうか？

整数の場合は，それぞれの電球ごとに，$1=2^0$を表すもの，$2=2^1$を表すもの，$4=2^2$を表すもの，$8=2^3$を，$16=2^4$を，$32=2^5$を，$64=2^6$を，$128=2^7$を，……と役割を割り振っておいて，オンになっている電球が表す数を足すことで，（電球の数さえ増やせば）いくらでも大きな数を表せる。

色の場合は，（たとえばRGBという表し方を用いるとすれば）色を赤，緑，青の3つの要素に分解して，それぞれの度合いを整数で表すと，この整数を電球のオンオフで表せる。そして，文字は1文字ずつ整数を対応させて，この整数をビットのオンオフで表現するわけだ。

ところで，3.14とか，2.718281828459041（フナイチワニワイチワニワシゴクオシイ）とか，1.41421356（ヒトヨヒトヨニヒトミゴロ）などといった小数は，どう扱うのだろうか。これは，整数の場合とよく似た論法を逆方向に使う。つまり，$\frac{1}{2}$，$\frac{1}{4}$，$\frac{1}{8}$，$\frac{1}{16}$，……を表す電球を用意するわけだ。数学で習ったように（忘れてしまった人もいるかもしれないが），$\frac{1}{2}=2^{-1}$，$\frac{1}{4}=2^{-2}$，$\frac{1}{8}=2^{-3}$，……となる。

0.5ならば，$\frac{1}{2}$を表す電球を1個だけオンにする。1.25ならば，1を表す電球と$\frac{1}{4}$を表す電球をオンにするわけだ。

しかし問題がある。たとえば，3.14を考えてみよう。3の方は1と2の電球をつければよいので，残りは0.14だ。0.14は$\frac{1}{8}$（0.125）より大きいので，まず$\frac{1}{8}$の電球をオンにする。残りが0.015（＝0.14－0.125）になる。ここまでを式に表すと次のようになる。

$$3.14=1\times2^1+1\times2^0+0\times2^{-1}+0\times2^{-2}+1\times2^{-3}+0.015$$

ここで，0のところと「1×」を省略するとすると次のようになる。

$$3.14=2^1+2^0+2^{-3}+0.015$$

以下，同様にやってみると

$$3.14=2^1+2^0+2^{-3}+2^{-7}+0.00718750$$
$$3.14=2^1+2^0+2^{-3}+2^{-7}+2^{-8}+0.00328125$$
$$3.14=2^1+2^0+2^{-3}+2^{-7}+2^{-8}+2^{-9}+.001328125$$
$$3.14=2^1+2^0+2^{-3}+2^{-7}+2^{-8}+2^{-9}+2^{-10}+0.00035156250$$

……

となって，いつまでたっても終わらない。だから，10進数で表した小数をコンピュータで表現するときは，多くの場合（0.5とか0.125などでうまく切りがつくのでなければ），ピッタリ表すことはできない。コンピュータの中では，小数は「近似値（きんじ）」として扱われているわけだ。

16進数による表記

上で見たように，コンピュータでは8ビット，つまり1バイトを単位としていろいろな情報を扱うことが多い。このため，日常生活で使う10進数ではなく「16進数」を用いて，数値を表すと便利な場合が多い。

16進数では，0，1，2，……，9までの10の数字に加えて，A，B，C，D，E，Fの6文字を使う。Aが10進数の10，Bが11，Cが12，Dが13，Eが14，そして，Fが10進数の15を表す。8ビットの情報を16進数で表すと，$2^8=2^4\times2^4$なので，16進数2桁で1バイトの値を表すことができて，切りがよい。

たとえば，上で出てきた「山と川」はシフトJISコードだと「36434, 33478, 37100」という数字の列になるが，16進数を使うと「8E52，82C6，90EC」となる。たとえば，16進の8E52が10進の36434になるか確認してみよう。

8E52 ➡ $8\times16^3+14$（＝E）$\times16^2+5\times16^1+2\times16^0=32768+3584+80+2=36434$

図0-3のような「漢字コード表」を見たことがある人も多いと思うが，漢字のコードはこの図のように16進数で表示されることが多い。

また，次の章でさっそく登場するが，色の指定にも16進数が使われる。上で見たように，赤，緑，青のそれぞれの成分を1バイト（0から255）で表現するので，それぞれの成分を16進数2文字，合計で6文字で表せるわけだ（図0-4）

アナログの世界を写す自然言語

さて，章の冒頭で何を議論していたのか思い出そう。プログラミング言語として自然言語を使うことはできるかどうかを考えていたのだった。

上で見たように，コンピュータではすべてのものを数字に置き換えて扱う。ワープロで入力する文字も数字の羅列として記憶されているし，「絵」はひとつひとつが24ビットの色を持つ点の集まりだし[脚注1]，「音」もそれを表す波（音波）を数値の羅列として表現している。つまりすべてが「デ

図0-3　漢字のコード表

図0-4　16進数による色の指定（MacOS Xの場合）

ジタル」なのだ。

これに対して，人間の世界はすべてをデジタルな形式（数字）には置き換えて扱うことのできない世界だ[脚注2]。数字に置き換えられるものは，人間が関わる世界の一部にすぎない。

文学作品，たとえば小説そのものを文字としてコンピュータに記憶することはできるが，その小説

［脚注1］第15章で少し紹介するように，線や図形などをほかの方法を使って表現することもできる。しかし，最終的に画面に表示されるときには，点の集まりである。

［脚注2］少なくとも現在は。将来，コンピュータがずっとずっと発達し，記憶容量も爆発的に増大して，処理も爆発的に速くなったとき，ひょっとすると，世界のすべてをデジタルに表現できてしまうかもしれない可能性がないとは，言い切れないかもしれない。

の良さはどこにあるとか，どのような印象を読者に与えるか，といったことは，まだコンピュータには扱えない。良い小説とは何かが，数字で表せるようになっていないからだ。

「もうちょっと紫っぽい色」などという表現は，人間には感覚的に理解できる。しかし，コンピュータで扱おうとすると，たとえば「RGBのRの要素を2，Bの要素を1だけ増やす」などといったようにすべてを数字で置き換えなければならない。これとても，コンピュータが「もうちょっと紫っぽい色」を理解したわけではなく，「もうちょっと紫っぽい色」と指定された場合には「RGBのRの要素を2，Bの要素を1だけ増やす」という操作を対応させるように，人間がプログラムの中に指定しておくのだ。

