ソフトウェア開発
第9回目授業

平野 照比古

2018/11/30

オブジェクトリテラルでプルダウンメニューを作成 (1)

連番のプルダウンメニュー作成関数を真似ればよい。

```
function makeSelectNumber(from, to, prefix, suffix, id, parent){
  let Select = makeElm("select", {"id":id}, parent);
  for(let i=from; i<=to; i++) {
    let option = makeElm("option",{ "value":i}, Select);
    makeTextNode(prefix+i+suffix,option);
  }
  return Select;
};
function makeSelectFromObj(Obj, id, parent){
```

for 文を for(.. in ..) で書き直せばよい。

オブジェクトリテラルでプルダウンメニューを作成 (2)

連番のプルダウンメニュー作成関数を真似ればよい。

```
function makeSelectFromObj(Obj, id, parent){
  let Select = makeElm("select", {"id":id}, parent);
  for(let opt in Obj) {
    let option = makeElm("option",{"value":opt}, Select);
    makeTextNode(Obj[opt],option);
  }
  return Select;
};
```

プルダウンメニューの表示文字列は引数内にあるので不要な形にした。

オブジェクトリテラルでラジオボタンを作成

連番のプルダウンメニュー作成関数を真似ればよい。

```
function makeRadioFromObj(Obj, id, name, parent){
  let Div = makeElm("div", {"id":id}, parent);
  for(let opt in Obj) {
    let radio = makeElm("input",
      {"value":opt, type:"radio", name:name}, Div);
    makeTextNode(Obj[opt],Div);
  }
  return Div;
```

- 全体をまとめるために div 要素内にラジオボタンを置く。
- ラジオボタンの説明文字列はラジオボタンと同じレベルの子要素にする。
- ラジオボタンの子要素では表示されない。

- clientX と pageX などの違いについての操作はほとんど合っていた。
- pageY = clientY + window.pageOffsetY の指摘がなかった。
- target と currentTarget の違いを確認するところで target.value のように要素の value を表示させていた誤答が目立った。

階層を持ったプルダウンメニューの作成

- 問題の意図はデータの持ち方を考えてもらうこと
- 重複がなく修正や追加がしやすい構造を考えることが重要
- ここでは地方毎にそれに属する地域をリストアップ
- 全国のデータはそれらを寄せ集めて作成

作成例 (1)

```
let Area = {
"北海道地方（北西部）": {
  "value":"201",
  "府県":[
    {"value":"301", "name":"宗谷地方"},
    {"value":"302", "name":"上川・留萌地方"}
]},
"北海道地方（東部）": {
  "value":"202",
  "府県":[
    {"value":"303", "name":"網走・北見・紋別地方"},
    {"value":"304", "name":"釧路・根室・十勝地方"}
]},
```

- 気象庁のページのソースを尊重してプルダウンメニューの value をオブジェクトリテラルに記述
- 地域のリストはこのオブジェクトのキーから作成
- 全国のリストは値から作成

作成例 (2)

```
let All = makeElm("select", {}, Form,
  {"click":[function(E){
     Form.replaceChild(selects[E.target.value],Form.children[1]);
  },false]});
let o = makeElm("option", {"value": "000"}, All);
o.innerText = "全国";
let selects = {"000":makeElm("select")};
let List = {};
for(let opt in Area) {//console.log(opt);
  let o = makeElm("option", {"value": Area[opt].value}, All);
  o.innerText = opt;
  selects[Area[opt].value] = makeElm("select");
  Area[opt]["府県"].forEach(function(fuken) {
    let tmp = makeElm("option",{value:fuken.value}, selects[Area[opt].value
    tmp.innerText = fuken.name;
    if(List[fuken.value] === undefined) {
      List[fuken.value] = 1;
      tmp = makeElm("option",{value:fuken.value}, selects["000"]);
      tmp.innerText = fuken.name;
    }
  });
}
Form.appendChild(selects["000"]);
```

Web サーバーソフト

- hypertext transfer protocol(HTTP) を利用して、クライアントプログラム (Web ブラウザなど) からの要求を処理
- このサービスを提供するプログラムとしては Apache が有名
- Apache によるサービスの設定ファイルは `httpd.conf`
  - ポート番号 (HTTP のポート番号は 80 番)
  - ドキュメントルート：Apache では Web サーバーに直接要求できるファイルはこのフォルダ (ディレクトリ) の下にあるものだけ

