সূচিপত্র

কোর্স এর মুল পাতা | HowToCode মুল সাইট | সবার জন্য প্রোগ্রামিং ব্লগ | পিডিএফ ডাউনলোড

বাংলায় পিএইচপি

প্রারম্ভিকা

বর্তমান যুগে ওয়েব অটোমেশনের জন্য একটি বিশ্বস্ত নাম - পিএইচপি । পিএইচপি একটি জনপ্রিয় প্রোগ্রামিং ল্যাঙ্গুয়েজ যার সূচনাই হয়েছিলো ওয়েব ডেভেলপমেন্ট এর জন্য । দিনে দিনে পিএইচপির জনপ্রিয়তা বাড়তে বাড়তে ওয়েব আজ তার আধিপত্য অদ্বিতীয় । বিশ্বের প্রায় ৮২% (রেফ: ১) ওয়েবসাইটই কোন না কোনভাবে পিএইচপির উপর নির্ভরশীল । ফেইসবুকের একটা বিরাট অংশ ডেভেলপ করা পিএইচপিতে । তারা পিএইচপির উপর এতটাই নির্ভরশীল যে ফেইসবুক নিজেরাই পিএইচপির উন্নয়নের জন্য নতুন পিএইচপি ইনজিন (HHVM) রিলিজ করে । উইকিপিডিয়াও কিন্তু ডেভেলপ করা পিএইচপিতে । পিএইচপির প্রবল জনপ্রিয়তা আর চাহিদার কথা চিন্তা করে গুগল সম্প্রতি তাদের এ্যাপ ইনজিন প্ল্যাটফর্মে পিএইচপি সাপোর্ট যোগ করে । আসলে যেখানে ওয়েবের ৮২%-ই পিএইচপি ব্যবহার করে সেখানে উদাহরণ খুজঁতে গেলে হাজার হাজার নমুনা পাওয়া যাবে । ওয়েব নির্ভর প্রজেক্টগুলোতে তাই পিএইচপি ডেভেলপারদের চাহিদাও ব্যাপক ।

রেফারেন্স: (১) http://php.net/usage.php

ওপেন সোর্স

বইটির কন্টেন্ট ওপেন সোর্স । আপনিও অবদান রাখতে পারেন গিটহাব রিপোজিটরিতে ।

ইনস্টলেশন

আর দশটা প্রোগ্রামিং ল্যাঙ্গুয়েজের মত পিএইচপিও কমান্ড লাইন থেকে চালানো সম্ভব । কিন্তু পিএইচপির জন্ম হয়েছিলো ওয়েব অটোমেশনের জন্য, পিএইচপির ব্যবহারও তাই মূলত সার্ভার কেন্দ্রিক । পিএইচপি ভালো করে শিখতে হলে সার্ভার এনভায়রনমেন্ট সম্পর্কে ভালো ধারণা থাকাটা জরুরী । নবীনদের জন্য পিএইচপির সাথে এ্যাপাচি হবে সার্ভার হিসেবে ভালো চয়েস । একই সাথে আমাদের আরও শিখে রাখা দরকার একটি ডাটাবেইজ সিস্টেম । অন্যান্য অধিকাংশ ডাটাবেইজ সিস্টেমের জন্য সাপোর্ট থাকলেও, পিএইচপির সাথে মাইসিকুয়েল (MySQL) এর প্রবল জনপ্রিয়তা চোখে পড়ার মত । তাই আমাদের ইন্সটলেশন সেকশনে আমরা দেখবো কিভাবে পিএইচপি, এ্যাপাচি এবং মাইসিকুয়েল ইন্সটল করা যায় ।

উইন্ডোজ

আমি ব্যক্তিগতভাবে বিশ্বাস করি মাইক্রোসফটের নিজেদের ডেভেলপমেন্ট প্ল্যাটফর্ম ডট নেট ছাড়া বাকি যে কোন ধরনের প্রোগ্রামিং এর জন্যই উইন্ডোজ একটি বাজে চয়েস । লিনাক্স বা ইউনিক্স এনভায়রনমেন্ট পিএইচপি ডেভেলপমেন্ট এর জন্য উৎকৃষ্ট । ইন ফ্যাক্ট, আপনি উইন্ডোজে পিএইচপির সব ফিচার পাবেন ও না । তাই আমি রিকমেন্ড করি পিএইচপির জন্য উবুন্টু বা ম্যাক ওস এক্স ব্যবহার করার জন্য । সেটা সম্ভব না হলে ভার্চুয়াল বক্সে লিনাক্স চালানোর জন্য । একেবারে নিরুপায় হলেই উইন্ডোজে পিএইচপি চালানো উচিৎ । উইন্ডোজে হয়তো আপনার কাজ চলে যাবে কিন্তু পিএইচপি ট্র্যাকে ক্যারিয়ারে উপরে উঠতে গেলে আপনাকে আজ হোক কাল হোক উইন্ডোজ ছাড়তেই হবে । বলাই বাহুল্য, আপনার ডেভেলপ করা এ্যাপ্লিকেশন ৯৯% ক্ষেত্রেই লিনাক্স হোস্টিং এ চলবে । সুতরাং, এগিয়ে থাকতে চাইলে এখনই সময় পিএইচপির জন্য অন্য কোন অপারেটিং সিস্টেম ব্যবহার করা ।

উইন্ডোজে পিএইচপি, এ্যাপাচি এবং মাইসিকুয়েল সেটাপ করার জন্য XAMPP জনপ্রিয় । এটি ডাউনলোড করে ইন্সটল করে নিলেই পেয়ে যাবেন আপনার ডেভেলপমেন্ট এনভায়রনমেন্ট ।

লিনাক্স

পিএইচপি ডেভেলপমেন্টের আসল মজাটা পাওয়া যায় লিনাক্সে । পিএইচপি, এ্যাপাচি এবং মাইসিকুয়েল সেটাপ করার জন্য আপনার লিনাক্স ডিস্ট্রোর প্যাকেজ ম্যানেজার ব্যবহার করুন । যারা উবুন্টু কিংবা উবুন্টু এর কোন ভ্যারিয়ান্ট ব্যবহার করছেন, তাদের জন্য এই ছোট্ট কমান্ডটি টার্মিনালে টাইপ করে এন্টার চাপলেই হবে:

```
sudo apt-get install lamp-server^
```

লিনাক্সের জন্যও XAMPP এর একটি ভার্সন আছে । কিন্তু সবকিছু রিপোজিটরি থেকে সেটাপ করে নেওয়াটাই বুদ্ধিমানের কাজ । এত ঝামেলা কম হয়, পরে প্রয়োজন হলে ট্রাবলশুটিং এও সমস্যা কম হবে । এছাড়া পিএইচপি রিলেটেড প্রচুর প্যাকেজ পাওয়া যাবে উবুন্টু সফটওয়্যার রিপোজিটরি থেকে যেগুলোও খুব সহজে ইনস্টল করে নিতে পারব

উবুন্টু বা ডেবিয়ান লিনাক্স বেইজড অপারেটিং সিস্টেমে লিনাক্স ইন্সটলেশনের জন্যে টার্মিনাল ওপেন করে নিচের কমান্ডগুলো একে একে একজিকিউট করতে হবে।

-Apache2 সার্ভার ইন্সটলেশন: sudo apt-get install apache2

-MySQL (ডাটাবেইজ) সার্ভার ইন্সটলেশন: sudo apt-get install mysql-server

-PhpMyAdmin (ডাটাবেইজ ভিজুয়াল এডিটর) ইন্সটলেশন: sudo apt-get install phpmyadmin

ম্যাক ওএস এক্স

ওসএক্স এ বাই ডিফল্ট এ্যাপাচি থাকে । পিএইচপি এবং মাইসিকুয়েল টা হোমব্রু এর মাধ্যমে ইনস্টল করে নেওয়া ভালো । বিস্তারিত ইন্সট্রাকশন পাওয়া যাবে এখানে - https://github.com/Homebrew/homebrew-php ।

এছাড়াও উইন্ডোজের মতই সবকিছু একই প্যাকেজে পাওয়া যাবে MAMP এর মাধ্যমে । তবে হোমব্রু এর মাধ্যমে সেটাপ করাটাই আমি রিকমেন্ড করি । প্রথমে একটু ঝামেলা মনে হলেও পরবর্তীতে এক্সটেনশন ইন্সটল করা কিংবা কমান্ড লাইন থেকে পিএইচপি রান করার জন্য হোমব্রু পিএইচপিই বেটার অপশন । সেই তুলনায় MAMP ব্যবহার করা সহজ কিন্তু কাস্টোমাইজেশন এর জন্য বেশ রেস্ট্রিক্টিভ ।

ল্যাঙ্গুয়েজ ব্যাসিকস

পিএইচপি কিভাবে কাজ করে?

পিএইচপি ইনজিন পিএইচপি ফাইল পড়ে লাইন বাই লাইন কোড এক্সিকিউট করে এবং আউটপুট দেয় । পিএইচপি ট্যাগের মধ্যে থাকা সব কোডই এই ইনজিনটি প্রসেস করে । প্রত্যেকটি ইনস্ট্রাকশন এর পর সেমিকোলন ব্যবহার করা হয় । সেমিকোলন দেখে পিএইচপি বুঝে নেয় তার বর্তমান কাজটি কোথায় এসে শেষ হবে ।

পিএইচপি ট্যাগ

যে কোন পিএইচপি ফাইল শুরুই করতে হয় বিশেষ একটি ট্যাগ ব্যবহার করে । এই ট্যাগটি দেখেই পিএইচপি ইনজিন বুঝতে পারে যে এই জায়গা থেকে পিএইচপি কোড শুরু । তখন সে ঐ ট্যাগের মধ্যবর্তী অংশ লাইন বাই লাইন এক্সিকিউট করে । আবার ট্যাগ শেষ হয়ে গেলে সে ধরে নেয় তার কাজ শেষ, আবার ট্যাগ না আসা পর্যন্ত ফাইলে যা আছে তা ঠিক তেমনভাবেই আউটপুট দিয়ে দেয় ।

পিএইচপি ট্যাগ শুরু করতে হয় `<?php` লিখে আর শেষ হয় `?>` লিখে । ফাইলটিতে যদি শুধু পিএইচপি কোডই থাকে তবে ট্যাগ ক্লোজ না করলেও সমস্যা হয় না । তবে যদি আমরা পিএইচপি দিয়ে ডাইনামিক প্রসেসিং এর পাশাপাশি কিছু স্ট্যাটিক অংশও রাখতে চাই সেক্ষেত্রে আমরা পিএইচপি ট্যাগ ওপেন এবং ক্লোজ করে নির্ধারন করে দিতে পারি কোন অংশটুকু ডাইনামিকালি পিএইচপি জেনারেট করবে আর কোন অংশ সবসময়ই একই থাকবে ।

যেমন:

```
<p>This is going to be ignored by PHP and displayed by the browser.</p>
<?php echo 'While this is going to be parsed.'; ?>
<p>This will also be ignored by PHP and displayed by the browser.</p>
```

এখানে আমরা প্রথমে এইচটিএমএল আউটপুট করলাম, এরপর একটা পিএইচপি ব্লক তারপর আবার আগের মতই এইচটিএমএল ।

কমেন্টস

কমেন্টস হলো কোড এর সেই অংশ যেটা পিএইচপি ইনজিন ইগনোর করে যায় । কমেন্টস এ কোন ইনস্ট্রাকশন থাকে না । মূলত কোড এ কমেন্ট করা হয় ছোট ছোট নোট আকারে । এই নোটগুলো কোড এর বিভিন্ন বিষয় ব্যখ্যা করে । যেমন একটি ভ্যারিয়েবল কি কাজ করে এটা আমরা ঐ ভ্যারিয়েবল এর পাশে কমেন্ট আকারে লিখে দিতে পারি ।

পিএইচপিতে ৩ ধরনের কমেন্টস করা সম্ভব:

```php
<?php

$name = "masnun"; // সিঙ্গল লাইন কমেন্ট - দুইটা স্ল্যাশ ব্যবহার করা হয়

/* মাল্টি লাইন কমেন্ট -
স্ল্যাশ এর পরে স্টার বা এ্যাস্টেরিস্ক দিয়ে শুরু হয় আবার একইভাবে শেষ হয় - শুধু স্ল্যাশ আর স্টারের সিরিয়ান টা

$age = 75; # একটা হ্যাশ ব্যবহার করে শেল স্টাইলে সিঙ্গল লাইন কমেন্ট করা যায়
```

সিঙ্গল লাইন কমেন্ট এর শুরু থেকে লাইনের শেষ পর্যন্ত কমেন্ট হিসেবে বিবেচিত হয় । মাল্টিলাইন কমেন্ট সাধারনত একাধিক লাইন জুড়ে হয় । তবে কমেন্টের শুরু (`/*`) আর শেষ (`*/`) এক লাইনেও হওয়া সম্ভব ।

ভ্যারিয়েবল ও ডাটা টাইপস

ভ্যারিয়েবল

প্রোগ্রামিং করতে গেলে প্রায়শই আমাদের বিভিন্ন ধরনের তথ্য উপাত্ত সংরক্ষণ করা লাগে । এই তথ্যগুলো আমরা কম্পিউটারের মেমোরীতে সংরক্ষন করে থাকি । ভ্যারিয়েবল হলো কম্পিউটার এর মেমোরীতে থাকা ছোট ছোট ব্লক যেখানে আমরা আমাদের প্রয়োজনমত ডাটা রাখতে পারি । এই মেমোরী ব্লকগুলোতে সংরক্ষিত ডাটা পরে এ্যাক্সেস করার জন্য আমরা আমাদের সুবিধামত নাম দিয়ে দেই । পিএইচপিতে ভ্যারিয়েবল তৈরি করা খুবই সহজ । সব ভ্যারিয়েবলই শুরু হবে ডলার সাইন (`$`) দিয়ে, ডলার সাইনের পরপরই ভ্যারিয়েবল এর নাম । এরপর ইকুয়াল সাইন (`=`) এর পর ঐ ভ্যারিয়েবল এর ভ্যালু ।

যেমন:

```
<?php
$name = "Abu Ashraf Masnun";
```

এখানে আমরা একটি ভ্যারিয়েবল তৈরি করলাম `$name` । ভ্যারিয়েবল এর নাম অবশ্যই আন্ডারস্কোর অথবা কোন এ্যালফাবেট দিয়ে শুরু হতে হবে । নামের শুরুতেই সংখ্যা ব্যবহার করা যাবে না । নামটি কেইস সেনসিটিভ । অর্থাৎ `$name` আর `$Name` সম্পূর্ণ আলাদা ভ্যারিয়েবল নাম ।

ডাটা টাইপ

আমাদের দৈনন্দিন জীবনে ব্যবহৃত ডাটা নানা ধরনের হয়ে থাকে । কোনটা টেক্সট, কোনটা সংখ্যা, সংখ্যার ভিতরে আবার কোনটা পূর্ণ সংখ্যা, কোনটা ভগ্নাংশ - এই সব ডাটার একেকটা কম্পিউটার একেক ভাবে সংরক্ষণ করে । এখান থেকেই মূলত ডাটা টাইপ কনসেপ্ট এর উৎপত্তি ।

পিএইচপিতে আমরা কোন ভ্যারিয়েবল এর টাইপ জানতে `gettype()` ফাংশনটি ব্যবহার করতে পারি । যেমন:

```
<?php
$age = 23;
echo gettype($age); // integer
```

