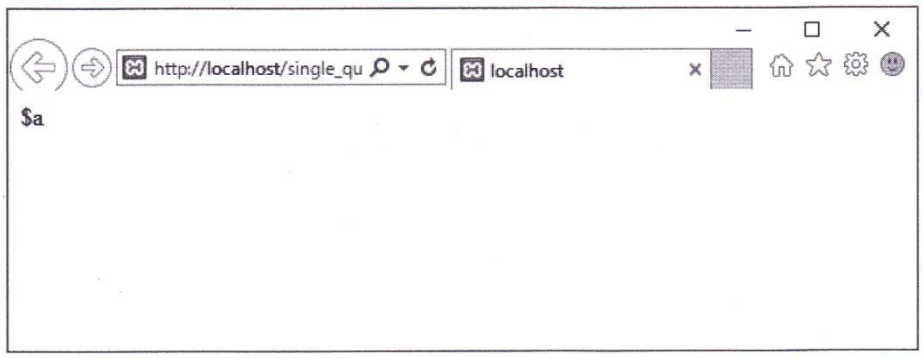
FIG 16-06 実行結果



このように、PHPで発行するSQL文に変数がある場合は注意が必要です。

関数

MySQLの関数についてはP.124で解説しましたが、PHPにも関数があります。

本書で扱うPHP関数

PHPには、1000以上の関数があります。ここでは、本書で登場する関数を紹介します。

TABLE 16-02 本書で登場する関数(次項の「date」は除く)

関数名	内容
date	現在の日時を返す(→P.363)
exec	コマンドを実行する(→P.353)
phpinfo	PHPの情報を表示する(→P.364)
nl2br	改行している場合、HTMLの改行タグを挿入する(→P.403)
preg_match	正規表現を使ってあいまい検索を行う(→P.463)
htmlspecialchars	タグなどの特殊文字列を変換する(→P.468)
isset	変数がセットされているか調べる(→P.480)
getenv	環境変数の値を得る(→P.365)
gethostbyname	ホスト名からIPアドレスを得る(参考)
gethostbyaddr	IPアドレスからホスト名を得る(→P.362)

ここでは、例としてdate、phpinfo、getenv、gethostbyaddr関数を紹介します。

date関数による日付・時刻表示

まずは、日付・時刻の処理に欠かすことのできない「date」関数を紹介します。ただし、このような日付・時刻に関する関数を使うときには、あらかじめタイムゾーンを正しく設定しておく必要があります（→P.341）。

書式 date関数の書式

```
date(日時の書式)
```

「date」は日時を返す関数です。引数に次のような文字列を指定すると、それぞれ対応する日時の値を返します。

TABLE 16-03 date関数に指定できる文字列

日時の書式	返す値
g	12時間単位の「時」
h	12時間単位の「時」の2桁表示
G	24時間単位の「時」
H	24時間単位の「時」の2桁表示
j	「日」
l	曜日の英語の文字列（Saturdayなどの文字を返す）
F	「月」の名前（Januaryなどの文字を返す）
n	「月」
m	「月」の2桁表示
s	「秒」（2桁）
Y	「年」
y	「年」の2桁表示

たとえば、

```
date("F")
```

とすると「December」のような現在の月の名前を返します。また、

```
date("n")
date("j")
```

とすると、現在の月と日を返します。Webページに、自動的に「○年○月○日」などと表示させるときに活躍します。なお、date関数の2番目の引数では、任意の日時（タイムスタンプ）を指定することもできます。

ちなみにPHPでは、日時の情報を「タイムスタンプ」という形式で処理します。この「タイムスタンプ」は、1970年1月1日0時から経過した秒数で表される整数です。

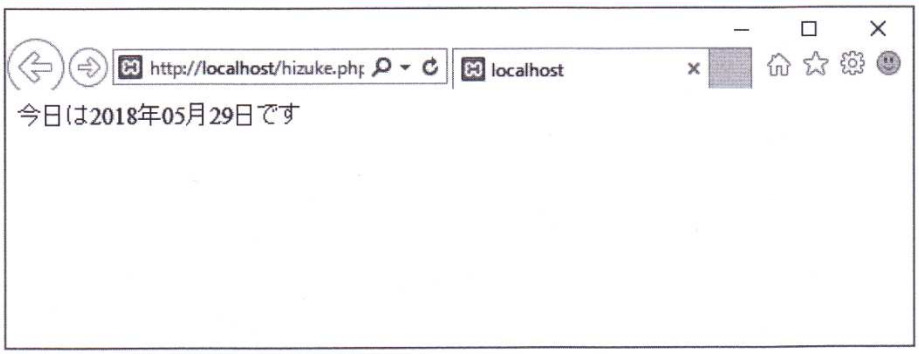
date関数の利用

では、現在の日付を表示するスクリプトを作成してみましょう。List16-06のように、現在の日付を「今日は××××年×月×日です」と表示するスクリプトを作ります。前述のとおり、文字列の結合には「.」を使います。