ポート番号の種類

- TCP/IP におけるサービスを識別するための番号
- ポート番号は大別して次の 3 種類

種類	範囲
System Ports(Well Known Ports)	1 番 ~ 1023 番
User Ports, (Registered Ports)	1024 番 ~49151 番
Dynamic Ports(Private or Ephemeral Ports)	49152 番 ~65535 番

Common Gateway Interface(CGI)

- Web コンテンツをダイナミックに生成する標準の方法
- Web サーバー上では Web サーバーと Web コンテンツを生成するためのインターフェイス
- この授業では CGI のプログラムは PHP(PHP:Hypertext Preprocessor) を使用

Web サーバーのインストール

- HTTP のサーバーを動かすためには Apache や PHP をインストールした後、いくつかの設定
- XAMPP は Apache、MySQL、Perl と PHP を一括してインストールパッケージ
  - インストール時に Apache をサービスとして実行するとパソコンを立ち上げた時に Apache を起動させる手間が省ける。
  - 標準の文字コードは UTF-8 である。
  - PHP の初期設定でタイムゾーンがヨーロッパ大陸になっている。
  - タイムスタンプを利用するプログラムの動作に注意すること。

PHP とは

日本 PHP ユーザー会のホームページより

> *PHP* は、オープンソースの汎用スクリプト言語です。 特に、サーバサイドで動作する *Web* アプリケーションの開発に適しています。 言語構造は簡単で理解しやすく、*C* 言語の基本構文に多くを拠っています。 手続き型のプログラミングに加え、(完全ではありませんが) オブジェクト指向のプログラミングも行うことができます。

PHP の使用に関する説明は日本 PHP ユーザー会から多く引用

PHP プログラムの実行方法

- XAMPP で指定されたドキュメントルートの下にファイルを作成してブラウザから実行
- コマンドプロンプトから実行
  - そのままでは XAMPP の下にある `php.exe` が実行できないので環境変数 `PATH` に追加
  - コントロールパネル ⇒ システム ⇒ システムの詳細設定を開き、下方にある「環境変数」ボタンを押す
  - 上方部のユーザーの環境変数にある環境変数「PATH」に XAMPP のフォルダを追加
  - コマンドプロンプトで `php <ファイル名>` で実行。漢字コードが化ける可能性あり。
- 実行したいファイルがあるところでコマンドプロンプトから
  `php -S localhost:8080`
  を実行。ブラウザから `localhost:8080/ファイル名` で実行可能となる。

PHP プログラムの書き方

- PHP のプログラムの拡張子は php
- クライアントからの要求で拡張子が php のファイルが要求されると、PHP のプログラムはサーバー側で処理
- その出力がクライアント側に送られる
- サーバー側で処理すべき PHP プログラムの部分は<?php と?>の間に記述
- それ以外の部分はそのままクライアント側に送られる。

本日の授業の注意

- 本日の授業では PHP のプログラムを対話型で実行する。
- XAMPP に含まれる PHP ではこの方法では実行できない。
- PHP の本家からダウンロードすれば実行可能
- 授業では Windows 上で Unix の環境を実現する Cygwin の PHP を用いる。
- 配布資料の PHP のプログラムの右のコメントは実行結果を示している。

データの型 —4 種類のスカラー型

- 論理値 (boolean)
  TRUE と FALSE の値をとる。小文字で書いてもよい。
- 整数 (integer)
  整数の割り算はない。つまり $1/2$ は 0 ではなく 0.5 となる。
- 浮動小数点数 (float, double も同じ)
- 文字列 (string)
  JavaScript と同様に文字列の型はあるが、文字の型はない。

データ型—2 種類の複合型

- 配列 (array)
  `array()` を用いて作成する。引数にはカンマで区切られた `key=>value` の形で定義する。`key=>`の部分はなくてもよい。このときは `key` として `0,1,2,...` が順に与えられる。
- オブジェクト (object)

データ型—2 種類の特別な型:

