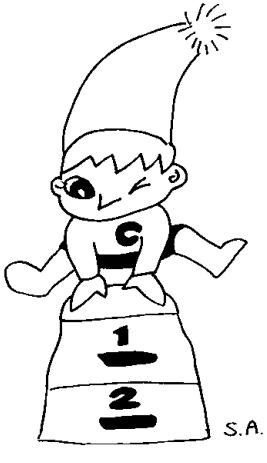
第2章

JavaScriptの第一歩

準備運動が終わったので，いよいよプログラミングの世界に入ろう。まず，JavaScript（ジャバスクリプト）というプログラミング言語を使って，ともかく「プログラム」を作ってみることにしよう。単純ではあるが，HTMLを使ってはできないことを実現してくれるプログラムを作る。



プログラミングを学ぶのに最適の言語――JavaScript？

実は，筆者は，JavaScriptは「ウェブ用」に限らずプログラミングを学び始めるのに最適な言語だと思っている。ほとんどのウェブブラウザにはJavaScriptの機能が組み込まれているので，新しいソフトをまったく買わなくてもJavaScriptのプログラムの勉強を始められる。また，ブラウザの環境と密接に結び付いているため，ブラウザの機能を拝借して，プログラムらしいプログラムを手軽に作成できる。ほかの言語だと，その言語を使えるようにする――処理系を準備する――だけでも初心者にとっては一苦労，さらにウィンドウを開いてボタンなどをきれいにレイアウトしてなどというところまでには何日も，いや何カ月も勉強しないと到達できない。しかし，JavaScriptならば「始めたその日のうちに」というのも決して無理な話ではない。

最初のプログラム――ダイアログボックスの表示

まず「ダイアログボックス」を表示するプログラムだ。56ページの図2-1のような「ダイアログボックス」は，ブラウザに限らずいろいろなソフトで表示される。これはHTMLだけを使ってはできないが，JavaScriptを使えば朝飯前だ。

プログラムを見てみよう。JavaScriptのプログラムは，HTMLファイルの中に埋め込む形で作るので，全体はHTMLのファイルになる。次のように，特別なタグを使ってJavaScriptのプログラム（ソースコード）をHTMLコードの中に埋め込むのだ。

プログラム例2-1　ダイアログボックスの例1――hello1.html

```
<html>
<head>
<title>Hello</title>
<meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
</head>
<body>
<script type="text/javascript" language="JavaScript">
confirm("Hello!");
</script>
<p>ごあいさつでした。</p>
</body>
</html>
```

では，初めて出てきたタグや属性について説明しよう。目新しいのは，次の3行だけだ。

```
<script type="text/javascript" language="JavaScript">
confirm("Hello!");
</script>
```

<s c r i p t>タグは，JavaScriptなどのプログラムを書くときに使うもので，このタグから</script>の前までがプログラムであることを示す。

メモ

JavaScriptのプログラムは「スクリプト（script）」とも呼ばれる。プログラミング言語には，「インタプリタ型の言語」と「コンパイラ型の言語」の2種類があり，JavaScriptなどのインタプリタ型言語のプログラムは「スクリプト」とも呼ばれることがある。この本の後ろで学ぶJavaは、（インタプリタ型言語のような特徴を持ってはいるが，どちらかといえば）コンパイラ型の言語である。

コンパイラ型の言語では「コンパイル」という操作を行ってからプログラムを実行する。これに対して，インタプリタ型の言語ではコンパイルなしに実行できる。つまり，実行しながらコンパイルに相当する操作（コンピュータが直接実行できる低レベルの命令に変換する操作）を行っているのだ。

この<script>タグには属性が2つ指定されている。type属性はスクリプトの型（MIMEコードといわれるもので，この場合はtext形式でjavascriptのコードを表す），language属性はスクリプトの言語（この場合はJavaScript）を指定している。現在のところ，ウェブページで使われるスクリプト言語としてはJavaScriptがもっとも多く使われており，実はこの2つの属性を書かなくても一般的なブラウザならどれでもJavaScriptとして解釈される。そうはいっても将来的なことを考えてtype="..."とlangauge="..."は指定しておこう。

<script>と</script>に挟まれた，次の1行がJavaScriptのプログラムで，これがダイアログボックスを表示するものだ。

```
confirm("Hello!");
```

この行は，confirmという名前の「関数（function）」を呼び出して「Hello!」という文字列をダイアログボックスに表示するJavaScriptの「文（sentence）」だ。JavaScriptのプログラム（スクリプト，ソースコード，あるいは単に「コード」とも呼ばれる）は，このような文が集まってできている。普通の文章（たとえば，この本）も「文」が集まってできているが，プログラムも「文」の集まりなのである。

さて，ものは試し，このページをブラウザで表示してどうなるか試してみよう。本書のホームページからリンクをたどるか，プログラム例2-1のような内容を持つファイルを作り，hello1.htmlという名前でテキスト形式で保存して，ブラウザに読み込む。まず図2-1のようなダイアログボックスが表示され［OK］をクリックすると，続いて図2-2のような文字列がブラウザウィンドウ内に表示さ