「翻訳ソフト」と称するプログラム，英語を日本語（らしい文字列の並び）に置き換えてくれるプログラムは開発されてはいるが，満足できる訳が出てくることは多くなく，優秀な翻訳者の訳に比べれば，はるかに劣っている。優秀な翻訳者が原文の表現する「意味」を考えて，それと等価な意味を持つ訳文に置き換えようとしているのに対して，翻訳ソフトは基本的に（この本で紹介するような）文字列のマッチング（検索）を繰り返しているだけだからだ。

コンピュータが扱う世界，（現在）扱える世界は，我々の扱う世界に比べてはるかに狭い。すべてをビットの列，電球のオンオフで表現できる世界だけだ。コンピュータが「意味」を理解しているわけではなく，単にビットのオンオフを繰り返しているにすぎない。それに意味を付けているのは人間の解釈（対応付け）なのである。

自然言語を使ってプログラムを書けるようになるためには，コンピュータが自然言語の意味を理解できるようになっていなければならない。そのためには，まず，コンピュータを使って，電球のオンオフで，人間世界を「理解」する方法を人間が考え出さなければならない。それは，実現されるとしても遠い遠い将来の話だ。

コンピュータの世界を記述するための言葉――プログラミング言語

そこで，この本で勉強するプログラミング言語が登場する。プログラミング言語は，ビット列，つまり電球のオンオフを使って表現できることがわかっている世界を記述するために（あるいは，まだ電球のオンオフで表現できていないことを，この仕組みで表現できるようにならないかを実験するために），人間が作り出した言語だ。

ビット列だけを扱うからといって，ビット列ばかりが並んでいるのではわけがわからないので，できるだけ人間がよく知っている世界に近づけようと，いろいろな人がいろいろなプログラミング言語を作ってきた。人間がよく知っている世界に近づけるためには，プログラミング言語を自然言語

に近づけるのがひとつの方法だ。そこで，「文（sentence）」とか「式（expression）」といった概念がプログラミング言語にも持ち込まれた。

コンピュータは数字を扱うのだから，数学の世界を記述するためにこれまで使われてきた数式などが使えると便利そうだ。そういうわけで，加減乗除などの演算ができるようになっている。これは，かなり初期の頃のプログラミング言語から取り入れられた。

さらに，コンピュータの用途が広がるにつれて，（自然言語や人工言語の）文字の並び――文字列――をそのまま扱えるようにしたいということで，文字列に関していろいろな操作が，しかも簡単にできるような機能を持ったPerlのような言語も登場してきた。

1990年代に入り，インターネットが一般的になってきて登場したのがJavaScriptやJavaなどの言語だ。JavaScriptは，ウェブページ（Webページ，ホームページ）を見るためのブラウザと密接に結び付いて，ホームページ上でいろいろな機能を実現することができる。Javaも，ウェブページ上の長方形の枠内に「アプレット（applet）」という特別なプログラムを表示できる機能を持っており，ウェブと深く結び付いている（Javaができるのはこれだけではない）。

Perlはウェブページが登場する前から存在していた言語だが，ブラウザとやりとりをして文字列の処理をするのに，とても便利だということで（それまでかなり広く使われていたのだが）ますます多くの人に使われるようになった。

この本ではインターネットに深く関係するこの3つの言語を，JavaScript，Perl，Javaの順番で紹介していく。

この本では触れないが，インターネットに関係して使われるプログラミング言語としてはこのほかにもいろいろな言語がある。たとえば，PHPは「データベース」を使っていろいろな情報を記録しておいて，それとウェブページとを関連付けて処理をするような場合に便利な機能がたくさん用意されている言語だ。Perlと似ているところがあるので，Perlを身に付けておけばPHPを学ぶのはかなり簡単になるはずだ。

Macromediaという会社が作ったFlashというソフトウェアには，画像やアニメーションなどをウェブページ上に表示したり，対話的に動かしたりするのに便利な機能を備えたActionScriptというプログラミング言語が付属している。ActionScriptには，アニメーション作成用の特別な機能が用意されているので，このような用途にはとても便利だ。しかし，この本で紹介する3つのプログラミング言語に比べると，応用の範囲はかなり狭くなる。

メモ

ウェブページにFlashで作ったアニメーションなどを表示するには，「プラグイン」というプログラムをブラウザに組み込む必要がある。プラグインは，ブラウザだけでは実現できない機能，たとえばアニメーションなど，をブラウザ内で実現するための補助プログラムだ。Flash用にはFlash用のプラグインをコンピュータに「インストール（install[脚注]）」する必要がある。なお，Flash用など著名なプラグインは，最近のブラウザでは標準で組み込んであるものも増えている。

ActionScriptはウェブ用のアニメーション作成用言語だが，ウェブ用に限らず，特定の分野に関して「狭く，深く」を狙った言語もたくさんある。たとえば，統計用の言語とか，シミュレーション用の言語といったものもある。自分の専門の分野に関係する言語もあるかもしれないので，検索エンジンなどで調べてみるとよいだろう。なお，「プログラミング言語」という言葉を狭い意味で使う場合，このような特定用途向けの言語は含めない場合もある。

インターネットはプログラミング言語にも大きな影響を与えたのだが，インターネットが広まる以前から広く使われており，今でも使われている言語としては，Fortran（フォートラン），COBOL（コボル），Pascal（パスカル），C言語（シー）とその後継であるC++（シープラスプラス）などがある。

いずれにしろ，ほとんどすべてのプログラミング言語に共通する手法や概念は，この本で学ぶことができるので，将来どのような言語を学ぶときにもまごつかずにすむはずだ。ただ，ここで学ぶ言語はかなり新しい部類の言語3つなので，すぐ上にあげたようなやや古い言語を使うときは，機能的に劣っていることで苦労することがあるかもしれない（C++は比較的新しいので，機能的にはそれほど困らないと思う）。

ウェブページを記述する言語——HTML

ウェブページ用の「言語」として一番大事なものが残っている。それはHTML（エイチティエムエル）（HyperText Markup Language，ハイパーテキスト用マークアップ言語）だ。