পিএইচপি তে বহুল ব্যবহৃত ডাটা টাইপ গুলো হলো:

বুলিয়ান

বুলিয়ান টাইপ ব্যবহার করে আমরা কোন কিছু সত্য না মিথ্যা তা প্রকাশ করে থাকি । আরেকটু গভীরভাবে চিন্তা করলে আমরা দেখবো যখন কোন ভ্যারিয়েবল ঠিক বিপরীতধর্মী দুইটা ভ্যালুর যে কোন একটা গ্রহন করে তখন আমরা সেটাকে সচারচর বুলিয়ান টাইপ দিয়ে প্রকাশ করি ।

পিএইচপিতে বুলিয়ান টাইপের ভ্যালু হতে পারে `TRUE` অথবা `FALSE` ।

উদাহরণ:

```
<?php

$isMarried = FALSE;
$isAlive = TRUE;
```

ইন্টিজারস

ইন্টিজার ব্যবহার করি আমরা পূর্ণ সংখ্যা প্রকাশ করার জন্য । এই পূর্ণ সংখ্যা ধনাত্বক বা ঋণাত্বক হতে পারে । যেমন: কারো বয়স ।

```
<?php

$age = 75;
```

লক্ষ্যনীয় যে ইন্টিজার ডেসিম্যাল, অক্টাল, হেক্সাডেসিম্যাল কিংবা বাইনারি ফরমাটেও প্রকাশ করা যায়:

```
<?php
$a = 1234; // ডেসিম্যাল
$a = -123; // ঋণাত্মক
$a = 0123; // অক্টাল, ডেসিম্যালে কনভার্ট করলে পাবো ৮৩
$a = 0x1A; // হেক্সাডেসিম্যাল, ডেসিম্যালে মান হবে ২৬
$a = 0b11111111; // বাইনারী, ডেসিম্যালে মান হবে ২৫৫
```

(উদাহরণটি পিএইচপি ম্যানুয়াল থেকে নেওয়া, কমেন্ট বাংলায় অনুবাদ করা)

ফ্লোটিং পয়েন্ট বা ডাবল

ভগ্নাংশ কিংবা দশমিক সংখ্যা প্রকাশ করার জন্য আমরা ফ্লোটিং পয়েন্ট টাইপ ব্যবহার করি । এটাকে ডাবল কিংবা রিয়াল নাম্বার ও বলা হয় ।

```
<?php

$temperature = 30.45;
```

স্ট্রিংস

স্ট্রিংস হলো অনেকগুলো ক্যারেক্টারের সমষ্টি । পিএইচপিতে এখনো ইউনিকোড এর সাপোর্ট আসে নি । স্ট্রিং ভ্যারিয়েবল তৈরি করতে হলে সাধারনত ডাবল কিংবা সিঙ্গল কোট ব্যবহার করা হয় ।

```
<?php
$name = "The Doctor";
$planet = 'Gallifrey';
```

এছাড়াও `heredoc` ফরম্যাটেও স্ট্রিং তৈরি করা যায়:

```
<?php

$doc = <<<DOC
Hello world!
DOC;
```

এখানে, আমরা প্রথমে `<<<` এবং পরপরই একটি আইডেন্টিফায়ার (আমাদের এক্ষেত্রে `DOC`) ব্যবহার করি । এরপর একটি লাইন ব্রেক দিয়ে আমাদের স্ট্রিং লেখা হয় ।ব্লক শেষ করার জন্য উপরোক্ত আইডেন্টিফায়ার টি নতুন লাইনে আবার লিখে সেমিকোলন দিয়ে স্টেটমেন্ট শেষ করা হয় । মনে রাখতে হবে শেষ লাইনে আইডেন্টিফায়ার এর আগে বা পরে কোন স্পেইস বা অন্য কোন ক্যারেক্টার ব্যবহার করা যাবে না ।

এ্যারে

এ্যারে হলো একটি তালিকা । যেখানে একটি ইনডেক্স এর বিপরীতে আমরা একটি ভ্যালু সংরক্ষণ করি । এ্যারে তৈরি করার জন্য আমরা বিল্ট ইন `array` কন্সট্র্যাক্ট ব্যবহার করি । ইনডেক্স এর বিপরীতে ভ্যালু ডিফাইন করার জন্য আমরা `=>` সিম্বল ব্যবহার করি ।

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);
```

এখানে `$array` একটি এ্যারে যার `foo` ইনডেক্স বা কি এর ভ্যালু `bar` এবং `bar` এর ভ্যালু `foo` । উদাহরণটি পিএইচপি ম্যানুয়াল থেকে নেওয়া ।

লক্ষ্যনীয় বিষয়: এ্যারে এর ইনডেক্স বা কি শুধুমাত্র স্ট্রিং বা ইন্টিজার হবে । তবে ইনডেক্স এর ভ্যালু যে কোন টাইপের হতে পারে ।

আমরা চাইলে, ইনডেক্স এর ভ্যালু স্কিপ করতে পারতাম । সেক্ষেত্রে পিএইচপি নিজে থেকেই ইনডেক্স এর ভ্যালু হিসেবে ক্রমিক সংখ্যা ব্যবহার করতো ।

```
<?php
$list = array('a', 'c', 3, 'wow', 5);
```

এখানে এই এ্যারের ইনডেক্সগুলো হবে - 0, 1, 2, 3, 4 - মনে রাখতে হবে, এই ইনডেক্স হলো জিরো বেইড, অর্থাৎ ইনডেক্স গননা শুরু হয় জিরো থেকে । প্রথম আইটেমের ইনডেক্স তাই জিরো হয়, যে কোন আইটেমের ইনডেক্স হয় তার নিউমেরিক্যাল পজিশন থেকে এক কম।

পিএইচপি 5.4 থেকে এ্যারে ডিফাইন করার জন্য শর্টহ্যান্ড ব্যবহার করা যায়:

```php
<?php

$array = [
    "foo" => "bar",
    "bar" => "foo",
];

$list = ['a', 'z', 2, 10];
```

এখানে আমরা `array` এর পরিবর্তে `[` এবং `]` এর মধ্যে কি-ভ্যালু ডিফাইন করি ।

পিএইচপির এ্যারে ডাটা টাইপটি একই সাথে এ্যারে, লিস্ট, ডিকশনারী, স্ট্যাক, কিউ, কালেকশান প্রভৃতির কাজ করতে পারে । এ্যারে নিয়ে বিস্তারিত আলোচনা থাকছে মাস্টারিং এ্যারে চ্যাপ্টারে ।

অবজেক্ট টাইপ

পিএইচপি ক্লাস থেকে `new` কিওয়ার্ড ব্যবহার করে অবজেক্ট ইন্সট্যান্স তৈরি করা যায় । অবজেক্ট সম্পর্কে আরো বিস্তারিত আমরা অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং চ্যাপ্টারে দেখবো ।

নাল টাইপ

যখন কোন ভ্যারিয়েবলের কোন ভ্যালু থাকে না তখন সেটা নাল টাইপ এর হয় । এই টাইপের একমাত্র গ্রহনযোগ্য ভ্যালু হলো - `null` - যার মানে ঐ ভ্যারিয়েবল এর কোন ভ্যালু নেই ।

টাইপ কনভার্শন

অটোমেটিক কনভার্শন

পিএইচপিতে আমরা ভ্যারিয়েবল ডিক্লেয়ার করার সময় এর টাইপ নির্ধারণ করে দিতে পারি না । ভ্যারিয়েবল এর ভ্যালুর উপর নির্ভর করে পিএইচপি নিজে থেকেই ডাটা টাইপ নির্বাচন করে নেয় । যেমন `$var` এর ভ্যালু হিসেবে যদি আমরা `hello world` পাস করি, তাহলে `$var` হবে স্ট্রিং, পরবর্তীতে যদি `$var` এর ভ্যালু হিসেবে `23` হয় তবে সেটি হবে ইন্টিজার ।

পিএইচপি কনটেক্সট অনুযায়ী ভ্যারিয়েবল এর টাইপ স্বয়ংক্রিয়ভাবে পরিবর্তন করে নেয় । যেমন, আমরা যখন যোগ করি তখন যদি উভয় পাশই ডাবল ভ্যালু হয় তখন রেজাল্ট ও হবে ডাবল । কিন্তু যে কোন একটি যদি ডাবল না হয় তাহলে রেজাল্ট হবে ইন্টিজার ।

এসব ক্ষেত্রে মূল ভ্যারিয়েবল এর টাইপ পরিবর্তন হয় না কিন্তু ফলাফলের পরিবর্তন হয় । পিএইচপির এই অটোমেটিক টাইপ কনভার্শন এর ব্যাপারে খেয়াল রাখা জরুরী । নাহলে অনাকাঙ্ক্ষিত ফলাফল পাওয়া অসম্ভব কিছুই না ।

ম্যানুয়াল টাইপ কনভার্শন

ম্যানুয়ালি টাইপ কনভার্ট করতে আমরা `settype()` ফাংশনটি ব্যবহার করি । এটি ঐ ভ্যারিয়েবল এর টাইপ এবং ভ্যালু দুটোই পরিবর্তন করতে পারে ।

```php
<?php
$age = 23;
$name = "masnun";

settype($age, "string");
settype($name, "integer");

var_dump($age);
var_dump($name);
```

কন্সট্যান্টস, এক্সপ্রেশনস ও অপারেটরস

অপরিবর্তনশীল কন্সট্যান্টস

কন্সট্যান্ট এর নাম শুনেই বোঝা যাচ্ছে এর কাজই হলো পরিবর্তন না হওয়া । কনস্ট্যান্ট হিসেবে আমরা খুব সিম্পল ভ্যালু সংরক্ষণ করতে পারি । `define` ব্যবহার করে আমরা কন্সট্যান্ট তৈরি করি ।

```
<?php
define("FOO",     "something");
define("FOO2",    "something else");
define("FOO_BAR", "something more");
```

কন্সট্যান্ট এর নাম সাধারনত বড় হাতের লেখা হয় । নামকরণের ক্ষেত্রে ভ্যারিয়েবলের মতই শুরুতে সংখ্যা ব্যবহার করা যায় না, নামটি কেইস সেনসিটিভ ।

এক্সপ্রেশনস

পিএইচপির খুব গুরুত্বেপূর্ণ একটি বিষয় হচ্ছে এক্সপ্রেশন । পিএইচপিতে আমরা মোটামোটি যাই লিখি তার সবই এক্সপ্রেশন । একটি এক্সপ্রেশন এর সবসময়ই একটি ভ্যালু থাকে ।

আমরা যখন একটি ভ্যারিয়েবল কিংবা কন্সট্যান্ট ডিফাইন করি তখন কিন্তু আমরা একটি এক্সপ্রেশন ব্যবহার করছি । যেমন:

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

এখানে আমরা `$name` ভ্যারিয়েবল এ `masnun` স্ট্রিংটি এ্যাসাইন করেছি । এখানে এই `"masnun"` অংশটুকু হলো একটি এক্সপ্রেশন যার ভ্যালু হলো স্ট্রিং `masnun` ।

আবার যখন এই ভ্যারিয়েবলটিকেই পুনরায় আরেকটি ভ্যারিয়েবল এ এ্যাসাইন করছি তখন:

```
<?php

$nickname = $name;
```

এখানে কিন্তু `$name` অংশটুকু একটি এক্সপ্রেশন যার ভ্যালু হচ্ছে `masnun` । এখানে আমরা `$nickname` এর ভ্যালু হিসেবে `$name` এক্সপ্রেশনের ভ্যালুকে এ্যাসাইন করেছি ।

এক্সপ্রেশনের ব্যাসিকটা বুঝে নেওয়া জরুরী কেননা পিএইচপি এক্সপ্রেশন ভিত্তিক ল্যাঙ্গুয়েজ, পিএইচপিতে নানা ধরণের জটিল জটিল এক্সপ্রেশন ব্যবহার করে পরবর্তীতে আমাদের নানা সমস্যার সমাধান করা লাগবে ।

অপারেটরস

অপারেটর এক বা একাধিক ভ্যালু কিংবা এক্সপ্রেশনের সম্মিলন ঘটিয়ে একটি নতুন ভ্যালু তৈরি করে । যেমন:

```
<?php
$a = 23;
$b = 5;

$sum = $a + $b + 2;
```

এখানে শেষ লাইনে এসে তিনটি ভিন্ন এক্সপ্রেশন এর সমন্বয়ে আমরা নতুন একটি ভ্যালু পাচ্ছি । `$a + $b + 2` এটি নিজেও কিন্তু একটি এক্সপ্রেশন । সুতরাং আমরা এভাবেও বলতে পারি, অপারেটরের কাজ হচ্ছে একাধিক এক্সপ্রেশনের সমন্বয়ে নতুন একটি বৃহদাকার এক্সপ্রেশন তৈরি করা ।

অপারেটর প্রিসিডেন্স

শুরুতেই আসুন একটি সহজ অঙ্ক করি:

```
30 - 4 * 30 / 5 +  4
```

বলুনতো এটার ফল কেন `10` হবে? কারণ আমরা জানি এখানে সবার আগে ভাগ এবং গুন এর কাজ করতে হবে এরপর যোগ বিয়োগ । এটাই গনিতের নিয়ম, এখানে এই যে আমরা ভাগ এবং গুনকে অগ্রাধিকার দিলাম, এটাই অপারেটর প্রিসিডেন্স । যখন পিএইচপিতে একাধিক এক্সপ্রেশনের মধ্যে আমরা এমন করে অপারেটর ব্যবহার করি তখন কোন কোন অপারেটর অগ্রাধিকার পায় - তাই সঠিক ফলাফল পেতে আমাদেরকে অপারেটর প্রিসিডেন্স সম্পর্কে বিস্তারিত জানতে হবে ।

অপারেটর প্রিসিডেন্স সম্পর্কে আরো বিস্তারিত জানতে পিএইচপি ম্যানুয়ালের এই চ্যাপ্টারটি দ্রষ্টব্য - http://php.net/manual/en/language.operators.precedence.php - পরবর্তীতে এই কন্টেন্টও বাংলায় বিশদভাবে ব্যখ্যা করে লেখার ইচ্ছা আছে ।

কমন অপারেটরস

গানিতিক অপারেটরগুলো আমরা সবাই কমবেশী চিনি:

```
<?php
$a = -$a; // মান ঋনাত্মক করা হলো
$sum = 2 + 3; // যোগ
$sub = 6 - 3; // বিয়োগ
$mul = 5 * 6; // গুন
$div = 24 / 3; // ভাগ
$mod = 13 % 2; // ১৩ কে ২ দিয়ে ভাগ করলে ভাগশেষ ১
$exp = 2 ** 3; // ২ টু দি পাওয়ার ৩ = ৮
```

এ্যাসাইনমেন্ট অপারেটর (`=`) এর সাথে আমরা ইতোমধ্যে পরিচিত হয়েছি, এটার মাধ্যমে আমরা ভ্যালু এ্যাসাইন করি ।

```php
<?php
$a = ($b = 4) + 5;
```

এখন `$b` এর মান `4` এবং `$a` এর মান হবে `9` । অবজেক্ট টাইপ ব্যতিত প্রায় সকল টাইপের ক্ষেত্রেই এ্যাসাইনমেন্ট অপারেটর ডান পাশের এক্সপ্রেশনের ভ্যালু কপি করে, তাই মূল এক্সপ্রেশনে ব্যবহৃত ভ্যারিয়েবলগুলোর ভ্যালু পরিবর্তন হয় না । তবে আমরা যদি একই চাই যে দুটি ভ্যারিয়েবল ই একই মেমরী ব্লক তথা একই ডাটাকে নির্দেশ করুক সেক্ষেত্রে আমরা একটা `&` যোগ করে দিতে পারি নিচের মত করে:

```php
<?php

$a = 3;
$b = &$a;

$a = 5;
echo $b; // এটার মানও পরিবর্তন হয়ে 5 হবে
```

এটাকে এ্যাসাইনমেন্ট বাই রেফারেন্স বলা হয় ।

কম্প্যারিজন অপারেটরগুলো দুটো এক্সপ্রেশনের ভ্যালু কম্পেয়ার করার জন্য ব্যবহার করা হয় । যেমন:

```php
<?php

$a == $b; // TRUE হবে যদি $a আর $b এর ভ্যালু টাইপ কনভার্শনের পর একই হয়
$a === $b; // TRUE হবে যদি $a এবং $b এর ভ্যালু একই এবং এরা একই টাইপের হয়
$a != $b; // TRUE যদি টাইপ কনভার্শনের পর ভ্যালু একই না হয়
$a <> $b; // TRUE যদি টাইপ কনভার্শনের পর ভ্যালু একই না হয়
$a !== $b; // TRUE যদি টাইপ কনভার্শনের পর ভ্যালু একই না হয় অথবা তারা একই টাইপ না হয়
$a < $b; // TRUE যদি $a, $b থেকে ছোট হয়
$a > $b; // TRUE যদি $a, $b থেকে বড় হয়
$a <= $b; // TRUE যদি $a, $b এর সমান অথবা $b থেকে ছোট হয়
$a >= $b; // TRUE যদি $a, $b এর সমান অথবা $b থেকে বড় হয়
```

মনে রাখতে হবে, যদি আমরা স্ট্রিং এর সাথে নাম্বার কম্পেয়ার করি তবে পিএইচপি অটোমেটিক্যালি স্ট্রিংকে নাম্বারে কনভার্ট করবে । এরপর দুই নাম্বার ভ্যালু কম্পেয়ার করবে । তবে `===` বা `!==` এর বেলায়, যেখানে টাইপ সহ কম্পেয়ার করা হয় - এসব ক্ষেত্রে টাইপ কনভার্শন অটোমেটিক্যালি হয় না ।

ইনক্রিমেন্টাল ও ডিক্রিমেন্টাল অপারেটরস:

```php
<?php
++$a // আগে $a এর ভ্যালু এক বাড়িয়ে তারপর $a এর ভ্যালু রিটার্ন করে
$a++ // আগে $a এর ভ্যালু রিটার্ন করে তারপর $a এর ভ্যালু এক বাড়িয়ে দেয়

--$a // আগে $a এর ভ্যালু এক কমিয়ে নেয় তারপর $a এর ভ্যালু রিটার্ন করে
$a-- // আগে $a এর ভ্যালু রিটার্ন করে তারপর $a এর ভ্যালু এক কমিয়ে দেয়
```

লজিকাল অপারেটরস:

```php
<?php
$a and $b // TRUE যদি দুটোই TRUE হয়
$a or $b // TRUE যদি যে কোন একটা TRUE হয়
$a xor $b // TRUE যদি যে কোন একটা TRUE হয় কিন্তু দুটোই TRUE না হয়
!$a // TRUE যদি $a TRUE না হয়
$a && $b // TRUE যদি দুটোই TRUE হয়
$a || $b // TRUE যদি যে কোন একটা TRUE হয়
```

নোট: `and`, `or` এবং `||`, `&&` একই কাজ করলেও এদের অগ্রাধিকার ভিন্ন । যেটা আমরা অপারটের প্রিসিডেন্স টেবিলে বিস্তারিত দেখতে পাবো ।

এছাড়া আমরা দুটো স্ট্রিং কে সংযুক্ত করতে `.` অপারেটর ব্যবহার করি । যেমন:

```php
<?php
$a = "Hello ";
$b = $a . "World!"; // এখন $b এর ভ্যালু হবে "Hello World!"
```

কন্ট্রোল স্ট্রাকচারস

কন্ট্রোল স্ট্রাকচার - নাম শুনেই আন্দাজ করা যায় কন্ট্রোল রিলেটেড ব্যাপার । একটা পিএইচপি প্রোগ্রামের ফ্লো কন্ট্রোল করাই কন্ট্রোল স্ট্রাকচারের কাজ । আমরা জানি পিএইচপিতে আমাদের কোড লাইন বাই লাইন এক্সিকিউট করা হয় । কখনো কখনো আমাদের এই এক্সিকিউশন কোন শর্তের উপর নির্ভর করে । শর্ত সাপেক্ষে প্রোগ্রাম এর কোন অংশটুকু পিএইচপি এক্সিকিউট করবে সেটা আমরা এই কন্ট্রোল স্ট্রাকচারগুলোর মাধ্যমে নির্ধারণ করে দিবো ।

কোড ব্লক

পিএইচপিতে আমরা কয়েকটা স্টেটমেন্টকে একটা গ্রুপ কিংবা ব্লক আকারে ডিফাইন করতে পারি কার্লি ব্রেইস (`{ }`) ব্যবহার করার মাধ্যমে । ব্লক তৈরি করার সুবিধা হলো যখন কোন শর্ত সাপেক্ষে নির্দিষ্ট কিছু স্টেটমেন্ট রান করতে হয় তখন আমরা ঐ স্টেটমেন্টগুলোকে একটি কোড ব্লকে রেখে পিএইচপি কে বলে দেই অমুক ঘটনা সত্য হলে এই কোড ব্লকের ভিতরে রাখা স্টেটমেন্ট রান করতে । কন্ট্রোল স্ট্রাকচারের সাথে কোড ব্লকের সম্পর্ক গভীর ।

ইফ, এলস (If, Else)

কোন একটি বিশেষ শর্ত সাপেক্ষে কোন কোড ব্লক এক্সিকিউট করতে আমরা ইফ, এলস ব্যবহার করি । ইফ এর পরে আমরা একটি এক্সপ্রেশন দেই, এটি যদি সত্য হয় (অর্থাৎ এটার বুলিয়ান ভ্যালু `true` হয়) তাহলে ইফ ব্লক রান করে । যদি আমরা সাথে একটি এলস ব্লকও দিয়ে দেই, তাহলে ইফ স্টেটমেন্ট `false` হলে এই বিকল্প ব্লক টি রান করে ।

উদাহরণ:

```php
<?php

$age = 10;

if($age > 18) {
    echo "You are an adult!";
} else {
    echo "You are not an adult yet!";
}
```

এখানে আমরা একটি ভ্যারিয়েবল নিয়েছি `$age` এরপর ইফ ব্লকে আমরা চেক করছি এই ভ্যারিয়েবল এর মান ১৮ অপেক্ষা বেশি কিনা । যদি ১৮ অপেক্ষা বেশি হয় তাহলে প্রথম ব্লক রান করবে । যদি না হয় তবে দ্বিতীয় ব্লক । উপরের কোডটি রান করলে আমরা দেখবো এলস ব্লক রান করেছে কারন এই ভ্যারিয়েবল এর মান আমরা শুরুতেই ১০ নির্ধারণ করে দিয়েছি যেটা নি:সন্দেহে ১৮ থেকে বড় না ।

আমরা চাইলে একাধিক শর্তও যোগ করতে পারি `elseif` ব্যবহার করে । এসব ক্ষেত্রে প্রথমে ইফ ব্লক চেক করা হবে, এটা যদি সত্য না হয় তাহলে এলসইফ ব্লকগুলো একটা একটা করে চেক করা হবে । যদি কোনটাই `true` না হয় তাহলেই এলস ব্লকের কোড রান করবে । যেমন:

```php
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
```

পুনশ্চ: একটি ইফ বা এলস কোড ব্লকের মধ্যে চাইলে আরো ইফ এলস ব্লক যোগ করা যায় । এটাকে নেস্টেড ব্লক বলা হয় ।

হোয়াইল লুপ (While)

লুপ কি জিনিস তা আমরা সবাই কম বেশি জানি । হোয়াইল লুপের কাজ হচ্ছে আমাদের দিয়ে দেওয়া এক্সপ্রেশন যতক্ষণ পর্যন্ত সত্য হবে ততক্ষন পর্যন্ত সাথে দেওয়া কোড ব্লকটি এক্সিকিউট করা । প্রতিবার লুপ এর শুরুতে ঐ এক্সপ্রেশনটি পিএইচপি চেক করবে, যদি সত্য হয় তাহলে কোড ব্লকটি রান করবে, কোড ব্লক রান করা শেষ হলে সে আবার এক্সপ্রেশনটি চেক করবে - এভাবে চলতেই থাকবে যতক্ষণ পর্যন্ত ঐ এক্সপ্রেশন `true` রিটার্ন করবে । সুতরাং আমরা বুঝতে পারছি যদি আমাদের লুপ ভেঙ্গে বের হতে হয় তাহলে এমন কোন পরিবর্তন আনতে হবে যাতে ঐ এক্সপ্রেশন এর ভ্যালু আর সত্য না হয় । উদাহরণ দেখে নেই:

```php
<?php

$i = 1;
while ($i <= 10) {
    echo $i++;
}

echo "I am outside the loop";
```

এই উদাহরণে দেখুন, আমরা শুরুতে `$i` এর ভ্যালু দিয়েছি ১ । এরপর হোয়াইল লুপে শর্ত দিয়েছি যে `$i` এর ভ্যালু যতক্ষন ১০ এর ছোট বা সমান হবে ততক্ষণ পর্যন্ত যেন সে কোড ব্লকটি রান করে । লক্ষ্য করুন এই কোড ব্লকের ভিতরে আমরা `$i` এর ভ্যালু শুধু আউটপুটই দেইনি, সেই সাথে এটার ভ্যালু ১ করে বাড়িয়ে দিয়েছি । প্রতিবার লুপের শুরুতে পিএইচপি চেক করবে ভ্যারিয়েবলটার ভ্যালু ১০ থেকে কম কিনা, এরপর সে ভ্যালুটা আউটপুট দিয়ে সাথে ১ যোগ করে দিবে । এভাবে লুপ চলতে চলতে একটা সময় এসে `$i` এর ভ্যালু বাড়তে বাড়তে ১০ এর বেশি হয়ে যাবে । তখন আর হোয়াইল লুপে দেওয়া শর্ত সত্য থাকবে না, পিএইচপি তখন লুপ থেকে বের হয়ে পরবর্তী কোড এক্সিকিউট করা শুরু করবে ।

ডু-হোয়াইল লুপ (do-while)

আমরা দেখেছি হোয়াইল লুপ প্রথমেই শর্তটি চেক করে । যদি সত্য হয় তবেই সে কোড ব্লক রান করে । যদি এমন হয় যে প্রথমবারেই শর্তটি মিথ্যা হলো, তখন? তখন আর কোড ব্লকটি একবারো রান করবে না । ডু-হোয়াইল লুপের ব্যাপারটা একটু ভিন্ন । এখানে আমরা প্রথমে কোড ব্লক দিয়ে দেই, পিএইচপি এটা রান করে আগে, এরপর হোয়াইল

লুপের কন্ডিশন টা চেক করে । যদি সত্য হয় তাহলে আবার কোড ব্লক রান করে, সত্য না হলে লুপ ভেঙ্গে বের হয়ে যায় ।

যেহেতু ডু-হোয়াইল লুপে কোড ব্লক প্রথমে রান করে এবং এক্সপ্রেশন এর ভ্যালু পরে চেক করা হয় তাই নিশ্চিতভাবে বলা যায় যে প্রদত্ত কোড ব্লকটি অন্তত একবার রান করবেই । ঠিক এই একবার রান করা নিশ্চিত করার জন্যই হোয়াইল এর পরিবর্তে ডু-হোয়াইল ব্যবহার করা হয় ।

কোড এক্সাম্পল দেখে নেই:

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
```

দেখুন এখানে `$i` এর ভ্যালু ০ অপেক্ষা বেশি না হলেও কোড ব্লকটি রান করে এবং ০ আউটপুট দেয় ।

ফর লুপ (for)

এটা একটু তুলনামূলকভাবে জটিল লুপ । এখানে আমরা ৩টা এক্সপ্রেশন দিয়ে দেই -

- প্রথমটা যখন প্রথমবার লুপ শুরু হবে তখন রান করা হবে ।
- ২য় টা হচ্ছে লুপের মূল শর্ত, হোয়াইল লুপের মত প্রতিবার লুপ শুরু হওয়ার সময় এটা চেক করা হবে । এটার বুলিয়ান ভ্যালু `true` হলেই কেবল লুপটি চলবে ।
- ৩য়টা প্রতিবার কোড ব্লক শেষ করে রান করা হয় ।

আমরা খুব সিম্পল একটা উদাহরণ দেখে নেই:

```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

এখানে,

- `$i = 1` এটা লুপ প্রথমবার শুরু হওয়ার সময়ই রান করবে । অর্থাৎ আমরা `$i` এর ভ্যালু ১ সেট করে নিলাম ।
- এবার হলো শর্ত পরীক্ষা করার পালা । প্রতিবার লুপের শুরুতে চেক করা হবে `$i` এর ভ্যালু ১০ এর সমান কিংবা কম আছে কিনা । থাকলে কোড রান করবে, না হলে লুপ শেষ করে বের হয়ে যাবে ।
- প্রতিবার লুপ শেষে আমরা ভ্যারিয়েবলটির ভ্যালু ১ করে বাড়িয়ে দিচ্ছি যাতে একটা সময় গিয়ে এটার ভ্যালু ১০ এর বেশি হয় ।

সংক্ষেপে এটাই হলো ফর লুপের কাহিনী । আমরা চাইলে যে কোন এক্সপ্রেশন ফাকা রাখতে পারি । যদি আমরা ভুলে বা ইচ্ছাকৃতভাবে ২য় এক্সপ্রেশনটি ফাকা রাখি, সেক্ষেত্রে ফর লুপটি চলতেই থাকবে কেননা এটি কতক্ষন পর্যন্ত চলতে হবে সেই নির্দেশনটা আমরা পিএইচপি কে দেই নি ।

ফর-ইচ লুপ (foreach)

কোন এ্যারে থেকে ভ্যালুগুলোকে একটা একটা নিয়ে কাজ করার জন্যই ফর-ইচ লুপ ব্যবহার করা হয় । এই লুপের এক্সপ্রেশন হিসেবে আমরা বলে দেই একটি এ্যারে থেকে কিভাবে প্রতিটা আইটেম নিতে হবে । এরপর সেই নির্দেশনা অনুযায়ী পিএইচপি আইটেমগুলোকে একটা একটা করে আমাদের কোড ব্লকে পাস করে । প্রতি লুপে এভাবে আমরা এ্যারের একেকটি আইটেম পাই । যতক্ষণ পর্যন্ত ঐ এ্যারের সবগুলো আইটেম একবার করে নেওয়া না হবে ততক্ষণ পর্যন্ত এই লুপ চলতেই থাকবে ।

আমরা আমাদের এক্সপ্রেশন এ বলে দিতে পারি যে আমরা কি শুধু এ্যারের ভ্যালুটা চাই নাকি ইনডেক্স ভ্যালু দুটোই চাই । নিচের উদাহরণ দেখি:

```php
<?php

// এই উদাহরণে শুধুই আইটেম টি নিবো
$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
    echo $value;
}

