

# PHPプログラム

---

```
<?php echo "Hello PHP"; ?>
```

- PHPプログラムはPHPタグ（`<?php` と `?>`）の中に記述する
- `echo`命令を使うとターミナルにデータを出力できる
- PHPプログラムはターミナル上で`php`コマンドを使って実行する

## basic1.php

```
<?php  
echo "Hello PHP";  
?>
```

PHPタグの内側はPHPのパースャによって解析されます。PHPのステートメント（文）の終端にはセミコロン;  
が必要です。

## Terminal

```
$ php basic1.php  
Hello PHP
```

## basic2.php

```
<?php  
echo "Hello PHP";  
echo "Hello Python";  
echo "Hello Java";  
?>
```

## Terminal

```
$ php basic2.php  
Hello PHPHello PythonHello Java
```

PHPの開始タグ`<?php`と終了タグ`?>`の中に記述したステートメントが上から順番に処理されているのがわかります。

## PHPタグ

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>PHP Sample</title>
</head>
<body>
  <h1>Hello <?php echo "PHP"; ?></h1>
</body>
</html>
```

- PHPタグの内側はPHPのパースによって解析される
- PHPタグの外側はPHPのパースに無視される（通常のテキストとして出力される）
- ファイルの末尾にある終了タグ`?>`は省略できる

### basic3.php

```
Hello Python

<?php
echo "Hello PHP";
?>

Hello Java
```

### Terminal

```
$ php basic3.php
Hello Python

Hello PHP
Hello Java
```

## basic4.php

```
<?php  
echo "Hello PHP";  
?>
```

ファイルがPHPプログラムのみで構成される場合、終了タグ：`?>`は省略すべきです。終了タグの外側に余計な空白や改行があると、それらも出力されてしまうためです。

## Terminal

```
$ php basic4.php  
Hello PHP
```

## コメント

```
<?php
// Single line comment
# Single line comment
/*
Multiple
lines
comment
*/
echo "Hello PHP";
```

- コメントとはプログラムの中にメモを残す仕組み
- 単一行のコメントには//や#を使う
- 複数行のコメントには/\* \*/を使う

### basic5.php

```
<?php
// This is My First Programming!
echo "Hello PHP";
```

//を#に変更してもコメントとして機能します。//はCやC++でサポートされているコメントの形式です。#はUnixのシェルでサポートされているコメントの形式です。PHPはその両方をサポートしています。

### Terminal

```
$ php basic5.php
Hello PHP
```

**basic5.php - 修正**

```
<?php
/*
  This is
  My First
  Programming!
*/
echo "Hello PHP";
```

**Terminal**

```
$ php basic5.php
Hello PHP
```

## phpコマンド

```
$ php basic1.php
```

- PHPプログラムを実行するためのコマンド
- `php FILENAME`のように実行するファイルを指定する
- `php`コマンドにオプションを指定することもできる

### phpコマンドのオプション

オプション	機能
<code>-v</code>	PHPのバージョンを表示する
<code>-a</code>	対話形式でPHPを実行する
<code>-i</code>	PHPの設定情報を表示する

ここでは指定可能なオプションの一部を記載しています。

#### Terminal - `-v`オプション

```
$ php -v
PHP 7.1.23 (cli) (built: Nov  7 2018 18:20:35) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
```

`-v`オプションを指定するとPHPのバージョンを確認できます。

#### Terminal - `-a`オプション

```
$ php -a
Interactive shell

php > echo "Hello";
Hello
```

`-a`オプションを指定すると対話形式でPHPを実行できます。`quit`と入力すると終了できます。

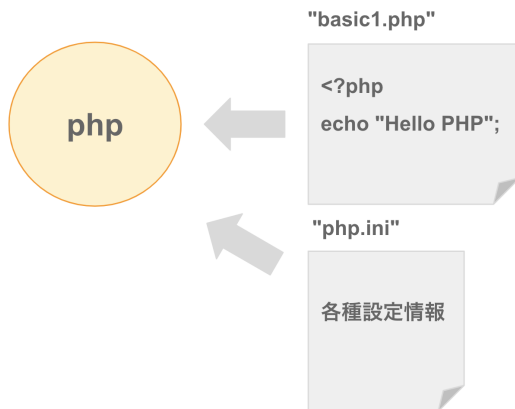
## Terminal - `-i`オプション

```
$ php -i
phpinfo()
PHP Version => 7.2.22

System => Darwin murayamamasahikonoMacBook-Pro.local 18.2.0 Darwin Kernel
Version 18.2.0: Thu Dec 20 20:46:53 PST 2018; root:xnu-
4903.241.1~1/RELEASE_X86_64 x86_64
Build Date => Sep  8 2019 15:17:40
...省略
```

`-i`オプションを指定するとPHPの設定情報を確認できます。同様の情報は後述する`phpinfo()`関数で取得することもできます。

## php.iniファイル



- **php.ini**ファイルはPHPの設定情報を管理する
- **/etc/php.ini**など規定のフォルダに配置する
- **php.ini**ファイルを編集することで、エラーの出力レベル、マルチバイト文字列の設定、タイムゾーンの設定などPHPの様々な設定をカスタマイズできる

Macに標準でインストールされているPHPの場合、デフォルトでは**php.ini**ファイルが作成されていません。以下の**cp**コマンドで**php.ini**ファイルをコピーして作成します。

```
sudo cp /etc/php.ini.default /etc/php.ini
```

**/etc**フォルダ以下を編集するには管理者権限が必要です。**sudo**コマンドは管理者権限で実行するコマンドです。

### info.php

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

### Terminal

```
$ php -S localhost:8000
```

**-S**オプションを指定するとPHPに付属しているビルトインWebサーバを起動できます。またキーボードから**ctrl - c**とタイプすることでビルトインWebサーバを停止できます。ビルトインWebサーバの詳細については後述します。



**Browser**

```
http://localhost:8000/info.php
```

# 変数

---



- 変数とはプログラムの中でデータを扱う仕組み
- 変数は箱のようなものでデータを代入したり、参照したりできる
- 変数はプログラムの実行時にメモリ上に確保される領域

## var1.php

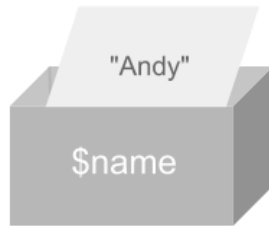
```
<?php
$name = "Andy";
echo "Hello ";
echo $name;
```

PHPの変数は\$nameのように先頭に\$がつきます。

## Terminal

```
$ php var1.php
Hello Andy
```

## 変数の命名規則



- PHPの変数名は先頭文字を\$記号で始める
- 変数名には半角英数字や\_アンダースコアを使う
- ただし\$記号の直後に数字を使うことはできない

### var2.php - 妥当な変数名

```
<?php
$name = "Andy";
$name2 = "Betty";
$name . $name2;
echo $name; # => AndyBetty
```

- 記号は2つの文字列データを連結するために使います。

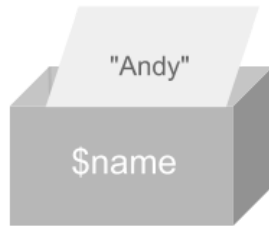
### Terminal

```
$ php var2.php
AndyBetty
```

### var3.php - 不正な変数名

```
<?php
name = "Andy";           # $記号で始まっていない
$2name = "Betty";        # $記号の直後に数字は使えない
$name - $name2;          # 変数名に-（ハイフン）は使えない
```

## 変数の命名規則 - 大文字小文字の取り扱い



- 変数名はアルファベットの太文字小文字を区別する
- 変数名には小文字を使うことが一般的
- 変数名に2つ以上の単語を組み合わせる場合はキャメルケース、スネークケースといった表記法がある

### var4.php

```
<?php
$name = "Andy";
$name = "Betty";
$name = "Carol";
echo $name; # => Andy
echo $Name; # => Betty
echo $NAME; # => Carol
```

上記の3つの変数の中では\$nameが一般的な名前の付け方です。

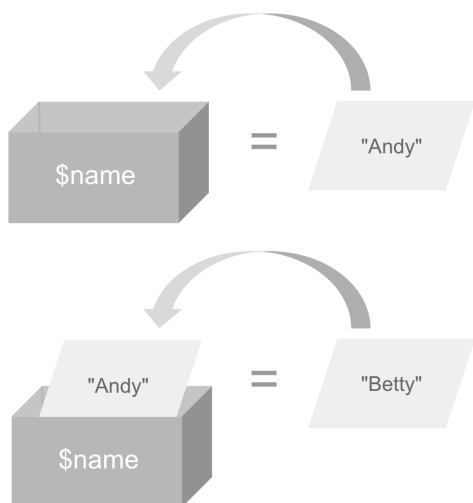
### 参考：First Nameという変数名を表現する場合

```
<?php
$firstName = "Andy"; # キャメルケース
$first_name = "Andy"; # スネークケース
```

### Terminal

```
$ php var4.php
AndyBettyCarol
```

## 変数の代入



- 変数にデータを代入するには `=` 記号を使う
- `=` 記号は代入演算子と呼ぶ
- 変数の中のデータは新たなデータの代入で上書きされる

演算子とは `=` や `+` などの計算記号のことです。

### var5.php

```
<?php
$name = "Andy";
echo $name; # => Andy
$name = "Betty";
echo $name; # => Betty
```

### 構文

```
$変数名 = データ;
```

1. 代入先となる変数名を先（左辺）に記述する
2. 次に代入演算子 `=` を記述する
3. 最後（右辺）に変数に代入するデータを記述する

プログラムは `=` の右辺が先に評価される

### よくある間違い

```
"Andy" = $name; # 変数名を先に記述していない
```

## Terminal

```
$ php var5.php  
AndyBetty
```

# データ型



- PHPで扱うデータにはいくつかの種類がある
- データの種類のことをデータ型と呼ぶ
- 主要なデータ型は以下の通り

データ型名	データ型名	使用例
整数型	intger, int	\$price = 1000; \$discount = -100
浮動小数点数型	float, double	\$pi = 3.14; \$tax = 1.10;
文字列型	string	\$message = "Hello "; \$name = 'Andy';
論理型	boolean, bool	\$student = true; \$student = false;
配列型	array	\$names = ["Andy", "Betty", "Carol"];
リソース型	resource	\$handle = fopen("names.csv", "r");
NULL型	null	\$empty = null;

他にもIterable、オブジェクト、コールバックといった型も存在します。これらについては本講座では取り扱いません。

## var6.php

```
<?php
$name = "Andy";
$age = 20;
echo "NAME:" . $name; # => NAME:Andy
echo "AGE:" . $age; #=> AGE:20
```

## Terminal

```
$ php var6.php
NAME:AndyAGE:20
```

## 整数型

1000

- 整数データをそのまま記述する（`''`は不要）
- `integer`型、`int`型と呼ぶ
- 2進数表記、8進数表記、16進数表記もサポートしている

### var7.php

```
<?php
$price = 1000;
$discount = -100;
echo "PRICE:" . $price; # => PRICE:1000
echo "DISCOUNT:" . $discount; # => DISCOUNT:-100
```

### Terminal

```
$ php var7.php
PRICE:1000DISCOUNT:-100
```

### 参考：10進数以外の表記について

- 2進数
  - データの先頭に0bを付与する
- 8進数
  - データの先頭に0を付与する
- 16進数
  - データの先頭に0xを付与する



## var8.php

```
<?php
$a = 10;    # 10進数
$b = 0b10;  # 2進数 先頭に0bを付与する（10進数の2と同じ）
$c = 010;   # 8進数 先頭に0を付与する（10進数の8と同じ）
$d = 0x10;  # 16進数 先頭に0xを付与する（10進数の16と同じ）
echo $a; # => 10
echo ",";
echo $b; # => 2
echo ",";
echo $c; # => 8
echo ",";
echo $d; # => 16
```

## Terminal

```
$ php var8.php
10,2,8,16
```

## 浮動小数点数型

3.14

- 実数（小数点を含むデータ）を扱うデータ型（`''`は不要）
- `float`型、`double`型と呼ぶ
- 丸め誤差を含む（通常はIEEE754倍精度フォーマットに従う）

### var9.php

```
<?php
$price = 1000;
$tax = 1.10;
echo $price * $tax; # => 1100
```

### Terminal

```
$ php var9.php
1100
```

## 文字列型



"Hello"

- データを`""`ダブルクォーテーションあるいは`' '`シングルクォーテーションで囲む
- `""`ダブルクォーテーションで囲む場合は変数展開可能
- `string`型呼ぶ

### var10.php

```
<?php
$name = "Andy";
echo $name; # => Andy
$name2 = 'Betty';
echo $name2; # => Betty
```

### Terminal

```
$ php var10.php
AndyBetty
```

### var11.php - 変数展開

```
<?php
$name = "Andy";