HTMLは，ウェブページそのものを記述するのに使われるもので，普通は「プログラミング言語」としては扱われない。ウェブページに表示する内容を記述するための言語で，ページに表示する文字や画像などのデータや，ある文字列がそのページでどのような役目をするものか（たとえば「タイトル（表題）」だとか「リンク」だとか）を表す。コンピュータへの命令を並べて，それを実行することにより何らかの操作を行うために使う「プログラミング言語」ではない。

[脚注]「設置する」あるいは「導入する」などの意。

メモ

HTMLの中にプログラムを「埋め込む」ことができる。この本で学ぶJavaScriptはHTMLに埋め込まれる形で実行できる言語だ。Javaのプログラムも少し形式は異なるが，HTMLの文書に埋め込んで実行できる。詳しくはこの後の各章で説明していく。

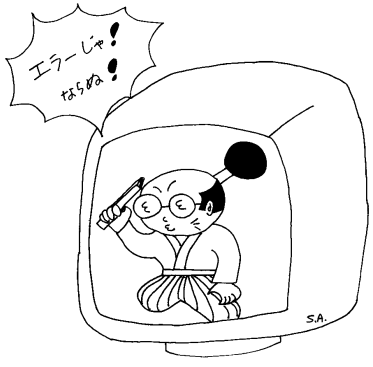
この本では，ウェブ用のプログラミング言語について学ぶが，ウェブ用のプログラミング言語では，HTMLで書かれたものをいろいろな形で使って，ウェブページを変化させる。このため，ウェブページを記述するための言語であるHTMLについても説明する。次の章でHTMLの概要をひととおり説明し，以降必要に応じてほかの（プログラミング）言語を学ぶ際に，HTMLに関して必要な知識を追加していく。

プログラムを作るための準備――心の準備

最後に，プログラミングを学ぶに際して準備しておいていただきたいことをあげておこう。まずは，心の準備だ。

この本は第0章から始まった。なぜ，0から始めたかというと，プログラミングではいろいろなことが，1ではなく，0から始まるからだ。まず，このことに慣れてもらおうと思ったわけだ。たとえば，これから学ぶJavaScriptやJava言語では，1年の最初の月を表す数字は0だ。これがよいか悪いかは別問題として，ともかくそう決まってしまっているので，そういうことに慣れてもらうしかない。

コンピュータは融通が利かない。これはすでに，ワープロソフトなどを使う場合でも，かなり経験しているのではないかと思うが，プログラムを作るとそう思う度合いがかなり増すはずだ。1,000行に渡るプログラムを書いたとき，たったひとつ「"」（引用符）を書くのを忘れても，そのプログラムは動かない。人間なら，1ページに2つや3つの誤字脱字があっても，問題なく理解できるが，プログラムを書くときは，とにかくひとつでも間違いがあれば，それで動かなかったり，誤動作をしたりする。昔々，「,」と書くべきところで「.」と書いてしまったがために，ロケットの打ち上げに失敗したことがあったというのは，この世界では有名な話だ。



最初は，この融通のなさに，ほとほと参る（頭に来る）ことと思う。しかし，逆に見れば，コンピュータはまことに忠実な僕（しもべ）でもある。どんなに単純な作業を，どんなに長いあいだするように命令しても，文句はまったく言わずに実行してくれる。この長所をこそ生かすべきだ。

プログラムを作るための準備——ものの準備

この本を読むためには，さしあたって（この本以外は）何もいらない。書いてあることを読むだけで，プログラミングの世界を仮想的に体験できるはずだ。実行した場合の様子は，図があるのでだいたい想像ができるだろう。プログラミングの世界がどんなものか覗いてみたい方は，そのような読み方をしてもかまわない。

しかし，プログラミングの技術を身に付けようとするには，プログラム（やHTMLの文書）を作ってみて実際に動かしてみなければならない。スポーツやもの作りと同様，ただ見ているのと，実際にやるのとは大違いだ。

実際にプログラムを作るためには，さしあたってインターネットのホームページ（ウェブページ）が見られるパソコンがありさえすればそれで十分だ[脚注1]。最初に学ぶHTMLの文書を作るのにも，その次に学ぶJavaScriptのプログラムを作るのにも，インターネットが見られて，JavaScriptが実行できるブラウザと，エディタあるいはワープロソフトがあれば事足りる。

ブラウザとしては，Internet Explorer（インターネット　エクスプローラ）でもNetscape Navigator（ネットスケープ　ナビゲータ）でも，あるいは少しマイナーだがOpera（オペラ）でも，Mozilla（モジラ）でも大丈夫だ。まずはJavaScriptが動けばどんなブラウザでもかまわない。

プログラムを書くためには，「エディタ」か「ワープロソフト」がいる。どちらかというとエディタの方が，プログラムの作成には向いている。Windows（ウィンドウズ）ならば「メモ帳」，Macintosh（マッキントッシュ）ならばSimpleTextかTextEdit，Linux（リナックス）やBSD（ビーエスディー）などUnix（ユニックス）系のOS（オーエス）ならば，vi（ブイアイ）やemacs（イーマックス），あるいはxedit（エックスエディット）などのエディタが，特に何もしなくても使えるようになっているはずだ。

メモ

ここから後の説明は，この章では読むだけにしておいて，次章以降で実際にファイルを作るときに，ここの内容を思い出しながら試して欲しい。

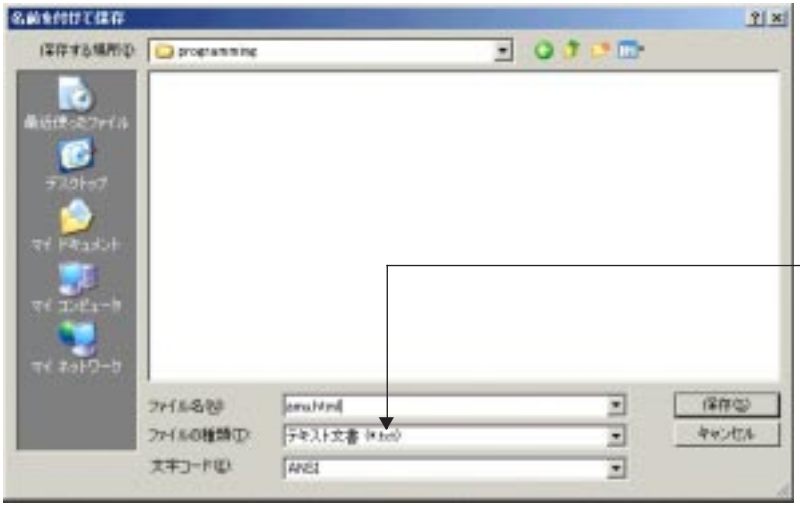
Windowsの「メモ帳」は，「スタート」メニューから「アクセサリ」[脚注2]を選択して「メモ帳」を選べば使える。