- リソース (resource)
  オープンされたファイル、データベースへの接続、イメージなど特殊なハンドルを保持する。
- ヌル (NULL)
  ある変数が値を持たないことを示す。

変数

- 変数は宣言なしで使用できる (宣言する方法すらない)。
- 変数名は $で始まる。
- 変数に値を代入すればその値はすべてコピーされる。
- コピーではなく参照にしたい場合には変数の前に &を付ける。

変数 — 例 (1)

```
<?php
$a = 1;
$b = $a;
print "1:\$a=$a, \$b=$b\n";// 1:$a=1, $b=1
$a = 2;
print "2:\$a=$a, \$b=$b\n";// 2:$a=2, $b=1
$b = &$a;
print "3:\$a=$a, \$b=$b\n";// 3:$a=2, $b=2
$a = 10;
print "4:\$a=$a, \$b=$b\n";// 4:$a=10, $b=10
?>
```

例 (1) 解説

- 3 行目で変数 $a の値を変数 $b の代入。
  両者の値は同じである。
- 5 行目で $a の値を変更しても、変数 $b の値は変化しない (6 行目)
- 7 行目では変数 $b に変数の参照を代入している。この場合には変数 $a の値を変更すると (9 行目) 変数 $b の値も変更される (10 行目)。

変数— 配列の場合

```
<?php
$a = array(1,2);
$b = $a;

print "1\n";
print_r($a);
print_r($b);

$a[0] = 20;
print "2:\$b[0]=$b[0]\n";

$b = &$a;
$a[1] = 30;
print "3:\$b=";
print_r($b);
?>
```

変数— 配列の場合 (実行結果)

```
Array
(
    [0] => 1
    [1] => 2
)
Array
(
    [0] => 1
    [1] => 2
)
2:$b[0]=1
3:$b=Array
(
    [0] => 20
    [1] => 30
)
```

例 (2)–解説

- 2 行目で配列の成分が 1 と 2 である配列を作成
  3 行目でこの配列を別の変数に代入
- 9 行目では配列の一部分の値を変えているが、3 行目で代入された変数の値には変化がない (10 行目)。
  これから単純な配列の代入は配列全体がコピーされている
- 関数 `print_r()` は配列などの構造を持つ変数の内容をわかりやすく表示する関数。
  より詳しい情報を知りたい場合には関数 `var_dump()` を利用。

可変変数

ある変数の値が文字列のとき、その前に$をつけると、その文字列が変数名として使える。

```
<?php
$a = 1;
$b = 2;

$c = "a";
print "\$\$c = " . $$c ."\n";  // $$c = 1

$c = "b";
print "\$\$c = " . $$c . "\n"; // $$c = 2
?>
```

可変変数– -解説

- 変数$a と $b にそれぞれ 1 と 2 を代入 (1 行目と 2 行目)。
- 変数$c に文字列"a"を代入し、$$c を出力させると$a の値が表示 (5 行目と 6 行目)。
- 同様に、変数$c に文字列"b"を代入して、$$c を出力させると$b の値が表示 (8 行目と 9 行目)。

変数のスコープ

- 変数は特に宣言しなくても使用できる。
- 指定しない限りローカルなスコープしか持たない。
- 関数外で定義された変数も、関数内から参照できない。
- 参照するためにはスーパーグローバル$GLOBAL を利用する。
- 関数内での変数は外部からの参照できない。
- PHP には外部ファイルを読み込むための関数 include() と一度だけ指定されたファイルを読み込む require_once() がある。このファイルの中で定義された変数は、読み込むテキストを直接記述したときと同じ

定義済み変数

スーパーグローバルと呼ばれるグローバルスコープを持つ定義済みの変数

変数名	意味
$GLOBALS	グローバルスコープで使用可能なすべての変数への参照
$_SERVER	サーバー情報および実行時の環境情報
$_GET	HTTP GET 変数
$_POST	HTTP POST 変数
$_FILES	HTTP ファイルアップロード変数
$_COOKIE	HTTP クッキー
$_SESSION	セッション変数
$_REQUEST	HTTP リクエスト変数
$_ENV	環境変数
$argc	コマンドプロンプトから実行したときに与えられた引数の数
$argv	コマンドプロンプトから実行したときに与えられた引数のリスト