LIST 16-06 hizuke.php

```
<?php
print "今日は".date("Y")."年".date("m")."月".date("j")."日です";
?>
```

FIG 16-07 実行結果



環境情報

phpinfo関数による環境情報

使用されているPHPに関するいろいろな情報を得る関数に、「phpinfo」関数があり、次のように記述することで情報が表示されました(→P.352)。

```
<?php
phpinfo();
?>
```

関数は、引数を取らなくても( )を付ける必要がありました(→P.127)。さて、この「phpinfo( );」というたったの1行によって、たくさんの環境情報が表示されました。

phpinfoが返すこうした情報の中に、**環境変数**というものがあります。これは、phpinfo関数による表示の中の「Apache Environment」で示されるブロックにあります。環境変数には、「Webサーバーのソフトウェア」や「クライアントのIPアドレス」

など重要な情報が保管されています。そして、表示の中の「REMOTE_ADDR」という項目に、あなたがお使いのマシンのIPアドレスが表示されているはずです。確認してみてください（localhostの場合は「127.0.0.1」等）。

getenv関数

次に、getenv関数を使って、アクセスしてきたクライアントの情報を表示するPHPスクリプトを作ってみることにしましょう。

getenv関数は「環境変数の値」を返す関数で、特定の引数を指定することで、対応する情報が得られます。

書式 getenv関数の書式

```
getenv(得たい情報の項目)
```

TABLE 16-04 getenv関数で指定する引数と、得られる情報

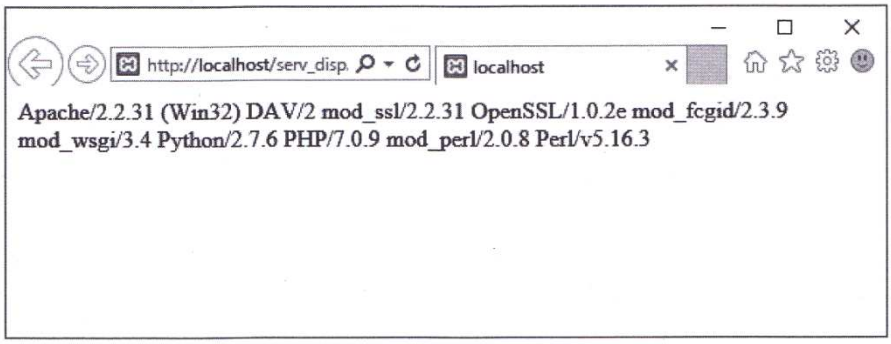
引数（得たい情報の項目）	得られる情報
SERVER_SOFTWARE	Webサーバーのソフトウエア
SERVER_PORT	使用しているポート
PATH	サーバーに設定されているPATH
REMOTE_ADDR	クライアントのIPアドレス
HTTP_USER_AGENT	クライアントのブラウザの情報

たとえば「Webサーバーのソフトウエア」は、「getenv ("SERVER_SOFTWARE")」で得ることができます。List16-07のスクリプトを実行してみましょう

LIST 16-07 serv_disp.php

```
<?php
print getenv("SERVER_SOFTWARE");
?>
```

FIG 16-08 実行結果

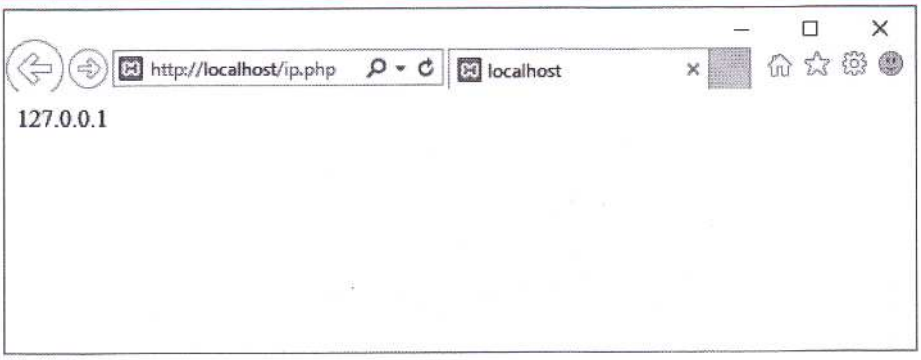


同様に「getenv ("REMOTE_ADDR")」(サンプル「ip.php」) で、クライアントのIPアドレスを得ることができます。

LIST 16-08 ip.php

```
<?php
print getenv("REMOTE_ADDR");
?>
```

FIG 16-09 実行結果



gethostbyaddr関数によるホスト名の取得

もう1つ、gethostbyaddr関数を使ってみましょう。gethostbyaddr関数は「IPアドレスからホスト名を得る」関数です。ちなみに「get-host-by-addr」と書けば、「アドレス (address) から (by) ホスト (host) を得る (get)」という意味がわかりやすくなると思います。