れるはずだ。ダイアログボックスの大きさやメッセージ，色などはブラウザによってさまざまなので，この図と少し違っているかもしれない。

図2-1　最初に表示されるダイアログボックス

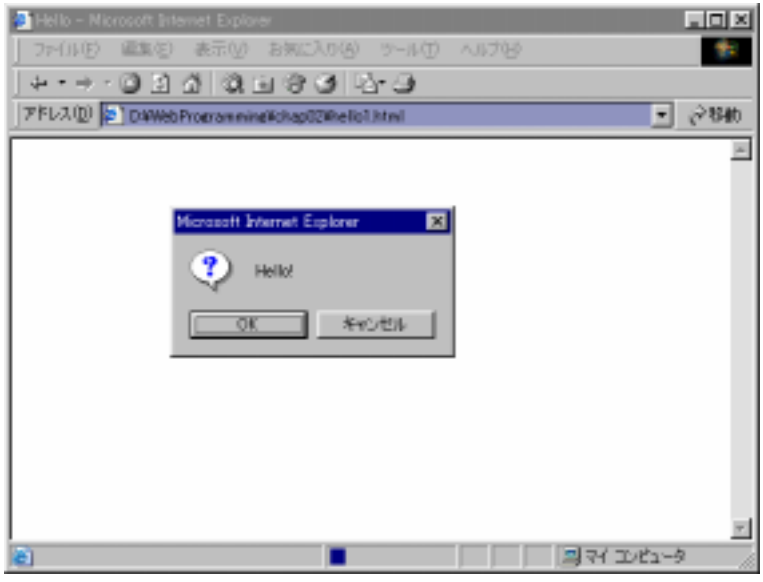
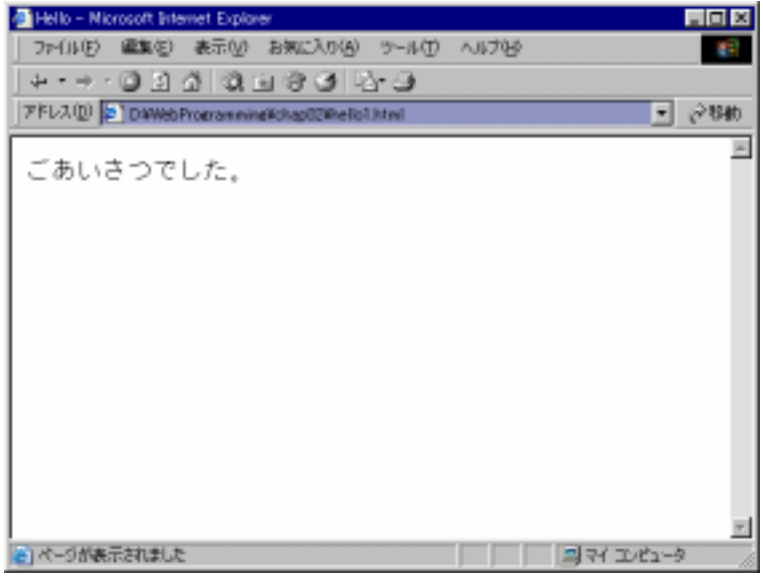


図2-2　［OK］を押すと続いて表示される内容



落とし穴

もしダイアログボックスが出なかった場合は，まず<script>と</script>に挟まれた文字がすべて同じように入力されているかチェックして欲しい。1文字でも違っていると動かないことがある。

また，ブラウザによってJavaScriptの実行が禁止されている可能性もある。この場合，［インターネットオプション］あるいは［初期設定］［設定］などのメニューを選択して表示される画面で，［セキュリティ］［Webコンテンツ］［詳細］などの項目を選択して，［JavaScriptスクリプト（アクティブスクリプト）を有効にする］のところにチェックを入れれば使えるようになる（図2-3，図2-4）。こうしておいてから，［再読込］を実行すれば，大丈夫なはずだ。この設定は，ブラウザやそのバージョンによって異なるので，うまくいかなかったらWEB HELPを参照して欲しい。

図2-3　JavaScriptを有効にする（Windows版Intenet Explorer 6.0の設定画面。［ツール｜インターネットオプション］メニューでウィンドウを表示し、［セキュリティ］タブの［レベルのカスタマイズ］をクリックする）

図2-4　JavaScriptを有効にする（Mac OS X用Intenet Explorer 5.1の設定画面）

これでも直らないときは，<script>...</script>の中に全角のスペースが入っていないか見て欲しい。HTMLについてのときも書いたが，全角のスペースは，半角のスペースとは違って区切り文字とは扱われないので（"ab"などの余計な文字が書いてあるのと同じことになってしまい），エラーになる。

引数――関数の動作を変化させる

上の例ではconfirmに「Hello!」という文字列を渡したが，これを別の文字列にすれば，当然表示される文字列も変わる。このように，関数に「渡す」もののことを「引数（ひきすう）」あるいは「パラメータ（parameter）」と呼ぶ。では，引数を次のように「Hi!」に変えて同じように実行してみよう。今度は「Hi!」という文字列が表示されたはずだ。

```
confirm("Hi!");
```

HTMLとして書く――document.write

今度は，confirmのかわりに，別の関数を使ってみよう。プログラム例2-1の<script>...</script>の部分を次のようにして，document.writeという関数[脚注]を使おう。

```
<script type="text/javascript" language="JavaScript">
document.write("<h1>Hello!</h1>");
</script>
```

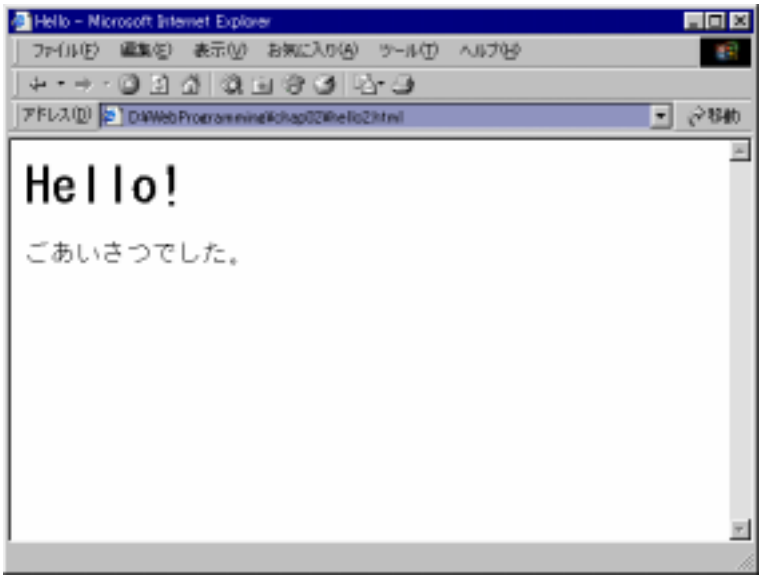
これをブラウザで表示すると，図2-5のようになる。

document.writeを使うと，引数に指定した文字列をHTMLのコードとして書き出す（出力する）ことができる。つまり次のように書いてあったのと同じことになるのだ。

```
<html>
<head>
<title>Hello</title>
<meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
</head>
<body>
<h1>Hello!</h1>          <!--     ←ここが入れ替わった  -->
<p>ごあいさつでした。</p>
```

［脚注］本当は「関数」というよりも「メソッド」と呼ぶ方が適当なのだが，しばらくは「関数」と呼んでおくことにする。メソッドも関数ではあるので，間違いというわけではない。

図2-5　document.writeを使う




```
</body>
</html>
```

この程度ならHTMLで直接上のように書けばよいので，わざわざJavaScriptを使う必要はないが，後の方でdocument.writeのありがたみが，だんだんわかってくるはずだ。ここでは，頭の隅に入れておいて欲しい。