これらの変数のうち、$_ENV、$argc と$argv 以外は CGI で使用

文字列について

- シングルクオート (’) ではさむ。
  中に書かれた文字がそのまま定義される。
- ダブルクオート (") ではさむ。
  改行などの制御文字が有効になる。また、変数はその値で置き換えられる。これを変数が展開されるという。
- ヒアドキュメント形式
  複数行にわたる文字列を定義できる。<<<の後に識別子を置く。文字列の最後は行の先頭に初めの識別子を置き、そのあとに文の終了を表す ; を置く。識別子の前に空白などを入れてはいけない。

文字列—実行例

```
<?php
print 'string1 \':abcd\n';
print "string2 \":abcd\n";    // string1 ':abcd\nstring2 ":abcd

$a = 1;
print 'string3 \':$a bcd\n';
print "string4 \":$a bcd\n"; //string3 ':$a bcd\nstring4 ":1 bcd
print "string5 \":{$a}bcd\n";//string5 ":1bcd

$b = array(1,2,3);
print "string6:$b\n";         //string6:Array
print "string7:$b[1]aa\n";  //string7:2aa
print "string7:$b[$a]aa\n"; //string7:2aa

print <<<_EOL_
string8:
  aa
  \$a = $a
_EOL_;           //string8:
                 //  aa
                 //  $a = 1
?>
```

文字列—実行結果 解説

- シングルクオートの文字列では改行を意味する \n がそのまま出力されているのに対し、ダブルクオートの文字列では改行に変換されている (1 行目)。
- シングルクオートの文字列では変数名がそのまま出力
- ダブルクオートの文字列では変数の値に変換。これを変数で展開されるという。
- 変数として解釈されるのは変数名の区切りとなる文字 (空白など)。そのためプログラムの 7 行目では変数の後に空白を入れている。
- 空白を入れたくない場合などは変数名全体を { と } で囲む。
- 変数が配列の場合には個々に指定しなくてはいけない。指定にはべつの変数が使用できる (5 行目と 6 行目)
- ヒアドキュメントでは途中の改行もそのまま出力される。また、変数は展開される。

式と文

- 文の最後には必ず; が必要である。
- その他はほとんど C 言語と同じ構文が使える。
- 演算子+は JavaScript と異なり、通常の数の演算となる。
- 文字列に対して+を用いると数に直されて計算される。
- 文字列の連接には. を用いる。

文字列の演算子の確認

```
<?php
print  1  + "2" . "\n";     // 3
print "1" + "2" . "\n";     // 3
print "1" . "2" . "\n";     // 12
print 1 . 2 . "\n";         // 12

print "1a" + "2" . "\n";  // 3
print "a1" + "2" . "\n";  // 2
print "0x1a" + "2" . "\n";// 28
```

文字列の演算子の確認—解説

- +演算子は常に数としての加法として扱われ、. は文字列の連接として扱われる
- 文字列が数として不正な文字を含むとその直前まで解釈した値を返す (7 行目と 8 行目)。
- 16 進リテラルも正しく解釈される (9 行目)。

比較演算子

変数の状態を調べる関数が引数の値によりどのようになるか、また、論理値が必要なところでどのようになるかを表している。

- gettype() 変数の型を調べる
- isempty() 変数が空であることを調べる
- is_null() 変数の値が NULL であるかを調べる
- isset() 変数の値がセットされていて、その値が NULL でないことを調べる
- if(変数) としたときに TRUE となるかどうか

PHP 関数による $x の比較

式	gettype()	empty()	is_null()	isset()	boolean : if($x)
$x = "";	string	TRUE	FALSE	TRUE	FALSE
$x = null;	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x が未定義	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	array	TRUE	FALSE	TRUE	FALSE
$x = false;	boolean	TRUE	FALSE	TRUE	FALSE
$x = true;	boolean	FALSE	FALSE	TRUE	TRUE
$x = 1;	integer	FALSE	FALSE	TRUE	TRUE
$x = 42;	integer	FALSE	FALSE	TRUE	TRUE
$x = 0;	integer	TRUE	FALSE	TRUE	FALSE
$x = -1;	integer	FALSE	FALSE	TRUE	TRUE
$x = "1";	string	FALSE	FALSE	TRUE	TRUE
$x = "0";	string	TRUE	FALSE	TRUE	FALSE
$x = "-1";	string	FALSE	FALSE	TRUE	TRUE
$x = "php";	string	FALSE	FALSE	TRUE	TRUE
$x = "true";	string	FALSE	FALSE	TRUE	TRUE
$x = "false";	string	FALSE	FALSE	TRUE	TRUE