$message1 = 'Hello $name'; # 変数を展開しない
echo $message1; # => Hello $name

$message2 = "Hello $name"; # 変数を展開する
echo $message2; # => Hello Andy
```

### Terminal

```
$ php var11.php
Hello $nameHello Andy
```

## 論理型



true

- 真を表す`true`と偽を表す`false`の2つのデータで構成されるデータ型
- `boolean`型、`bool`型と呼ぶ
- 論理型データの確認には`var_dump`関数を使う

### var12.php

```
<?php
$name = "Andy";
$student = true;

echo $name; # => Andy
echo ",";
var_dump($student); # => bool(true)
```

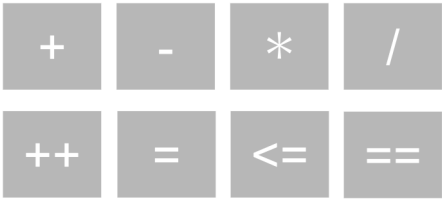
論理型データは`echo`命令で出力すると`true`の場合は`1`、`false`の場合は`""`空文字に置き換わってしまいます。ここでは`var_dump`関数を使ってデータを確認しています。

### Terminal

```
$ php var12.php
Andy,bool(true)
```

論理型のデータ（`true`、`false`）は後で学習する`if`文や`for`文などと相性の良いものになっています。

# 演算子



- プログラム上で使う計算記号のこと
- 四則演算や、値の大小比較などを行う
- 主要な演算子は以下のとおり

名前	役割	例
代数演算子	加算、減算、乗算、除算、剰余、累乗	<code>+, -, *, /, %, **</code>
加算子/減算子	変数の値を1増やす、1減らす	<code>++, --</code>
文字列演算子	文字列の結合	<code>.</code>
代入演算子	変数への代入	<code>=, +=, -=, *=, /=, .=</code>
比較演算子	値の大小比較	<code>&gt;, &gt;=, &lt;, &lt;=, ==, !=</code>
論理演算子	論理積、論理和、否定	<code>&amp;&amp;,   , !</code>

PHPには他にも、ビット演算子、エラー制御演算子、実行演算子、配列演算子、型演算子などが用意されています。

## var13.php

```
<?php
$price = 1000;
$count = 10;
$total_price = $price * $count;
echo $total_price; # => 10000
```

## Terminal

```
$ php var13.php
10000
```

## 代数演算子



- 四則演算  $+$   $-$   $*$   $/$
- 余り  $\%$
- 累乗  $**$

### var14.php

```
<?php
$x = 5;
$y = 2;
$z = $x + $y;
echo $z; # => 7
echo ",";
$z = $x - $y;
echo $z; # => 3
echo ",";
$z = $x * $y;
echo $z; # => 10
echo ",";
$z = $x / $y;
echo $z; # => 2.5
echo ",";
$z = $x % $y;
echo $z; # => 1
echo ",";
$z = $x ** $y;
echo $z; # => 25
```

### Terminal

```
$ php var14.php
7,3,10,2.5,1,25
```

$\%$  剰余演算子は変数の値が2の倍数や3の倍数などとチェックしたい場合に利用します。2の倍数かチェックするには、求めた剰余（余り）が0かどうかで判定できます。

## var15.php

```
<?php
$age = 20;
$age = $age + 1; # 20 + 1と同じ意味
echo $age; # => 21
```

## Terminal

```
$ php var15.php
21
```

## 加算子/減算子



- 加算子 `++` (変数の値を1増やす)
- 減算子 `--` (変数の値を1減らす)
- インクリメント、デクリメントと呼ぶ

### var16.php

```
<?php
$page = 20;
$page++; # $page = $page + 1; と同じ
echo $page; #=> 21
```

繰り返し処理 (`for`文) や配列の取り扱いなど、変数の値を1増やしたいというケースは頻繁に発生します。加算子`++`を使うとプログラムをシンプルに表現できます。

### Terminal

```
$ php var16.php
21
```

### var16.php - 修正

```
<?php
$page = 20;
$page--; # $page = $page - 1; と同じ
echo $page; #=> 19
```

### Terminal

```
$ php var16.php
19
```



## 文字列演算子



- `.` 演算子は文字列を連結する
- 整数型データなど異なるデータ型も文字列として連結できる
- 改行コードは `PHP_EOL` 定数

### var17.php

```
<?php
$message = "Hello ";
$name = "Andy";
$message = $message . $name;
echo $message; # => Hello Andy
```

#### Terminal

```
$ php var17.php
Hello Andy
```

### var18.php

```
<?php
$label = "AGE:";
$age = 20;
echo $label . $age; # => AGE:20
```

#### Terminal

```
$ php var18.php
AGE:20
```

## var19.php

```
<?php
$name = "Andy";
$name2 = "Betty";
echo $name . PHP_EOL;
echo $name2 . PHP_EOL;
```

## Terminal

```
$ php var19.php
Andy
Betty
```

改行コードはWindowsやmacOS、Linuxなどプラットフォームによって異なります。Windowsは`\r\n`、macOSやLinuxでは`\n`です。PHP\_EOL定数はプラットフォームに合わせた改行コードを出力します。

## 代入演算子



- 代入演算子 `=`
- 複合代入演算子 `+=`, `-=`, `*=`, `%=`, `.=`
- 複合代入演算子を使えば計算と代入をまとめてできる

### var20.php

```
<?php
$price = 1000;
$price += 500; # $price = $price + 500; と同じ
$price *= 1.10; # $price = $price * 1.10; と同じ
echo $price; # => 1650
```

### Terminal

```
$ php var20.php
1650
```

### var21.php

```
<?php
$message = "Hello ";
$message .= "Andy";
echo $message; # => Hello Andy
```

### Terminal

```
$ php var21.php
Hello Andy
```

## 比較演算子



- 2つのデータを比較する
- `>`, `>=`, `<`, `<=`, `==`, `!=`
- 比較演算子の評価結果は論理型データ(`true`, `false`)になる

### var22.php

```
<?php
$apple = 100;
$banana = 200;
$result = $apple > $banana;
var_dump($result); # => bool(false)
```

論理型のデータは`echo`で出力せずに`var_dump`関数で出力しています。

### Terminal

```
$ php var22.php
bool(false)
```

### var22.php - 修正

```
<?php
$apple = 100;
$banana = 200;
$result = $apple < $banana;
var_dump($result); # => bool(true)
```

値の大小比較には `>=` や `<=` を使うこともできます。この場合、以上（以下）といった比較になります。

### var22.php - 修正

```
<?php
$apple = 100;
$banana = 200;
$result = $apple == $banana;
var_dump($result); # => bool(false)
```

## var22.php - 修正

```
<?php
$apple = 100;
$banana = 200;
$result = $apple != $banana;
var_dump($result); # => bool(true)
```

演算子の優先度についても復習しておきましょう。代入演算子`=`を見つけたら、`=`の後の部分（`$apple != $banana`）が先に処理されると考えてください。これは代入演算子`=`に比べて比較演算子（`>`など）の方が処理の優先度が高いためです。

# トレーニング

---

## var\_tr1.php

次のプログラムがあります。

```
<?php
$message = "PHP Training";
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php var_tr1.php
PHP Training
```

---

## var\_tr2.php

次のプログラムがあります。

```
<?php
$color1 = "Red";
$color2 = "Green";
$color3 = "Blue";
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php var_tr2.php
Red, Green, Blue
```

---

## var\_tr3.php

次のプログラムがあります。

```
<?php
$price = 100;
$count = 3;
$tax = 1.1;
```

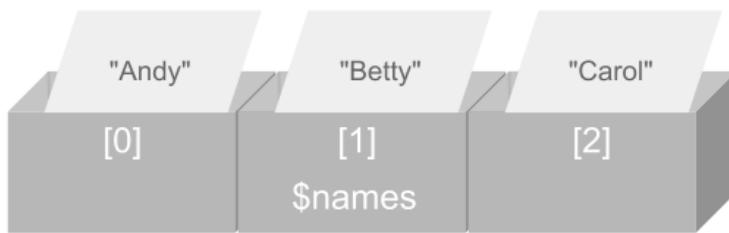
次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php var_tr3.php
330
```

---

# 配列



- 配列はプログラム上で複数のデータを管理する仕組み
- 配列は変数的一种であり、配列変数とも呼ぶ
- `[]` の中にデータを定義する

## 構文

```
$変数名 = [データ, データ, データ];
```

## array1.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

echo "Hello ";
echo $names[0];
echo "Hello ";
echo $names[1];
echo "Hello ";
echo $names[2];
```

配列変数の `[]` に指定する数値のことを要素番号や添字、インデックス、キーなどと呼びます。

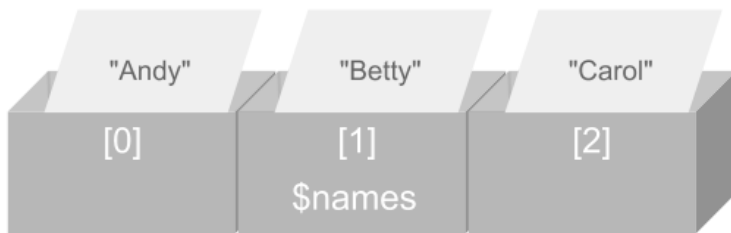
## Terminal

```
$ php array1.php
Hello AndyHello BettyHello Carol
```

ここで学習する配列は、後で学習する連想配列と区別して添字配列と呼ぶこともあります。



## 要素番号の指定



- 配列を構成する一つひとつのデータを要素と呼ぶ
- 要素番号は0番から割り振られる
- 存在しない要素番号を指定するとエラー

### array2.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

echo $names[3];
```

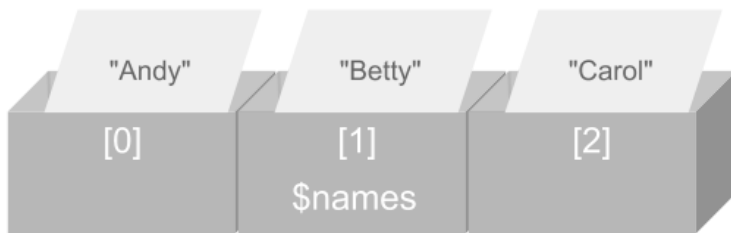
### Terminal

```
$ php array2.php
PHP Notice:  Undefined offset: 3 in /Users/murayama/Desktop/code-
php/array2.php on line 4

Notice: Undefined offset: 3 in /Users/murayama/Desktop/code-php/array2.php
on line 4
```

PHPのエラーメッセージにはError（エラー）、Warning（警告）、Notice（注意）など様々な種類があります。ここではPHP Noticeと先頭に出力されています。

## 配列の出力



- 配列変数は`echo`で出力できない
- 配列変数の出力には`var_dump`関数を使う
- `var_dump`関数は変数の詳細な情報を出力する

### array3.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

echo $names;
```

### Terminal

```
$ php array3.php
PHP Notice: Array to string conversion in /Users/murayama/Desktop/code-
php/array3.php on line 4

Notice: Array to string conversion in /Users/murayama/Desktop/code-
php/array3.php on line 4
```

`Array to string conversion`という出力から「配列を文字列に変換している」と読み取れます。

### array3.php - 修正

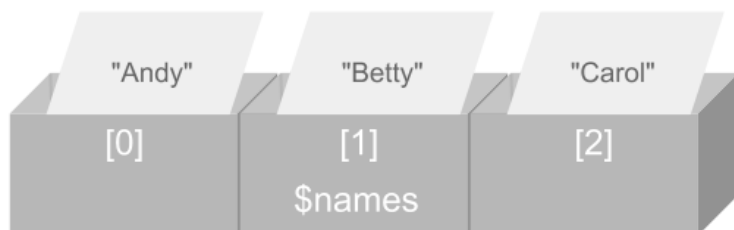
```
<?php
$names = ["Andy", "Betty", "Carol"];

var_dump($names);
```

## Terminal

```
$ php array3.php  
array(3) {  
    [0]=>  
    string(4) "Andy"  
    [1]=>  
    string(5) "Betty"  
    [2]=>  
    string(5) "Carol"  
}
```

## 配列の要素数の取得



- 配列の要素数を取得するには`count`関数を使う
- `count`関数は要素数を整数型データ (`int`型) で返す
- `count`関数は後述するfor文と組み合わせて使うことが多い

### array4.php

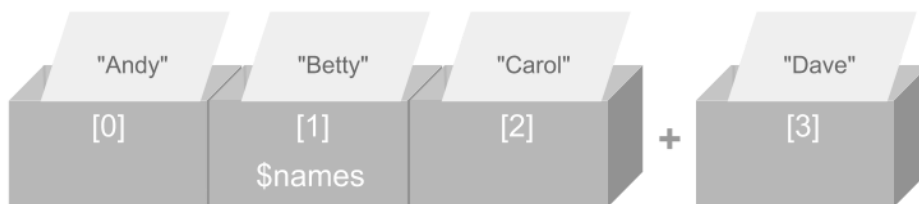
```
<?php
$names = ["Andy", "Betty", "Carol"];
$count = count($names);

echo $count;
```

### Terminal

```
$ php array4.php
3
```

## 要素の追加



- 配列変数に要素を追加できる
- 追加時は要素番号を省略して代入する
- 追加した要素は配列の後部に追加される

### 構文