書式 gethostbyaddr関数の書式

```
gethostbyaddr(得たいホストのIPアドレス)
```

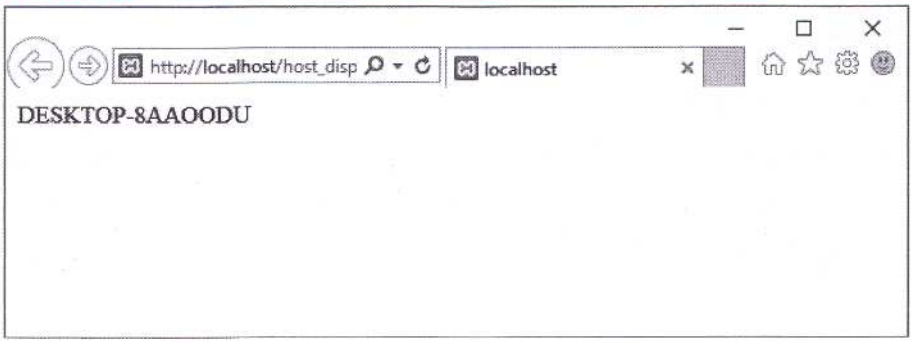
さて、前項の「getenv ("REMOTE_ADDR")」で得たIPアドレスを引数にして、gethostbyaddr関数を設定すれば、クライアントのホスト名を得ることができますね。

List16-09は、クライアントのホスト名を表示するスクリプトです。

LIST 16-09 host_disp.php

```
<?php
print gethostbyaddr(getenv("REMOTE_ADDR"));
?>
```

FIG 16-10 実行結果



gethostbyaddr関数の利用

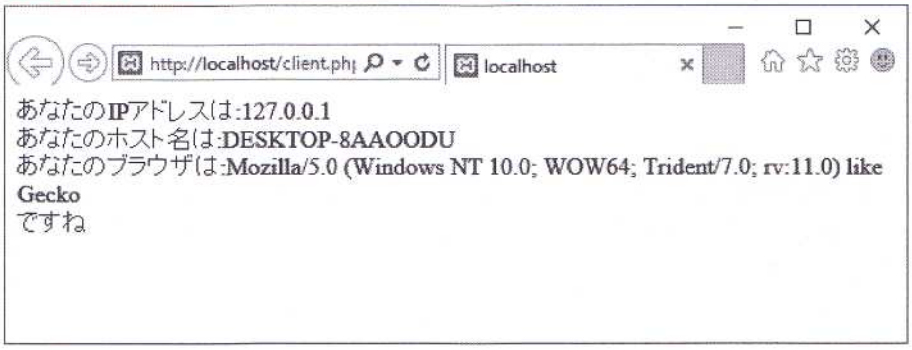
それでは、getenv関数とgethostbyaddr関数を使って、クライアントの情報を送り返すPHPスクリプトを作ってみることにしましょう。もっとも、このページに来訪した方にはちょっと失礼に当たるかもしれませんが…。

このスクリプトではgetenv関数を使って、アクセスしてきた「クライアントのIPアドレス」と「クライアントのブラウザ情報」を取得し、getenv関数で取得したIPアドレスをgethostbyaddr関数の引数とすることで、「クライアントのホスト名」を取得しています。

LIST 16-10 client.php

```
<?php
print "あなたのIPアドレスは:";
print getenv("REMOTE_ADDR");
print "<br>";
print "あなたのホスト名は:";
print gethostbyaddr(getenv("REMOTE_ADDR"));
print "<br>";
print "あなたのブラウザは:";
print getenv("HTTP_USER_AGENT");
print "<br>ですね";
?>
```

FIG 16-11 実行結果



比較演算子

以降で解説する「繰り返し」処理や「条件分岐」では、設定した条件に一致しているかどうかを判断する部分があります。このとき使う記号が**比較演算子**です。PHPでは、次のような「比較演算子」を使います。

TABLE 16-05 比較演算子

比較演算子	内容
a==b	aはbに等しい
a>b	aはbより大きい
a>=b	aはb以上
a<b	aはb未満
a<=b	aはb以下
a<>b	aはbと等しくない

条件が正しいことを「TRUE」あるいは「真」、条件が誤っていることを「FALSE」または「偽」であるといいます。「等しい」を表す「==」は、「=」を2つ記述します。「=」ではありませんので、注意してください。