// এখানে আমরা ইনডেক্স এবং ভ্যালু দুটোই নিবো
$arr2 = array("a" => "abdullah", "b" => "bahar");
foreach($arr2 as $k => $v) {
    echo $k. "=". $v;
}
```

প্রথম উদাহরণে আমরা পিএইচপিকে বলে দিচ্ছি `$arr` এ্যারে থেকে একেকটি আইটেম নিয়ে সেটিকে `$value` ভ্যারিয়েবলে রাখতে । এরপর কোড ব্লকটি রান করতে । ২য় উদাহরণে, আমরা বলে দিচ্ছি ইনডেক্স এর ভ্যালু টা `$k` এবং আইটেম এর ভ্যালুটা `$v` ভ্যারিয়েবলে রেখে কোড ব্লকটি রান করাতে ।

সুইচ (switch)

সুইচ ব্যবহার করে আমরা খুব সহজে কোন একটা এক্সপ্রেশন এর ভ্যালুর উপর নির্ভর করে অনেকগুলো সিদ্ধান্ত থেকে এক বা একাধিক বেছে নিতে পারি । এর আগে আমরা ইফ-এলস, এলস-ইফ এর ব্যবহার দেখেছি । যখন অনেকগুলো শর্ত পরপর চেক করার প্রয়োজন হয় তখন আমরা এলস-ইফ ব্যবহার করি, অনেকটা নিচের মত করে -

```php
<?php
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}
```

সাধারনত এরকমভাবে একগাদা এলস-ইফ ব্লক এ্যভয়েড করার জন্য সুইচ পারফেক্ট । এখানে আমরা কয়েকটা কন্ডিশন চেক করছি, প্রথমে দেখছি `$i` এর ভ্যালু শুন্য কিনা, এরপর চেক করছি এটির মান ১ কিনা, এরপর আবার দেখছি ২ কিনা । সুইচ এর মাধ্যমে এভাবে বার বার চেক না করে, একবারেই চেক করে মান অনুসারে সিদ্ধান্ত নেওয়া যায় । উপরের কোডই আমরা সুইচ ব্যবহার করে এভাবে লিখতে পারি -

```
<?php
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
```

এখানে, আমরা `$i` এর ভ্যালুর উপর নির্ভর করে আমাদের সিদ্ধান্ত নিবো তাই এটাকে সুইচ এ পাস করা হয়েছে । এরপর আমাদের কোড ব্লকের ভিতরে কতগুলো `case` বা সম্ভাব্য ঘটনা বিচার করা হয়েছে । `case` এর সাথে আমরা প্রত্যাশিত ভ্যালু দিয়ে একেকটি কেইস ডিফাইন করি । যেমন, `case 0` মানে হলো `$i` এর মান যখন 0 হবে তখন এই কেইসটি সত্য । এরপর কোলন (`:`) দিয়ে আমরা আমাদের প্রয়োজনীয় কোড লিখি । এখানে আমরা একটা আউটপুট দিয়ে রেখেছি । এরপর যে কোন কেইস মিলে গেলে আমরা চাই সুইচ ব্লক থেকে বের হয়ে যেতে, তাই `break` এর ব্যবহার । `break` সম্পর্কে আমরা একটু পরেই জানবো । আপাতত জেনে রাখি, কোন কেইস এর পর ব্রেক না থাকলে ঐ কেইসটি ম্যাচ করেই পিএইচপি থেমে যাবে না, বরং পরবর্তী কেইসগুলাও ম্যাচ করে দেখবে । একাধিক কেইসও এভাবে ম্যাচ করা সম্ভব । তবে অধিকাংশ ক্ষেত্রেই আমরা চাইবো একটি কেইস ম্যাচ করলে সুইচ থেকে বের হয়ে যেতে ।

ব্রেক (break)

যে কোন ধরণের কন্ট্রোল স্ট্রাকচার থেকে বের হয়ে আসতে ব্রেক ব্যবহার করা হয় । যেমন:

```
<?php
$array = ["a", "b", "c", "d"];
foreach($array as $val) {
    echo $val;
    if($val == "b") {
        break;
    }
}
```

এই উদাহরণে আমরা দেখছি, এ্যারে থেকে সবগুলো আইটেম নেওয়ার আগেই লুপটি বন্ধ হয়ে যাচ্ছে, কারণ যখনই আইটেম এর ভ্যালু `b` হচ্ছে তখনই আমরা `break` স্টেটমেন্ট ব্যবহার করে লুপ থেকে বের হয়ে আসছি । ফলে বাকি আইটেমগুলো আর আউটপুটে আসছে না ।

কন্টিনিউ (continue)

ব্রেক এর কাজ লুপ থেকে বের হয়ে যাওয়া, কন্টিনিউ এর কাজ বর্তমান লুপের বাকি অংশ স্কিপ করে সরাসরি পরবর্তী লুপ শুরু করা ।

```
<?php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2) {
        continue;
    }
    print "$i\n";
}
```

এই কোডটি রান করলে 2 আউটপুটে আসবে না, কারণ যখনই `$i` এর ভ্যালু ২ হবে তখনই পিএইচপি কন্টিনিউ স্টেটমেন্টটি দেখবে । তখন আর কোড ব্লকের বাকি অংশ রান না করেই পরবর্তী লুপে প্রবেশ করবে । তাই `print "$i\n";` এই অংশটি ২ এর বেলায় এক্সিকিউট করা হবে না, আউটপুটেও তাই ২ আসবে না । অর্থাৎ লুপিং এর সময় কন্টিনিউ ব্যবহার করলে পিএইচপি সাথে সাথে ব্লকের বাকি অংশ স্কিপ করে পরবর্তী আইটেমে চলে যায় ।

এক্সারসাইজ

1. একটা স্ক্রিপ্ট লিখেন যেটা চলতি মাসের নাম নিবে এবং তার উপর ভিত্তি করে নিচের মেসেজ প্রিন্ট করবে।
    - যদি মাসের নাম November হয় তাহলে প্রিন্ট করবে, Winter is comming.
2. যদি November না হয়, তাহলে প্রিন্ট করবে, You better stay in South.

হিন্টস : date('F', time()) এইটা ব্যবহার করে আপনি, মাসের নাম নিতে পারবেন।

2.ধরুন আপনি Zimbabwe এর একজন বাসিন্দা । আপনার বন্ধু বিভিন্ন দেশে ঘুরে ঘুরে খেলা দেখে । সে আপনাকে একদিন বলল, " ওমুক দেশটা খুব সুন্দর " । বন্ধুর কথা শুনে, আপনার ঐদেশে ঘুরতে যেতে ইচ্ছে হল। কিন্তু সে আপনাকে দেশেটির নাম বলেনি। বরং আপনাকে একটি চিরকুট দিল। যাতে লেখা - "Bangladesh is beautiful country" । এবং সে আপনাকে কতগুলো দেশের নামের লিস্ট দিল যেগুলো হল -"England", "Bangladesh", "Sri Lanka", "India" ।

এখন এই দুটো তথ্যের উপর ভিত্তি করে আপনাকে ঐদেশের নাম কি সেটা বলতে হবে।

হিন্টস : স্ট্রিং কে অ্যারেতে কনভার্ট করার জন্য explode() ফাংশনটি ব্যবহার করতে পারেন। ডিটেইলস : http://php.net/manual/en/function.explode.php

আমরা পরবর্তিতে আরোও এক্সারসাইজ দেয়ার চেষ্টা করব।

ফাংশনস

ফাংশন মুলত কিছু স্টেটমেন্টের সমন্বয়ে তৈরি একটি কোড ব্লক যা একটি প্রোগ্রামে বার বার রিইউজ করা সম্ভব। পিএইচপি স্ট্যান্ডার্ড লাইব্রেরীতে অসংখ্যা বিল্টইন ফাংশন রয়েছে যেগুলো আপনি খুব সহজেই ব্যবহার করতে পারেন । এছাড়াও আপনি চাইলে আপনার প্রয়োজনমত ফাংশন লিখে নিতে পারেন

বিল্টইন ফাংশন

পিএইচপি তে বাই ডিফল্ট যেসব ফাংশন আগেই তৈরি করা থাকে সেগুলোকেই আমরা বিল্ট ইন ফাংশন বলি । এগুলো আমাদের জন্য আগেই তৈরি করা তাকে, আমাদের কাজ শুধু কল করা ।

```
<?php
var_dump("This is .....");
?>
```

ইউজার ডিফাইনড ফাংশন

আমাদের নানা কাজে নানা ধরণের ফাংশন প্রয়োজন হয়, এই সব ক্ষেত্রে আমরা নিজেরাই ফাংশন তৈরি করে নিতে পারি বা অন্যের তৈরি করা ফাংশন ব্যবহার করতে পারি । যে সব ফাংশন ইউজার অর্থাৎ আমাদের (আমার নিজের বা অন্য কোন ডেভেলপারের) তৈরি করা সেগুলোকে ইউজার ডিফাইনড ফাংশন বলা হয় ।

```
<?php
function sayHello($name)
{
    return "Hello {$name}!";
}
?>
```

এখানে আমরা একটি ইউজার ডিফাইনড ফাংশন দেখছি, এই ফাংশনটি একটি নাম এ্যাক্সেপ্ট করে একটি সুন্দর গ্রিটিং রিটার্ন করে ।

ডিফাইনিং ফাংশন

যেকোন ইউজার ডিফাইনড ফাংশন লিখতে হলে আপনাকে নিচের সিনট্যাক্স অনুসরণ করতে হবে:

```php
<?php
function functionName($arg1,$arg2)
{
    // কোড ব্লক
}
?>
```

প্রথমে ফাংশন কিওয়ার্ড, তারপর ফাংশনের নাম, এরপর ব্রাকেটে প্যারামিটার লিস্ট, এরপর ফাংশনের মূল বডি যেটি কিনা একটি কোড ব্লক ।

ফাংশনের নাম অবশ্যই ইউনিক হতে হবে, নাম করনের ক্ষেত্রে পিএইচপির সচারচার নিয়মগুলোই অনুসরন করা হয় । ফাংশনের প্যারামিটার ফাকা থাকতে পারে । অধিকাংশ ফাংশনই প্রসেসিং এর পর একটা ভ্যালু রিটার্ন করে, এটাকে রিটার্ন ভ্যালু বলা হয় । তবে কখনো কখনো ফাংশন ভ্যালু নাও রিটার্ন করতে পারে । এই ধরনের ফাংশনকে ভয়েড ফাংশন বলা হয় ।

প্যারামিটার ও আর্গুমেন্ট

একটি ফাংশন যেসব ইনপুট গ্রহন করে এগুলোই হলো প্যারামিটার । প্যারামিটার গুলো ভ্যারিয়েবল হিসেবে ডিফাইন করা হয় এবং ফাংশন বডির ভিতরে ঐ প্যারামিটারগুলোর ভ্যালু আমরা ঐ ভ্যারিয়েবলগুলো থেকে পাই ।

আর ফাংশন কল করার সময় ঐ প্যারামিটারগুলোর জন্য ভ্যালু পাস করার জন্য যে এক্সপ্রেশন ব্যবহার করি সেটাই হলো আর্গুমেন্ট ।

সহজ ভাষায়, ফাংশন ডিফাইন করার সময় ব্যবহৃত ভ্যারিয়েবলগুলো প্যারামিটার, কল করার সময় ফাংশন টাকে যেই ভ্যালু পাস করে কল করি তা হলো আর্গুমেন্ট ।

উদাহরণ:

```php
<?php
function testFunc($name, $age)
{

}

testFunc("masnun", 5*4)
```

এখানে `$name` এবং `$age` হলো প্যারামিটার, `"masnun"` এবং `5*4` হলো আর্গুমেন্ট । তবে প্রায়শই আমরা দেখি এই দুটো টার্ম এর একটার জায়গায় আরেকটা টার্ম ব্যবহৃত হতে ।

প্যারামিটার ছাড়া ফাংশন

```php
<?php
function functionName()
{
   echo "Hi i don't have any argument!";
}
?>
```

এই ফাংশনটিতে কোন প্যারামিটার নেই। অর্থাৎ একে কল করার সময় কোন আর্গুমেন্ট পাস করাতে হবে না। অনেকটা এই রকম করে `functionName();` । প্যারামিটার না নিলে প্যারামিটার লিস্ট এর ব্রাকেট টা আমরা এভাবে ফাকাই রাখবো ।

প্যারামিটার সহ ফাংশন

```php
<?php
function functionName($arg1,$arg2)
{
   echo $arg1;
   echo $arg2;
}
?>
```

প্যারামিটার গুলো ব্রাকেটের মধ্যে কমা দিয়ে আলাদা করা হয় । এই ফাংশনটিতে দুইটি প্যারামিটার আছে `$arg1` এবং `$arg2` , এই ফাংশনটিকে কল করতে হলে ফাংশনের মধ্যে এই দুইটি প্যারামিটার এর জন্য আর্গুমেন্ট পাস করাতে হবে। অনেকটা এই রকম করে `functionName('This is arg 1','This is 2');` , যদি ফাংশনটি কল করার সময় আর্গুমেন্টগুলো না থাকে তবে আর্গুমেন্ট মিসিং এরর দেখাবে। তাই এগুলো রিকোয়ারড আর্গুমেন্ট ।

পিএইচপি ফাংশনে একটি প্যারামিটারের ডিফল্ট ভ্যালু ডিফাইন করে দেওয়া যায় । যেমন:

```php
<?php
function functionName($arg1="default value")
{
   echo $arg1;
}
?>
```

এই ফাংশনটি কল করার সময় কোন আর্গুমেন্টের ভ্যালু না দিলেও চলবে (`functionName();`) । সেক্ষেত্রে `$arg1` এর ভ্যালু হবে তার ডিফল্ট ভ্যালু । এভাবেই আমরা যে কোন প্যারামিটারের ডিফল্ট ভ্যালু ডিফাইন করে দিয়ে সেই প্যারামিটারটিকে অপশনাল প্যারামিটারে পরিনত করতে পারি ।

রিটার্ন ভ্যালু

পূর্বে ব্যবহৃত ইউজার ডিফাইনড ফাংশনগুলোতে কোন ভ্যালু রিটার্ন করা হয় নি, সরাসরি আউটপুট দেওয়া হয়েছে। সরাসরি আউটপুট দেখানোর পাশাপাশি ফাংশনগুলো ভ্যালু রিটার্ন করতে পারে যেগুলো আমরা কোন এক্সপ্রেশনে ব্যবহার করতে পারি ।

`return` কিওয়ার্ডটি ব্যবহার করে ভ্যালু রিটার্ন করা হয়।

```
<?php
function functionName($arg1,$arg2)
{
   return $arg1.$arg2;
}
?>
```

ভ্যারিয়েবল ফাংশন

আমরা চাইলে একটি ফাংশনকে ভ্যারিয়েবল ব্যবহার করে কল করতে পারি । এক্ষেত্রে আমরা ফাংশনটির নাম একটি ভ্যারিয়েবল এ এ্যাসাইন করি । এরপর ঐ ভ্যারিয়েবলটির পর `()` ব্যবহার করে ফাংশনটি কল করি । উদাহরণ দেখি:

```
<?php
function foo() {
    echo "In foo()<br />\n";
}

function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br />\n";
}

// This is a wrapper function around echo
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // This calls foo()

$func = 'bar';
$func('test');  // This calls bar()