```
$配列変数[] = データ;
```

### array5.php

```
<?php
$names = ["Andy", "Betty", "Carol"];
$names[] = "Dave";

echo $names[3];
```

### Terminal

```
$ php array5.php
Dave
```

### array5.php - 修正

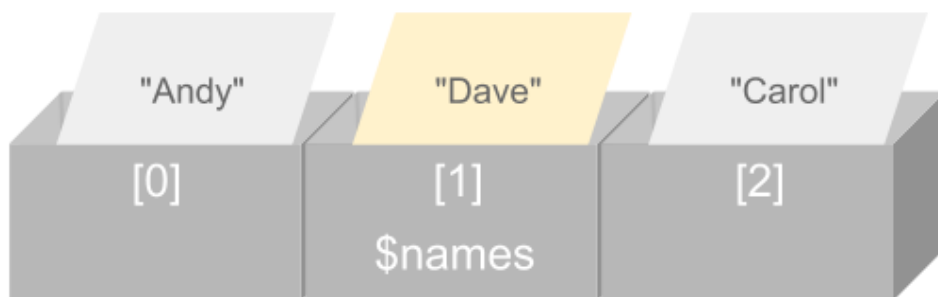
```
<?php
$names = ["Andy", "Betty", "Carol"];
$names[] = "Dave";

echo count($names);
```

## Terminal

```
$ php array5_2.php  
4
```

## 要素の変更



- 既存の要素に新たなデータを代入できる
- 変更時は、配列変数に要素番号を指定して代入する
- 存在しない要素番号を指定すると、指定した要素番号で追加される

### 構文

```
$配列変数[要素番号] = データ;
```

### array6.php

```
<?php
$names = ["Andy", "Betty", "Carol"];
$names[1] = "Dave";

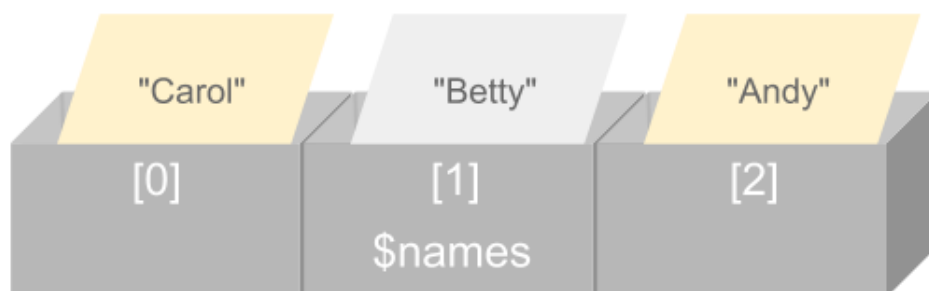
echo $names[0];
echo $names[1];
echo $names[2];
```

### Terminal

```
$ php array6.php
AndyDaveCarol
```

## 要素の変更

### 要素の入れ替え



- 2つの要素を入れ替えることができる
- ただし要素の入れ替えには工夫が必要
- 一時的にデータを代入する変数を用意する

### array7.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

$names[0] = $names[2];
$names[2] = $names[0];

var_dump($names);
```

### Terminal

```
$ php array7.php
array(3) {
  [0]=>
  string(5) "Carol"
  [1]=>
  string(5) "Betty"
  [2]=>
  string(5) "Carol"
}
```



## array7.php - 修正

```
<?php
$names = ["Andy", "Betty", "Carol"];

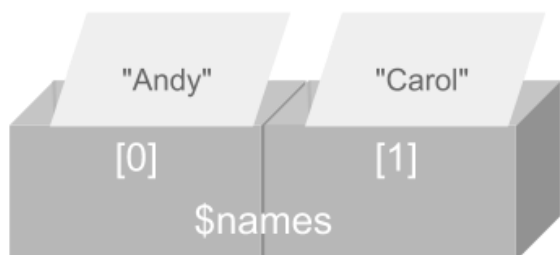
$temp = $names[0];
$names[0] = $names[2];
$names[2] = $temp;

var_dump($names);
```

## Terminal

```
$ php array7.php
array(3) {
  [0]=>
  string(5) "Carol"
  [1]=>
  string(5) "Betty"
  [2]=>
  string(4) "Andy"
}
```

## 要素の削除



- 配列の要素を削除するには関数を使う
- `unset`関数を使うと要素番号に空きが生じるので注意
- `array_splice`関数を使うと配列の要素を削除できる

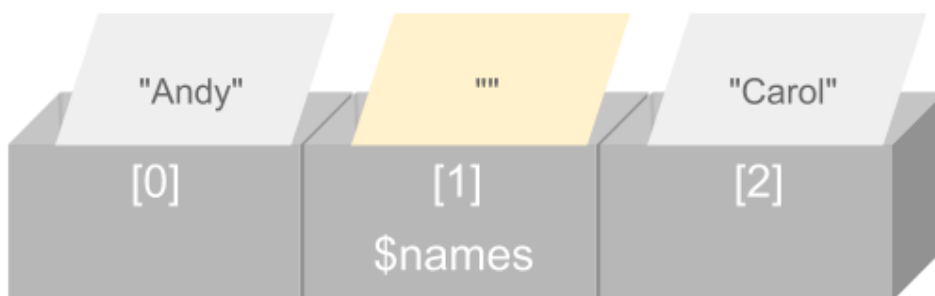
### array8.php

```
<?php
$names = ["Andy", "Betty", "Carol"];
$names[1] = "";

var_dump($names);
```

### Terminal

```
$ php array8.php
array(3) {
  [0]=>
  string(4) "Andy"
  [1]=>
  string(0) ""
  [2]=>
  string(5) "Carol"
}
```



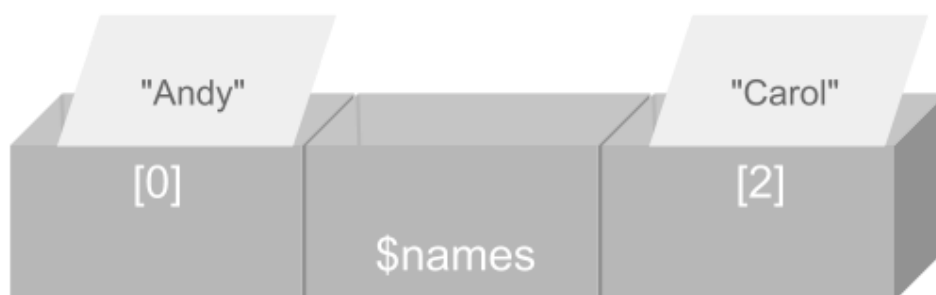
## array9.php - unset関数

```
<?php
$names = ["Andy", "Betty", "Carol"];
unset($names[1]);

var_dump($names);
```

### Terminal

```
$ php array9.php
array(2) {
  [0]=>
    string(4) "Andy"
  [2]=>
    string(5) "Carol"
}
```



unset関数による配列の要素の削除は、要素番号に空きが生じるため注意が必要です。配列の要素番号に空きがあると扱いにくいものになってしまいます。

## array10.php - array\_splice関数

```
<?php
$names = ["Andy", "Betty", "Carol"];
array_splice($names, 1, 1);

var_dump($names);
```

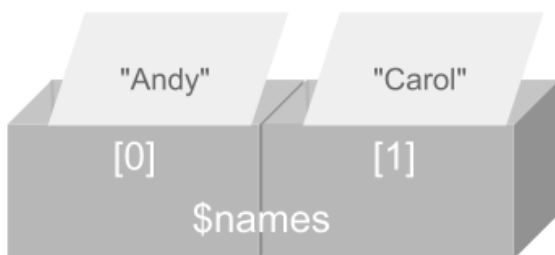
- array\_splice関数の引数には以下の3つを指定しています。

1. 削除対象の配列：\$names
2. 削除対象の要素番号（開始位置）：1
3. 削除対象の要素数：1

引数とは関数に与えるデータのことです。関数の仕組みの詳細については後述します。

### Terminal

```
$ php array10.php
array(2) {
  [0]=>
    string(4) "Andy"
  [1]=>
    string(5) "Carol"
}
```



# トレーニング

---

## array\_tr1.php

次のプログラムがあります。

```
<?php
$points = [10, 20, 30];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php array_tr1.php
60
```

---

## array\_tr2.php

次のプログラムがあります。

```
<?php
$colors = [];

// TODO $colorsに"Red", "Green", "Blue"を追加します。

var_dump($colors);
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php array_tr2.php
array(3) {
  [0]=>
  string(3) "Red"
  [1]=>
  string(5) "Green"
  [2]=>
  string(4) "Blue"
}
```

## array\_tr3.php

次のプログラムがあります。

```
<?php
$colors = ["Red", "Green", "Blue"];

// TODO $colorsから"Red"を削除します。

var_dump($colors);
```

次の実行結果となるようにプログラムを作成してください。

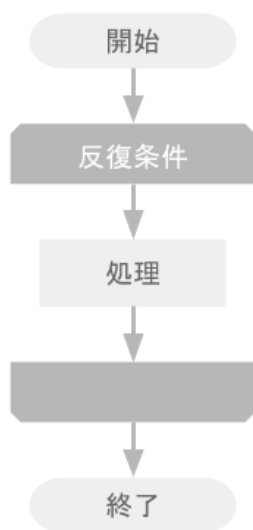
### 実行結果

```
$ php array_tr3.php
array(2) {
  [0]=>
  string(5) "Green"
  [1]=>
  string(4) "Blue"
}
```

---

# while文

---



- while文は反復構造（繰り返し）を定義する
- 反復条件( )には比較演算子（<など）を使うことが多い
- while文の{ }の中に記述した処理が繰り返し実行される

## 構文

```
while (反復条件) {  
    // 処理  
}
```

## loop1.php

```
<?php  
$i = 0;  
while ($i < 3) {  
    echo $i . PHP_EOL;  
    $i++;  
}
```

## Terminal

```
$ php loop1.php  
0  
1  
2
```

## 反復条件

- while文の反復条件( )で繰り返しを制御する
- 反復条件によっては一度も実行されないこともある
- 反復条件は評価されると論理値 (true, false) になる

### loop1.php - 修正

```
<?php
$i = 0;
while ($i <= 0) {
    echo $i . PHP_EOL;
    $i++;
}
```

### Terminal

```
$ php loop1.php
0
```

### loop1.php - 修正

```
<?php
$i = 0;
while ($i > 5) {
    echo $i . PHP_EOL;
    $i++;
}
```

### Terminal

```
$ php loop1.php
```

反復処理が一度も実行されないなので、上記のように何も出力されないようになります。



## loop1.php - 修正

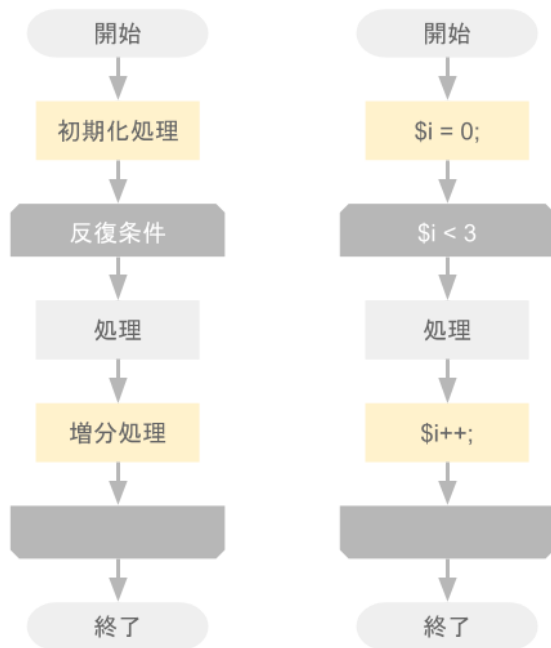
```
<?php
$i = 0;
while (true) {
    echo $i . PHP_EOL;
    $i++;
}
```