たとえば、「2>1」は正しいので、「TRUE」(真)になります。

「繰り返し」処理

プログラムで実行することの利点の1つは、同じ処理を何百回何千回だろうと、疲れを知らず間違えることなく繰り返して処理を実行してくれるところです。ここでは、そんな「繰り返し」処理を行う「for」と「while」の構文を紹介します。

forによる繰り返し

forとは

forは、**カウンタ変数**という「数を数えるための変数」を用意し、これを「イチ、ニ、サン…」のように増やしながら、指定した範囲にある処理を繰り返します。forには、あらかじめ「繰り返すのはカウンタ変数が10以下のとき」というような「条件」を設定しておきます。そしてこの「条件」に合わなくなったら(条件の結果が「FALSE」になっ

たら）繰り返しが終了する、といった仕組みです。

forの構文は、次のようになります。

書式 for

```
for(初期値;繰り返しの条件;変化){
        繰り返し実行する処理
}
```

「{ }」で囲まれた「繰り返し実行する処理」の部分の処理を繰り返します。この部分は1行だけでなく、何行あってもかまいません。forの後の（）内は、どのように繰り返すか、ということを設定しています。

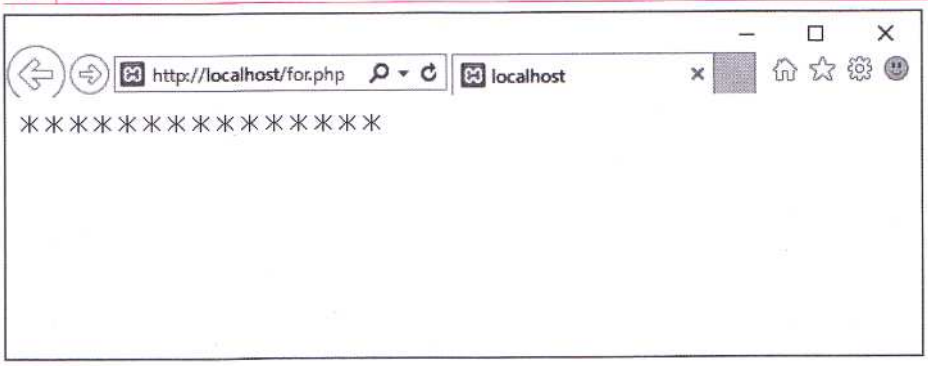
forを使う

用語だけではイメージしにくいですね。実際に実行してみるのが一番です。List16-11のスクリプトを入力して、試してみましょう。List16-11は、「＊」の文字を15回繰り返して表示する例です。

LIST 16-11 for.php

```
<?php
for($i=1;$i<=15;$i=$i+1){                                    1
    print "＊";
}
?>
```

FIG 16-12 実行結果



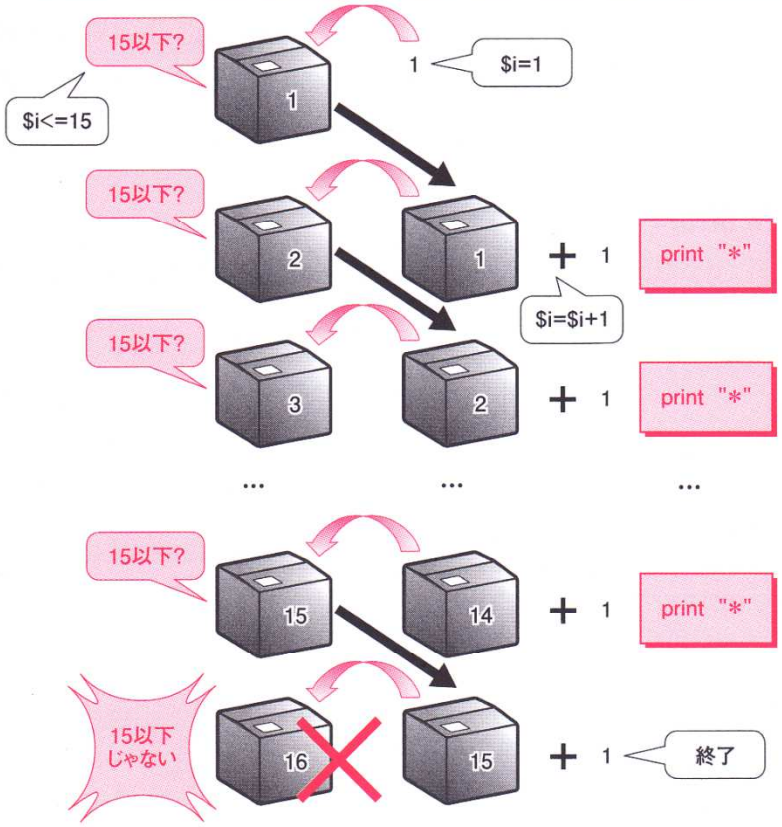
もしうまく動作しない場合は、1つ1つの文字を確認してください。「;」や「{」などの記号を間違えていませんか？

forの流れ

さて、List16-11の内容を見ていきましょう。1の部分の、forの( )内の記述は、次の意味を持ちます。

- $i=1　→　変数$iが、一番最初は「1」(初期値)
- $i<=15　→　$iが15以下のうちは繰り返しなさい(繰り返しの条件)
- $i=$i+1　→　繰り返すときは1ずつプラスしなさい(変化)

