■PHPスクリプト言語のまとめ

作成:2013.2.15/2015.7.7修正 yoshi

分野	PHP	説明
動作環境と言語仕様		
スクリプトの動作コード	UTF-8,EUC-JP	php5.2以降は、開発内部コードがUTF-8となった
コード変換	日本語の変換用に「**mbstring**」「**mb_convert_kana**」の関数がある。	php.iniの初期設定ファイルで「mbstring」ONが必要
HTMLコードのスクリプト記述	スクリプトの範囲指定:HTMLタグコードに対して、スクリプトの範囲を指定する形式。	ASP,JSPもPHPと同様にHTMLコードベースの範囲指定。
スクリプトのコード記述(範囲)	<?php …PHPコード ?> 又は <script language="php"> …PHPコード </script>	(**<?** …PHPコード **?>**も可能であるが、php.iniでオプション設定でONが必要)
ファイルの拡張子	xxx.**php**	スクリプトのインタープリター動作種別として利用される。
コメント行	「//」又は「#」で始まる以降(1行のみ有効)、「/*」から「*/」の範囲(複数行の有効)	
ユーザ定義関数	関数:「**fuction rei( ){ … }**」で定義し、「**rei();**」で実行	ユーザ定義関数以外にPHP組込み関数あり
別モジュールのロード方法	**include**("xxx.php");	ファイルの読み込み失敗した場合に警告 (E_WARNING)を発するもののスクリプトの処理は続行
	require "xxx.php";	ファイルの読み込み失敗した場合に E_COMPILE_ERROR レベルの致命的なエラー発生し、処理は中断
	require_once "xxx.php"; 又は **include_once**("xxx.php");	**require_once,include_once**とも既にロードしている場合は、ロードしない

文字列出力	**print** "あいうえお"."abc"; //文字連結での出力例
	echo "あいうえお","abc"; //コンマ区切りごとの出力例
	echoの省略形:**<?=出力文字列?>** …short_open_tag=onのとき	

HTMLコードの記述:ヒアドキュメント	$bgn_name = **<<<_EOL_** <div align="center"> … **_EOL_;**	ヒアドキュメントの仕様として「<<<」で、終了文字列に「;」が必要。
ヒアドキュメントの文字列表示	**echo <<<_EOL_** <div align="center"> … **_EOL_;**	echoの場合は、'='が不要
HTTPヘッダー送信(開始方法)	**header**("Content-Type: text/html; charset=UTF-8"); //↑「header」関数を利用	
HTMLコード内での変数出力(print/echo文)	echo "リンク"; //↑**{ }** で囲む必要ある(**<?php … ?>**で囲む方法もある)	変数有りのHTMLコード出力をヒアドキュメントの場合は気にしなくても良い。
CGI(フォームデータの入力)		
CGIの方法	HTTP要求のGETとPOSTについて、別々の関数が必要で、**$_SERVER["REQUEST_METHOD"]**でGET,POSTの値を得ることができる。**$_REQUEST**では、両方の可能	
GET	$inp_foo = **$_GET**['foo'];	method:get又はquery stringでのCGIによるデータ取得
POST	$inp_foo = **$_POST**['foo'];	method:postでのCGIによるデータ取得
URLエンコード	rawurlencode($str); //→空白を+に変換しない urlencode($str);	GETで日本語を送信したい時は、HTTPエンコードする必要がある。
URLデコード	rawurldecode(($str); //→+を空白に変換しない urldecode(($str);	
変数と定数		
可変変数	変数に「**$**」を先頭に付けると、変数の値を変数名とすることができる (例:$a="abc"; **$$a**= "値"; $abcの値は「値」となる)	
変数のスコープ	**<?php … ?>**内は、グローバル変数で、fuction内はローカル変数のため、関数内でグローバル変数を利用するために「**global** 」を先頭に付ける	
静的変数(クロージャー)	「**static**」を変数の前に付加(関数実行後もその値を保持)	
引数の参照渡し(リファレンス)	関数の引数のところで「&」を先頭に付ける。	(例:function rei($x,&$y){ … })→function内で引数を変更した場合は、参照元が変更される
定数の定義	**define** ("Infile","../dir/input_file.txt",true);	
ループ・分岐処理		
for文	**for** (開始条件式; 終了判断式; 増分式) { … }	各式は空にすることもできますし、複数の式をカンマで区切って指定することもできます。
if、else文	**if** (条件){ … } **else** { … }	
if文(次)	**elseif** (条件){ … }	
ループ処理の次へ進む	**continue;**	
ループ処理を抜ける	**break;**	
while文	**while** (条件式){ … }	
do while文	**do** { … } **while** (条件式)	

switch文	**switch** ($i) {
	case 0: …
	break;
	case 1: …
	}
		同じ変数を異なる値と比較し、値に応じて異なったコードを実行したいとき