いろいろな値を論理値として評価した結果が JavaScript と異なる場合があるので注意が必要である。

比較演算子==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

制御構造

- PHP の分岐、繰り返しなどの制御構造は C 言語と同じである。
- JavaScript のオブジェクトのプロパティを列挙するのに利用した `for(... in ...)` に相当する `foreach` がある。
  この構文は形をとる。
  `foreach(「配列名」 as [キー =>] 値)`
  「キー」と「値」のところは単純な変数を置くことができる。
  「キー」の部分は省略可能である。

foreach の使用例 (1)

```
 <?php
$a = array("red","yellow","blue");
foreach($a as $val) {
  print "$val\n";
}
foreach($a as $key=>$val) {
  print "$key:$val\n";
}
```

この部分の出力は次のとおりである。

```
red
yellow
blue
0:red
1:yellow
2:blue
```

単純な配列のときは「キー」の値は $0, 1, 2, \ldots$ を順にとる (出力 4 行目から 6 行目)。

foreach の使用例 (2)

```
$a = array("red"=>"赤","yellow"=>"黄","blue"=>"青");
foreach($a as $val) {
  print "$val\n";
}
foreach($a as $key=>$val) {
  print "$key:$val\n";
}
```

- 1 行目では JavaScript のオブジェクトリテラルに似た連想配列を定義
- 2 行目から 4 行目と 5 行目から 7 行目で以前と同じ形で配列の内容を表示

この部分の出力は次のようになる。

```
赤
黄
青
red:赤
yellow:黄
blue:青
```

PHP の配列に関する注意

```
$b = array();
$b[3] = "3rd";
$b[0] = "0";
print count($b)."\n";
foreach($b as $key=>$val) {
  print "$key:$val\n";
}
for($i=0;$i<count($b);$i++) {
  if(isset($b[$i])) {
    print "$i:$b[$i]\n";
  }
}
?>
}
```

この部分の出力は次のようになる。

```
2
3:3rd
0:0
```

PHP の配列に関する注意 (解説)

- 1 行目で配列を初期化し、その後、4 番目と先頭の要素に値を代入
- 関数 count() は配列の大きさ (正確には配列にある要素の数) を求めるものである。2 つの値しか定義していないので 2 となる。
- foreach で配列の要素を渡るときは定義した順に値が設定

$a[$key]=$val の関係が成立

配列に関する関数

- list()
  配列の個々の要素をいくつかの変数にまとめて代入できる。これは関数ではなく、言語構造
- array_pop() と array_push()
- in_array($needle, $array)
  与えられた配列 ($array) 内に指定した値 ($needle) が存在するかを調べる
- shuffle()
  与えられた配列の要素をランダムに並べ替える
- 配列の切り出しと追加
  array_slice(), array_splice()

関数の特徴

- 関数はキーワード function に引き続いて () 内に、仮引数のリストを書く。そのあとに {} 内にプログラム本体を書く。
- 引数は値渡しである。参照渡しをするときは仮引数の前に&を付ける。
- 関数のオーバーロードはサポートされない。
- 関数の宣言を取り消せない。
- 仮引数の後に値を書くことができる。この値は引数がなかった場合のデフォルトの値となる。デフォルトの値を与えた仮引数の後にデフォルトの値がない仮引数を置くことはできない。
- 関数は使用される前に定義する必要はない。
- 関数内で関数を定義できる。関数内で定義された関数はグローバルスコープに存在する。ただし、外側の関数が実行されないと定義はされない。
- 関数の戻り値は return 文の後の式の値。複数の値を関数の戻り値にしたいときは配列にして返す。