メモ

あっさり説明したが，HTMLの文書（コード）は，第1章では我々（人間）が書いた。しかし，ここでは，プログラムが書いていることに注意して欲しい。第1章で説明したウェブエディタも，人間がワープロと同じように入力するものを，裏で（一所懸命）HTMLコードに直しているのだ。そして，HTMLコードに直すためのプログラムは，人間が書いたものなのである。

なにやら，話がこんがらがってしまったと思うが，ようするに，JavaScriptは（人間のかわりに）HTMLのコードを書いてHTML文書を作成することができるのだ。だから，人間ならとても書く気にならないような複雑な文章や，単純な作業の繰り返しによってできあがるきれいな表なども，プログラムを作ればいくつでも，好きなだけできる。これが，JavaScript（など，ウェブ用のプログラム言語）のもっとも強力なところだ。

イベント駆動型のスクリプト――何かが起こったときに実行される

上の例では，<body>...</body>のあいだに，<script>...</script>を埋め込んで，JavaScriptのスクリプトを実行したが，スクリプトの実行方法には別種のものがある。上で見たものを「<body>部埋め込み型のスクリプト」と呼ぶことにすると，以下で紹介するのは「イベント駆動型のスクリプト」と呼ぶことができるものだ。

これも，例を見よう。

プログラム例2-2　ダイアログボックスの例2――hello3.html

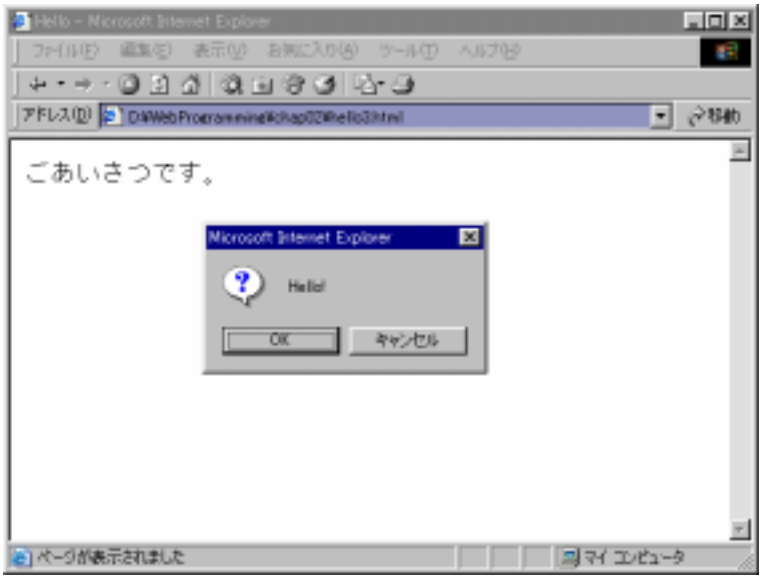
```
 1  <html>
 2  <head>
 3  <title>Hello</title>
 4  <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
 5  <script type="text/javascript" language="JavaScript">
 6
 7  function sayHello() {
 8      confirm("Hello!"); // ダイアログボックスを表示
 9  }
10  </script>
11  </head>
12
13  <body onload="sayHello()">
14  <p>ごあいさつです。</p>
15  </body>
16  </html>
```

上のプログラム（スクリプト）をファイルに入力して保存し，実行してみよう（各行の頭にあるのは，説明のための行番号だ。これは入力しないこと）。図2-6のように表示されるはずだ。

結果は最初のプログラム（プログラム例2-1）とほとんど同じだったはずだ。しかしプログラムの内容は違っている。先ほどは<body>...</body>に囲まれていた，<script>タグが<head>...</head>に囲まれている。また，<body>タグにonload属性が指定されており，その値が"sayHello()"になっている。だいたい想像が付くかもしれないが，このページは次のように動作する。

1. <head>...</head>にあるJavaScriptのコードが解釈され，記憶される。
2. <body>...</body>のあいだの文字列やタグが解釈され，ウィンドウに文字列「ごあいさ

図2-6　「イベント駆動型」のスクリプトの実行



つです。」が表示される。

3. <body>タグのonload属性に指定されている関数sayHello()が実行される。

この結果，図2-6のようにダイアログボックス内に「Hello!」が，ウィンドウ内に「ごあいさつです。」の文字列が表示されるのである。

メモ

プログラム例2-1とプログラム例2-2では，実は少し動作が異なっている。気付いたであろうか？　2-1ではダイアログボックスが表示された後にブラウザに文字が表示され，2-2ではダイアログボックスが表示されているときにはすでにブラウザに文字が表示されている。本文でも簡単に説明されているが，プログラムには同じ動作をするのにいくつかの方法があり，また似たような動作をするが微妙に異なる場合があるのである。

onload属性の値として指定されている関数sayHello()だが，これは7行目のfunctionの後に書かれている文字列と同じであることに注意して欲しい。onload="xxx"と指定された場合，xxxという名前の関数が実行される約束になっているのだ。この例は，ブラウザにページがロードされるという「イベント（出来事）」に付随する形で起動されるスクリプトである。

メモ

ここから，この本では関数（やメソッド）の名前には( )を付けてそれが関数であることを明示する。まだ，それほど混乱はしないが，長いプログラムを書くようになると，このような約束事をしておいた方がわかりやすくなる。

このプログラムの7行目から9行目はsayHello( )という名前の関数の「定義」で，これがHTMLファイルの読み込み時に実行される。sayHello( )の後にある「{」は，sayHello( )の定義がここから始まることを示すもの。HTMLでいえば開始タグだ。同様に最後の「}」は，終了タグのようなもので，sayHello( )の定義の終わりを示す。内容は，おわかりだろう。confirm( )はダイアログボックスを表示する関数だから，引数として"Hello!"を渡されれば，この文字列をボックス内に表示する。

confirm( )の呼び出しの後にある「 // ダイアログボックスを表示」は「コメント」で，プログラムの動作にはまったく関係しないものだ。「//」から行の終わりまでに書かれた文字は（JavaScriptを解釈するプログラム――インタプリタ――には）無視される。HTMLではコメントを<!-- と-->に挟んで書いた。言語によってコメントの表し方はいろいろなのだ。

機械にとっては無意味でも，人間にとってはこのようなメモがあると，他の人が作ったプログラムを読む場合や，自分が作ったプログラムを1カ月後に（あるいは筆者のように20年後に！）読む場合に，何をしているのかを理解するための大きな助けになる。