［脚注1］最近では携帯電話やPDA（Personal Digital Assistant，携帯情報端末）でもウェブページを見ることができるが，携帯電話だけではプログラムを作るのは，今のところは不可能だ。PDAでも，かなりの不便を覚悟しないと難しいだろう。

［脚注2］Windows XPなどの場合，設定によっては「すべてのプログラム」という項目を経由する必要がある。

プログラムを書き終えたら「ファイル」メニューから「名前を付けて保存...」を選択して，図0-5のように，ファイル名を「拡張子[脚注1]」付きで指定する。

図0-5　拡張子を指定してファイルを保存



拡張子の付け方は次のとおりだ。「こうしなければならない」と決まっているものもあるし，「こうしておいた方が後が楽だ」というものもある。いずれにしても，下の規則に従っておけば問題はない。

- HTMLファイルの場合は`xxx.html`あるいは`xxx.htm`。
- JavaScriptのファイルの場合は`xxx.js`（普通はHTMLファイルの中に書かれるので，JavaScriptのファイルを単独で保存することはあまりない）。
- Perlのプログラムが入ったファイルの場合は`xxx.pl`。
- Javaのプログラムが入ったファイルの場合は`xxx.java`。

Macintosh（略してMac（マック）[脚注2]）でも同じようにできる。Mac OS 9あるいはこれ以前を使っている場合は，SimpleTextというエディタが標準で用意されている。プログラムを書いたら，「別名で保存...」を選択して，Windowsの場合と同じように`xxx.html`，`xxx.pl`，`xxx.java`のように拡張子を付けて保存すればよい。

Mac OS Xならば，ApplicationsフォルダにあるTextEditを使うことができる。プログラムを書いたら，「別名で保存...」を選択して，Windowsの場合と同じように`xxx.html`，`xxx.pl`，

[脚注1] ファイルの名前の最後に付ける，そのファイルの種類を表す文字列。
[脚注2] この本では「マック」といえばMacintoshのことを指す。

プログラム，アプリケーション，ソフトウェア，システム

ここで少し用語の整理をしておこう。プログラムは「コンピュータにやってもらう一連の操作を記述した文書」だが，もうひとつ「その記述した文書に従って動作するもの」のことも「プログラム」と呼ぶことがある。

例をあげよう。この本では「仮名で書いた文字列を漢字に変える」つまり「仮名漢字変換」をしてくれるものを作る。図0-6のようなものだ。［よみがな］の欄にひらがなの文字列を入れて，［変換］のボタンを押すと［漢字］の右の欄に漢字が表示される。

このようなものを作るのに「プログラム」を書くが，「プログラム」を書いた結果できたもの，仮名漢字変換をしてくれるもの，のことも「プログラム」と呼ぶ。「仮名漢字変換プログラム」とか「表計算プログラム」などという場合の「プログラム」は「プログラミングの結果できた，仮名漢字変換をしてくれる仕組み」や「プログラミングの結果できた，表計算をしてくれる仕組み」のことを指している。

この意味の「プログラム」は，「アプリケーション」とか「ソフトウェア（略して「ソフト」）」，さらには「システム」と呼ばれることも多い。「仮名漢字変換ソフト」「表計算のアプリケーション」「仮名漢字変換システム」などと呼ばれる。細かく見ると，それぞれの言葉の意味する範囲は少しずつずれており，感覚的に図に表すと図0-7のようになる。

システムという言葉は，どちらかというと，かなり大規模なものを指す場合が多い。また，システムはコンピュータの世界に限らず，「換気システム」などといったように使う場合も多い。だから，他の3単語の意味するところとはだいぶずれがある。

図0-6　本書で作る仮名漢字変換プログラム

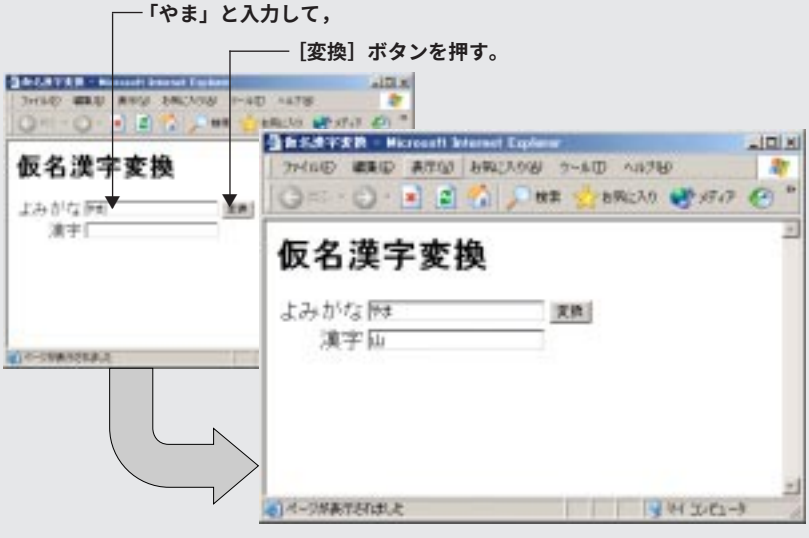
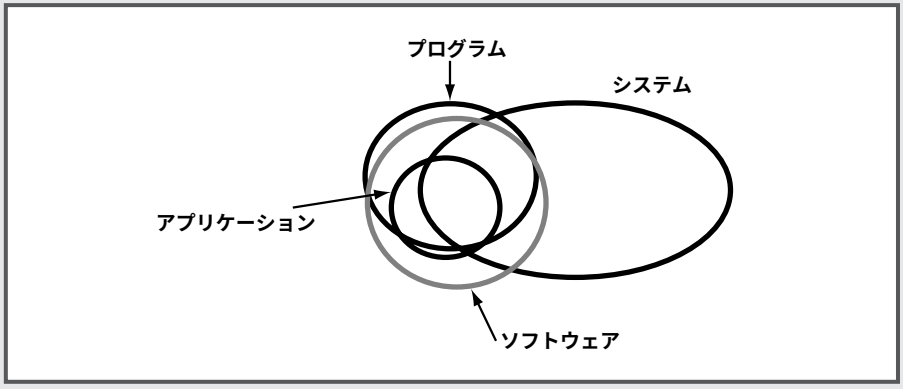