switch文の条件に不一致のとき	**default**	
三項演算子 (?:)	$res = (条件式)**?** (trueの式) **:** (falseの式)	javascriptの実行動作と異なるので、入れ子の場合は、()で括る必要あり。
配列・連想配列		
配列・連想配列の相違点	配列操作の関数が豊富(連想配列の文字列キーでも配列の処理が可能、二重配列の定義も可能)	
配列の定義	$sample = **array**("a","b","c"); //↑「array()」を利用	
連想配列の定義	$sample = **array**("a" =>"10","b"=>"11","c"=>"12"); //↑「array()」を利用	配列と連想配列の区別がないので、変数名で区別が付くようにする
配列・連想配列の利用	$sample[1]["a"] = 'xxx'; //配列、連想配列とも、[ ]で区別	
配列(連想配列)ループ:foreach	**foreach**($sample as $data){ … } 、 **foreach**($sample as $key => $value){ … } //↑「as」を記述して、キーと値を得る。	
連想配列ループ:while	**while**(**list**($key, $value) = **each**($sample)){ … } //↑「list」、「each」を利用して、キーと値を得る。	
list処理	**list**($a,$b,$c) = $sample;	

分野	PHP	説明
配列の内容出力	**print_r** ($sample_ary); **var_dump**($sample_ary);	
配列の先頭の要素を削除	$xxx = **array_sift**($sample);	
配列の先頭に要素を追加	**array_unshift**($sample,$xxx);	
配列の末尾の要素を削除	$xxx = **array_pop**($sample);	
配列の末尾に要素を追加	**array_push**($sample,$xxx);	
配列の任意の要素を取得	$xxx = **array_slice**($sample,3,4);	(配列,取り出し開始位置,取り出し個数)
配列の任意の要素を置換	**array_splice**($sample,2,3,array("a","b","c");	(配列,削除開始位置,削除個数,置換する要素のリスト)
配列の要素数を取得	$xxx = **count**($sample); //↑「count」の関数利用	
配列の文字列結合(結合文字指定)	$str = **implode**(",",$ary);	joinは、implodeのエイリアス

配列のチェック	if(**is_array**($sample)){ … }
	//↑「is_array」の関数を利用　又は
	if(**gettype**($sample) == "array"){ … }
	//↑「gettype」の関数を利用	

配列の要素を検索	**array_search**($str,$sample);	
配列・連想配列のキーを取得	$keyarray = **array_keys**($sample);	
文字列操作		
文字列連結:[.]利用	$outstr　= $str.' あいうえお'; $outstr .= $nextstr;	
整数に変換	$number = **intval** ($str); //↑変数の文字列で0-9の数字を整数化(先頭文字が0-9以外の場合は、0を返す) $out = **settype**($number,"integer"); //↑"integer"の替わりに"int"も同じ	
文字列検索(部分文字の位置)	$pos = **strpos** ($str,$keywd [,$start]); //$startは、開始位置	見つかった場合は、文字列の位置(ポジション:1文字目が0)を返す。見つからない場合は、falseを返す
文字列分割(分割文字指定)to配列	$strarray = **explode**("-",$str);	"x"は、分割文字で、正規表現を利用できないのがその分高速処理。
変数にデータがあるかチェック	if(**isset**($str)){ … }	「isset」関数は、未定義とNULLのとき**false**を返す。
文字列の長さ	**strlen** ($str) //マルチバイト用の関数ある「**mb_strlen**」	
先頭と文末の空白文字を削除	**trim**($str);	
文末の指定文字の削除	**chop**:　//rtrim() のエイリアス	
正規表現		
正規表現の相違点	POSIX拡張正規表現(ereg_xxx)とPerl互換型正規表現:PCRE(preg_xxx) … Perl5.6以上とは一部レベルが低い PCREは、Perlの"g"パターン修飾子は利用不可	POSIXは、IEEE 規定されていますが、Perlの方が高機能です。
文字列一致(正規表現)	**preg_match**("/xxx/",$str); **preg_match_all**("/xxx/",$str,$ary,$flag);	/xxx/は、正規表現記述
文字列置換(正規表現)	**preg_replace**("/xxx/",$str,$repdata); **preg_replace_callback**("/xxx/",create_function(){…},$repdata));	/xxx/は、正規表現記述
文字列分割(正規表現)	$outarray = **preg_split**("/xxx/",$str); //↑Perl互換型の正規表現 $outarray = **split**("/xxx/",$str); //↑POSIX拡張正規表現	/xxx/は、正規表現記述
文字列検索(正規表現)	$outarray = **preg_grep**("/xxx/",$inarray);	/xxx/は、正規表現記述
ファイル操作		
ファイルのopen	define("FILE_PATH","./test/testfile.txt"); $fp = **fopen**(FILE_PATH,"**r**")　or die ("not fileread ! ¥n");	ファイルハンドル:[$fp] 読込モード:["r"] … ["r+"]は、+ writeが可 書込モード:["w"] … ["w+"]は、+ readが可 追加書込モード:["a"] … ["a+"]は、+ readが可
ファイル読込(1ファイル全体)	$fdata = **file** ("./test/testfile.txt")　or die ("not fileread ! ¥n");	オープン処理をせずに、ファイル全体を読み込む。レコードごとに配列として格納される。
ファイル読込(区切り)→配列入力	$fdata = **fgetcsv**($fp,0,"¥t");	レコード長を"0"とすると、1レコードの終わりまで。 tab,コンマ(,)などのデリミタとエンクロージャ(フィールドを囲む"など)… デフォルト(省略)は、「,」「"」
ファイル書込(区切り)←配列出力	**fputcsv**($fp,$fdata,"¥t");	tab,コンマ(,)などのデリミタとエンクロージャ(フィールドを囲む"など)… デフォルト(省略)は、「,」「"」
ファイルのバイナリーデータ処理	**fread** ($fp,FILE_PATH); **fwrite**($fp,"test data¥n");	fopenでバイナリーモード:"**b**"の追加が必要
ファイルのwrite	**fwrite**($fp,"test data¥n");　←**fputs**も同じ	レコード単位
ファイルのclose	**fclose**($fp);	