メモ

JavaScriptではもう1種類「/* ... */」という形式のコメントも書くことができる。この形式の場合...の部分は複数行にまたがって記述できる。たとえば次のようなものもコメントだ。

```
/*********************************************
 *        『プログラミングは難しくない！』サンプル
 *        作成者　難野何兵衛
 *        作成日 2002.02.14
 *********************************************/
```

条件により処理を変える——if文

さて，もう少しおもしろくしてみよう。最初は英語で「Hello!」といったが，今度は日本語にしてみよう。英語だといつも，Hi!とかHello!ですんで便利だが，日本語は，時刻によってあいさつを変えるので，次のようにダイアログボックスのメッセージを変えてみる。

- 午前4時から午前10時前までは「おはようございます」
- 午前10時以後，午後7時前までは「こんにちは」
- それ以外の場合は「こんばんは」

ダイアログボックスにメッセージを表示するには，同じ関数confirm()を使えばよい。現在の時刻を見て，それに従ってconfirm()に渡す引数を変えて呼び出せばよいわけだ。ページが表示されるときに，onload()で呼び出すことにして，同じ関数sayHello()の定義だけを書き換える。

メモ

sayHello()という名前を付けた関数はいつもまったく同じ働きをしなければいけない，ということはない。名前を同じにして機能を変更（追加）していってもよいし，途中で名前を変えてもかまわない。ただし，「名は体を表す」ようにしておかないと，後で読んだときに他人だけでなく自分も混乱する。その関数の機能が変わってしまったら名前も変えるのが原則だ。また，関数の名前を変えたときは，<body>タグの属性onloadの値も同じように変更することを忘れないこと。

プログラム例2-3　日本語でのあいさつの例——hello4.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello</title>
<meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
<script type="text/javascript" language="JavaScript">

function sayHello() {
    var today; // 日付に関する「オブジェクト」を保持する変数todayを宣言
    var hour; // 時刻を保持する変数hourを宣言
    today = new Date();  // Dateオブジェクトを作りtodayに「代入」
    hour = today.getHours();  // todayから時刻を取り出してhourに代入

    if (4 <= hour && hour < 10) {  // 午前4時以降10時前まで
        confirm("おはようございます。");
```

```
16      }
17      else if (10 <= hour && hour < 19) {  // 午前10時以降午後7時前まで
18          confirm("こんにちは。");
19      }
20      else { // その他の場合。つまり，午後7時以降，翌朝7時前まで
21          confirm("こんばんは。");
22      }
23  }
24  </script>
25  </head>
26
27  <body onload="sayHello()">
28  <p>ごあいさつです。</p>
29  </body>
30  </html>
```

メモ

このリストの先頭の「<!DOCTYPE ...」で始まる行は，2行に渡っているが，これは印刷の都合で，実際は途中に改行を入れないで1行に書いてある（途中で，Enterあるいはreturnキーを押さずに書いた）。1行に書いても，エディタやワープロの設定によっては，画面上でも折り返されてしまうが，自分で改行を入れた場合とは意味が異なる。

以後，このように行番号が飛んでいる場合は，前の行の続きであることを示す。その場合，改行を途中に入れないで入力して欲しい。改行を入れてしまうと，うまく動かなくなったり，動作が違ってしまう場合がある。

違っているのは，sayHello()の定義だけなので，この中だけ説明しよう。まず，最初の2行。

```
 9      var today; // 日付に関する「オブジェクト」を保持する変数todayを宣言
10      var hour; // 時刻を保持する変数hourを宣言
```

この2行はいずれも「変数」を「宣言」している「文」だ。変数をどう使うかは以下で追い追い説明する。まずここでは，「このプログラムでは，todayとhourという変数を使いますよ」と宣言している。varはvariable（変数）を省略したもので，「予約語（reserved word）」と呼ばれるJavaScriptのプログラムで特別な役目をする単語だ。

メモ

なぜ変数を宣言するかというと，まずプログラムを読む人に，「このプログラムではtodayとhourという変数を使いますよ。覚えておいてください」と伝えるためだ。同時に，プログラムを実行してくれるJavaScriptの「処理系」に対して，「このプログラムではtodayとhourという変数を使うので，そのための場所を用意しておいてください」と伝える役目もする。

実はJavaScriptやこの後学ぶPerlなどの言語では，変数は宣言せずに使ってもよい。JavaScriptやPerlはどちらかというと，気軽に小さなプログラムを作って実行してみようという雰囲気の言語なので，これらの言語を設計した人はいちいち変数の宣言はしなくてもよいことにしたわけだ。

これに対して，C，C++や，この本の終わりの方で学ぶJavaなどの言語では使用する変数は必ず宣言する。なぜ面倒なことをあえてやるのかというと，変数を宣言することによりエラーを未然に防げるからだ。たとえば，宣言することにより，変数の入力ミスなどを比較的簡単に見つけられる。上のプログラムの14行目でhourをourと間違えて入力してしまっても，JavaScriptではエラーにならず実行できてしまうが，ourには時刻ではなく（自動的に）0が入るためプログラムがおかしな動作をしてしまう。これに対して，Javaなどの言語では，ourと書いてあると「宣言されていない変数が使われている」というエラーが出てプログラムは実行できないのである。

簡便さを優先するか，エラーを未然に防ぐ方を優先するか，言語によってその設計思想が違っているわけで，どちらがありがたいか，一概にはいえない。用途によってどちらの言語を使うかを考えて使うのがよいであろう。

では，次の行。

```
11      today = new Date();  // Dateオブジェクトを作りtodayに「代入」
```

new Date()によって，Dateという日付に関するさまざまな情報を保持している「オブジェクト」が生成され，そのオブジェクトが変数todayに「代入」されて記憶される。「オブジェクトとはいったい何か？」を説明するのにはかなりのスペースが必要なので，これはもう少しプログラミングについていろいろな知識を身に付けてから第8章や第14章で説明する。ここでは，以下で説明するように使えると覚えておいて欲しい。

メモ

日常生活においてもそうだと思うが，プログラミングを学ぶ際には，細かいことは置いておいて，当面無条件で覚えた方が楽なことが結構ある。そういったことをいちいち突き詰めていると，まったく先に進めなくなる場合もあるので，いったん「棚上げ」して，後で棚から下ろして解決するという姿勢も大切だ。ただ，何から何まで棚上げしていては，これまたまったく知識が身に付かないので，程度問題ではある。ここでは，オブジェクトとはいったい何なのか，その定義は，といったことはいったん棚上げして，単にDateというのはオブジェクトなのだと覚えておいて欲しい。そして，Dateを使えば，どのようなことができるのか「雰囲気をつかむ」ことに集中して欲しい。

変数に何かを「代入」するというのは，その変数用に用意されている記憶領域（箱）にその何かを保存しておくことをいう。たとえば，上の例ではnew Date()によって生成されたDateオブジェクトを扱うのに必要な情報が，todayという名前が付けられた場所に記憶されるわけだ。

次の行は，todayに代入されたDateオブジェクトを活用した例だ。