## Terminal

```
$ php loop1.php
0
1
2
3
...無限に繰り返す
```

ターミナルでctrlキーとcキーを同時に押すとプログラムは停止します。

## 初期化処理と増分処理



- 反復を制御するためにカウンタ変数を使う
- 初期化処理：カウンタ変数を初期化する
- 増分処理：カウンタ変数を増加（減少）する

カウンタ変数には`$i`を使うことが一般的です。またカウンタ変数の初期値には0、増分処理はインクリメントを使うことが多いです。

### loop1.php - 修正

```
<?php
$i = 0;
while ($i < 3) {
    echo $i . PHP_EOL;
    $i++;
}
```

### Terminal

```
$ php loop1.php
0
1
2
```

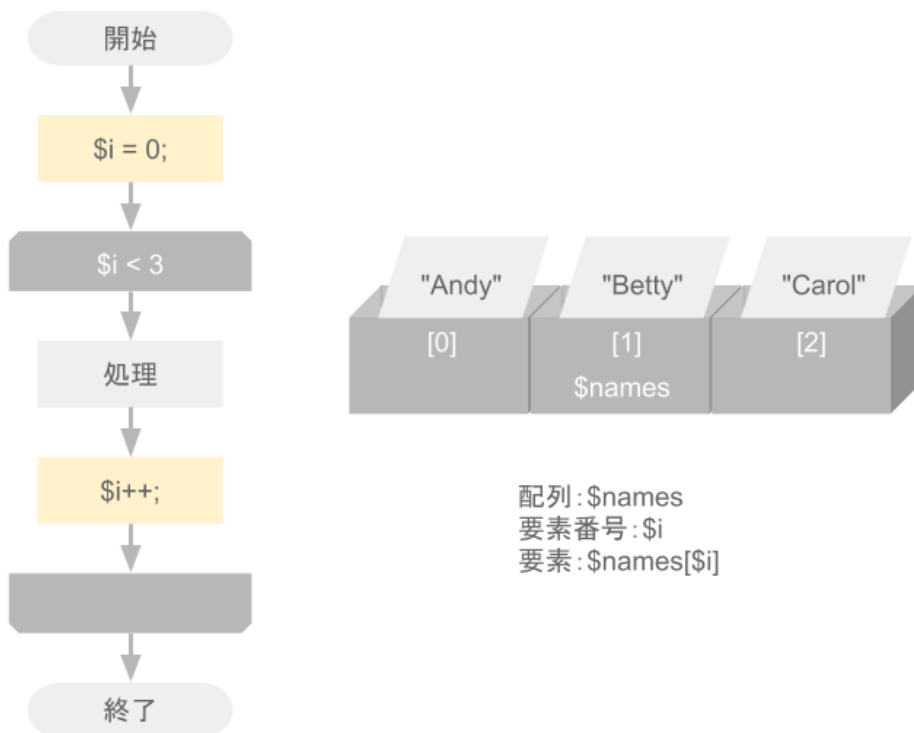
## loop1.php - 修正

```
<?php
$i = 3;
while ($i > 0) {
    echo $i . PHP_EOL;
    $i--;
}
```

## Terminal

```
$ php loop1.php
3
2
1
```

## 配列と反復構造



- 配列は反復構造と一緒に利用することが多い
- 配列の要素番号にカウンタ変数を使う
- カウンタ変数の初期値を0とし、配列の要素数の上限までインクリメントする

### array1.php

```
<?php  
$names = ["Andy", "Betty", "Carol"];  
  
echo "Hello ";  
echo $names[0];  
echo "Hello ";  
echo $names[1];  
echo "Hello ";  
echo $names[2];
```

同じようなコードを繰り返し記述している点に注目します。

## loop2.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

$i = 0;
while ($i < 3) {
    echo "Hello ";
    echo $names[$i] . PHP_EOL;
    $i++;
}
```

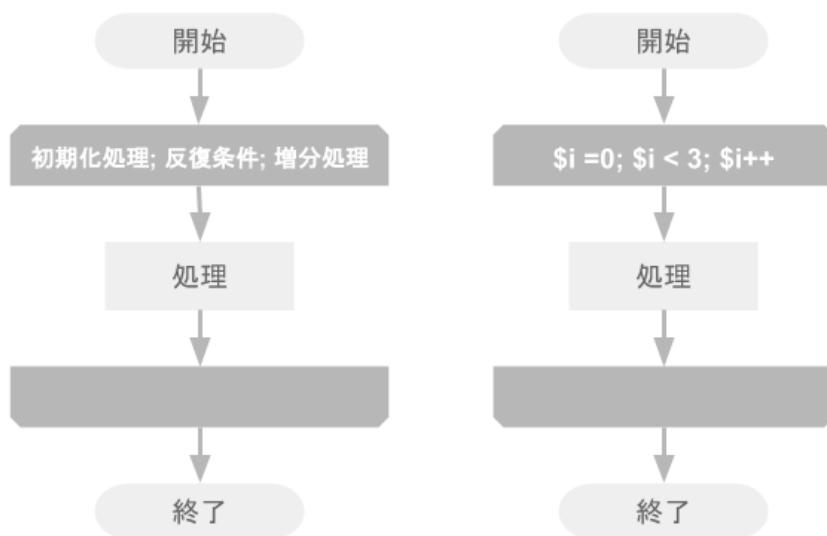
反復条件の `$i < 3` の部分は `$i < count($names)` と実装できます。

## Terminal

```
$ php loop2.php
Hello Andy
Hello Betty
Hello Carol
```

# for文

---



- for文は反復構造（繰り返し）を定義する
- 初期化处理、反復条件、増分処理をまとめて記述する
- 反復構造の全体像を把握しやすい

## 構文

```
for ( 初期化处理; 反復条件; 増分処理 ) {  
    // 処理  
}
```

## loop2.php

```
<?php  
$names = ["Andy", "Betty", "Carol"];  
  
$i = 0; // 初期化处理  
while ($i < 3) { // 反復条件  
    echo "Hello ";  
    echo $names[$i] . PHP_EOL;  
    $i++; // 増分処理  
}
```

## loop3.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

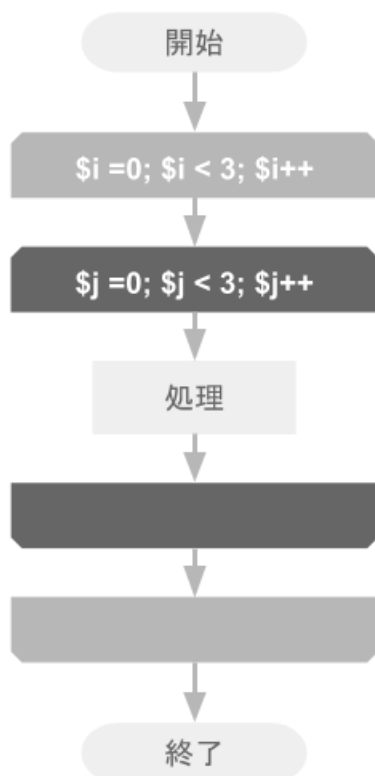
for ($i = 0; $i < 3; $i++) {
    echo "Hello ";
    echo $names[$i] . PHP_EOL;
}
```

反復条件の `$i < 3` の部分は `$i < count($names)` と実装できます。

## Terminal

```
$ php loop3.php
Hello Andy
Hello Betty
Hello Carol
```

## ネストしたループ



- 反復構造の中で反復構造を定義できる
- このような入れ子構造をネストと呼ぶ
- カウンター変数には\$i、\$j、\$kと名前をつけることが一般的

### loop4.php


```
<?php
for ($i = 0; $i < 3; $i++) {
    for ($j = 0; $j < 3; $j++) {
        echo $i + $j;
        echo " ";
    }
    echo PHP_EOL;
}
```

### Terminal


```
$ php loop4.php
0 1 2
1 2 3
2 3 4
```



## breakとcontinue



```
<?php
for ($i = 0; $i < 10; $i++) {
    if ($i == 5) {
        break;
    }
    echo $i . PHP_EOL;
}
echo "end" . PHP_EOL;
```



```
<?php
for ($i = 0; $i < 10; $i++) {
    $x = $i % 2;
    if ($x == 1) {
        continue;
    }
    echo $i . PHP_EOL;
}
echo "end" . PHP_EOL;
```

- **break**、**continue**を使うと反復処理を制御できる
- **break**を使うと反復処理を終了する
- **continue**を使うと次の反復処理（ループ）に進む

### loop5.php - break

```
<?php
for ($i = 0; $i < 10; $i++) {
    if ($i == 5) {
        break;
    }
    echo $i . PHP_EOL;
}
echo "end" . PHP_EOL;
```

### Terminal

```
$ php loop5.php
0
1
2
3
4
end
```

## loop6.php - continue

```
<?php
for ($i = 0; $i < 10; $i++) {
    $x = $i % 2;
    if ($x == 1) {
        continue;
    }
    echo $i . PHP_EOL;
}
echo "end" . PHP_EOL;
```

## Terminal

```
$ php loop6.php
0
2
4
6
8
end
```

# トレーニング

---

## loop\_tr1.php

次のプログラムがあります。

```
<?php
$i = 1;
while ($i <= 9) {
    # TODO
}
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php loop_tr1.php
1
3
5
7
9
```

---

## loop\_tr2.php

次のプログラムがあります。

```
<?php
$colors = ["Red", "Green", "Blue"];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php loop_tr2.php
1:Red
2:Green
3:Blue
```

## loop\_tr3.php

次のプログラムがあります。

```
<?php
$points = [10, 20, 30];
$total = 0;
```

次の実行結果となるようにプログラムを作成してください。

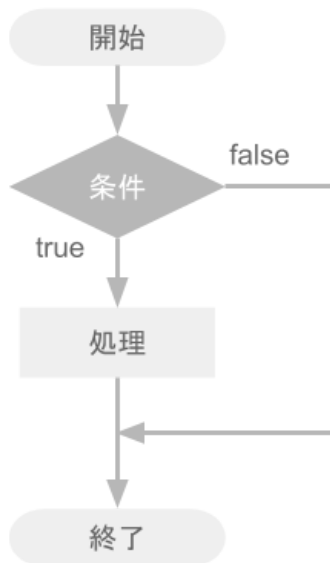
### 実行結果

```
$ php loop_tr3.php
60
```

---

# 分岐構造

## if文



- `if`文を使うと分岐構造を定義できる
- `if`文の条件`()`が成立する場合、直後の処理ブロック`{}`が実行される
- `if`文には`else`や`else if`を組み合わせることもできる

## 構文

```
if (条件) {  
    // 処理  
}
```

if文の構文はwhile文と同じです。if文は条件が成立したときに一度だけ処理を実行します。

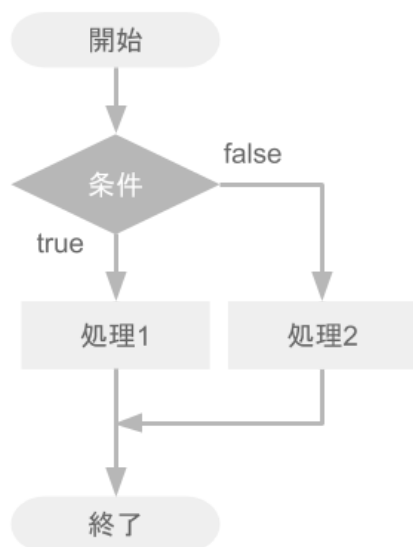
## choice1.php

```
<?php  
$dice = 6;  
  
if ($dice >= 4) {  
    echo "Win";  
}
```

## Terminal

```
$ php choice1.php  
Win
```

## if - else



- `if`文には`else`を組み合わせることができる
- `if`文の条件`()`が成立する場合、直後の処理ブロック`{}`が実行される
- `if`文の条件`()`が成立しない場合、`else`の処理ブロック`{}`が実行される

### 構文

```
if (条件) {  
    // 処理1  
} else {  
    // 処理2  
}
```

### choice2.php

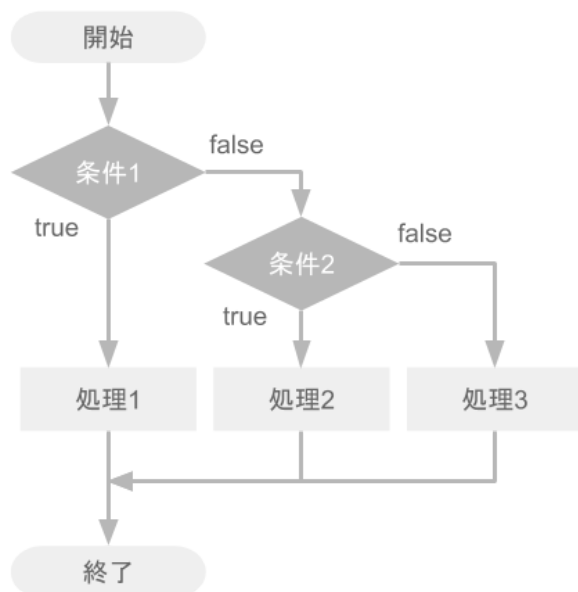
```
<?php  
$dice = 3;  
  
if ($dice >= 4) {  
    echo "Yes";  
} else {  
    echo "No";  
}
```