図0-7　プログラム，アプリケーション，ソフトウェア，システムの意味する範囲



プログラム，アプリケーション，ソフトウェアの3つは，コンピュータの機械（目に見えるもの，ハードウェア）ではなく，コンピュータ内部で行われていることを概念的に（抽象的に）表現する場合に使われる言葉だ。

この3つの中では，アプリケーション（application[脚注]，応用ソフトとも呼ばれる）の示す範囲が一番狭い。ワープロとか表計算とか，お絵かきソフトなど，具体的な仕事（作業）をするために作られたもののことをいう。

アプリケーションと相対する言葉としては，「システムソフトウェア」といわれるものがある。Windows（ウィンドウズ）とかMac OS（マックオーエス）とか，Linux（リナックス）などのオペレーティングシステム（operating system，略してOS（オーエス））などがその代表だ。また，スキャナとかプリンタなど周辺機器を処理するためのプログラムも含まれることがある。「システムソフトウェア」は具体的，応用的な作業というよりは，コンピュータを利用する上で基本的な機能を提供するために書かれたプログラムの集まりのことを指す。

システムソフトウェアもアプリケーションもソフトウェアの一種だ。そして，プログラムもソフトウェアということができる。こうしてみると，ソフトウェアが一番広い概念だといってよさそうだ。しかし，ソフトウェアもその構成要素を見ればプログラム（の集まり）であることには違いない。逆にいうと，プログラムはソフトウェアを含む概念のようにも感じられる面がある。

さらに，アプリケーションはプログラムには違いないが，「プログラム」というと少し小さめの印象があるのに対して，アプリケーションといわれると何万行にも及ぶプログラムを書いてやってできるようなもの，という印象もある。

このようなわけで，これら4つの単語の境界は曖昧なまま，かなり感覚的に使われることが多い。こういった意味の違いを，コンピュータに「理解」させるのは至難の業だ。人間でさえ，よくわかっていないのだから。

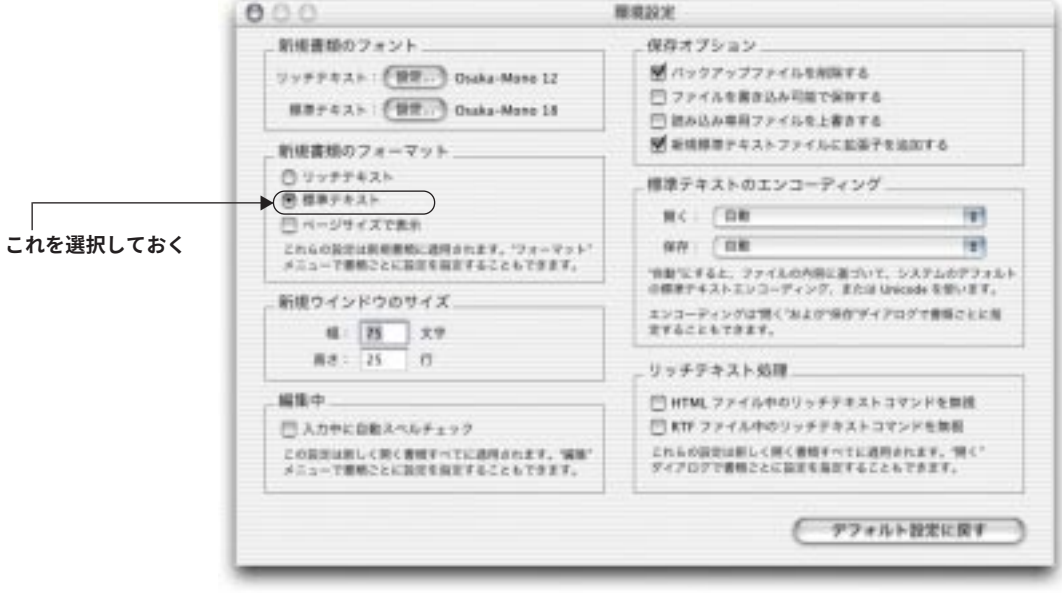
［脚注］「応用する」「適用する」といった意味のapplyの名詞形。

xxx.javaのように拡張子を付けて保存すればよい。TextEditを使う場合は，「環境設定」の「新規書類のフォーマット」のうちの「標準テキスト」をチェックしておいた方がよい（図0-8）。これを選択しておかないと，「フォーマット」メニューで「標準テキストにする」を毎回選択する必要がある。

Windowsでも，Macでも，少し（2,000円とか3,000円とか）お金を出せばとても使いやすいエディタが手に入る（無料（フリーウェア）のものもある）。プログラミングに慣れてきたら，検討してもよいだろう。Windowsならば秀丸（ひでまる）とかEmEditorなど多数ある。Mac OS用にもたくさんあるが，Jeditが一番人気が高いようだ。いずれもインターネットの検索エンジンにプログラム名を入力して検索すれば，各プログラムのホームページがわかるはずだ。そこからダウンロードして試してみてから買うことができる。

LinuxなどのUnix系のOSを使っている人は，標準で付いてくるviやemacs，xeditなどで機能的には十分だ。とくに，emacsやviには機能が山ほど詰まっている。Unix系OSを自分で使っている人で「初心者」は少ないだろうから，マニュアルなどを参照して，いずれかの使い方をマスターして欲しい。学校などで，Unixの環境が使えるという人ならば，「ログイン」をしたら，（まだ起動されていなかったら）X Windowを起動し，メニューからEmacsを選ぶか，ターミナル（ktermなど）など「コマンド」が入力できるウィンドウを開いて，「emacs」と入力すればよいだろう。X Windowから起動したのなら，マウスを使った編集やメニューの選択も利用できる。多くのメッセージ