```
12      hour = today.getHours();  // todayから時刻を取り出してhourに代入
```

todayにはDateオブジェクトが記憶されているわけだが，Dateオブジェクトには，getHours()という関数（メソッド）が用意されている。today.getHours()で，現在の時刻（24時間制）を取り出して返してくれる。したがって，たとえば，午後1時7分25秒にこのプログラムを実行した（このHTMLページをブラウザに読み込んだ）とすれば，today.getHours()からは13が戻ってきて，この値が変数hourに記憶される（ちなみにtoday.getMinutes()，today.getSeconds()なども用意されていて，それぞれ7，25が戻ってくる。さらに，年や月を得るためにtoday.getFullYear()，today.getMonth()なども利用可能だ）。

メモ

代入は「=」を使って表すが，この「=」は，プログラミング以外の「普通の世界」での使われ方とは違っている。普通の「=」は，左辺と右辺が等しいことを表す（たとえば，19－3＝16）。しかし，JavaScriptの「=」は代入を表すもので，右辺で計算した結果（や右辺でできたオブジェクトなど）を，左辺の表す場所に記憶するという意味を表すだけなのだ。

次の文を見よう。

```
x = 3;
```

JavaScriptでは，これはxが3である（3と等しい）ことを表しているのではなく，3という値がxという名前を付けられた場所に記憶されるということを表しているものなのだ。気持ちとしては，次のような表記の方がピッタリだ。

```
x ← 3;
```

メモ

```
hour = today.getHours();
```

など，文の最後に「;」が付いているが，JavaScriptではこの「;」を省略できる。つまり，次のように書いてもOKだ。

```
hour = today.getHours()
```

次のように，同じ行に複数の文を書くときは，間に「;」を入れて，区切りを明示する必要がある。

```
today = new Date();   hour = today.getHours()
```

1行にいくつの文を書いてもよいが，普通はせいぜい2文くらいしか書かない[脚注]。なお，JavaScript以外の多くのプログラミング言語では，文の最後の「;」は，たとえ1行に1文しか書かれていなくても省略はできない。本書では，これからほかの言語も学ぶので，習慣にする意味も込めて，「;」を省略しないで書くことにする。

13行目にあき（空行）があるのは，この上とこの下で処理の内容が若干異なるため，区切りの意味だ。このように，いろいろな工夫をしてプログラムの内容を把握しやすくすることはとても大切だ。

さて，次がこのプログラムの肝だ。

```
14      if (4 <= hour && hour < 10) {  // 午前4時以降10時前まで
15          confirm("おはようございます。");
16      }
17      else if (10 <= hour && hour < 19) {  // 午前10時以降午後7時前まで
18          confirm("こんにちは。");
19      }
20      else { // その他の場合。つまり，午後7時以降，翌朝7時前まで
21          confirm("こんばんは。");
22      }
23  }
```

ifは，「もし～ならば」という意味だから，「if (4 <= hour && hour < 10)」で，「もしhourが4以上で」&&「hourが10より小さい」という条件が満たされれば，ということになる。「&&」は「かつ」を意味する記号で（この2文字でandと読む），この場合，まとめると「もしhourが4以上で，かつ，hourが10より小さい場合」に，その下の行，つまり

```
15          confirm("おはようございます。");
```

を実行することになる。

elseは，「でなければ」という意味なので，else ifで「でなくて，もし～ならば」ということになる。この場合，「『もしhourが4以上で，かつ，hourが10より小さい場合』でない場合で，かつ，『もしhourが10以上で，かつ，hourが19より小さい場合』」にその下の行，つまり

```
18          confirm("こんにちは。");
```

[脚注] このときの「行」とは改行文字（EnterあるいはReturnのキーを入力したときに入る文字）によって区切られた「行」をいう。エディタやワープロソフトで折り返されて2つの「行」に渡って表示されていても改行文字が入っていなければ，プログラムでは1「行」として扱う。プログラムの処理系（インタプリタやコンパイラ）は改行文字までをひとつの単位として処理をするのだ。

が実行される。

最後のelseは，「でなければ」というわけで，上の2つのいずれにも入らない場合，

```
21          confirm("こんばんは。");
```

を実行する。

メモ

ifもelseも，varと同じように予約語で，特別な目的に使われる。このため，if，else，varなどの名前を変数として使うことはできない。たとえば次のような文はエラーになってしまって実行できない。

```
if = 3;
var = else + if;
```

if文の形式

このようにif ... else if ... else ...を使うといろいろな場合に分けてそれぞれの場合でどのような処理をするかを記述できる[脚注]。まとめると，if文は次のような形式を持つ。

```
if (<条件1>) {
    <処理部1>
}
else if  (<条件2>) {
    <処理部2>
}
else if  (<条件3>) {
    <処理部3>
}
...
else {
    <処理部n>
}
```

else if ...の部分はいくつでも書くことができるし，ひとつもなくてもかまわない。また，else ...の部分も省略可能だ。if文が使われないプログラムはまずないというほどよく使われるので，これからもいろいろな例が登場する。else ...が省略されているものなどもそのうちお目にかかるはずだ。

[脚注] このような処理を「分岐（ぶんき）」処理などと呼ぶ。if文は分岐処理に使われるわけだ。

<処理部>には，ひとつ以上の文が並ぶ。つまり，if文の中には複数の文がその要素として入るわけだ。具体例は以下にあるので，しばらくお待ちを。

メモ

実は<処理部>に文をひとつだけ書けばよいというときは，「{」と「}」を省略できる。しかし，経験上，こうするといろいろな問題を引き起こす誘因になるので，たとえ単純な文がひとつだけの場合でも「{」と「}」で囲むことを勧めたい。なぜかは，「棚上げ」しておこう。そのうちわかる。

条件の条件

<条件>の部分には上で見たように真（true）か偽（false）になる条件を書く。たとえば，「4 <= hour && hour < 10」という式は，hourが4以上10未満の場合にtrue（真）となり，それ以外の場合false（偽）となる。JavaScript（やこの後に登場するPerl）には，ちょっと奇妙なところがあって，この条件が数字や文字列でもよい。たとえば次のようなif文も構文的には「正しい」文だ。