FIG 16-13 forの流れ



この例では、「$i」をカウンタ変数として使っています。カウンタ変数は、「$kurikaesi」でも「$p」でも何でもかまわないのですが、なぜか昔から「カウンタ変数は$i」と相場が決まっています。

さて、初期値として「$i=1」が設定されています。つまり、「カウントの最初は1だよ」ということです。そして「繰り返しの条件」が「$i<=15」となっています。つまり「カウンタ変数$iが15以下のうちは、この処理を繰り返しなさい」という設定です。15以下というのは、当然15も含むわけです。

「変化」は「$i=$i+1」となっています。これは「$iを1ずつ増やす」という意味です。プログラムの世界では、「=」は右のもの($i+1)を左のもの($i)に代入する、という意味でしたね(→P.356)。「$i」に1を足して、その1を足した結果を「$i」に代入する、つ

まり「$iを1増やせ」ということになります。

初期値は「$i=1」で「1」です。これに「$i+1」で「1」足した値「2」を「=」で左の$iに代入して$iが「2」。次にまた「$i+1」で1足した値「3」を$iに代入して$iが「3」…と1ずつ増えていきます。

「$i=$i+1」を省略して書くと、「$i++」となります。これをインクリメントといいます。つまり、List16-12の例を次のように記述しても結果はまったく同じです。「$i=1」では「1から」、「$i<=15」で「15以下の間」、「$i++」で「1を足せ」と処理しています（1）。

LIST 16-12 for_variation.php

```
<?php
for($i=1;$i<=15;$i++){
    print "＊";                                    1
}
?>
```

whileによる繰り返し

次はwhileによる繰り返しです。今度は「繰り返しを行う条件」が正しい（TRUE）間は、「繰り返しの処理」の部分を繰り返します。次のような構文になります。

書式 while

```
while(繰り返しを行う条件){
        繰り返しの処理
}
```

whileでは、特にカウンタ変数の初期値を設定する部分がないので、自分で初期値を設定しなければいけません。そして、繰り返していくうちに必ず「繰り返しを行う条件」が誤り（FALSE）になるようにしなければいけません。もし、いつまでも「繰り返しを行う条件」が正しい構文を作ってしまうと、永遠に処理が繰り返されることになるので注意が必要です。

whileを使う

さて、先ほどはforで「＊」を15回書き込みましたが、今度はwhileを使って同じことをやってみることにしましょう。

List16-13では、まず最初にカウンタ変数「$i」を用意して、これに「1」を代入します。そして、「print "＊";」を実行するたびに「$i」を1ずつ増やし、これをwhileで繰り返します。繰り返しの条件は「変数$iが15以下」とします。

LIST 16-13 while.php

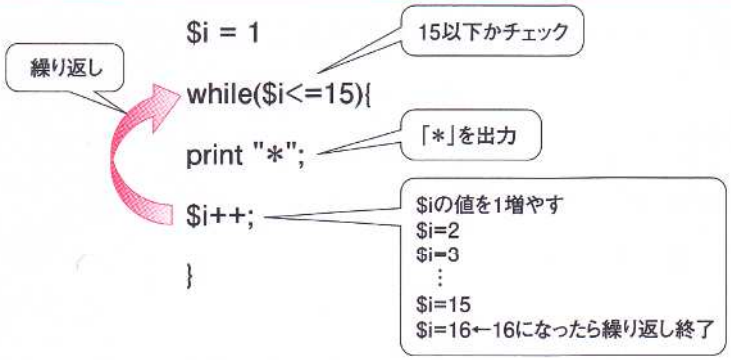
```
<?php
$i=1;                          1
while($i<=15){                 2
    print "*";
    $i++;                      3
}
?>
```

「$i=1;」で$iに1を代入し（1）、そして条件を調べます。最初、「$i」は「1」なので、「<=15」（15以下）という条件は正しくなり（TRUE）、処理「print "*"」が実行され、「*」が表示されます（2）。

次に「$i++;」で$iが（→前ページ）1増えて「2」になります（3）。そしてまたwhileの部分に戻り、「while ($i<=15)」の条件が正しいかが調べられます。