図0-8　Mac OS Xに標準添付されているTextEditの環境設定



が英語なので，まごつく場合もあるかもしれないが，じっくり読めば何とかなるだろう。

viも便利なのだが「モード」があるので，慣れるまでが大変だ。とりあえずターミナルで「vi」と入力してviの画面になったら，小文字の「i」を押すと入力できるので，文字を入力したら，「Esc」と書かれているキーを押せばよい。「Esc」を押した後で文字を消すには，小文字の「x」を押す。ファイルを保存するには（Escを押して「入力モード」を終えてから）「:w <ファイル名>」とする。終了するには「:q」だ。

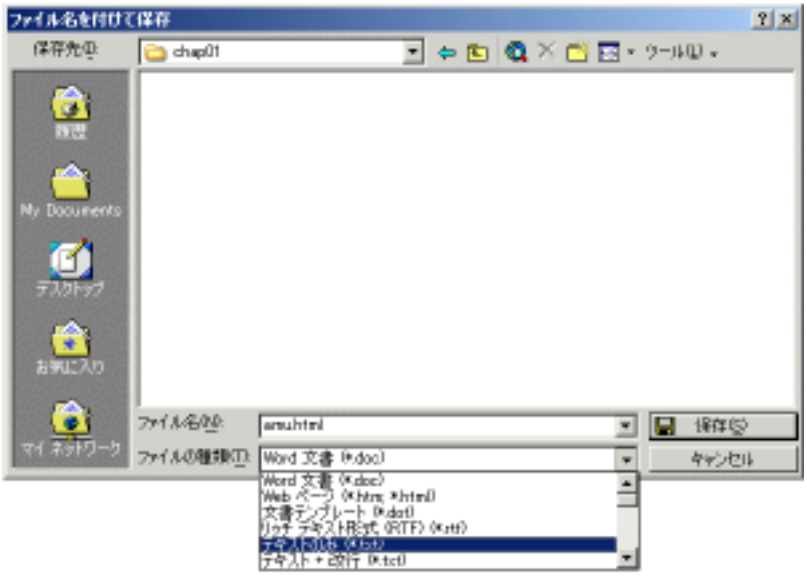
ワープロを使ったプログラムの作成

プログラムの作成は，ワープロでもできないことはない。注意するのは，文字の大きさとか，書体とかを無視して「テキスト形式」で保存する必要があることだ。

よく使われている，Windows（ウィンドウズ）版のMicrosoft（マイクロソフト） Word（ワード）を例にして説明しよう。新しいファイルを作って，プログラムを書いたならば「ファイル」メニューで「名前を付けて保存」を選択して，「ファイルの種類」のところで「テキストのみ（*.txt）」を選んで保存する必要がある（図0-9）。

ただし，ここでHTMLファイルならばxxx.html（あるいはxxx.htm），Javaのプログラムならばxxx.javaといった具合に，.txtではなく別の拡張子を指定する必要がある（図0-9はHTMLファイルを保存しているところ）。このあたりは，非常にまぎらわしいので注意が必要だ。

図0-9　Wordを使ってファイルの保存

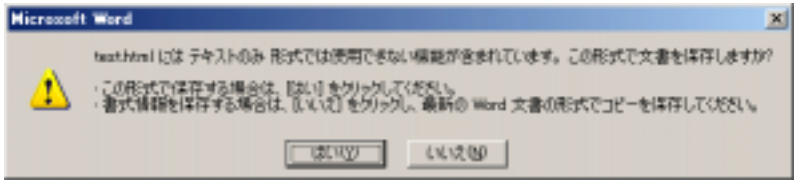


メモ

このとき，たとえウェブページを作るためにHTMLファイルを作っている場合でも「Webページ」という項目を選んではいけない。これを選んでしまうと，作ったばかりのHTMLのプログラムをウェブページでそのまま表示するように，Wordが勝手に操作をしてしまう（HTMLの文書そのものを表示するようなHTMLの文書を作ってしまう）。これは，コンピュータが融通がきかないからこうなっているのではなく，このソフトを作った人々がこうするのがよいと考えて作ったものだ（ろうと思うが，変な規則だと筆者は思う）。いずれにしろ，基本的に，ワープロはプログラムを作る道具という位置付けはされていない。

さらに，テキスト形式で保存しようとすると，図0-10のようなメッセージが表示される。このメッセージには「はい」と答えて，そのままテキストとして保存しておけばよい。プログラムを書く際には，文字の大きさや書体などの情報は無視してかまわない[脚注]。

図0-10　Wordのファイルをテキスト形式で保存しようとすると現れる「ダイアログボックス」



メモ

プログラムを書くとき「文字の大きさや書体などは無視してよい」ということは，どんな書体や大きさの文字を使ってもよいということだ。（筆者のように）「昔に比べて小さい字を読むのは疲れて困る」という人は，大きなサイズの文字を使うようにエディタやワープロを設定するとよい。また，多くのエディタでは背景や文字の色も指定できる。背景色を濃いめの色，たとえば黒にして，文字を明るい色，たとえば白にしておくとさらに目の疲れを防止できるだろう。

エディタやワープロの「初期設定」などの項目を調べてみるとよい。どのエディタやワープロでも，文字の書体（フォント）や大きさを指定できるはずだ。また，有料のプログラムならばほとんどのものに，文字や背景の色を指定する機能も付いている。

［脚注］もしも将来，自分がワープロソフトを作るとしたら，もちろんこのような情報をきちんと処理するプログラムを作る必要がある。ここでいっていることは，自分が書くプログラムを構成する文字については，その書体や大きさは任意のものでよいということだ。

フォルダを使った整理

プログラムを書いたら，コンピュータにある「ハードディスク」にファイルとして保存する。これは，ワープロや表計算ソフトなどの場合と同じだ。Windowsならば「マイ ドキュメント」の下に，Macならば「書類」あるいは「Documents」の下に，上で見たような拡張子を付けて保存しておけばよいだろう。