HTTPサーバから読み込む
 	
fopen()をHTTPサーバーに対して行う
	$fh = **fopen**("http://www.xxx.com/","**r**")
	while (($line = **fgets**($fh) !== false) {
	　print $line;
	}
	fclose($fh);
		「header」にある「charset」コードを判断して、コード変換処理をする必要があります。

ファイルアップロード	define ("UPLOAD_DIR","./test/");
	$outfile=UPLOAD_DIR.**$_FILES**["upfile"]["**name**"];
	if (!**$_FILES**["upfile"]["**error**"]){
	　move_uploaded_file(**$_FILE**["upfile"]["**tmp_name**"],$outfile);
	}
		formで「enctype="multipart/form-data"」と「input type="**file**" name="upfile" 」の場合に、サーバー側でのCGI処理内容。テンポラリファイルに一時的に保存して、**$_FILES**のエラーがない場合に所定のディレクトリへ保存する形態をとる。

オブジェクト指向(クラス)		
クラスの相違点	C++言語やJavascriptのような「クラスとオブジェクト」の機能	

クラス	**class** Hello_class{
	　var $name = "CXMedia";
	　function DisplayHello(){
	　　print "こんにちは、".$this->name."さん。";
	　}
	}
		"this"は、インスタンス化されたものを表し、「$this -> name」は自分自身を指す。

オブジェクト	$obj1 = **new** Hellow_class(); $obj2 = **new** Hellow_class();	new演算子でインスタンス化し、$obj1と$obj2は、クラスHellow_classをインスタンス化したオブジェクトです。

分野	PHP	説明
プロパティとメソッド	$obj1 = new Hello_class ("Sunrise"); $obj2 = new Hello_class ( ); $obj1 -> DisplayHello ( ); // ↑「こんにちは、Sunriseさん。」と表示 $obj2 -> DisplayHello ( ); // ↑「こんにちは、CXMediaさん。」と表示	クラス内で定義された変数(name)をプロパティ(Property)、関数(DisplayHello ( ))をメソッド(Method)と呼ぶ。"->"演算子を使って呼び出せる。

コンストラクタ	**class** Hello_class{
	  var $name;
	  function _construct ($name="CXMedia"){
	  $this -> name = $name;
	  }
	  function DisplayHello ( ){
	  print "こんにちは、".$this->name."さん。";
	  }
	}
		_construct又は、クラス内でクラス名と同じ名前のメソッドをコンストラクタ(Constructor)と呼ぶ。変数を初期化する場合に利用。

継承(インヘリタンス)		

	// コンストラクタ //
	class Hello_class{
	  var $name;
	  function _construct ($name="CXMedia"){
	  $this -> name = $name;
	  }
	  function DisplayHello ( ){
	  print "こんにちは、".$this->name."さん。";
	  }
	}
	// 継承のクラス //
	class Hello_class2 **extends** Hello_class {
	  function DisplayHello ( ) {
	  print "あなたは誰？";
	  Hello_class::DisplayHello ( );
	  }
	}
		Hello_class2の中でHello_classのメソッドと同じ名前のメソッドDisplayHello ( )が定義されているが、同じ名前のメソッドが定義されていると、オーバーライド(Override)されてHello_class2の方のメソッドが有効になり、それ以外のプロパティ、メソッドは継承されたまま。
		Hello_class::DisplayHello ( )のように子クラスから親クラスのメソッドを呼び出せる。

	// オブジェクト // $obj1 = new Hello_class2 ("Sunrise"); $obj1 -> DisplayHello ( ); // ↑「あなたは誰？こんにちは、Sunriseさん。」と表示	