**Terminal**

```
$ php choice2.php  
No
```



## if - else if



- `if`文には`else if`を組み合わせることができる
- `if`文の条件()が成立しない場合、`else if`の条件()が評価される
- `else if`は複数記述できる

### 構文

```
if (条件1) {  
    // 処理1  
} else if (条件2) {  
    // 処理2  
} else {  
    // 処理3  
}
```

### choice3.php

```
<?php  
$dice = 4;  
  
if ($dice >= 5) {  
    echo "Win";  
} else if ($dice >= 3) {  
    echo "Draw";  
} else {  
    echo "Lose"  
}
```

## Terminal

```
$ php choice3.php  
Draw
```

PHPの`else if`は`elseif`のように半角スペースを付けて記述することもできます。

## 論理演算子

- 論理演算子によって複数の条件を組み合わせることができる
- `&&`は論理積（かつ、AND）、`||`は論理和（あるいは、OR）を意味する
- `!`は否定を意味し、評価結果が反転する

### choice4.php - 論理積

```
<?php
$diceA = 4;
$diceB = 3;

if ($diceA >= 4 && $diceB >= 4) {
    echo "Win";
} else {
    echo "Lose"
}
```

#### Terminal

```
$ php choice4.php
Lose
```

### choice5.php - 論理和

```
<?php
$diceA = 4;
$diceB = 3;

if ($diceA >= 4 || $diceB >= 4) {
    echo "Win";
} else {
    echo "Lose"
}
```

#### Terminal

```
$ php choice5.php
Win
```

## choice6.php - 否定

```
<?php
$dice = 4;

if (!($dice > 3)) {
    echo "Win";
} else {
    echo "Lose"
}
```

この場合if (\$dice <= 3)と記述するのと同じ結果になります。

## Terminal

```
$ php choice6.php
Lose
```

## 異なるデータ型の比較

- `==`演算子は型の相互変換が発生する
- `===`演算子は型の相互変換が発生しない
- `(int)`のように記述することでデータ型を変換できる（キャスト）

### choice7.php - `==`による比較

```
<?php
$diceA = 6;
$diceB = "6";

if ($diceA == $diceB) {
    echo "Win";
} else {
    echo "Lose";
}
```

#### Terminal

```
$ php choice7.php
Win
```

### choice8.php - `===`による比較

```
<?php
$diceA = 6;
$diceB = "6";

if ($diceA === $diceB) {
    echo "Win";
} else {
    echo "Lose";
}
```

#### Terminal

```
$ php choice8.php
Lose
```

## choice9.php - 型の変換 (キャスト)

```
<?php
$diceA = 6;
$diceB = "6";

$diceC = (int)$diceB;

if ($diceA === $diceC) {
    echo "Win";
} else {
    echo "Lose";
}
```

### Terminal

```
$ php choice9.php
Win
```

## if文の応用

- 反復構造と分岐構造を組み合わせることができる
- 配列の中の特定の要素だけを処理できる
- 制御構造を組み合わせるとインデントが深くなる

### choice10.php

```
<?php
$names = ["Andy", "Betty", "Carol"];

for ($i = 0; $i < 3; $i++) {
    if ($names[$i] != "Andy") {
        echo "Hello ";
        echo $names[$i] . PHP_EOL;
    }
}
```

### Terminal

```
$ php choice10.php
Hello Betty
Hello Carol
```

# トレーニング

---

## choice\_tr1.php

次のプログラムがあります。

```
<?php
$score = $argv[1];
```

`$argv`はコマンドライン引数などと呼ばれる特別な変数です。`php`コマンドの実行時にデータを設定できます。また`$argv[0]`にはファイル名が代入されています。

次の実行結果となるようにプログラムを作成してください。

### 実行結果

- `$argv[1]`が80以上の場合

```
$ php choice_tr1.php 80
80 : A
```

- `$argv[1]`が60以上の場合（かつ、79以下の場合）

```
$ php choice_tr1.php 60
60 : B
```

- `$argv[1]`が上記以外の場合（59以下の場合）

```
$ php choice_tr1.php 59
59 : C
```



## choice\_tr2.php

次のプログラムがあります。

```
<?php  
$scores = [90, 72, 58, 80];
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php choice_tr2.php  
90 : A  
72 : B  
58 : C  
80 : A
```

`$scores`の要素が80以上の場合A、60以上の場合B、その他の場合Cと出力します。

---

## choice\_tr3.php

次のプログラムがあります。

```
<?php
$id = $argv[1];
$password = $argv[2];
```

コマンドライン引数を2つ受け取ります。

次の実行結果となるようにプログラムを作成してください。

### 実行結果

- `$argv[1]`がAndyかつ`$argv[2]`がsecretの場合

```
$ php choice_tr3.php Andy secret
OK
```

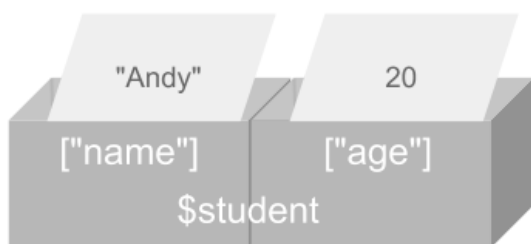
- 上記以外の場合

```
$ php choice_tr3.php Betty pass
NG
```

---

# 連想配列

---



- 連想配列は配列の一種
- 要素番号の代わりにキーを使って値を管理する
- キーと値の関連付けはダブルアロー演算子 `=>` を使う

## 構文

```
$連想配列 = [キー => 値, キー => 値];
```

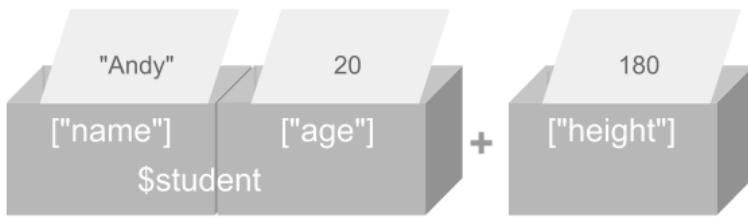
## a\_array1.php

```
<?php
$student = ["name" => "Andy", "age" => 20];
echo $student["name"] . PHP_EOL;
echo $student["age"] . PHP_EOL;
```

## Terminal

```
$ php a_array1.php
Andy
20
```

## 要素の追加



- 連想配列に要素を追加できる
- 追加時には新たなキーを指定して値を代入する
- count関数でキーと値の組み合わせの数を取得できる

### a\_array2.php

```
<?php
$student = ["name" => "Andy", "age" => 20];
$student["height"] = 180;

echo $student["height"] . PHP_EOL;
```

#### Terminal

```
$ php a_array2.php
180
```

### a\_array2.php - 修正

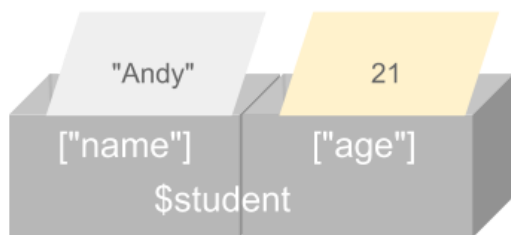
```
<?php
$student = ["name" => "Andy", "age" => 20];
$student["height"] = 180;

echo count($student);
```

#### Terminal

```
$ php a_array2.php
3
```

## 要素の変更



- 連想配列の要素を変更できる
- 変更時には既存のキーを指定して値を代入する
- 存在しないキーを指定すると追加扱いとなる

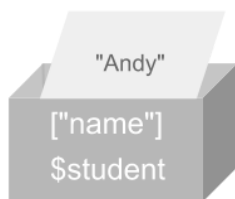
### a\_array3.php

```
<?php
$student = ["name" => "Andy", "age" => 20];
$student["age"] = 21;
echo $student["age"];
```

### Terminal

```
$ php a_array3.php
21
```

## 要素の削除



- 連想配列の要素を削除できる
- 配列の要素を削除するには関数を使う
- 連想配列の要素（キーと値）の削除にはunset関数を使う

### a\_array4.php

```
<?php
$student = ["name" => "Andy", "age" => 20];
unset($student["age"]);

var_dump($student);
```

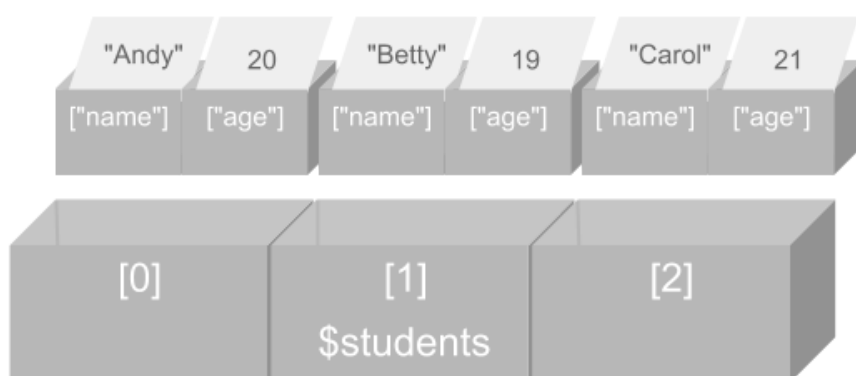
### Terminal

```
$ php a_array4.php
array(1) {
    ["name"]=>
        string(4) "Andy"
}
```

## 連想配列の応用

name	age
Andy	20
Betty	19
Carol	21

- 添字配列と連想配列を組み合わせることができる
- 添字配列と連想配列を組み合わせると表形式のデータを表現できる
- 要素番号とキーの組み合わせで必要なデータにアクセスする



### a\_array5.php

```
<?php
$students = [
    ["name" => "Andy", "age" => 20],
    ["name" => "Betty", "age" => 19],
    ["name" => "Carol", "age" => 21]
];

for ($i = 0; $i < 3; $i++) {
    if ($students[$i]["age"] >= 20) {
        echo "Hello ";
        echo $students[$i]["name"] . PHP_EOL;
    }
}
```

## Terminal

```
$ php a_array5.php  
Hello Andy  
Hello Carol
```



# トレーニング

---

## a\_array\_tr1.php

次のプログラムがあります。

```
<?php
$score = ["english" => 90, "math" => 88, "science" => 80];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php a_array_tr1.php
258
```

---

## a\_array\_tr2.php

次のプログラムがあります。

```
<?php
$scores = [
    ["english" => 90, "math" => 88, "science" => 80],
    ["english" => 64, "math" => 72, "science" => 72],
    ["english" => 90, "math" => 92, "science" => 94]
];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php a_array_tr2.php
252
```

`math`キーの値の合計を求めます。

---

## a\_array\_tr3.php

次のプログラムがあります。

```
<?php
$countries = [
    "Japan" => ["Tokyo", "Osaka", "Nagoya"],
    "England" => ["London", "Birmingham", "Glasgow"],
    "France" => ["Paris", "Marseille", "Lyon"]
];
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

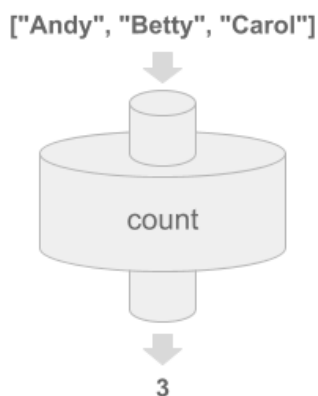
```
$ php a_array_tr3.php
London
Birmingham
Glasgow
```

**England**キーの値を出力します。

---

# 関数

---



- 関数とは予め定義された処理のこと
- 関数はいつでも呼び出すことができる
- 組み込み関数とユーザ定義関数の2種類がある

## function1.php

```
<?php
$names = ["Andy", "Betty", "Carol"];
$count = count($names);

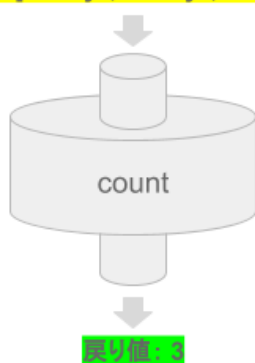
echo $count;
```

## Terminal

```
$ php function1.php
3
```

## 引数と戻り値

引数: ["Andy", "Betty", "Carol"]



`$count = count($names);`

↑  
戻り値

↑  
引数

- 関数呼び出し時に渡すデータを引数と呼ぶ
- 関数から返却されるデータを戻り値と呼ぶ
- PHPマニュアルを読むと引数と戻り値を確認できる
  - <https://www.php.net/manual/ja/>

### 構文 - 関数呼び出し