これを繰り返し、最後は「$i++;」で$iが増えて「16」になると、条件と一致しなくなる（FALSE）ので、16回目は書き出すことなく終了します。

FIG 16-14 whileの流れ



do～whileによる繰り返し

whileと似ている構文に、「do ～ while」があります。whileと違うのは、条件を調べるのが処理の後に来るという点です。

書式 do ～ whileの構文

```
do{
        繰り返しの処理
}while(繰り返しを行う条件)
```

次を実行すると、List16-13と同じ結果が得られます。

LIST 16-14 do_while.php

```
<?php
$i=1;                                    1
do{
    print "*";
    $i++;                                2
}while($i<=15)                           3
?>
```

「$i=1」では最初は「1」(1)、「$i++」で「1を足せ」(2)、「while ($i<=15)」で「15以下の間」(3)と処理しています。

注意すべき点は、whileの場合は最初に条件を調べるので、もしそこで条件に合っていなければ「1回も処理することなく終わる」ということです。しかしdo ～ whileの場合には、条件を調べるのが最後なので、「最低でも1回は実行」します。

条件分岐

ifによる条件分岐

ifは、条件によって実行する内容を変化させるものです。PHPスクリプトでは、次のような構文になります。

書式 ifの構文

```
if(条件){
    条件が正しいとき実行する処理
else{
    条件が正しくないときに実行する処理
}
```

()内の「条件」が合っていれば(TRUE)、「条件が正しいとき実行する処理」を実行し、そうでなければ(「条件」が合っていなければ：FALSE)「条件が正しくないときに実行する処理」を実行します。

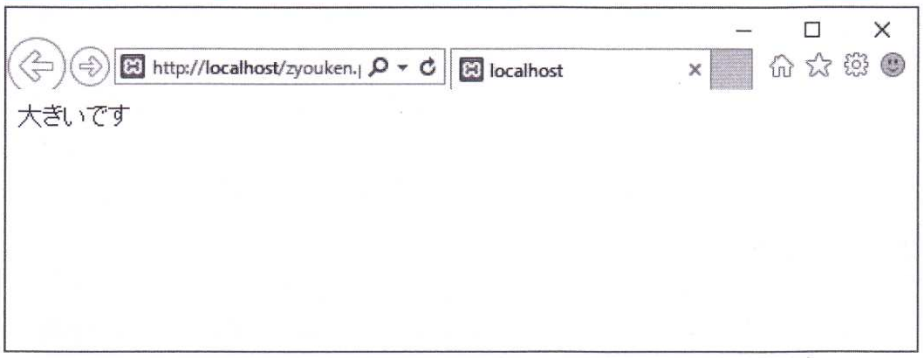
ifの流れ

たとえば、List16-15を実行してみましょう。

LIST 16-15 zyouken.php

```
<?php
if(200>100){                    1
    print "大きいです";          2
}else{                          3
    print "小さいです";          4
}
?>
```

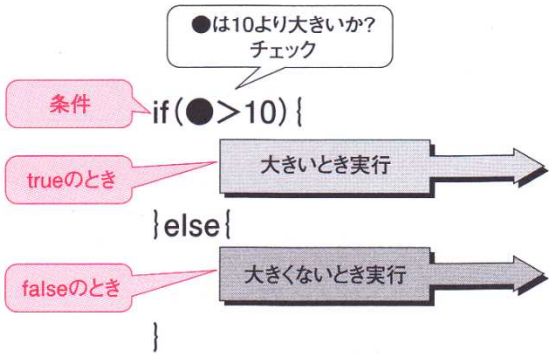
FIG 16-15 実行結果



「A>B」は、「AはBより大きい」ということを表します（→P.368）。「200と100ではどちらが大きいか」といえば、もちろん200ですね。つまり、1では「もし200は100より大きいならば」（200>100）という条件を調べています。これは「正しい」ため、2の構文中の「条件が正しいとき実行する処理」が実行されることになります。

もし、1の条件が正しくない場合（3）は、4を実行します。今回はここは実行されないことになります。

FIG 16-16 ifの流れ



もちろん実際のスクリプトでは、こんな当たり前の「200>100」というような条件ばかりを設定するわけではありません。たとえば「データベースに接続できたら」とか、「指定した文字が含まれていたら」というような、処理上の重要な判断でも使います。

なお、構文中の「else {条件が正しくないとき～}」の部分は、必要なければ省略することができます。この場合、条件が正しいときだけ指定した処理を行い、条件が正しくないときは何も実行しません。

三項演算子

条件によって値を変化させるだけなら、実に簡単な構文があります。次は「三項演算子」です。

書式 三項演算子

```
(条件)?条件が正しいときの値: 条件が正しくないときの値;
```

「三項演算子」はifの記述を簡潔にして、値自体を条件によって変化させます。「?」「:」などの記号の入力に注意してください。次は前ページで紹介した「200>100なら"大きいです"と表示、そうでなければ"小さいです"と表示」する例を、三項演算子で表したものです。