$func = 'echoit';
$func('test');  // This calls echoit()
?>
```

[উদাহরনটি পিএইচপি ম্যানুয়াল থেকে নেওয়া]

ফাংশনে ভ্যারিয়েবলের ব্যবহার

ইতোমধ্যে আমরা দেখেছি ফাংশনে কিভাবে আর্গুমেন্ট পাস করতে হয়। মুলত এই আর্গুমেন্ট গুলো সংরক্ষণ করা হয় লোকাল ভ্যারিয়েবল এ। এই ভ্যারিয়েবলগুলোকে ফাংশনের বাইরে থেকে এক্সেস করা যায় না। ফাংশনটি যখন কল করা হবে তখন এই ভ্যারিয়েবলগুলো মেমোরিতে সংরক্ষিত হবে। ফাংশন শেষে এই ভ্যারিয়েবলগুলোর আর কোন অস্বিত্ব থাকবে না ।

পিএইচপিতে মুলত দুই ধরনের ভ্যারিয়েবল এর ধারণা প্রচলিত, একটি হচ্ছে `global` ভ্যারিয়েবল আর অন্যটি `local` ভ্যারিয়েবল। সাধারণত কোড ব্লকের বাইরের পিএইচপিতে ভ্যারিয়েবল গুলো গ্লোবাল হয়, এসব ক্ষেত্রে আপনার প্রোগ্রামে একই নামে দুটি ভ্যারিয়েবল লিখলে ২য় টির দ্বারা প্রথমটির ভ্যালু প্রতিস্থাপিত হবে। অন্যদিকে ফাংশন / মেথডে ব্যবহারিত ভ্যারিয়েবলগুলো সাধারনত লোকাল ভ্যারিয়েবল হয়। ঐ ভ্যারিয়েবলগুলোকে শুধুমাত্র ওই মেথড / ফাংশনের মধ্যে ব্যবহার করা যাবে।

```
<?php
$myvar = 100; // $myvar একটি গ্লোবাল ভ্যারিয়েবল

function myFunc()
{
    $myvar = 50; // $myvar একটি লোকাল ভ্যারিয়েবল
}
?>
```

চাইলে গ্লোবাল ভ্যারিয়েবলগুলোকে সরাসির ফাংশন / মেথডের ভেতরে ব্যবহার করা যায়। এর জন্য `global` স্টেটমেন্ট ব্যবহার করতে হবে। নিচের উদাহরনটি দেখুনঃ

```
<?php
$myvar = 100; // $myvar একটি গ্লোবাল ভ্যারিয়েবল

function myFunc()
{
    global $myvar;
    echo $myvar // $myvar একটি গ্লোবাল ভ্যারিয়েবল
}
?>
```

এখানে আমরা ফাংশনের ভিতর থেকেও গ্লোবাল `$myvar` ভ্যারিয়েবলটি এ্যাক্সেস করেছি ।

পিএইচপিতে যখন কোন ফাংশনে / মেথডে কোন লোকাল ভ্যারিয়েবল ব্যবহার করা হয় তখন ফাংশনটি কল হওয়ার সময় ভ্যারিয়েবলটি তৈরি হয় এবং কল শেষ হলে ধ্বংস হয়ে যায়। অনেক সময় আমাদের ওই ফাংশনটি পরবর্তীতে কল করা হলে ভ্যারিয়েবলটির আগের মান জানার দরকার হতে পারে। `static` স্টেটমেন্ট ব্যবহার করে ওই ভ্যারিয়েবলটির ভ্যালু পরবর্তী কলের জন্য সংরক্ষন করা সম্ভব । নিচে একটি উদাহরন দেওয়া হল ।

মনে করা যাক আমাদের একটি ফাংশন আছে যার নাম `query()` এখন আমাদের পুরো প্রোগ্রামে সকল ডাটাবেস কুয়েরির জন্য আমরা এই ফাংশনটি ব্যবহার করব। এই কারনে আমারা ফাংশনের মধ্যে একটি `static` ভ্যারিয়েবল ব্যবহার করব। যা কতবার এই ফাংশনটিকে কল করা হয়ে তার হিসাব রাখবে।

```php
<?php
function query()
{
    static $count;
    $count += 1;
    $queryResult = "This is Query Result";
    return array('query' => $queryResult, 'count' => $count);
}
// চারবার কল করা হয়েছে
var_dump(query());
var_dump(query());
var_dump(query());
var_dump(query());
?>
```

আউটপুটঃ

```
array (size=2)
  'query' => string 'This is Query Result' (length=20)
  'count' => int 1
array (size=2)
  'query' => string 'This is Query Result' (length=20)
  'count' => int 2
array (size=2)
  'query' => string 'This is Query Result' (length=20)
  'count' => int 3
array (size=2)
  'query' => string 'This is Query Result' (length=20)
  'count' => int 4
```

এনোনিমাস ফাংশন

এই জাতীয় ফাংশনগুলোর কোন স্পেসিফিক নাম থাকে না। উদাহরন হিসাবে নিচের কোডটি দেখুন।

```php
<?php
$data = function (){
    $val = array();
    for($i=0;$i<10;$i++)
    {
        $val[] = $i;
    }
    return $val;
};

var_dump($data());
?>
```

এখানে আমরা ফাংশনটির কোন নাম দেইনি কিন্তু বাকি অংশগুলো ঠিক রেখেছি । এই ফাংশনটির কোন নাম না থাকলেও আমরা `$data` ভ্যারিয়েবলটি ব্যবহার করে ফাংশনটি কল করতে পারি ।

এনোনিমাস ফাংশনে স্বাভাবিক ফাংশনের মত করেই আর্গুমেন্ট ব্যবহার করা যায়, নিচের কোড দেখুনঃ

```
<?php
$data = function ($limit){
    $val = array();
    for($i=0;$i<$limit;$i++)
    {
        $val[] = $i;
    }
    return $val;
};

var_dump($data(20));
?>
```

রিকার্সিভ ফাংশন

কোন ফাংশন যখন নিজে নিজেকে কল করে তখন তাকে রিকার্সিভ ফাংশন বলা হয়ে থাকে। নিচে ফ্যাক্টোরিয়াল এর উদাহরন দেওয়া হলো । এখানে `$n` এর ভ্যালু কমিয়ে কমিয়ে ফাংশনটি রিকার্সিভলি কল করা হয় । যখন `$n` এর ভ্যালু শূন্য হয় তখনই সে থেমে যায় । যে শর্তের উপর নির্ভর করে ফাংশনটি নিজেকে আবার কল করে বা থেমে যায় এটাকে বেইজ কন্ডিশন বা বেইজ কেইস বলা হয় । রিকার্সিভ ফাংশনে বেইজ কেইস সেট করে দেওয়া জরুরী নাহলে এই রিকার্শন থিওরেটিক্যালি থামবে না । প্র্যাক্টিকালি পিএইচপি একটি নির্দিষ্ট লেভেল এর রিকার্শন এর পর ইরর থ্রো করবে ।

যদি রিকার্সিভ ফাংশন থেকে ডাটা রিটার্ন করতে হয় তবে মূল ফাংশনের ভেতরে নিজেকে আবার কল করার সময় ফাংশনের সামনে `return` লাগিয়ে কল করতে হবে।

```
<?php
function fact($n) {
  if ($n === 0) { // our base case
     return 1;
  }
  else {
     return $n * fact($n-1); // <--calling itself.
  }
}

var_dump(fact(10));
?>
```

এখানে ফাংশনটি নিজেকে কল করে অপেক্ষা করতে থাকে সেটির রিটার্ন ভ্যালুর জন্য । সেই ফাংশনটি আবার নিজেকে কল করে অপেক্ষা করতে তাকে । এভাবে একটা নেস্টেড অবস্থা তৈরি হয় । এবং সাধারনত সব শেষে কল করা ফাংশন (যেটি বেইজ কেইস ম্যাচ করে) সেটি আগে ভ্যালু রিটার্ন করে এবং কন্ট্রোল তার আগের কলারকে

ফিরিয়ে দেয় । এভাবেই রিকার্শন কাজ করে ।

সাধারণত ফাংশন নেস্টিং ৯৯ বার পর্যন্ত লিমিট করা থাকে। তাই Fibonacci জাতীয় প্রোগ্রাম for / while / if দিয়ে করা উচিত।

মাস্টারিং এ্যারে

আমরা ডাটাটাইপ চ্যাপ্টারে প্রথম এ্যারে এর সাথে পরিচিত হই । পিএইচপিতে এ্যারে খুবই গুরুত্বপূর্ন কনসেপ্টগুলোর মধ্যে অন্যতম । এই চ্যাপ্টারে তাই আমরা এ্যারে সংশ্লিষ্ট বিষয়গুলো দেখবো ।

ডিফাইনিং এ্যারে

ডাটা টাইপ চ্যাপ্টারে আমরা এ্যারে কিভাবে ডিফাইন করতে হয় তা দেখেছি । আবারো একবার দ্রুত দেখে নেই:

এ্যাসোসিয়েটিভ এ্যারে

এই এ্যারেতে একটা কি (key) এর বিপরীতে একটা ভ্যালু স্টোর করা হয় ।

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// PHP 5.4 থেকে শর্টহ্যান্ড ব্যবহার করা যায়
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

ইনডেক্সেড এ্যারে

এখানে আমরা কোন কি ডিফাইন করি না । পিএইচপি নিজে থেকেই ক্রমিক সংখ্যা ব্যবহার করে ইনডেক্স এর জন্য ।

```
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
```

মিক্সড এ্যারে

এ ধরণের এ্যারে তে একই সাথে আমরা অটো ইনডেক্স এর পাশাপাশি নিজেদের প্রয়োজনীয় কি ডিফাইন করে দেই । যেমন:

```
<?php
$array = array(23, 87, 32, "name" => "masnun", 43);
```

এখানে পিএইচপি প্রথম ৩টি আইটেমের ক্ষেত্রে ইন্টিজার ব্যবহার করবে । `name` কি টি স্ট্রিং । এরপর আবার পরের আইটেমটির জন্য আগের ইন্টিজার ভ্যালুর পরবর্তী ক্রমিক সংখ্যাটি ব্যবহার করবে ।

কুইক নোটস

- এ্যারে তে লাস্ট আইটেম এর পর কমা দেওয়া অপশনাল । তবে মাল্টিলাইনে শেষ লাইনের শেষে কমা দেওয়া রিকমন্ডেড ।
- এ্যারের ভ্যালু যে কোন টাইপ হতে পারে । কিন্তু কি (key) এর টাইপ অবশ্যই স্ট্রিং অথবা ইন্টিজার হতে হবে ।
- কি এর টাইপ যদি স্ট্রিং হয় এবং ঐ স্ট্রিং যদি ভ্যালিড ইন্টিজারে কনভার্ট করা সম্ভব হয় তাহলে পিএইচপি ঐ কি এর টাইপ অটোমেটিক্যালি ইন্টিজার করে ফেলে । অর্থাৎ আপনার কি যদি হয় `"3"` তাহলে পিএইচপি ওটাকে `3` এ কনভার্ট করে ব্যবহার করবে ।
- ফ্লোটিং পয়েন্ট নাম্বার কিংবা বুলিয়ান হলে সেটা ইন্টিজারে কনভার্ট করে নেয় অনুরূপভাবে ।
- Null হলে সেটা এম্পটি স্ট্রিং এ পরিবর্তন করে নিবে ।
- অন্য কিছু কি হিসেবে ব্যবহার করতে গেলে Illegal key offset এরর পাওয়া যাবে ।
- কি (key) অপশনাল । যদি কি এর কোন ভ্যালু না দেওয়া হয় তাহলে পিএইচপি আগে ব্যবহার করা সবচেয়ে বড় ইন্টিজার কি এর ভ্যালু এক বাড়িয়ে নতুন কি তৈরি করে নেয় । কোন ইন্টিজার কি না থাকলে শূন্য থেকে শুরু করে । ইনডেক্সেড এ্যারে তে আমরা একই ঘটনা দেখেছি ।
-
- এ্যারে গুলো জিরো বেইজড ইনডেক্স ব্যবহার করে । অর্থাৎ কি ডিফাইন না করে দিলে, প্রথম কি এর ভ্যালু হয় `0` । এরপর প্রতিবার এক এক করে বাড়ে ।

উদাহরণ:

```php
<?php
$array = array(
    1    => "a",
    "1"  => "b",
    1.5  => "c",
    true => "d",
);
var_dump($array);
```

এ্যাক্সেসিং এ্যারে

আমরা এ্যারে ডিফাইন করলাম। এবার ব্যবহার করার পালা । এ্যারে থেকে কোন এলিমেন্ট এর ভ্যালু পাওয়ার জন্য সেটার কি (key) দিয়ে আমরা নিচের মত করে এ্যাক্সেস করতে পারি:

```php
<?php
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
```

অর্থাৎ, এ্যারের ভ্যারিয়েবল এর পর থার্ড ব্রাকেটে আমরা কি পাস করি । `$array["foo"]` থেকে আমরা `$array` এর `foo` কি এর ভ্যালু পাই । আমরা এই উদাহরনে দেখছি এ্যারের ভিতরে আমরা আরো এ্যারে তৈরি করতে পারি । যে এ্যারের ভিতরে আরো এ্যারে থাকে সেটাকে আমরা মাল্টি ডাইমেনশনাল এ্যারে বলি । মাল্টি ডাইমেনশনাল এ্যারের ক্ষেত্রে আমরা প্রথমে একটি কি এর ভ্যালু বের করে নেই । সেটিও যদি এ্যারে হয় তবে পুনরায় আবার সেটির কি দিয়ে সংশ্লিষ্ট ভ্যালু বের করতে পারি ।

ইনডেক্সড এ্যারের ক্ষেত্রে কি গুলোর ভ্যালু নিউমেরিক অর্থাৎ ইন্টিজার হয় । আমরা জানি এই ইনডেক্স শুরু হয় শূন্য থেকে । প্রথম আইটেমটি তাই আমরা পাই `$array[0]` তে । এভাবে অন্যান্য আইটেমগুলিও আমরা তাদের নিজ নিজ ইনডেক্স ব্যবহার করে এ্যাক্সেস করা যায় । মিক্সড এ্যারের ক্ষেত্রে নিউমেরিক কি গুলো ইন্টিজার ভ্যালু ও স্ট্রিং কি গুলো তাদের স্ট্রিং ভ্যালু ব্যবহার করে এ্যাক্সেস করা হয় ।

মজার ব্যাপার হলো থার্ড ব্রাকেট এর পরিবর্তে আমরা সেকেন্ড ব্রাকেটও ব্যবহার করতে পারি । এটা ট্রাই করে দেখুন:

```php
<?php
$array = array(1,2,3);
var_dump($array{1});
```

এই যে কি দিয়ে কোন এ্যারে থেকে ঐ কি এর ভ্যালু এ্যাক্সেস করা - এটাকে ডিরেফারেন্সিং বলা হয় ।

পিএইচপি 5.4 থেকে আমরা সরাসরি ফাংশন থেকে রিটার্ন করা এ্যারে এ্যাক্সেস করতে পারি:

```php
<?php
function getArray() {
    return array(1, 2, 3);
}

// on PHP 5.4
$secondElement = getArray()[1];
```

এখানে আমরা `getArray()` এর ভ্যালু হিসেবে একটি এ্যারে পাই এবং সাথে সাথে আমরা সেটা ডিরেফারেন্স করছি । পিএইচপির আগের ভার্সন গুলোতে আমরা সরাসরি এভাবে ডিরেফারেন্স করতে পারতাম না । তখন আমাদের করতে হত নিচের মত করে:

```
<?php
$tmp = getArray();
$secondElement = $tmp[1];
```

অর্থাৎ ফাংশন এর রিটার্ন ভ্যালু প্রথমে একটি ভ্যারিয়েবল এ স্টোর করে নিয়ে তারপর সেই ভ্যারিয়েবল থেকে ভ্যালু বের করতে হত ।

এ্যারে মডিফাই করা

আমরা এ্যারে তে নতুন আইটেম যোগ করতে পারি, এক্সিস্টিং আইটেম এর ভ্যালু পরিবর্তন করতে পারি কিংবা পারি কোন আইটেম ডিলিট করে দিতে । আসুন দেখি এগুলো কিভাবে করা যায়:

নতুন আইটেম যোগ করা

কি সহ যোগ করা:

```
<?php

$array['key_name'] = 'val';
```

এক্ষেত্রে আমরা থার্ড ব্রাকেট এ কি এর নাম দিয়ে দেই এবং সাথে সাথে ভ্যালু ও এ্যাসাইন করি ।

কি ছাড়া যোগ করা:

```
<?php

$array[] = "value";
```

এখানে আমরা কি এর কোন নাম দেইনি । সরাসরি ভ্যালু এ্যাসাইন করেছি । এক্ষেত্রে পিএইচপি ঐ এ্যারের ইন্টিজার কি গুলোর মধ্যে সবচেয়ে যেটা বড় তার পরের ইন্টিজার ভ্যালু টা কি হিসেবে ব্যবহার করবে । যেমন:

```
<?php

$array = array("name" => "masnun", 23 => 'blah');
$array[] = 'aha';
var_dump($array);
```

এখানে সবচেয়ে বড় ইন্টিজার কি এর ভ্যালু ছিলো `23` , তাই `aha` এর কি হবে `24` (23 + 1) । এ্যারে ইনডেক্সিং এর ক্ষেত্রে পিএইচপির এই বিহ্যাভিয়র টা আমাদের মনে রাখা জরুরী ।

ভ্যালু পরিবর্তন করা

কি দিয়ে এ্যাক্সেস করে আমরা একটি এলিমেন্ট পাই । ঐ এলিমেন্ট এর ভ্যালু আমরা নতুন করে এ্যাসাইন করতে পারি যেমন করে আমরা ভ্যারিয়েবল এর মান পরিবর্তন করি ।

```
<?php
$array = array("name" => "masnun");
$array['name'] = "new name";
```

এখানে আমরা `name` কি এর ভ্যালু পরিবর্তন করে দিলাম । ইনডেক্সেড এ্যারের ক্ষেত্রেও ঠিক একইভাবে আমরা ভ্যালু পরিবর্তন করি তাদের নিউমেরিক ইনডেক্স ব্যবহার করে:

```
<?php
$array = array(100, 233, 456);
$array[1] = 21;
```

এখানে আমরা ২য় আইটেমটির ভ্যালু পরিবর্তন করে দিলাম ।

এ্যারে থেকে আইটেম রিমুভ করা

আমরা `unset` ফাংশনটি ব্যবহার করে ভ্যারিয়েবল রিমুভ করে থাকি । এটা এ্যারের উপরও একইভাবে কাজ করে কেননা এ্যারেও মূলত ভ্যারিয়েবল এরই কালেকশন । এ্যারে থেকে একটা আইটেম রিমুভ করতে আমরা তার কি সহ এই ফাংশনটি কল করি:

```
<?php

unset($array[3]);
```

আমরা সম্পূর্ণ এ্যারে ধরে ডিলিট করে দিতে চাইলে সরাসরি ঐ এ্যারেটি এই ফাংশনে পাস করে দিবো -

```
<?php

unset($array);
```

খেয়াল রাখতে হবে, `unset` শুধু ঐ কি এবং তার ভ্যালুই রিমুভ করবে । কিন্তু এ্যারে টা রি-ইনডেক্স করবে না । মানে আপনি যদি ৩য় আইটেমটি মুছে ফেলেন, তাহলেও ৪র্থ আইটেমটির ইনডেক্স `3` ই থাকবে, এক কমে `2` হয়ে যাবে না । অর্থাৎ ৪র্থ আইটেমটি ৩য় আইটেমের স্থানে সরে আসবে না । আমাদের যদি একটা আইটেম রিমুভ করার পর এই ভ্যালুগুলো পুনরায় ইনডেক্স করার প্রয়োজন হয় তবে আমরা `array_values` ফাংশন ব্যবহার করতে পারি ।

এ্যারে সংশ্লিষ্ট বেশ কিছু প্রয়োজনীয় ফাংশন দেখবো আমরা পরবর্তী চ্যাপ্টারে ।

কমন এ্যারে ফাংশনস

পিএইচপিতে এ্যারে নিয়ে কাজ করার জন্য প্রচুর ফাংশন রয়েছে । এগুলোর পূর্নাঙ্গ তালিকা পাওয়া যাবে ম্যানুয়ালে - Array Functions সেকশনে । এই চ্যাপ্টারে আমরা বেশি প্রচলিত কিছু এ্যারে সংশ্লিষ্ট ফাংশন সম্পর্কে জানবো । পরবর্তীতে বাকি এ্যারে ফাংশনগুলোও কাভার করা হবে এখানে ।

এই চ্যাপ্টারটির বেশীরভাগ কন্টেন্ট, বিশেষ করে উদাহরণগুলো পিএইচপি ম্যানুয়াল থেকে নেওয়া । এখানে বাংলায় ব্যখ্যা করা হয়েছে ফাংশনগুলো । নবীনদের জন্য যতটুকু প্রয়োজন ঠিক ততটুকু রাখা হয়েছে । এ্যাডভান্সড কিছু জিনিস ইচ্ছাকৃতভাবেই সংযোজন করা হয়নি ।

উদাহরণগুলোর আউটপুট ইচ্ছাকৃতভাবেই বইতে দেখানো হয়নি । কোড নিজে থেকে রান করে আউটপুট দেখে বোঝার চেষ্টা করার সুযোগ দেওয়ার জন্যই এমনটি করা হয়েছে ।

count()

একটি এ্যারেতে কতগুলি এলিমেন্ট আছে তা জানতে আমরা এই ফাংশনটি ব্যবহার করে থাকি ।

```php
<?php

$array = array(1,2,3);
echo count($array);
```

এই ফাংশনটি সেকেন্ড প্যারামিটার হিসেবে `COUNT_NORMAL` কিংবা `COUNT_RECURSIVE` কন্সট্যান্ট এ্যাক্সেপ্ট করে । এই প্যারামিটারটি অপশনাল । ডিফল্ট ভ্যালু হিসেবে `COUNT_NORMAL` থাকে । এই মোডে সে শুধু প্রদত্ত এ্যারের কতগুলো আইটেম আছে সেটা হিসেব করে । যখন আমরা `COUNT_RECURSIVE` ব্যবহার করি তখন এই ফাংশনটি মাল্টি ডাইমেনশনাল এ্যারের ক্ষেত্রে সব গুলো এ্যারের এলিমেন্ট হিসেব করে । অর্থাৎ মূল এ্যারের মধ্যে অন্য কোন এ্যারে থাকলে সেগুলোর এলিমেন্টও গননায় রাখা হবে ।

```php
<?php
$food = array('fruits' => array('orange', 'banana', 'apple'),
              'veggie' => array('carrot', 'collard', 'pea'));

// রিকার্সিভ মোড
echo count($food, COUNT_RECURSIVE); // মোট ৮টি এলিমেন্ট

// নরমাল মোড
echo count($food); // ২টি আইটেম
```

array_key_exists

আমাদের এ্যারেতে নির্দিষ্ট নামের কোন কি আছে কিনা তা জানার জন্য আমরা এই ফাংশনটি ব্যবহার করি । ঐ নামের ফাংশন থাকলে আমরা বুলিয়ান `true` পাই, আর না থাকলে `false` ।

```
<?php
$search_array = array('first' => 1, 'second' => 4);
if (array_key_exists('first', $search_array)) {
    echo "The 'first' element is in the array";
}
```

array_keys

নাম শুনেই বোঝা যাচ্ছে কোন এ্যারে এর কি গুলো পাবো আমরা এই ফাংশন থেকে । এই ফাংশনটির ২য় প্যারামিটার হিসেবে আমরা একটা ওয়ার্ড দিয়ে দিতে পারি, সেক্ষেত্রে শুধুমাত্র যে সকল কি এর মধ্যে ঐ ওয়ার্ডটি থাকবে সেগুলোই রিটার্ন করবে । ৩য় প্যারামিটারটি হচ্ছে `===` কম্প্যারিজনের জন্য । অর্থাৎ, এটার ভ্যালু ট্রু হলে আগের দেওয়া ওয়ার্ডটি সার্চ করার সময় টাইপ বিবেচনা করা হবে । ২য় এবং ৩য় প্যারামিটার অপশনাল ।

উদাহরণ:

```
<?php
$array = array(0 => 100, "color" => "red");
print_r(array_keys($array));

$array = array("blue", "red", "green", "blue", "blue");
print_r(array_keys($array, "blue"));
```

array_values

আগের ফাংশনটির সাথে মিল রেখেই এটি এ্যারের ভ্যালুগুলো রিটার্ন করে । যেমন:

```
<?php
$array = array("size" => "XL", "color" => "gold");
print_r(array_values($array));
```

in_array

এই ফাংশনটি একটি এ্যারেতে একটি নির্দিষ্ট ভ্যালু আছে কিনা তা জানায় । ৩য় প্যারামিটারটি অপশনাল । বুলিয়ান `true` পাস করলে সার্চ করার সময় টাইপও ম্যাচ করে ।

```php
<?php
$os = array("Mac", "NT", "Irix", "Linux");
if (in_array("Irix", $os)) {
    echo "Got Irix";
}
if (in_array("mac", $os)) {
    echo "Got mac";
}
```

array_pop

এই ফাংশনটি প্রদত্ত এ্যারের শেষ আইটেমটি রিটার্ন করে । এবং একই সাথে ঐ আইটেমটি এ্যারে থেকে রিমুভ করে দেয় ।

উদাহরণ:

```php
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
```

array_push

আগের ফাংশনটির ঠিক উল্টো কাজ করে এই ফাংশনটি । এটির কাজ কোন এ্যারের শেষে এক বা একাধিক আইটেম যোগ করা । যেমন:

```php
<?php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
```

এই ফাংশনের প্রথম আর্গুমেন্টটি হবে একটি এ্যারে । এরপর আমরা যে এলিমেন্টগুলো যোগ করতে চাই সেগুলো যোগ করবো ।

array_shift

`array_pop` এ্যারের শেষ থেকে আইটেম বাদ দিতো, `array_shift` এর কাজ শুরু থেকে বাদ দেওয়া । এটি এ্যারের প্রথম আইটেমটি রিটার্ন করে এবং ঐ আইটেমটি এ্যারেটি থেকে রিমুভ করে দেয় ।

উদাহরণ:

```php
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_shift($stack);
print_r($stack);
```

array_unshift

নাম শুনেই বুঝতে পারার কথা এটা কি করে । `array_push` এর মত করেই এই ফাংশনটি এ্যারের শুরুতে আইটেম যোগ করে । উদাহরণ:

```php
<?php
$queue = array("orange", "banana");
array_unshift($queue, "apple", "raspberry");
print_r($queue);
```

array_flip

এই ফাংশনটি এ্যারের কি আর ভ্যালু ইন্টারচেইনজ করে দেয় । অর্থাৎ কি গুলো হয়ে যায় ভ্যালু আর ভ্যালুগুলো হয়ে যায় কি । যেমন:

```php
<?php
$trans = array("a" => 1, "b" => 1, "c" => 2);
$trans = array_flip($trans);
print_r($trans);
```

array_reverse

এই ফাংশনটি এ্যারের আইটেমগুলোর অর্ডার বা ক্রমিক উল্টো করে দেয় । অর্থাৎ প্রথম আইটেমটি শেষে আর শেষের আইটেমটি শুরুতে আসে । এবং অন্যান্য আইটেমগুলিও একইভাবে উল্টো অর্ডারে নিয়ে আসা হয় ।

আমরা যদি আইটেমের অর্ডার পরিবর্তন হলেও তার আগের কি এর ভ্যালু ঠিক রাখতে চাই তাহলে ২য় আর্গুমেন্টটির ভ্যালু বুলিয়ান `true` পাস করলেই হবে ।

```php
<?php
$input  = array("php", 4.0, array("green", "red"));
$reversed = array_reverse($input);
$preserved = array_reverse($input, true);

print_r($input);
print_r($reversed);
print_r($preserved);
```

array_merge

দুই বা ততোধিক এ্যারে মার্জ করে একটি নতুন এ্যারে তৈরি করে এই ফাংশনটি । স্ট্রিং কি এর ক্ষেত্রে যদি একই নামের কি থাকে তবে শেষের এ্যারের একই নামের কি এর ভ্যালু ব্যবহৃত হয় । নিউমেরিক ইনডেক্স এর ক্ষেত্রে আইটেমগুলো একটার পর একটা এ্যাড করে নেয় । কি নিয়ে কোন কনফ্লিক্ট হয় না ।

```
<?php
$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
$result = array_merge($array1, $array2);
print_r($result);
```

array_rand

কোন এ্যারে থেকে এক বা একাধিক র‍্যান্ডম আইটেম বেছে নেয় এই ফাংশনটি । ঐ নির্বাচিত আইটেমগুলোর কি রিটার্ন করে । প্রথম প্যারামিটার হিসেবে এ্যারেটি এ্যাক্সেপ্ট করে । ২য় আর্গুমেন্ট হিসেবে আমরা পাস করতে পারি কতগুলো আইটেম বেছে নিবে । ২য় প্যারামিটারটি অপশনাল ।

যখন একটি মাত্র এন্ট্রি বেছে নেয় তখন এই ফাংশনটি সরাসরি তার কি রিটার্ন করে । তবে একাধিক আইটেম এর বেলায় সে কি গুলো একটি এ্যারেতে রিটার্ন করে ।

```
<?php
$input = array("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$rand_keys = array_rand($input, 2);
echo $input[$rand_keys[0]] . "\n";
echo $input[$rand_keys[1]] . "\n";
```

array_search

একটি এ্যারের ভিতরে সার্চ করার জন্য এই ফাংশনটি ব্যবহৃত হয় । যদি এ্যারেতে ওয়ার্ডটি থাকে তবে ঐ আইটেম এর কি টা রিটার্ন করে । যদি ওয়ার্ডটি স্ট্রিং হয় তবে সার্চটি হবে কেইস সেনসিটিভ । অর্থাৎ আপার কেইস ও লোয়ার কেইস এর ভ্যারিয়েশন তখন ম্যাটার করবে । (Masnun আর masnun তখন এক হবে না) । ২য় প্যারামিটারটি অপশনাল । এটি স্ট্রিক্ট (`===`) কম্প্যারিজন এর জন্য । এটির ভ্যালু বুলিয়ান `true` পাস করলে সার্চ করার সময় টাইপও মিলিয়ে দেখা হবে ।

```
<?php
$array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');