```
$変数 = 関数名(引数, 引数)
```

### function1.php

```
<?php
$names = ["Andy", "Betty", "Carol"];
$count = count($names);

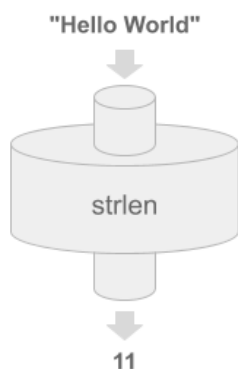
echo $count;
```

# 文字列を扱う関数

---

- strlen — 文字列の長さを得る
- strtolower — 文字列を小文字にする
- substr — 文字列の一部を返す
- explode — 文字列を文字列により分割する
- implode — 配列要素を文字列により連結する

## strlen — 文字列の長さを得る



- 引数
  - 文字列長を調べる対象となる文字列
- 戻り値
  - 文字列の長さ（バイト数）

### string1.php

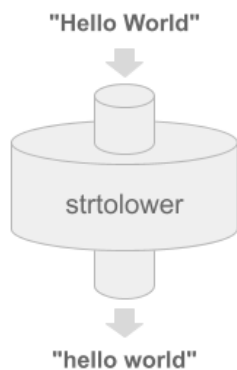
```
<?php
$str = "Hello World";
$length = strlen($str);
echo $length . PHP_EOL;
```

### Terminal

```
$ php string1.php
11
```

スペースやタブ、改行コードなども文字数としてカウントされるので注意してください。

## strtolower — 文字列を小文字にする



- 引数
  - 変換対象となる文字列
- 戻り値
  - 小文字に変換した文字列

### string2.php

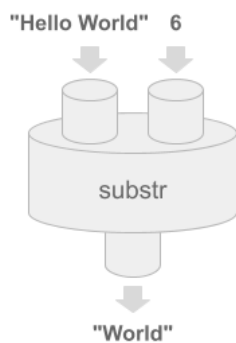
```
<?php
$str = "Hello World";
$lower_case = strtolower($str);
echo $lower_case . PHP_EOL;
```

### Terminal

```
$ php string2.php
hello world
```

大文字に変換するstrtoupper関数も用意されています。

## substr — 文字列の一部を返す



- 引数
  1. 部分文字列の取得対象となる文字列
  2. 開始位置
  3. 開始位置からの文字数（省略可能）
- 戻り値
  - 文字列の一部

### string3.php

```
<?php
$str = "Hello World";
$sub = substr($str, 6);
echo $sub . PHP_EOL;
```

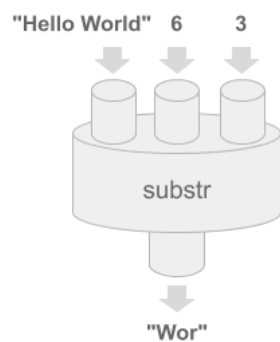
### Terminal

```
$ php string3.php
World
```

文字列も配列と同様に先頭は0からカウントします。



## string3.php - 修正

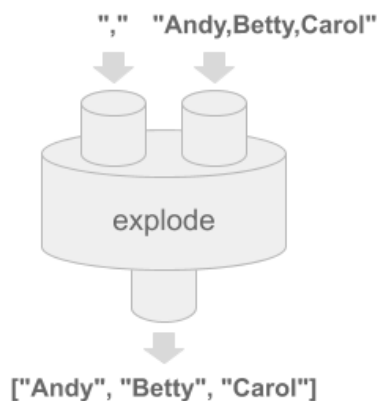


```
<?php
$str = "Hello World";
$sub = substr($str, 6, 3);
echo $sub . PHP_EOL;
```

## Terminal

```
$ php string3.php
Wor
```

## explode — 文字列を文字列により分割する



- 引数
  1. 区切り文字
  2. 変換対象となる文字列
- 戻り値
  - 区切り文字で分割された配列

### string4.php

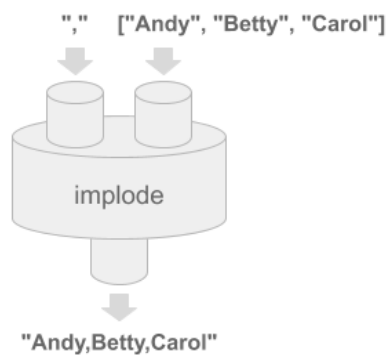
```
<?php
$str = "Andy,Betty,Carol";
$array = explode(",", $str);
print_r($array);
```

`$array`には配列が代入されます。配列はechoで出力できないため、`print_r`関数で出力しています。

### Terminal

```
$ php string4.php
Array
(
    [0] => Andy
    [1] => Betty
    [2] => Carol
)
```

## implode — 配列要素を文字列により連結する



- 引数
  1. 区切り文字
  2. 変換対象となる配列
- 戻り値
  - 区切り文字で連結された文字列

### string5.php

```
<?php
$array = ["Andy", "Betty", "Carol"];
$str = implode(",", $array);
echo $str . PHP_EOL;
```

### Terminal

```
$ php string5.php
Andy,Betty,Carol
```

## マルチバイト文字列関数

- 日本語のようなマルチバイト文字列を扱う場合はマルチバイト用の文字列関数を使う
- PHPの実行環境において`mbstring`モジュールが有効になっている必要がある
- `mb_strlen`関数や`mb_substr`関数などが存在する

### string6.php

```
<?php
$message = "ハローワールド";
$length = mb_strlen($message);
echo $length . PHP_EOL;
```

### Terminal

```
$ php string6.php
7
```

# トレーニング

---

## string\_tr1.php

次のプログラムがあります。

```
<?php
$colors = ["Red", "Green", "Blue"];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php string_tr1.php
RED
GREEN
BLUE
```

---

## string\_tr2.php

次のプログラムがあります。

```
<?php
$colors = ["Red", "Green", "Blue"];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php string_tr2.php
RGB
```

---

## string\_tr3.php

次のプログラムがあります。

```
<?php
$str = "Hyper-Text-Markup-Language";
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php string_tr3.php
HTML
```

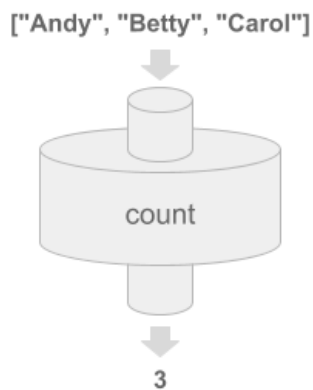
---

## 配列を扱う関数

---

- `count` — 変数に含まれるすべての要素を数える
- `array_reverse` — 要素を逆順にした配列を返す
- `array_sum` — 配列の中の値の合計を計算する
- `array_slice` — 配列の一部を展開する
- `sort` — 配列をソートする

## count — 変数に含まれるすべての要素を数える



- 引数
  - 要素数を調べる対象となる配列
- 戻り値
  - 要素数

### arr1.php

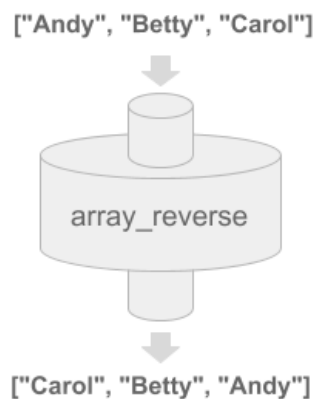
```
<?php
$array = ["Andy", "Betty", "Carol"];
$length = count($array);
echo $length . PHP_EOL;
```

### Terminal

```
$ php arr1.php
3
```



## array\_reverse — 要素を逆順にした配列を返す



- 引数
  - 逆順の対象となる配列
- 戻り値
  - 要素を逆順にした配列

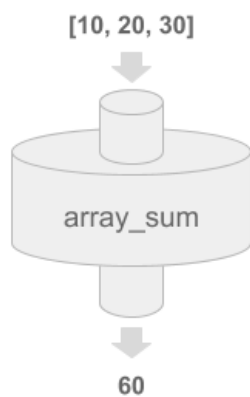
### arr2.php

```
<?php
$array = ["Andy", "Betty", "Carol"];
$reversed_array = array_reverse($array);
print_r($reversed_array);
```

### Terminal

```
$ php arr2.php
Array
(
    [0] => Carol
    [1] => Betty
    [2] => Andy
)
```

## array\_sum — 配列の中の値の合計を計算する



- 引数
  - 計算対象となる配列
- 戻り値
  - 要素の値の合計値

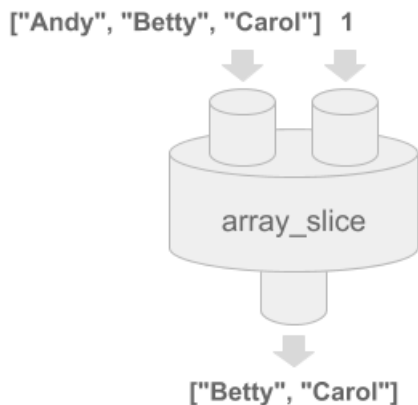
### arr3.php

```
<?php
$array = [10, 20, 30];
$sum = array_sum($array);
echo $sum . PHP_EOL;
```

### Terminal

```
$ php arr3.php
60
```

## array\_slice — 配列の一部を展開する



- 引数
  1. 一部分の取得対象となる配列
  2. 開始位置
  3. 開始位置からの要素数（省略可能）
- 戻り値
  - 切り取った配列

### arr4.php

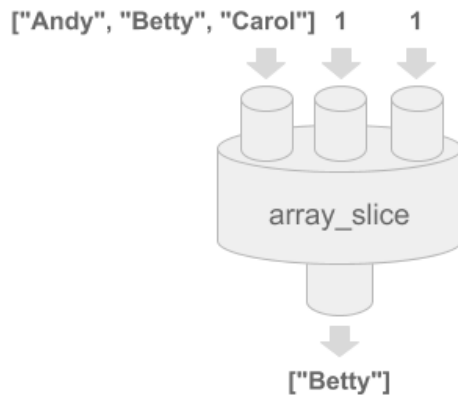
```
<?php
$array = ["Andy", "Betty", "Carol"];
$sliced_array = array_slice($array, 1);
print_r($sliced_array);
```

### Terminal

```
$ php arr4.php
Array
(
    [0] => Betty
    [1] => Carol
)
```

配列の要素番号は0からカウントします。

## arr4.php - 修正

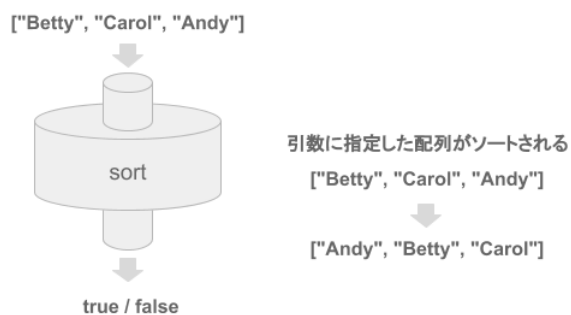


```
<?php
$array = ["Andy", "Betty", "Carol"];
$sliced_array = array_slice($array, 1, 1);
print_r($sliced_array);
```

## Terminal

```
$ php arr4.php
Array
(
    [0] => Betty
)
```

## sort — 配列をソートする



- 引数
  - ソートの対象となる配列
- 戻り値
  - 並び替えに成功した場合 `true` 失敗した場合 `false`

### arr5.php

```
<?php
$array = ["Betty", "Carol", "Andy"];
sort($array);
print_r($array);
```

ここではsort関数の戻り値は利用していません

### Terminal

```
$ php arr5.php
Array
(
    [0] => Andy
    [1] => Betty
    [2] => Carol
)
```

# トレーニング

---

## arr\_tr1.php

次のプログラムがあります。

```
<?php
$scores = [90, 72, 58, 80];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php arr_tr1.php
300
```

---

## arr\_tr2.php

次のプログラムがあります。

```
<?php
$scores = ["english" => [90, 72, 58],
           "math" => [80, 82, 78],
           "science" => [94, 66, 80]];
```

次の実行結果となるようにプログラムを作成してください。

実行結果

```
$ php arr_tr2.php
english:220
math:240
science:240
```

## arr\_tr3.php

次のプログラムがあります。