LIST 16-16 sankou.php

```
<?php
print (200>100)?"大きいです":"小さいです";
?>
```

また次は三項演算子を使って「11時までは"午前です"と表示、そうでなければ"午後です"と表示」させる例です。

LIST 16-17 am_pm.php

```
<?php
print (date("G")<12)?"午前です":"午後です";
?>
```

複数の条件を設定したifの構文

ifでは、条件式を何重にも設定することができます。次はその構文です。

書式 複数の条件を設定したifの構文

```
if(条件1){
    条件1が正しいとき実行する処理
}elseif(条件2){
    条件2が正しいときに実行する処理
}elseif(条件3){
    条件3が正しいときに実行する処理
    ⋮
}else{
    すべての条件が正しくないときに実行する処理
}
```

複数の条件があるifは、

「条件1」が合っていれば（TRUEなら）「条件1が正しいとき実行する処理」の部分を実行し、合っていない場合は次へ…

「条件2」が合っていれば（TRUEなら）「条件2が正しいとき実行する処理」の部分を実行し、合っていない場合は次へ…

⋮

と繰り返して、すべての条件が正しくなければ「すべての条件が正しくないときに実行する処理」の部分を実行します。

COLUMN ▶ **ストアドプロシージャでの条件分岐**

本書では扱っていませんが、MySQLのストアドプロシージャ（→P.261）でも「if」を使った条件分岐ができます。

複数の条件を設定したif文の流れ

では、ちょっと複雑な処理を考えてみましょう。アクセスしたときの時間に応じて変化する、挨拶メッセージを作ってみます。現在の時間は「date ("G")」で得ることが

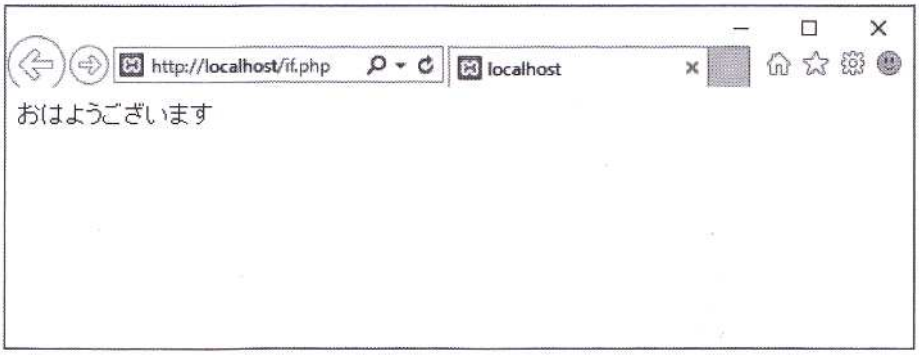
できましたね（→P.363）。つまり、「date（"G"）の値が18以上」「date（"G"）の値が9以上」…という条件を設定して、ifで処理を分岐させることになります。

具体的には、0時から「眠くないですか」、6時から「おはようございます」、9時から「こんにちは」、18時から「こんばんは」と表示されるList16-18を作ってみましょう。

LIST 16-18 if.php

```
<?php
if (date("G")>=18){
    print "こんばんは";
}elseif(date("G")>=9){
    print "こんにちは";
}elseif(date("G")>=6){
    print "おはようございます";
}else{
    print "眠くないですか";
}
?>
```

FIG 16-17 実行結果



実行したときの時間によって、対応する文字列が表示されます。

switchを使った分岐

変数の値によって、それぞれ異なる処理を実行する場合には「switch」を使います。たとえば、変数「$i」が「1」のときは「××」、変数「$i」が「2」のときは「△△」、変数「$i」が「3」のときは「○○」…を実行する、というような処理ができます。

このような処理は、前項のifを組み合わせても同じように実行できます。しかし、「switch」を使うと、よりわかりやすく記述することができます。

switchの構文は、次のようになります。

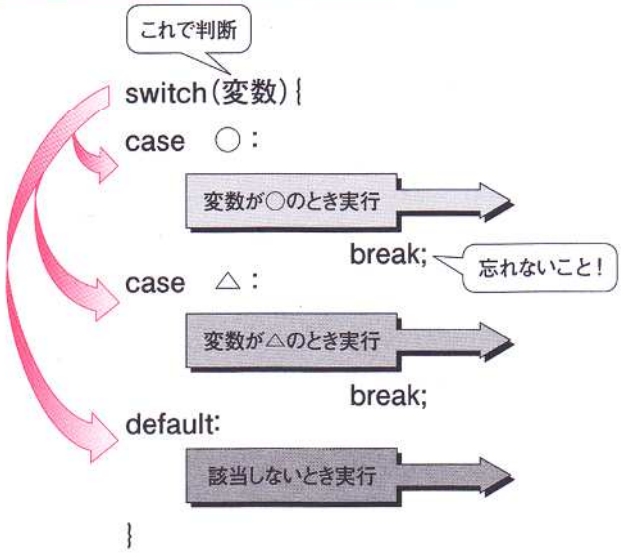
書式 switchの構文

```
switch(変数){
case 変数の値1:
    処理1
    break;
case 変数の値2:
    処理2
    break;
...
default:
    すべての条件が正しくないときに実行する処理
}
```

switchの後に設定する「変数」がcaseの後に記述した値と一致した場合、それぞれの「処理」が実行されます。それぞれの処理の最後には必ずbreakを付けてください。「break」は、処理を終了する命令です。