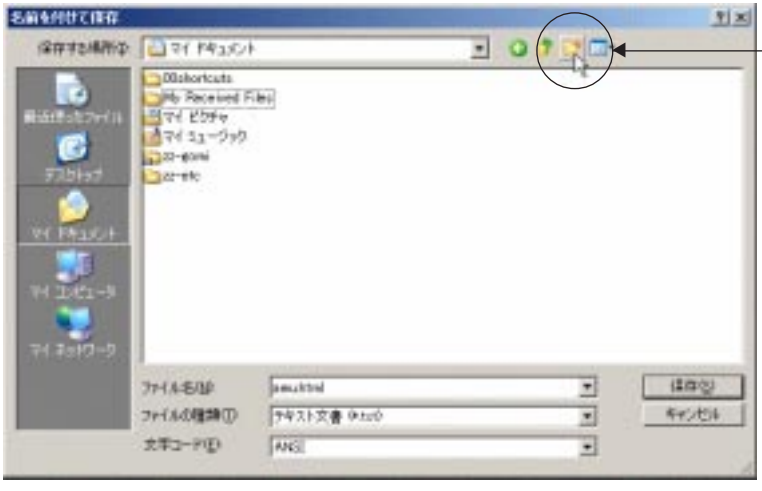
できれば，「マイ ドキュメント」や「Documents」「書類」などの直下ではなく，この本専用の「フォルダ（Unix系のOSやMS-DOS（エムエスドス）の言葉でいえばディレクトリ）」を作っておくとよいだろう（フォルダを作るのはC:\の直下に作ってもよい）。たとえば「プログラム」とか「プログラミングは難しくない！」とかしてもよい。が，先々のことを考えると「programming」などと英語にしておいた方が都合がいいだろう。PerlやJavaのプログラムを作ったり実行するとき，キーの入力が少なくてすむし，他のコンピュータにコピーするときに，「文字化け」の危険がなくなる。

フォルダを作るには，たとえば「メモ帳」の「名前を付けて保存」で表示される「ダイアログボックス」（図0-11）で，右上の「新しいフォルダの作成」のアイコンをクリックするのが簡単だろう。すると「新しいフォルダ」という名前の新しいフォルダができるので「programming」などと入力すればよい。

メモ

当然，WindowsのエクスプローラやMacOS XのFinder（ファインダ）を使ってフォルダを作ってもよい。

図0-11　「メモ帳」のフォルダを作るアイコン



もっときちんと整理したい人は，第1章，第2章，……，第16章に対応して，chap01，chap02，……，chap16とprogrammingフォルダの下に，「サブフォルダ」を作っていくとよいだろう。chapはchapter（章）の略で，01，02と頭に0を付けるのは桁をそろえるためだ。また，こうしておくと，一覧したときにchap01から順番に並んでくれる。

次章の最初の例題を保存する際に，「マイ ドキュメント」の下もしくはC：\の直下に「programming」というフォルダを作り，続いてprogrammingをダブルクリックしてその下に行き「chap01」というフォルダを作って，その下にファイルを保存すればよい。

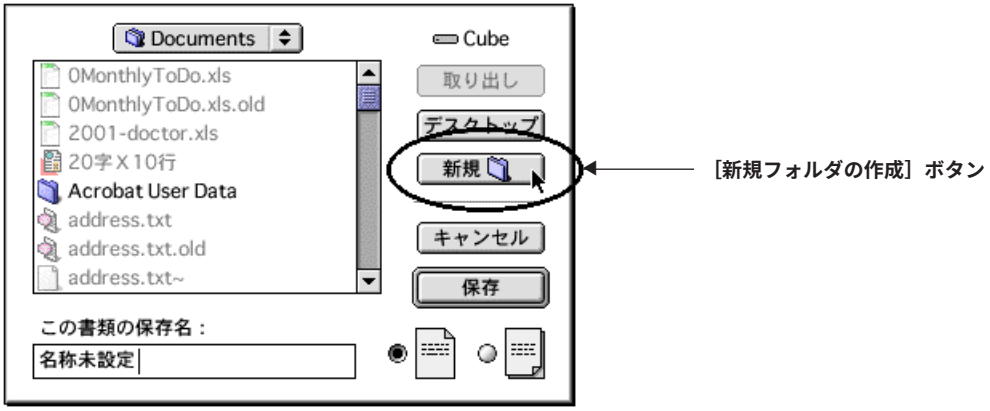
Mac OS XのTextEditなら，「別名で保存...」のダイアログボックスで同じことができる。図0-12の左の図のように，フォルダの内容が表示されない設定になっていたら，下向き矢印をクリックして（図左），内容を表示する設定にしてから，「新規フォルダ」のボタン（図右）を選べばよい。

なお，左下の「拡張子を隠す」にチェックマークが入っていたら，これをとっておいた方がよいだろう。プログラミングをするには，拡張子と付き合っていかなければならない。

それより前のMac OSのSimpleTextを使うのならば，「新規」の後ろにフォルダのアイコンの付いたボタンをクリックすればよい（図0-13）。

図0-12　Mac OS XのTextEditの保存ダイアログ

図0-13　Mac OS 9のSimpleTextの「別名で保存...」のダイアログボックス



拡張子の表示

上で見たように，インターネットや多くのパソコンでは「拡張子」を使って，ファイルの種類を区別している。しかし，WindowsやMacではその設定によっては，拡張子が表示されない。プログラミングをするときには，拡張子を常に表示しておいた方が便利なので，最後にそれを確認しておこう。