```
<?php
$names = ["Carol", "Andy", "Betty", "Daniel"];
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php arr_tr3.php
array(4) {
  [0]=>
  string(6) "Daniel"
  [1]=>
  string(5) "Carol"
  [2]=>
  string(5) "Betty"
  [3]=>
  string(4) "Andy"
}
```

変数`$names`の要素をアルファベットの降順で出力します。

---

# ファイルシステムを扱う関数

---

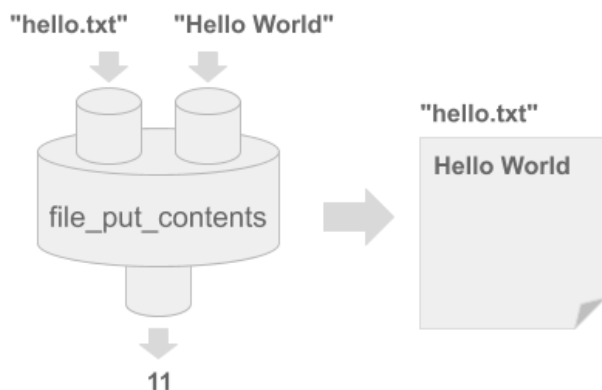
- `file_put_contents` — 文字列をファイルに書き込む
- `file_get_contents` — ファイルの内容を全て文字列に読み込む
- `file` — ファイル全体を読み込んで配列に格納する
- `mkdir` — ディレクトリを作る
- `fopen` — ファイルまたはURLをオープンする
- `fclose` — オープンされたファイルポインタをクローズする
- `fgets` — ファイルポインタから1行取得する
- `fwrite` — バイナリセーフなファイル書き込み処理

## テキストファイルについて

本講座で取り上げるファイルシステムを扱う関数のほとんどは、テキストファイルを対象に扱うものです。テキストファイルとはアルファベットや数字、日本語のようなテキストデータで構成されているファイルのことです。そのためJPEGやPNGのような画像データや音声データ、動画データのようなファイルはここで紹介する関数で扱うことはできないので注意してください。



## file\_put\_contents — 文字列をファイルに書き込む



- 引数
  1. ファイル名
  2. ファイルに書き込む文字列
  3. フラグ（省略可能）
    - FILE\_APPEND定数を指定すると追記モード
- 戻り値
  - ファイルに書き込まれたバイト数
    - ただし、書き込みに失敗した場合はfalse

### fs1.php

```
<?php
$str = "Hello World" . PHP_EOL;
$file = "hello.txt";
file_put_contents($file, $str);
```

### Terminal

```
$ php fs1.php
```

ターミナル上には何も表示されません。カレントディレクトリ上に"hello.txt"というファイルが生成されます。

### hello.txt

```
Hello World
```

## 上書きモードと追記モード

- file\_put\_contents関数はデフォルトで上書きモードで動作する
- 第3引数にFILE\_APPEND定数を指定すると追記モードで動作する

### fs1.php - 修正

```
<?php
$str = "Hello PHP" . PHP_EOL;
$file = "hello.txt";
file_put_contents($file, $str);
```

### Terminal

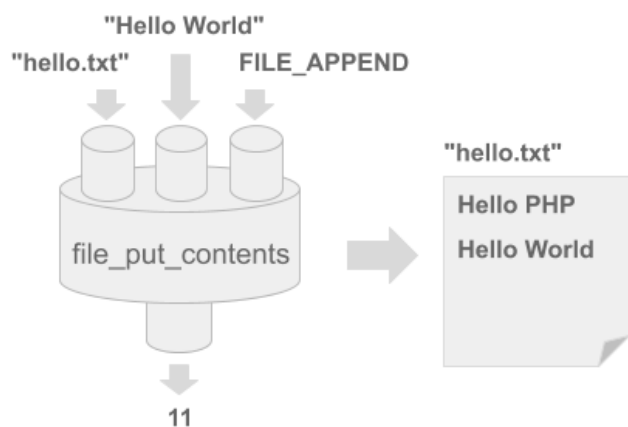
```
$ php fs1.php
```

### hello.txt

```
Hello PHP
```

hello.txtの内容は上書きされてHello PHPに置き換わっていることがわかります。

## fs1.php - 修正



```
<?php
$str = "Hello World" . PHP_EOL;
$file = "hello.txt";
file_put_contents($file, $str, FILE_APPEND);
```

## Terminal

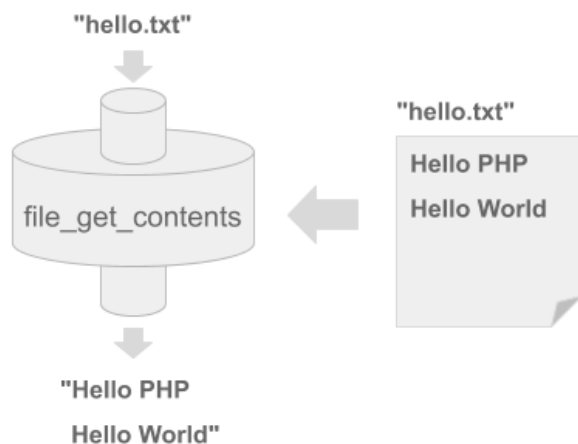
```
$ php fs1.php
```

## hello.txt

```
Hello PHP
Hello World
```

hello.txtの内容にHello Worldが追記されていることがわかります。

## file\_get\_contents — ファイルの内容を全て文字列に読み込む



- 引数
  - 読み込み対象となるファイル名
- 戻り値
  - 文字列（ファイルの内容）
    - ただし、読み込みに失敗した場合はfalse

### fs2.php

```
<?php
$file = "hello.txt";
$str = file_get_contents($file);
echo $str;
```

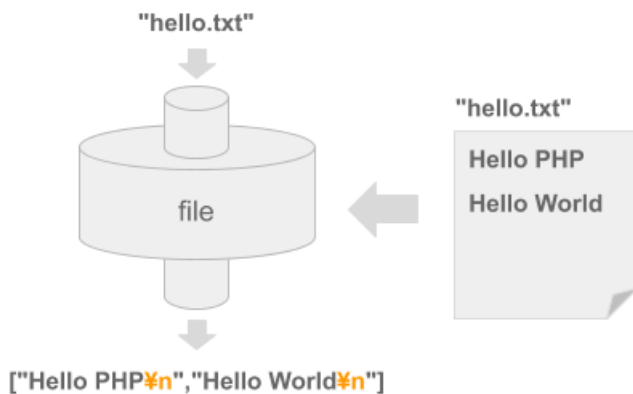
### hello.txt

```
Hello PHP
Hello World
```

### Terminal

```
$ php fs2.php
Hello PHP
Hello World
```

## file — ファイル全体を読み込んで配列に格納する



- 引数
  1. 読み込み対象となるファイル名
  2. フラグ
    - FILE\_IGNORE\_NEW\_LINES定数指定すると改行コードを除去する
- 戻り値
  - 配列（ファイルの内容を行ごとに要素に変換したもの）
    - ただし、読み込みに失敗した場合はfalse

### fs3.php

```
<?php
$file = "hello.txt";
$array = file($file);
print_r($array);
```

### Terminal

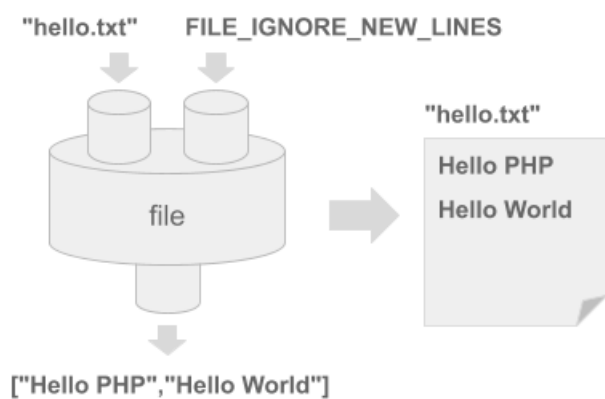
```
$ php fs3.php
Array
(
    [0] => Hello PHP

    [1] => Hello World

)
```

各要素の文字列データに改行コードが存在します。

## fs3.php - 修正

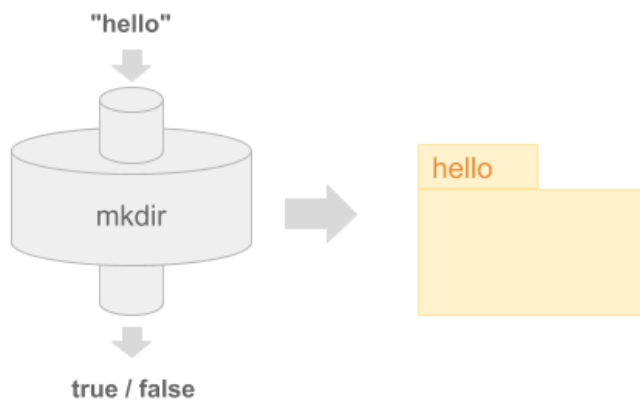


```
<?php
$file = "hello.txt";
$array = file($file, FILE_IGNORE_NEW_LINES);
print_r($array);
```

## Terminal

```
$ php fs3.php
Array
(
    [0] => Hello PHP
    [1] => Hello World
)
```

## mkdir — ディレクトリを作る



- 引数
  - ディレクトリ名
- 戻り値
  - 成功した場合はtrue、失敗した場合はfalse

### fs4.php

```
<?php
$dir = "hello";
mkdir($dir);
```

### Terminal

```
$ php fs4.php
```

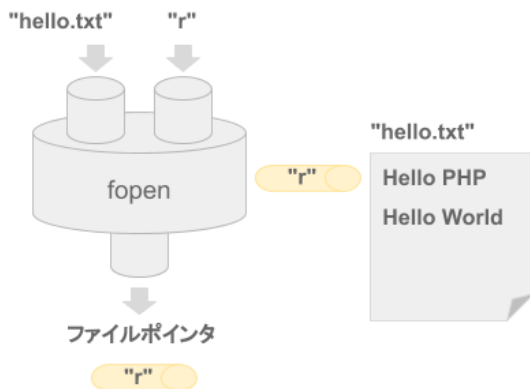
ターミナル上には何も表示されません。カレントディレクトリ上にhelloという名前のディレクトリが生成されます。

## ファイルの読み込み

- fopen関数は戻り値にファイルポインタを返す
- fgets関数はファイルポインタからデータを読み込む
- fclose関数でファイルポインタを破棄する

ファイルの入出力を細かく制御したい場合はfopen関数やfgets関数、fclose関数などを組み合わせて制御します

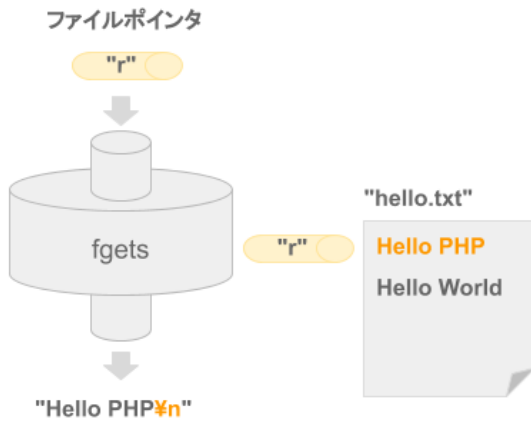
### fopen — ファイルまたはURLをオープンする



- 引数
  1. ファイル名
  2. アクセス形式
    - "r"
      - 読み込みモード
    - "w"
      - 書き込みモード（上書き）
    - "a"
      - 書き込みモード（追記）
- 戻り値
  - ファイルポインタ
    - オープンに失敗した場合はfalse

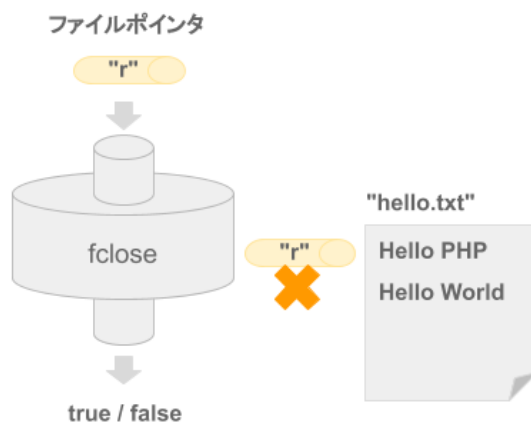


## fgets — ファイルポインタから1行取得する



- 引数
  1. ファイルポインタ
  2. 読み込みデータ長（省略可能）
    - 省略時は行末まで読み込む
- 戻り値
  - 文字列（1行のデータ）
    - 読み込むデータがない場合はfalse
    - 読み込みに失敗した場合はfalse

## fclose — オープンされたファイルポインタをクローズする



- ◦ 引数
  - ファイルポインタ
- ◦ 戻り値
  - 成功した場合はtrue、失敗した場合はfalse

**fs5.php**

```
<?php
$file = "hello.txt";
$handle = fopen($file, "r");