ユーザー定義関数の使用例

```
<?php
function example($a, $as, &$b, $f=false) {
  print "\$a = $a\n";
  print_r($as);
  print "\$b = $b\n";
  if(!isset($x)) $x = "defined in function";
  print "\$x = $x\n";
  if($f) {
    print "\$GLOBALS['x'] = ". $GLOBALS['x']."\n";
  }
  $a = $a*2;
  $as[0] += 10;
  $b = $b*2;
  return array($a,$as);
}
```

ユーザー定義関数の使用例—解説

- 1 行目での関数では 3 番目の引数が参照渡し、4 番目の引数がデフォルトの値が設定されている。
- 6 行目の関数 `isset()` は与えられた変数に値がセットされているかどうかを確かめるものである。
- ここでは`$x` は仮引数ではないのでローカルな変数となり、`isset($x)` は `false` となる。`!isset($x)` は `true` となるので変数`$x` には`"defined in function"`が代入される。
- 9 行目ではデフォルトの引数のチェックのための部分である。
- 11 行目から 13 行目までは変数に値を代入して、呼び出し元の変数が変わるかどうかのチェックをする。
- 14 行目は初めの二つの仮引数を配列にして戻り値としている。

関数の動作チェック

```
$a = 10;
$as = array(1,2);
$b = 15;
$x = "\$x is defined at top level";
```

ここでは、関数に渡す引数の値を設定している。

```
example($a, $as, $b, true);
print "\$a = $a\n";
print_r($as);
print "\$b = $b\n";
```

実行結果

20 行目で呼び出した関数内での出力結果は次のようになる。

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 15
$x = defined in function
$GLOBALS['x'] = $x is defined at top level
```

関数の動作チェック—解説

- 仮引数の値は正しく渡されている。
- 6 行目は判定が `true` になるのでここで新たに値が設定され、それが 7 行目で出力される。
- デフォルトの仮引数に対して `true` が渡されているので、9 行目が実行される。スーパーグローバル`$GLOBAL` にはグローバルスコープ内の変数が格納されている。ここでは 19 行目に現れる変数の値が表示される。

関数の動作チェック—続き

21 行目から 23 行目の出力は次のようになる。

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 30
```

関数の動作チェック—続き (解説)

- 初めの 2 つの引数は配列であっても書き直されていない。11 行目と 12 行目の設定は戻り値にしか反映されない。
- 参照渡しの変数$b は書き直されている。

関数の動作チェック—続き

```
list($resa) = example($a, $as, $b);
print "\$resa = $resa\n";
?>
```

- 24 行目の関数呼び出しはデフォルトの引数がない。
- したがって、引数$f の値が false に設定され、9 行目は実行されない。
- list() は配列である右辺の値のうち先頭から順に指定された変数に代入
- ここでは関数内の 11 行目で計算された値を変数$resa に代入

関数の動作チェック—続き

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 30
$x = defined in function
$resa = 20
```

可変変数

ある変数の値が文字列のとき、その前に$をつけると、その文字列が変数名として使える。

```
<?php
$a = 1;
$b = 2;

$c = "a";
print "\$\$c = " . $$c ."\n";  // $$c = 1

$c = "b";
print "\$\$c = " . $$c . "\n"; // $$c = 2
?>
```

可変変数– -解説

- 変数$a と $b にそれぞれ 1 と 2 を代入 (1 行目と 2 行目)。
- 変数$c に文字列"a"を代入し、$$c を出力させると$a の値が表示 (5 行目と 6 行目)。
- 同様に、変数$c に文字列"b"を代入して、$$c を出力させると$b の値が表示 (8 行目と 9 行目)。

可変関数

文字列が代入された変数の後に () をつけると、その文字列の関数を呼び出すことができる。

```
<?php
function add($a, $b) { return $a+$b;}
function sub($a, $b) { return $a-$b;}
$f = "add";
print "$f(5,2) = " . $f(5,2) ."\n";// add(5,2) = 7
$f = "sub";
print "$f(5,2) = " . $f(5,2) ."\n";// sub(5,2) = 3
?>
```

可変関数–解説

- 2 行目と 3 行目で 2 つの関数 add() と sub() を定義
- 4 行目で変数$f に文字列"add"を代入して、5 行目で可変関数として呼び出すと、関数 add が呼び出されている。
- 同様に、変数$f に文字列"sub"を代入して、可変関数として呼び出すと、関数 sub が呼び出されている。