```
if (i+j) {
  <実行部>
}
```

この場合，「i+j」を実行した結果（「評価」した結果）が0になれば，ifの条件としては偽（false）と判定される。それ以外の値になった場合は真（true）として扱われる。<条件>が文字列でもかまわない。その場合，空文字列（""）以外は真として扱われるのだ。慣れないと，とても奇妙な感じがするが，まあ，そのうちだんだん慣れるので，心配はいらない。

メモ

プログラムは一般的には上から下に順番に実行される。ただし，if文の例でわかるように，いつもすべての部分が実行されるわけではなく，飛ばされてしまう部分もある。また，次の章で見るように，同じ部分が何回か繰り返される場合もある。ただし，その繰り返される部分についてみれば上から下に順番に実行されることに変わりはない。

メモ

前章のHTMLのところでも説明したが，プログラムを書くときも字下げ（インデント）をしてわかりやすくすることはとても大切だ。

```
if (4 <= hour && hour < 10) {  // 午前4時以降10時前まで
    confirm("おはようございます。");
}
```

たとえば，上の例ではifの条件が満たされた場合に実行されるのが，confirm( )だけであることが一目瞭然だ。この後の例ではもう少し複雑になる。字下げの効果をよく確かめて欲しい。

なお，字下げにもいろいろな流儀がある。たとえば，次の例のように「{」を別の行に書く人もいる。

```
if (4 <= hour && hour < 10)    // 午前4時以降10時前まで
{
    confirm("おはようございます。");
}
```

大事なことは，自分の流儀を確立して，それを一貫して使うことだ。少なくとも同じファイルの中では同じ流儀で書こう。

さらにif文——if文の入れ子

さて，夏と冬で暗くなる時間はだいぶ違う。筆者が今まで住んだことのある日本の中央部では冬には6時頃には真っ暗で，午後6時30分に「こんにちは」はちょっと変な感じだ。そこで，上のプログラムを「改良」して，4月から9月までは上と同じ，これ以外の月については，1時間早めて午後6時までは「こんにちは」としてみよう[脚注]。

プログラム例2-4　季節を考慮した日本語でのあいさつの例——hello5.html

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
 2  <!-- hello5.html -->
 3  <html>
 4  <head>
 5  <title>Hello</title>
 6  <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
 7  <script type="text/javascript" language="JavaScript">
 8
 9  function sayHello() {
10      var today;
11      var hour;
12      var month; // 月を保持する変数monthを宣言
13      today = new Date();  // Dateオブジェクトを作りtodayに代入
14      hour = today.getHours();
15      month = today.getMonth()+1;  // todayから月を取り出してmonthに代入
16
17      if (4 <= month && month <= 9) {   // 4月から9月まで
```

[脚注] こうしても，12月頃だと5時過ぎには暗くなってしまうので，まだ変だが，そこは我慢されたい。

```
18          if (4 <= hour && hour < 10) {
19              confirm("おはようございます。");
20          }
21          else if (10 <= hour && hour < 19) {
22              confirm("こんにちは。");
23          }
24          else { // 午後7時以降，翌朝4時より前まで
25              confirm("こんばんは。");
26          }
27      }
28      else {   // 1～3月と10～12月
29          if (4 <= hour && hour < 10) {
30              confirm("おはようございます。");
31          }
32          else if (10 <= hour && hour < 18) {
33              confirm("こんにちは。");
34          }
35          else { // 午後6時以降，翌朝4時より前まで
36              confirm("こんばんは。");
37          }
38      }
39  }
40  </script>
41  </head>
42
43  <body onload="sayHello()">
44  <p>ごあいさつです。</p>
45  </body>
46  </html>
```

新しいところだけ見ていこう。15行目のtoday.getMonth()は，上でちょっと紹介したが，月が返ってくる。注意しなければならないのは，0から11のいずれかの値が返ってくることだ。1を足さないと，日本などで使われている「月」の数字とは合わなくなる。

メモ

第0章で紹介したように，コンピュータで扱うときは，0から始めるのが「便利」だということになっていて，プログラミング言語の世界では，多くのものが1からではなく，0から始まる。しかし，月を0から始めるのは，さすがにやりすぎだと思う。東洋系の人が決めたら，きっと1から12にしただろう。なんと思いやりのない人たちなのだろう。自分たちは数字を月には使わないのだから，数字を月に使う人たちに合わせてくれてもよさそうなものなのに……

このプログラムでは，if文の中にさらにif文が入っている。つまり，if文が入れ子になっているわけだ。外側のif文は次のような構造をしている。

```
17      if (4 <= month && month <= 9) {   //4月から9月まで
            <A>
28      else {   // 1～3月と10～12月
            <B>
```

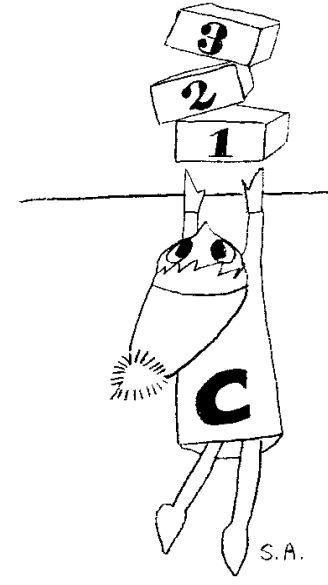
<A>の部分は4月から9月までのあいだという前提のもとで実行されるし，<B>の部分はそれ以外のときという前提のもとで実行される。<A>，<B>の部分はそれぞれ前に見たのとほとんど同じなので説明は省略しよう。

このように，if文を入れ子にする（ネストする）ことによって，いろいろな場合分けをして処理ができる。第1章でHTMLのリスト（<ul>と<ol>）の入れ子が出てきたが，コンピュータの世界ではこの概念は頻繁に登場する。「入れ子の場合は字下げに気を付ける」というのも，共通だ。

メモ

「文の入れ子」は，日常使う「自然言語」にもある現象だ。「私は，正君が昨日韓国へ行ったことを知っています。」という文は，全体でひとつの文になっているが，この中に含まれる「正君が昨日韓国へ行った」というのも，主語と述語を持つ文である。