$line = fgets($handle);
echo $line;

fclose($handle);
```

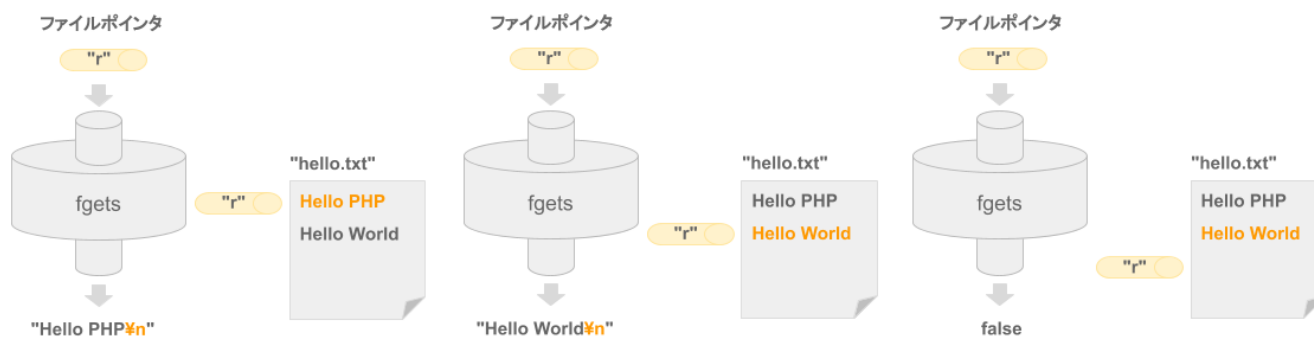
**hello.txt**

Hello PHP

**Terminal**

```
$ php fs5.php
Hello PHP
```

## fs5.php - 修正



```
<?php
$file = "hello.txt";
$handle = fopen($file, "r");
while(($line = fgetc($handle)) !== false) {
    echo $line;
}
fclose($handle);
```

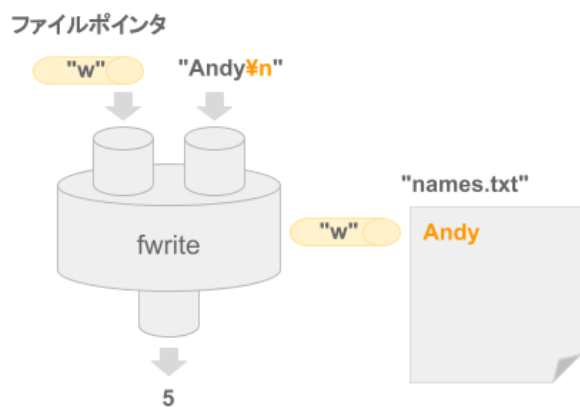
## Terminal

```
$ php fs5.php
Hello PHP
Hello World
```

## ファイルの書き込み

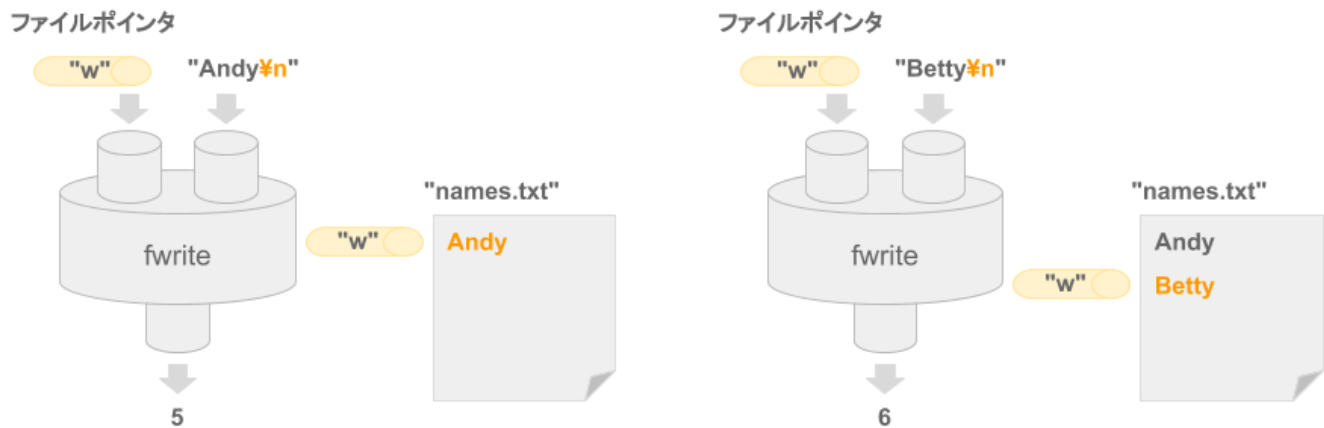
- fopen関数は戻り値にファイルポインタを返す
- fwrite関数はファイルポインタにデータを書き込む
- fclose関数でファイルポインタを破棄する

### fwrite — バイナリセーフなファイル書き込み処理



- 引数
  1. ファイルポインタ
  2. ファイルに書き込むデータ
- 戻り値
  - 書き込んだバイト数
    - 失敗した場合はfalse

## fs6.php



```
<?php
$names = ["Andy", "Betty"];
$file = "names.txt";
$handle = fopen($file, "w");
for ($i = 0; $i < count($names); $i++) {
    fwrite($handle, $names[$i] . PHP_EOL);
}
fclose($handle);
```

## Terminal

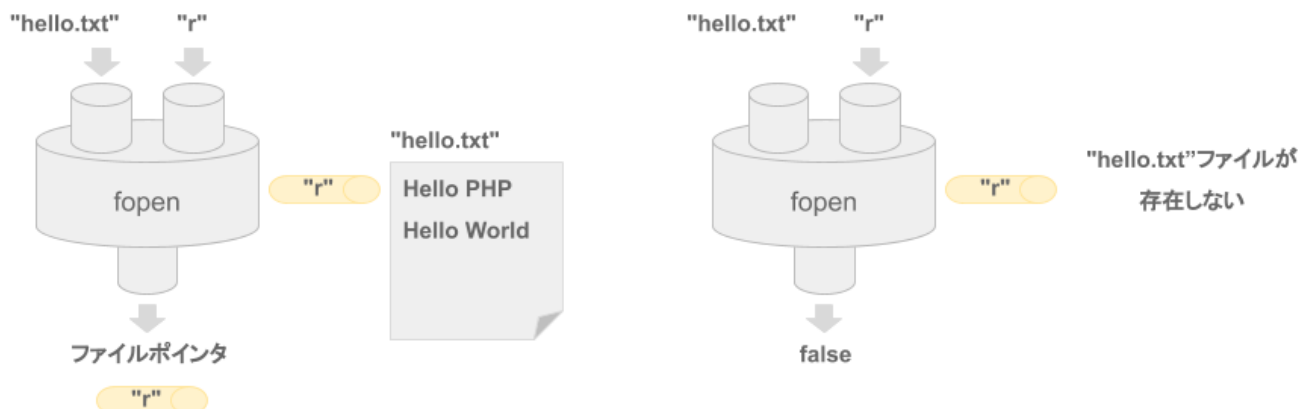
```
$ php fs6.php
```

ターミナルには何も表示されません。カレントディレクトリ上に `names.txt` というファイルが生成されます。

## names.txt

```
Andy
Betty
```

## ファイルの読み込み（書き込み）に失敗する場合



- ファイルシステム関数は、プログラムの実行環境の影響を受ける
  - 他のプログラムがファイル进行处理（ロック）している場合
  - ファイルへのアクセス権限がない場合
  - ファイル書き込み時にディスク容量が足りない場合 など
- ファイルシステムを扱う関数のほとんどは、実行時にエラーが発生した場合、戻り値に`false`を返す
- 関数呼び出し時に、エラー制御演算子`@`を付けると警告メッセージを抑制できる

### fs5.php - 修正

```
<?php
$file = "heloo.txt";
$handle = fopen($file, "r");
if ($handle === false) {
    die("can't open file");
}
while(($line = fgets($handle)) !== false) {
    echo $line;
}
fclose($handle);
```

die関数を呼び出すとその場でメッセージを出力して異常終了します。

### Terminal

```
$ php fs5.php
PHP Warning:  fopen(heloo.txt): failed to open stream: No such file or
directory in /Users/murayama/Desktop/test.php on line 3

Warning: fopen(heloo.txt): failed to open stream: No such file or
directory in /Users/murayama/Desktop/test.php on line 3
can't open file
```

`can't open file`メッセージだけでなく、Warningメッセージも出力されています。Warningメッセージを非表示にするには、エラー制御演算子`@`を使います。

### fs5.php - 修正

```
<?php
$file = "heloo.txt";
$handle = @fopen($file, "r");
if ($handle === false) {
    die("can't open file");
}
while(($line = fgets($handle)) !== false) {
    echo $line;
}
fclose($handle);
```

### Terminal

```
$ php $fs5.php
can't open file
```

# トレーニング

---

## fs\_tr1.php

次のファイル`score.txt`を準備します。

```
90  
72  
58  
80
```

次のプログラムがあります。

```
<?php  
$file = "score.txt";
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php fs_tr1.php  
300
```

---



## fs\_tr2.php

次のプログラムがあります。

```
<?php
$scores = [
    [90, 88, 80],
    [64, 72, 72],
    [90, 92, 94]
];
$file = "score.csv";
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php fs_tr2.php
```

以下の内容でscore.csvファイルが作成されること

```
90,88,80
64,72,72
90,92,94
```

---

## fs\_tr3.php

次のプログラムがあります。

```
<?php
$file = "score.csv";
```

fs\_tr2.phpで作成したscore.csvファイルを読み込みます。

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php fs_tr3.php
742
```

## ユーザ定義関数



functionキーワードによって  
関数の名前、引数、戻り値を定義する

- 自作する関数のことをユーザ定義関数と呼ぶ
- 関数を定義するにはfunctionキーワードを使う
- 関数を定義することで、再利用性、可読性、メンテナンス性を高めることができる

### 構文

```
function 関数名(引数, 引数) {  
    // 処理  
    return 戻り値;  
}
```

引数や戻り値 (return) は省略可能です。引数は, で区切って複数定義できます。

### function2.php

```
<?php  
function calc($price) {  
    $result = $price * 1.08;  
    return $result;  
}  
  
$price1 = calc(100);  
echo $price1 . PHP_EOL;  
  
$price2 = calc(1000);  
echo $price2 . PHP_EOL;
```

### Terminal

```
$ php function2.php  
108  
1080
```

## 引数の定義



- 引数は複数定義できる
- 引数を取らない関数も定義できる
- 関数呼び出し時に引数を正しく指定する必要がある

### function3.php

```
<?php
function sum($x, $y, $z) {
    return $x + $y + $z;
}

$result = sum(100, 200, 300);
echo $result . PHP_EOL;
```

#### Terminal

```
$ php function3.php
600
```

### function4.php

```
<?php
function hello() {
    return "Hello";
}

$result = hello();
echo $result . PHP_EOL;
```

#### Terminal

```
$ php function4.php
Hello
```

## 戻り値の定義



- 戻り値はreturnキーワードで指定する
- returnキーワードを使わない場合、戻り値のない関数となる
- returnキーワードに到達すると関数は終了する

### function5.php

```
<?php
function greet($name) {
    return "Hello $name" . PHP_EOL;
}

$message = greet("Andy");
echo $message;
```

### Terminal

```
$ php function5.php
Hello Andy
```

### function6.php

```
<?php
function greet($name) {
    echo "Hello $name" . PHP_EOL;
}

greet("Andy");
```

### Terminal

```
$ php function6.php
Hello Andy
```

**function7.php**

```
<?php
function greet() {
    echo "Hello" . PHP_EOL;
}

greet();
```

**Terminal**

```
$ php function7.php
Hello
```

# トレーニング

---

## function\_tr1.php

次のプログラムがあります。

```
<?php
function double($x)
{
    return $x * 2;
}

$scores = [90, 72, 58];
# TODO double関数を呼び出して実行結果のとおり出力します。
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php function_tr1.php
180
144
116
```

## function\_tr2.php

次のプログラムがあります。

```
<?php
# TODO repeat関数を定義します。

$hello2 = repeat("Hello", 2);
echo $hello2 . PHP_EOL;

$world3 = repeat("World", 3);
echo $world3 . PHP_EOL;
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php function_tr2.php
HelloHello
WorldWorldWorld
```

---

## function\_tr3.php

次のプログラムがあります。

```
<?php
# TODO search関数を定義します。

$scores = [90, 72, 58, 80, 66, 50];
$target = $argv[1];

$found = search($scores, $target);
if ($found === true) {
    echo "Found" . PHP_EOL;
} else {
    echo "Not Found" . PHP_EOL;
}
```

次の実行結果となるようにプログラムを作成してください。

### 実行結果

```
$ php function_tr3.php 66
Found
```

コマンドライン引数に66を指定します。

```
$ php function_tr3.php 65
Not Found
```

コマンドライン引数に65を指定します。

---