一致する値がない場合は、「default:」の次に書かれた処理が実行されます。

FIG 16-18 switchの流れ



switch文の流れ

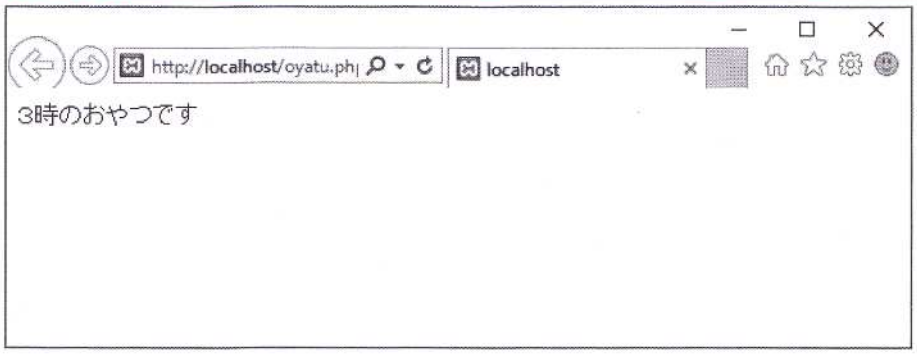
たとえばList16-19は、10時のときは「10時のおやつです」、15時のときは「3時のおやつです」、10時と15時以外のときは「おやつではありません」と表示する例です。

LIST 16-19 oyatu.php

```
<?php
switch(date("G")){
case 10:
    print "１０時のおやつです";
    break;
case 15:
    print "３時のおやつです";
    break;
default:
    print "おやつではありません";
}
?>
```

繰り返しになりますが、それぞれの処理の後に「break」を記述することを忘れてはいけません。万が一「break」を記述しなかった場合、その後の処理も実行されてしまいます。また「case 変数の値1:」などの「:」(コロン)にも注意をしてください。

FIG 16-19 実行結果



もう1つ、次のような連続模様を、switchを使って作成する方法を考えてみましょう。

◎★○▽▲◎★○▽▲◎★○▽▲◎★○▽▲◎★○▽▲◎★○▽▲◎★○▽▲…

「for」(→P.368)で「1 ～ 5」までの値を変数に代入し、「switch」でそれぞれの値ごとに別の処理をさせます。これをさらに「for」で繰り返す、という方法です。

List16-20は、変数「$x」が「1なら◎」「2なら★」「3なら○」「4なら▽」「5なら▲」を書き出す処理を8回繰り返すことで、上の「◎★○▽▲」を8回繰り返す連続模様を書き出します。

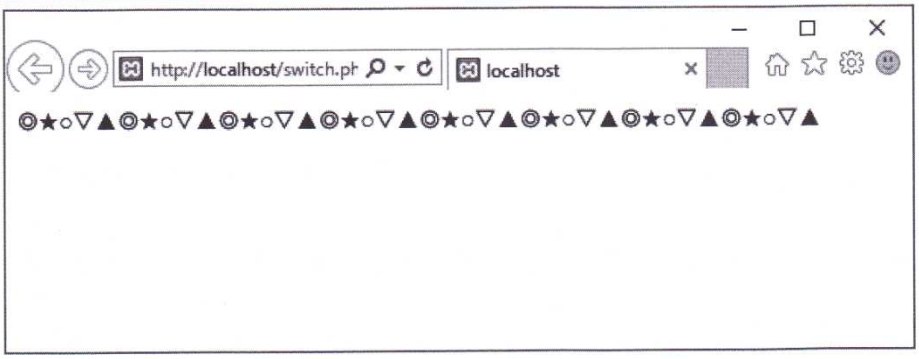
LIST 16-20 switch.php

```
<?php
for($x=1;$x<=8;$x++){                                          3

    for($y=1;$y<=5;$y++){                                      2
        switch($y){                                            1
        case 1:
            print "◎";
            break;
        case 2:
            print "★";
            break;
        case 3:
            print "○";
            break;
        case 4:
            print "▽";
            break;
        case 5:
            print "▲";
            break;
        }
    }

}
```

FIG 16-20 実行結果



処理の内容を見ていきましょう。

まず、以下の部分で変数「$y」が「1」なら「◎」、「2」なら「★」というように処理を分岐させます（1）。