自然言語の場合は，「私は，克子さんが，正君が昨日韓国へ行ったことをうらやましがっていることを知っています。」のように「文を含む文を含む文」が現れることはあまりないが，プログラミング言語の場合は，このような入れ子の構造が何重にもなる。if文の中にif文があり，さらにその中に（次の章で出てくる）whileループが，さらにその中にforループが入るといった構造もまれではない。処理をするのは人間ではなくコンピュータなので，何重の入れ子になっても問題なく処理してくれるのだ。



実は，このような何重にも入れ子になった文を処理をするときに，コンピュータも「棚上げ」をする（この棚のことを「スタック」と呼ぶ）。最初の文の処理中に，入れ子になっている2番目の文の始まりが出てきたら，最初の文の処理を棚上げして（スタックに積んで），2番目の処理に移る。その処理の途中で3番目の入れ子が始まったら，2番目の文の処理も棚上げして，3番目の処理を行う……といった具合だ。*n*番目（一番内側）の文の処理が終わったら，スタックに積んだ*n*－1番目（ひとつ手前）の処理を再開する。……と進んで，最初の文の処理が終われば，*n*重の入れ子の文の処理が終わるのである。

いろいろな「場合分け」

さて，プログラム例2-4では，最初に「4月から9月まで」と「それ以外の月」とに分けて，さらに時刻によって処理を分けたが，時刻の方に最初に注目することもできる。そうした場合，たとえば次のように処理しても同じことになる（17〜38行に相当）。

```
if (4 <= hour && hour < 10) {
    confirm("おはようございます。");
}
else if (10 <= hour && hour < 18) {
        confirm("こんにちは。");
}
else if (18 <= hour && hour < 19) {
    if (4<=month && month<=9) { // 4月から9月
        confirm("こんにちは。");
    }
    else { // 1〜3月，10〜12月
        confirm("こんばんは。");
    }
}
else { // 午後7時以降，翌朝4時より前まで
    confirm("こんばんは。");
}
```

夏と冬で分けなければいけないのは，18時から19時の間だけなので，この部分を分けて処理しているわけだ。ほかのときは夏と冬で分ける必要がない。おわかりいただけるかな？

さらに，少し複雑になるが，次のようにしてもよい（もともと20行必要だったのが10行になる！）。

```
if (4 <= hour && hour < 10) {
    confirm("おはようございます。");
}
else if ((10 <= hour && hour < 18) ||
         ((18 <= hour && hour< 19) && (4<=month && month<=9) ) )  {
    confirm("こんにちは。");
}
else {
    confirm("こんばんは。");
}
```

この例では，「おはようございます。」「こんにちは。」「こんばんは。」を出力するときはいつかを考

えて，それぞれの条件を分けた。「||」は「または」を表すものでこの2文字でor（オア）と読む。すぐ上の例では，「こんにちは。」を出すところと「こんばんは。」を出すところが2カ所ずつあるが，この例ではどれも1カ所ずつになっている。

「こんにちは。」を表示する条件は複雑だ。

```
else if ((10 <= hour && hour < 18) ||
         ((18 <= hour && hour< 19) && (4<=month && month<=9)))
```

これは，「10 <= hour && hour < 18」が成り立つ場合，あるいは「(18 <= hour && hour< 19) && (4<=month && month<=9)」が成り立つ場合に実行される。2つ目の条件はさらに分けると「18 <= hour && hour< 19」であって，かつ「4<=month && month<=9」である場合に成り立つことになる。2つ以上の条件が複雑に組み合わさる場合は，「(」と「)」を使って，強い結び付きの方をまとめるようにしよう。結局，「10時から18時のあいだであるか，あるいは18時から19時のあいだであって，4月から9月のあいだである」場合ということを表現していることになる。

メモ

この例の場合，実は，まとめるための「(. . .)」を使わずに次のように書いても同じように動作する。

```
else if (10 <= hour && hour < 18 ||
         18 <= hour && hour< 19 && 4<=month && month<=9)
```

「&&」，「||」，「<」などの「演算子（オペレータ）」には，結び付きの強さの順番（優先度）が付けられていて「<=」や「<」は「&&」よりも優先度が強く，「&&」は「||」よりも優先度が強いのだ。しかし，「(. . .)」を使うと優先度に関わらず結び付きの強さを制御できる。こうしておくと，優先度の強さに関係なくまとまりを明示できるし，勘違いを防げるので，とくに複雑な条件の場合は，「(. . .)」を使って，まとめることをお勧めする。

プログラミングが初めての人なら，一番わかりやすかったのは，おそらく，夏と冬をまず分けてしまう最初の方法だったろう。だが，経験を積んでいくと，一番最後のように「処理ごとに場合分けする」という習慣が付いてくるはずだ。

「処理ごとに場合分けする」方が，プログラムは短くなるし，わかりやすくなるのが普通だ。しかし，例外もあり，場合分けが複雑になりすぎて理解が困難になることもある。いずれにしろ，どういう場合分けをしたのか，論理的に明確にしておくことが大切だ。理解を助けるために，コメントも加えよう。

このように，同じ処理を実現するのに，いろいろな方法（アルゴリズム）が考えられる。複数のア

ルゴリズムを思いついた場合，とくに最初のうちは，どうすれば一番わかりやすくなるかを考えて決めるのがよかろう。プログラムを書くときに，まず留意するのは，それが自分にとって，そして他の人にとってわかりやすいかどうかなのだ。

よくあるエラー

プログラミングはエラーとの戦いだ。ここまでの例はいずれも比較的短いものばかりだが，何も問題がなくすぐに実行できたという人は，とても優秀な人か，とても運がよかった人だろう。

プログラムを作成する際には，まず頭の中でどうすれば問題――たとえば，時刻に従って3つのあいさつのいずれかを出す――を解決できるかをしっかり考える必要がある。しかし，考えておいても実際にプログラムを作ってみるとすぐにはうまく動かない。そのような場合，まず疑ってかかるのは次のような単純なエラーだ。