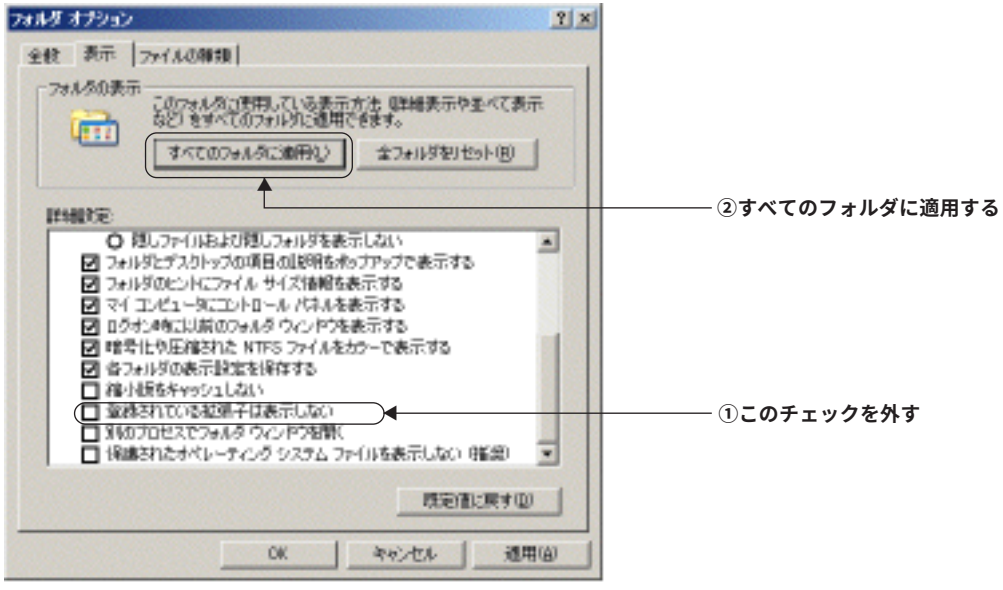
Windowsの場合，通常，画面（デスクトップ）の左上にある「マイ コンピュータ」あるいは「マイドキュメント」のアイコンをダブルクリックしたりして開くと「エクスプローラ」のウィンドウが開いて，ファイルやフォルダがリストされる（デフォルトではアイコンが表示される）。エクスプローラの「ツール」メニューから「フォルダオプション...」という項目を選ぶと小さめのウィンドウが表示される。上に並んでいる「タブ」のうち「表示」のタブを選ぶと，図0-14のような，表示の設定を指定するためのリストが表示される。

このウィンドウを下の方にスクロールしていくと，「登録されている拡張子は表示しない」という項目（オプション）がある。これに，チェックマークがはいっていたら外せばよい。これで，ファイルの拡張子が表示されるようになる。

Macintoshの場合，Mac OS 9あるいはそれ以前の場合は，特に何もしなくても，拡張子は表示される。したがって，そのままで大丈夫だ。Mac OS Xの場合は，Windowsと同じように拡張子が隠せるようになった[脚注]ので，それが隠れないように設定しておく必要がある。

［脚注］初心者が，「拡張子」という難しいものを意識しなくてもすむようにという配慮だろう。しかし，拡張子の有無の違いでファイルが見つからなくなるといった問題も起こる。したがって，この「配慮」が本当に初心者にとって便利かどうかは疑問の余地がある。

図0-14　Windowsの「フォルダオプション」の設定（Windows XPの場合）

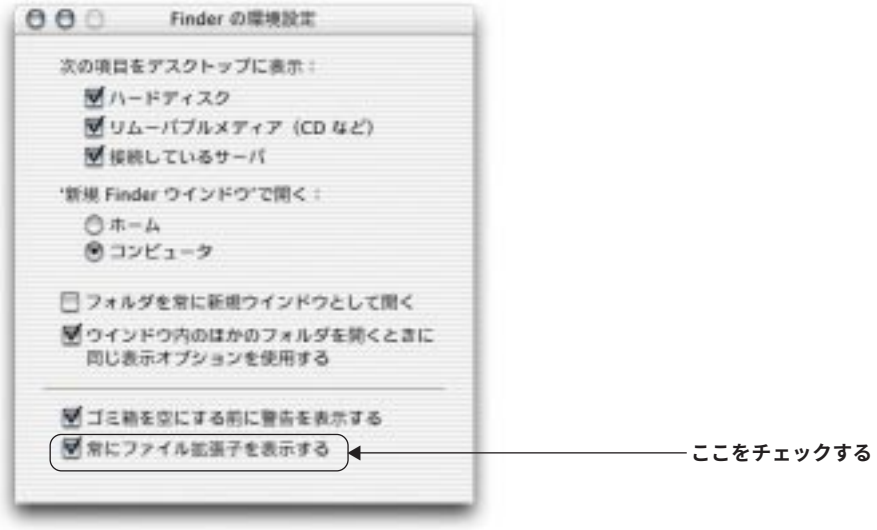


Macの画面右上に表示されているハードディスクのアイコンをクリックすると，Finder（ファインダ）という，ディスクの中身（ファイルやフォルダ）をリストしてくれるプログラムのウィンドウが表示される。そこで，左上のりんごマークのすぐ右にある「Finder」というメニューから「環境設定...」を選ぶと，図0-15のような「Finderの環境設定」のウィンドウが開く。その一番下にある「常にファイル拡張子を表示する」の項目をチェックしておけばよい。

Unix系のOSでは，Mac OS 9の場合と同様，何もしなくても拡張子は表示されるはずだ。

これで，準備は完了だ。次章からプログラミングを始めよう。

図0-15　Mac OS Xの「Finder の環境設定」



まとめ

プログラミング言語とは何か，コンピュータ内で数や色，文字などをどのように表現するを紹介しながら説明した。

■コンピュータ内での表現

数の表現　コンピュータ内部では0と1の並びで表されている

色の表現　色を成分に分けて，それぞれの成分を数字で表して表現する。RGBという形式がもっともよく使われる

文字の表現　文字を数字と対応させて表現する。対応の仕方を記述したものをコード系という

情報の単位

- **ビット**　0か1か，オンかオフかで2種類の状態を表せる。binary digitの略
- **バイト**　8ビットで，多くの欧米の（自然）言語で使われる文字を表現するのに十分な量
- **キロバイト（KB），メガバイト（MB），ギガバイト（GB），テラバイト（TB）**　2^{10}バイト，2^{20}バイト，2^{30}バイト，2^{40}バイトのこと

16進数による表記　1バイト分を，16進数（0－9，A－F）2文字で表現できる

プログラミング言語の種類　用途によっていろいろなプログラミング言語が用意されている。この本で学ぶJavaScript，Perl，Javaなどはウェブ用に便利な機能が揃っている

練習問題

■ 0.00000000000000000000000000000000000000001234といった非常に小さい小数を32ビット（32個の電球）で表現するにはどうすればよいか考えよ。$\frac{1}{2}$，$\frac{1}{4}$，$\frac{1}{8}$，……とビット（電球）を割り振っていっては，32ビットでは足りないことに注意。

■ この章の例ではマイナスの数（負数）は登場しなかった。−1234などをどう表せばよいか考えよ。

※答えは筆者のホームページ（http://www.musha.com/）にある，本書のサポートページで。