- **引用符（「"」や「'」）やかっこ（「{}」や「()」がきちんと組になっていない。**――たとえば，引用符がきちんと組になっていないと，処理系（コンパイラやインタプリタ）はとにかくペアになっている引用符を探そうとして，とんでもないところまでが引用符に囲まれていると思い込む。この結果，訳のわからないエラーメッセージが表示されることになる。
- **変数や関数（メソッド）のつづりが間違っている。**――たとえば，`hour`を`our`とつづったり，`aisatsu`を`aisatu`とつづったりしてしまうこの種のエラーは，変数の宣言を必ず行う必要がある言語だと，大部分は処理系が見つけてくれるが，JavaScript（やPerl）などの場合，その場でエラーにはならない。つづりを間違えた変数には，数値の0や文字列の""（空文字列）が入ってしまうのだ。
- **複数の箇所に使われている変数や関数の名前の変えたときに，1カ所（一部）だけ直して，すべてを変更するのを忘れる。**
- **ほかの言語と間違えて，違うキーワードを使ってしまう。**――これはとくにウェブ用のプログラムを作っているときにはいろいろな言語を使う必要があるので，注意しなければならない。付録Bに各言語で使う主な構文の一覧をあげたので，将来参考にして欲しい。

そして，さらに問題なのは，こういったエラーをしたときに処理系が人間と同じように思考してくれないことだ。引用符を閉じ忘れてとんでもないところまで引用符の終わりを探しに行って，そのあげくに「オブジェクトが見つかりません」などと訳のわからないメッセージが出たりする（日本語版のブラウザでも，プログラムのエラーメッセージは英語で書かれていることもある）。処理系は，鍵になる文字列を頼りに「構文解析」を行っていくので，その鍵が見つからないとどこまでも探しに行ってしまったりするのだ。

どのような場合にどのようなエラーや症状になるかは，JavaScriptの処理系によって（つまりブラウザのバージョンや動いているOSによって）だいぶ違うし，これからも変わっていくはずなので，この本のホームページで代表的な例を紹介する WEB HELP 。いずれにしろ，何度かエラーを経験して慣れてくると，どのようなエラーをするとどのようなメッセージが出るか見当が付くようになってくる（なってこなければ，プログラマとして生きていくのは難しかろう）。

プログラミングと英語

さて，上で見てきたように，プログラミングにはif，elseなどの英単語が使われている。また，上の例では変数の名前にも英単語を用いた。予約語に英語が使われているのは，コンピュータの世界の「共通語」が英語だからだろう。一時期，予約語にも日本語を使ったプログラミング言語を作ろうという試みがいくつかあったが，結局のところ，ほとんど広まらず，世界で広く使われている言語はすべて英語が予約語として使われている。

変数名や関数名としては，日本語（漢字やひらがな，カタカナ）が使える言語もあるが，多くの言語では使えない。JavaScriptでも，変数や関数の名前は半角の英文字あるいは「_」（アンダースコア）で始まり，英字または数字，または「_」が続いたものと決められている。バージョンによっては「$」（ドル記号）も使えるが，あえて使う必要はなかろう。

最近のコンピュータでは，ファイル名に日本語を使うこともできるのが普通だ。しかし，HTML文書やJavaScriptのプログラムを保存する場合は，ファイル名としてアルファベットや数字，それにハイフン（-）やアンダーバー（_）だけを使っておく方がよい。たとえば，ホームページを公開するプロバイダのコンピュータが日本語をきちんと処理してくれずに，転送したときにおかしな文字に変わってしまう（「文字化け」する）場合があるからだ。こうなると文書が表示できないので，ファイル名に日本語は使わずに，hello5.htmlといったようにしておくのがよい[脚注]。

ところで「文」を区切るのになぜ「.」ではなく「;」を使うのだろうか？　「.」は3.14など小数点としても使われる。小数点を表す「.」なのか，文の終わりを表す「.」なのか，紛らわしくなるのを避けたかったのだ。プログラムでは「曖昧（あいまい）さ」を扱うのがとても難しい。そのため，何でも，一意（ユニーク）に決まるようにしておくと処理がとても楽になる。

では文の終わりを表す記号として「?」や「!」はどうだろう？　確かに文の終わりにはなるが，質問ばかり（感嘆ばかり）していてはいつまでたっても仕事が終わりそうにない。というわけで，「.」ほど強くはないが，「:」よりも強い区切りとして使われ，ほとんど文の区切りと同じように扱われる「;」が，プログラミングの世界の「文」の区切りとして使われることになったのだろう。

［脚注］いずれも，いわゆる「半角」の英数字や記号だ。全角の文字は漢字やひらがなと同じように扱われるので，ここで述べたような文字化けの回避にはつながらない。

難しいのは論理的なエラー

単純な間違い――構文的なエラー――は，最初は大変でも時間をかければ解決するだろう。しかし，一番難しいのは論理的なエラー，つまりプログラムによる問題解決の方法である「アルゴリズム」が間違っている場合だ。プログラムが複雑になってくると，「こうすればうまくいくはずなのにどうしても動かない」といったケースが出てくる。

このような場合は途中の変数の状態などを調べることにより，自分の考えていた筋書きどおりに動いているかどうかを途中でチェックすることができる。プログラム例2-4で使ったconfirm()などの関数を挿入すれば，その時点の変数の値などを調べられる。また，document.write()を使って，ウェブページに途中結果を書いてしまうという方法もある（第6章で「フォーム」を使ったデバッグの方法を説明する）。

JavaScriptに関しては今のところそれほど広く使われていないが，処理系に付随していることの多い「デバッガ」を使うとこの作業がより簡単になる。confirm()などで出力しなくても，途中で実行を止めて変数の状態などを調べられる。

いくらプログラミングが上達しても，エラー（バグ）から逃れることはできない。あわてずに落ち着いて対処しよう。しばらくやってもエラーの原因がわからなかったら，休養をいれて頭をリフレッシュするのがよい。できれば，周りに相談できる人がいると深みにはまらずにすむ場合が多い（その人にアルゴリズムを説明するだけで，自分の頭の中が整理されてバグの原因がわかる場合も多いのだ）。

まとめ

この章では初めてのJavaScriptプログラムを作ってみた。また，とてもよく使われる，条件によって行う処理を変えるif ... elseの構文――条件分岐――や関数について説明した。

■ プログラム関連の概念

関数	一連の処理をまとめて名前を付けたもの
引数	関数に渡して関数の動作を変化させるもの
if文（条件分岐）	if (<条件1>) {...} else if (<条件2>){...} ... else {...} （else if以降は省略可）

■ HTML

<script>...</script>	JavaScriptのスクリプト（プログラム）を囲むHTMLタグ

`<body onload="関数()">`	ページが読み込まれたときに「関数」を実行

■ JavaScriptの関数（メソッド）

`confirm()`	ダイアログボックスを表示する
`document.write()`	HTMLのコードをJavaScriptから出力する