$key = array_search('green', $array); // $key = 2;
$key = array_search('red', $array);   // $key = 1;
```

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

(অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর ভূমিকা, কিছু সাধারন ধারনা এবং কিছু টেকনিক্যাল কনসেপ্ট এখানে পরবর্তীতে যোগ করা হবে)

টপিক লিস্ট

ক্লাস এবং অবজেক্ট এর পার্থক্য

একটি বাড়ি তৈরি করতে গেলে যেমন আমরা শুরুতে একটি নকশা বা ব্লু প্রিন্ট তৈরি করে নেই, পিএইচপিতেও তেমনি কোন অবজেক্ট কেমন হবে তা ডিফাইন করে দেওয়া হয় ক্লাস এর মাধ্যমে । অর্থাৎ অবজেক্ট এর ব্লু প্রিন্ট হলো ক্লাস, ক্লাস থেকে তৈরি করা হয় অবজেক্ট । একই ক্লাস থেকে তৈরি করা অবজেক্টগুলোর প্রত্যেকটি হলো ঐ ক্লাসের অবজেক্ট এর একটি ইন্সট্যান্স ।

ক্লাস ডিফাইন করা

ক্লাস ডিফাইন করা খুবই সহজ, প্রথমে `class` কিওয়ার্ড, এরপর ক্লাসের নাম এরপর কোড ব্লকে থাকবে ক্লাস বডি । খুব সিম্পল একটি ক্লাসের উদাহরণ হতে পারে এরকম:

```php
<?php
class SimpleClass
{
}
```

এটা আসলে একটি ফাকা ক্লাস । এটি কোন কাজই করে না ।

অবজেক্ট তৈরি করা

কোন ক্লাস থেকে ঐ ক্লাসের অবজেক্ট তৈরি করার জন্য আমরা `new` কিওয়ার্ডটি ব্যবহার করে থাকি । যেমন:

```php
<?php
$instance = new SimpleClass();
```

এখানে `$instance` একটি অবজেক্ট যার ক্লাস হলো `SimpleClass` । যদি কোন ভ্যারিয়েবল এ স্ট্রিং টাইপের ডাটা থাকে তবে ঐ ভ্যারিয়েবল এর আগে `new` ব্যবহার করেও নতুন অবজেক্ট তৈরি করা সম্ভব । এক্ষেত্রে ঐ ভ্যারিয়েবল এর যে ভ্যালু সেই নামের ক্লাস থেকে পিএইচপি অবজেক্ট তৈরি করার চেষ্টা করবে ।

```php
<?php
$className = 'SimpleClass';
$instance = new $className(); // SimpleClass()
```

প্রোপার্টি

কোন ফিচার বা বৈশিষ্ট্য বোঝাতে আমরা প্রোপার্টি ব্যবহার করতে পারি । যেমন: একজন মানুষের উচ্চতা বোঝানোর জন্য আমরা `Person` ক্লাস এ `height` নামে একটি প্রোপার্টি তৈরি করতে পারি ।

প্রোপার্টি গুলোকে সচরাচর ফিল্ড বা এ্যাট্রিবিউট নামেও ডাকা হয় । প্রোপার্টি ডিফাইন করা খুবই সহজ, প্রথমে ভিজিবিলিটি কিওয়ার্ড (`public` , `protected` কিংবা `private`) এর যে কোন একটি লিখতে হবে এবং তারপর আমরা যেভাবে ভ্যারিয়েবল ডিক্লেয়ার করি সেভাবেই আমাদের প্রোপার্টি ডিফাইন করবো । ভিজিবিলিটি নিয়ে আমরা পরবর্তীতে কোন চ্যাপ্টারে আলোকপাত করবো । আসুন আমরা দেখে নেই প্রোপার্টি কিভাবে ব্যবহার করা যায়:

```php
<?php

class Person
{
    public $age;
}


$person = new Person();
$person->age = 32;

$anotherPerson = new Person();
$anotherPerson->age = 45;

var_dump($person->age);
var_dump($anotherPerson->age);
```

এখানে আমরা `age` নামে একটি প্রোপার্টি ডিফাইন করলাম । পরবর্তীতে ঐ ক্লাসের দুটো ইনস্ট্যান্স তৈরি করে নিলাম এবং তাদের বয়স সেট করে দিলাম । লক্ষ্য করুন, কোন অবজেক্ট ইনস্ট্যান্স থেকে তার প্রোপার্টি এ্যাক্সেস করার জন্য আমরা `->` সিম্বলটি ব্যবহার করছি । এবং যখন প্রোপার্টি এ্যাক্সেস করছি তখন প্রোপার্টির নামের আগে ভ্যারিয়েবল সাইন নেই । অর্থাৎ, `$person->$age` নয়, বরং `$person->age` এর মাধ্যমে আমরা `age` প্রোপার্টি এ্যাক্সেস করতে পারি ।

এই অপারেটর (`->`) টি অবজেক্ট অপারেটর নামে পরিচিত।

যদি আমরা প্রোপার্টির নামের আগে ভ্যারিয়েবল সাইন ব্যবহার করে এ্যাক্সেস করি তখন সেটি ভ্যারিয়েবল ভ্যারিয়েবল এর মত করে কাজ করবে । প্রথমে `$age` এর ভ্যালু বের করে নিয়ে এরপর `$person->(value of $age)` এভাবে কল করা হবে । এভাবে আমরা একটি অবজেক্ট ইন্সট্যান্স থেকে ডাইনামিক্যালি তার প্রোপার্টি এ্যাক্সেস করতে পারি ।

আমরা চাইলে প্রোপার্টির একটি ইনিশিয়াল ভ্যালুও দিয়ে দিতে পারি । তবে এই ইনিশিয়াল ভ্যালু অবশ্যই কন্সট্যান্সট এক্সপ্রেশন হতে হবে (অর্থাৎ কোন ভ্যারিয়েবল বা ফাংশন ব্যবহার করা চলবে না) । যে কোন ফিক্সড ভ্যালু (যেমন: স্ট্রিং বা ইন্টিজার) কিংবা কোন কনস্ট্যান্ট ব্যবহার করা যেতে পারে ।

```php
<?php
class Person
{
    public $name = "masnun";
}
```

এটাকে প্রোপার্টি ইনিশিয়ালাইজেশন বলা হয় ।

মেথড

কোন কাজ করার জন্য আমরা মেথড ব্যবহার করি । মেথড আসলে ফাংশন যেটা ক্লাসের ভিতরে থাকে এবং ঐ ক্লাসের সকল প্রোপার্টি এবং মেথড এ্যাক্সেস করতে পারে ।

মেথড এর একটা উদাহরন দেখি:

```php
<?php

class Person
{
    public $age;

    public function getAge()
    {
        return $this->age;
    }
}


$person = new Person();
$person->age = 32;

$anotherPerson = new Person();
$anotherPerson->age = 45;

var_dump($person->getAge());
var_dump($anotherPerson->getAge());
```

এখানে আমরা `getAge()` নামে একটি মেথড ডিফাইন করেছি যেটার কাজই হচ্ছে ঐ অবজেক্ট ইন্সট্যান্স এর `age` প্রোপার্টির ভ্যালু রিটার্ন করা ।

আমরা দেখলাম `$this` এই ভ্যারিয়েবলটির মাধ্যমে আমরা ঐ অবজেক্ট ইনস্ট্যান্সটি এ্যাক্সেস করেছি । এটি সম্পর্কে আমরা আরো বিস্তারিত জানবো "স্ট্যাটিক ও নন-স্ট্যাটিক কনটেক্সট" সেকশনে । আপাতত আমাদের মনে রাখতে হবে `$this` ভ্যারিয়েবলটি যে ক্লাসে ব্যবহার করা হয়, এটি তার প্রত্যেকটি ইনস্ট্যান্সে নিজ নিজ ইনস্ট্যান্স কে পয়েন্ট করে ।

নন স্ট্যাটিক কনটেক্সট

আমরা আগের সেকশনে প্রোপার্টি দেখার সময় দেখেছি `$this` এর ব্যবহার । আমরা জেনেছি কোন ক্লাসের ভিতর যদি এই ভ্যারিয়েবলটি ব্যবহার করা হয় তাহলে ঐ ক্লাসের যতগুলো ইনস্ট্যান্স তৈরি করবো আমরা প্রত্যেকটির ভিতরে `$this` কিওয়ার্ড ঐ অবজেক্ট এ পয়েন্ট করবে ।

আগের উদাহরনটিই আরেকবার দেখে নেই:

```
<?php

class Person
{
    public $age;

    public function getAge()
    {
        return $this->age;
    }
}


$person = new Person();
$person->age = 32;

$anotherPerson = new Person();
$anotherPerson->age = 45;

var_dump($person->getAge());
var_dump($anotherPerson->getAge());
```

এখানে দেখুন, আমরা যখন `$person->getAge()` কল করছি তখন আমরা `$person` এর `age` প্রোপার্টি পাচ্ছি, আবার যখন `$anotherPerson->getAge()` কল করছি তখন পাচ্ছি `$anotherPerson` এর বয়স । অর্থাৎ, একই `$this` ভ্যারিয়েবলটি `$person` অবজেক্টের ভিতর `$person` কে এবং `$anotherPerson` ভিতরে `$anotherPerson` কে নির্দেশ করছে ।

এর ফলে, একটি ক্লাস থেকে তৈরি করা সব ইনস্ট্যান্সই তার নিজের প্রোপার্টি বা মেথড এ্যাক্সেস করতে পারে এই `$this` ভ্যারিয়েবল এর মাধ্যমে । তাই আলাদা আলাদা ইনস্ট্যান্সে একই প্রোপার্টির ভিন্ন ভিন্ন ভ্যালু থাকলেও আমরা এই ভ্যারিয়েবলটির মাধ্যমে ঐ ইনস্ট্যান্সের ভ্যালুটি জেনে নিতে পারছি খুব সহজেই!

এই যে একই ক্লাস থেকে তৈরি করা অবজেক্ট ইনস্ট্যান্স গুলোর ভ্যালু আলাদা আলাদা হতে পারে এটাই হলো নন-স্ট্যাটিক কনটেক্সট । এই কনটেক্সট এ কোন প্রোপার্টি বা মেথড শুধু ঐ ইনস্ট্যান্স স্পেসিফিক হয় ।

স্ট্যাটিক কনটেক্সট

কখনো কখনো কিছু প্রোপার্টি বা মেথড আমাদের সব ইনস্ট্যান্সের জন্যই কমন হয় । এই প্রোপার্টি গুলো বা মেথড গুলো আলাদা আলাদা ইনস্ট্যান্স এর জন্য আলাদা হওয়ার দরকার নেই, বরং ঐ ক্লাসের সবার জন্যই একই । এই মেথড বা প্রোপার্টি তাই সবাই এক সাথে শেয়ার করতে পারে । যেমন ধরুন, আমি চাই একটি `$count` প্রোপার্টি যেটির ভ্যালু সব অবজেক্ট ইনস্ট্যান্স শেয়ার করুক । অর্থাৎ যে কোন ইনস্ট্যান্স থেকেই আমি এই প্রোপার্টির ভ্যালু একই পাই । এক্ষেত্রে আমাকে এই প্রোপার্টি-টিকে স্ট্যাটিক হিসেবে ডিক্লেয়ার করতে হবে । তখন আমার ঐ ক্লাস থেকেই আমি সরাসরি এটি এ্যাক্সেস করতে পারবো, আমার অবজেক্ট ইনস্ট্যান্স তৈরি না করলেও চলবে । এটাই হচ্ছে স্ট্যাটিক কনটেক্সট । একটি উদাহরন দেখলে আরো ভালো বোঝা যাবে --

```php
<?php

class Person
{
    public static $count;

    public static function getCount()
    {
        return self::$count;
    }
}


Person::$count = 34;
var_dump(Person::$count);

$person = new Person();
var_dump($person->getCount());

Person::$count = 23;
var_dump(Person::$count);

var_dump($person->getCount());


$anotherPerson = new Person();
var_dump($anotherPerson->getCount());
```

উদাহরনটি একটু জটিল, তাই কয়েকবার ভালো করে পড়ুন । কোড রান করে আউটপুট ভালো করে মিলিয়ে নিন ।

দেখুন, এখানে `Person` ক্লাসে `$count` একটি স্ট্যাটিক প্রোপার্টি এবং `getCount()` একটি স্ট্যাটিক মেথড । এখান থেকে লক্ষ্যনীয়:

- স্ট্যাটিক মেথড বা প্রোপার্টি ডিফাইন করতে আমরা `static` কিওয়ার্ডটি ব্যবহার করি ।
- `$this` এর মত `self` এর মাধ্যমে আমরা স্ট্যাটিক কনটেক্সট এ প্রোপার্টি বা মেথড এ্যাক্সেস করি ।
- নন স্ট্যাটিক কনটেক্সট এ `->` ব্যবহার করা হয় এ্যাক্সেস করার জন্য । স্ট্যাটিক কনটেক্সট এ `::` ।
- স্ট্যাটিক কনটেক্সট এ প্রোপার্টির নামের আগে ভ্যারিয়েবল সাইন থাকে । নন-স্ট্যাটিক কনটেক্সট এ থাকে না । স্ট্যাটিক কনটেক্সট এ তাই ভ্যারিয়েবল ভ্যারিয়েবল এর মত করে এ্যাক্সেস করতে চাইলে আরেকটি ভ্যারিয়েবল সাইন যোগ করতে হয় ।

- স্ট্যাটিক মেথড কিংবা প্রোপার্টি কোন ইনস্ট্যান্স তৈরি না করেই সরাসরি ক্লাস এর নাম দিয়েই এ্যাক্সেস করা যায় ।
- স্ট্যাটিক প্রোপার্টি বা মেথড ঐ ক্লাসের সব ইনস্ট্যান্সই এ্যাক্সেস করতে পারে । এর ভ্যালু সব ইনস্ট্যান্সেই একই থাকে । এটা নন-স্ট্যাটিক কনটেক্সট এ (যেমন ইনস্ট্যান্স এর ভিতর থেকে) পরিবর্তন করা যায় না ।

এই সিম্বল টি (`::`) স্কোপ রেসুলেশন অপারেটর নামে পরিচিত।

নন-স্ট্যাটিক কনটেক্সট থেকে স্ট্যাটিক কনটেক্সট এ্যাক্সেস করা যায় কারন স্ট্যাটিক কনটেক্সট সবার জন্য একই । কিন্তু স্বাভাবিকভাবেই এর উল্টোটা করা সম্ভব হয় না ।

`$this` এবং `self`

এতক্ষনে আমরা বুঝে ফেলেছি এ দুটোর পার্থক্য । তবু বলি - `$this` নির্দেশ করে অবজেক্ট ইনস্ট্যান্স কে, `self` নির্দেশ করে ঐ ক্লাস কে ।

ইনহেরিট্যান্স

আমরা যেমন আমাদের বাবা-মার গুনাবলী বংশানুক্রমিকভাবে পাই, তেমনি ভাবে পিএইচপিতে ও একটি ক্লাস অন্য আরেকটি ক্লাস কে এক্সটেন্ড করে তার সব প্রোপার্টি বা মেথড ব্যবহার করতে পারে । এটাই ইনহেরিট্যান্স । একটি সহজ উদাহরন দেখি:

```
<?php

class ParentClass
{
    public $name;

    public function getName()
    {
        return $this->name;
    }
}


class ChildClass extends ParentClass
{

}

$child = new ChildClass();
$child->name = "Abul";

var_dump($child->getName());
```

এখানে লক্ষ্য করুন `ChildClass` টি `ParentClass` কে এক্সটেন্ড করেছে । এর ফলে `ChildClass` এ আমরা `name` বা `getName()` ডিফাইন না করলেও `ParentClass` থেকে সে এই প্রোপার্টি এবং মেথড এ্যাক্সেস করতে পারছে । এটাই সহজ ভাষায় ইনহেরিট্যান্স । এক্ষেত্রে আমরা বলতে পারি, `ChildClass` টি `ParentClass` কে ইনহেরিট করেছে । এখানে আমরা `extends` কিওয়ার্ডটি ব্যবহার করে বলে দেই কোন ক্লাসটি এক্সটেন্ড করছে আর কোনটি থেকে এক্সটেন্ড করা হচ্ছে । যেই ক্লাস টি এক্সটেন্ড করে, সেটিকে চাইল্ড ক্লাস এবং যেটি থেকে এক্সটেন্ড করা হয় সেটিকে প্যারেন্ট ক্লাস বলি আমরা । একটি ক্লাস যখন আরেকটি ক্লাস কে এক্সটেন্ড করে তখন প্যারেন্ট ক্লাস এর সব প্রোপার্টি এবং মেথডই চাইল্ড ক্লাস না ডিফাইন করলেও এ্যাক্সেস করতে পারবে ।

এখানে `ChildClass` এর `name` এবং `getName()` যে `ParentClass` থেকেই এসেছে তা এই উদহরনটি থেকে আরও পরিস্কারভাবে বোঝা যাবে:

```php
<?php

class ParentClass
{
    public $name = "Name of The ParentClass";

    public function getName()
    {
        return $this->name;
    }
}


class ChildClass extends ParentClass
{

}

$child = new ChildClass();
var_dump($child->getName());
```

এখানে দেখুন, আমরা `$name` এর ভ্যালু `ParentClass` এ ইনিশিয়ালাইজ করেছি । `ChildClass` হুবহু সেই ভ্যালুই গ্রহন করেছে । সুতরাং কোন সন্দেহ নেই যে এটি ইনহেরিটেন্স এরই ফল!

ভিজিবিলিটি

আমরা যদি এর আগে অবজেক্ট ওরিয়েন্টেড পিএইচপি কোড দেখে থাকি তাহলে হয়তো `public` , `protected` এবং `private` কিওয়ার্ডগুলোর ব্যবহার দেখেছি । আজকে আমরা এগুলো কেন ব্যবহার করা হয় তা জানবো ।

কিওয়ার্ড গুলোর আভিধানিক অর্থ চিন্তা করলে কিছুটা পরিষ্কার হয়েই যায় । যেটা `public` সেটা সবার জন্যই উন্মুক্ত । যেটা `private` সেটা ব্যক্তিগত, অর্থাৎ শুধুই আমার জন্য । তাহলে `protected` টা কি হবে? যেটা `protected` সেটা শুধুই আমার এবং আমার উত্তরাধিকারীদের জন্য ।

কোন ক্লাসের যে মেথড ও প্রোপার্টিগুলো পাবলিক হয় সেগুলো অন্য যে কোন জায়গা থেকেই এ্যাক্সেস করা সম্ভব । প্রটেক্টেড হলে শুধু মাত্র ঐ ক্লাস এবং যে সব ক্লাস ঐ ক্লাস কে ইনহেরিট করে তারাই শুধু এ্যাক্সেস করতে পারবে । প্রাইভেট হলে শুধু মাত্র ঐ ক্লাসের ভিতর থেকেই এটা এ্যাক্সেস করা যাবে, বাইরের কেউ বা কোন চাইল্ড ক্লাসও এটার এ্যাক্সেস পাবে না ।

আমরা পিএইচপি ম্যানুয়ালের এই উদাহরন টা দেখি:

```php
<?php
/**
 * Define MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private
```

এখানে দেখুন, `$obj->public` টা আমরা ক্লাসের বাইরে থেকেও অবজেক্টের প্রোপার্টি হিসাবে ব্যবহার করতে পারছি কিন্তু বাকি দুটো এ্যাক্সেস করতে গেলে আমরা ফ্যাটাল এরর পাবো । অন্যদিকে `$obj->printHello()` যেহেতু ঐ ক্লাসের ভিতরেই ডিফাইন করা, তাই ক্লাসের ভিতরে আমরা প্রাইভেট এবং প্রটেক্টেড প্রোপার্টিরও এ্যাক্সেস পাচ্ছি ।

এবার দেখি ইনহেরিট্যান্সের বেলায় কি হয় । আমরা এখন আগের ক্লাস টাকে এক্সটেন্ড করে আরেকটা ক্লাস বানাবো:

```php
class MyClass2 extends MyClass
{

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
$obj2->printHello();
```

এখানে আমরা দেখছি আমাদের চাইল্ড ক্লাসে আমরা প্যারেন্ট এর প্রটেক্টেড প্রোপার্টি এ্যাক্সেস করতে পারলেও প্রাইভেট প্রোপার্টি আনডিফাইনড থেকে যাচ্ছে ।

কুইক নোটস

- `var` কিওয়ার্ড ব্যবহার করে প্রোপার্টি ডিফাইন করলে তা সবসময়ই পাবলিক হবে ।
- ফাংশনের ক্ষেত্রে ভিজিবিলিটি কিওয়ার্ড মিস করলে সেটা পাবলিক হিসেবে গন্য হবে ।
- একই ক্লাস থেকে তৈরি করা অবজেক্ট গুলো একে অপরের প্রাইভেট এবং প্রটেক্টেড মেম্বার গুলো এ্যাক্সেস করতে পারবে । কারন তারা যেহেতু একই ক্লাস থেকে তৈরি হয়েছে তাই তারা জানে ইন্সট্যান্সগুলোর প্রোপার্টি ও মেথড কিভাবে ডিফাইন করা হয়েছে ।

কন্সট্রাক্টরস

যে কোন ক্লাসে আমরা একটি বিশেষ মেথড ডিফাইন করে দিতে পারি । পিএইচপি যখন একটি ক্লাস থেকে অবজেক্ট ইন্সট্যান্স তৈরি করবে তখন নতুন তৈরি হওয়া অবজেক্টটির এই বিশেষ মেথডটি কল করবে । প্রত্যেকটি ইন্সট্যান্স তৈরি হওয়র পরপরই পিএইচপি এই মেথডটি কল করে বিধায় অবজেক্টের নানা বিধ ইনিশিয়ালাইজেশনের কাজ এই মেথডে করা সম্ভব । এই মেথডটি অবজেক্ট তৈরি করার সময় গুরুত্বপূর্ন ভূমিকা পালন করে বিধায় এটাকে কন্সট্রাক্টর ফাংশন বা মেথড বলে ।

আমরা একটি উদাহরন দেখি:

```
<?php
class TestClass {
   function __construct() {
       print "From the constructor\n";
   }
}

$bc = new TestClass();
```

কন্সট্রাক্টরস ও ইনহেরিট্যান্স

চাইল্ড ক্লাস গুলোতে যদি আমরা নিজেদের কনস্ট্রাক্টর ডিফাইন করি তাহলে আর প্যারেন্ট এর কন্সট্রাক্টর অটোমেটিক্যালি কল হয় না । আমাদের কে এক্সপ্লিসিটলি প্যারেন্ট এর কন্সট্রাক্টর কল করার প্রয়োজন হয় ।

```
<?php
class TestClass {
   function __construct() {
       print "From the constructor\n";
   }
}

class SubClass extends TestClass {
   function __construct() {
       parent::__construct();
       print "In SubClass constructor\n";
   }
}

$test = new SubClass();
```

এখানে `parent::__construct();` এর মাধ্যমে আমরা প্যারেন্ট এর কন্সট্রাক্টর কল করলাম ।

ডেস্ট্রাক্টরস

একটা অবজেক্ট এর কাজ যখন শেষ হয়ে যায়, যখন আর কোন রেফারেন্স থাকে না ঐ অবজেক্ট এর তখন ঐ অবজেক্ট এর ডেস্ট্রাক্টর মেথডটি কল করা হয় ।

```php
<?php
class MyDestructableClass {
   function __construct() {
       print "In constructor\n";
       $this->name = "MyDestructableClass";
   }

   function __destruct() {
       print "Destroying " . $this->name . "\n";
   }
}

$obj = new MyDestructableClass();
```

সাধারনত অবজেক্ট এ ব্যবহৃত গুরুত্বপূর্ন রিসোর্স ডি-এ্যালোকেট করার জন্য ডেস্ট্রাক্টর মেথড বেশ কাজে দেয় । কনস্ট্রাক্টর এর মত ডেস্ট্রাক্টরের বেলায় প্যারেন্ট এর ডেস্ট্রাক্টর এক্সপ্লিসিটলি কল করতে হয় ।

ইন্টারফেইস

ইন্টারফেইসের মাধ্যমে আমরা বলে দেই একটা ক্লাসের কোন কোন মেথড অবশ্যই থাকা লাগবে । কিন্তু আমরা এর মূল ইম্প্লিমেন্টেশনটা নির্দিষ্ট করে দেই না ।

আমরা একটা ইন্টারফেইস ডিফাইন করি `interface` কিওয়ার্ডটি দিয়ে । এবং যে সব ক্লাস এই ইন্টারফেইস মেনে চলে তারা এই ইন্টারফেইসকে `implement` করে ।

ইন্টারফেইস ডিফাইন করা হয় সাধারনভাবে ক্লাসের মত করেই । ইন্টারফেইসের মেথডগুলোর শুধু সিগনেচার (কি কি প্যারামিটার নেয়) ডিফাইন করে দেওয়া হয় কিন্তু এই মেথডগুলোর বডি ডিফাইন করা হয় না ।

পিএইচপি ম্যানুয়াল থেকে একটি উদাহরন দেখে নেই:

```php
<?php

// Declare the interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implement the interface
// This will work
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}
```

এখানে আমরা `iTemplate` নামে একটি ইন্টারফেইস ডিফাইন করেছি । আমাদের `Template` ক্লাসটি এই ইন্টারফেইস ইম্প্লিমেন্ট করে । খেয়াল করুন আমরা দেখছি কিভাবে ইন্টারফেইসে শুধু মেথড সিগনেচার এবং আমাদের মূল ক্লাসে তার ইম্প্লিমেন্টেশন তৈরি করা হয়েছে ।

আমরা যদি ইম্প্লিমেন্টেশন ক্লাসে ইন্টারফেইসের কোন মেথড ডিফাইন করতে ভুলে যাই সেক্ষেত্রে আমরা ফ্যাটাল এরর পাবো ।

পিএইচপিতে একটি ক্লাস অনেকগুলো ইন্টারফেইস ইম্প্লিমেন্ট করতে পারে তবে যদি দুইটি ইন্টারফেইসের একই নামের মেথড থাকে তাহলে সঙ্গত কারনেই কোন ক্লাস এই দুটি ইন্টারফেইস একই সাথে ইম্প্লিমেন্ট করতে পারবে না । সেটা করলে ঐ মেথডের নাম নিয়ে কনফ্লিক্ট তৈরি হবে ।

ইন্টারফেইস হলো অনেকটা ডেভেলপারের সাথে চুক্তি করার মতো । আমাদের চুক্তি মেনে নিতে হলে তাকে অবশ্যই আমাদের বলে দেওয়া মেথড ইম্প্লিমেন্ট করতে হবে । যখন কোন ক্লাস আমাদের ডিফাইন করা ইন্টারফেইস ইম্প্লিমেন্ট করে তখন আমরা ধরে নিতে পারি আমাদের বলে দেওয়া মেথডগুলো ঐ ক্লাসে আছে । এর ফলে আমরা ক্লাসের প্রয়োজনীয় ডিজাইন সম্পর্কে নিশ্চিত হতে পারি ।

এ্যাবস্ট্রাকশন

কিছু ক্লাসকে আমরা এ্যাবস্ট্রাক্ট হিসেবে ডিক্লেয়ার করতে পারি । এসব ক্লাস থেকে সরাসরি অবজেক্ট ইনস্ট্যান্স তৈরি করা সম্ভব হয় না । কিন্তু এদের কে ইনহেরিট করা সম্ভব । কোন ক্লাসের একটি মেথড এ্যাবস্ট্রাক্ট হলে সেটিকে এ্যাবস্ট্রাক্ট ক্লাস হিসেবে ডিক্লেয়ার করতে হবে ।

এ্যাবস্ট্রাক্ট মেথড গুলোর বেলায় শুধুই মেথড সিগনেচার ডিফাইন করে দিতে হয় । মূল ইম্প্লিমেন্টেশন দেওয়া হয় না । প্যারেন্ট ক্লাসে ডিফাইন করা সকল এ্যাবস্ট্রাক্ট মেথড অবশ্যই চাইল্ড ক্লাসে ইম্প্লিমেন্ট করতে হবে । এসময় ভিজিবিলিটি একই অথবা বেশী ওপেন (প্রাইভেট থাকলে প্রাইভেট কিংবা প্রটেক্টেড, প্রটেক্টেড থাকলে প্রটেক্টেড কিংবা পাবলিক) রাখা আবশ্যক । একই সাথে ফাংশনের সিগনেচারও ম্যাচ করতে হবে, আপনি চাইলেই চাইল্ড ক্লাসে কোন মেথডের একটি প্যারামিটার যোগ বা বাদ দিতে পারবেন না ।

এ্যাবস্ট্রাক্ট ক্লাস অনেকটা ইন্টারফেইসের মত শুধু এখানে শুধু মাত্র নির্দিষ্ট মেথড গুলো আমরা এ্যাবস্ট্রাক্ট রেখে বাকি মেথডগুলোর ইম্প্লিমেন্টেশন তৈরি করে দিতে পারি ।

```php
<?php
abstract class AbstractClass
{
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "\n";
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') ."\n";
?>
```

এখানে একই এ্যাবস্ট্রাক্ট ক্লাস থেকে আমরা দুটি ক্লাস তৈরি করেছি । এবং প্রত্যেকটি সাবক্লাসে আমরা এ্যাবস্ট্রাক্ট মেথডগুলো নিজেদের মত করে ইম্প্লিমেন্ট করেছি । কিন্তু `printOut()` মেথডটি মূল ক্লাসেই ডিফাইন করা ।

ট্রেইটস

আমরা দেখেছি ক্লাস ইনহেরিট্যান্সের মাধ্যমে আমরা প্যারেন্ট ক্লাস থেকে চাইল্ড ক্লাসে মেথড ইনহেরিট করতে পারি । অর্থাৎ প্যারেন্ট ক্লাসে কোন মেথড তৈরি করা থাকলে আমরা সেটা চাইল্ড ক্লাসে ব্যবহার করতে পারি । কিন্তু অনেক সময় দেখা যায় একই ক্লাস হায়ারার্কিতে নেই এমন দুটি ক্লাসের কিছু কমন মেথড থাকতে পারে । অর্থাৎ এমন দুটি ক্লাস যারা একটি আরেকটিকে এক্সটেন্ড করে না কিন্তু দুটি ক্লাসেরই কমন মেথড থাকছে `getName()` নামে যেটা একই কাজ করে । এসব ক্ষেত্রে কোড রিইউজে সহায়তা করতে পিএইচপি 5.4.0 ভার্সন থেকে ট্রেইটস এর প্রচলন ।

ট্রেইটস ডিফাইন করা হয় ক্লাসের মত করেই তবে `trait` কিওয়ার্ডটি ব্যবহার করে । একটি ট্রেইটের ভিতরে একই ধরনের কিছু ফাংশনালিটি সম্পন্ন মেথড ডিফাইন করা থাকে । এরপরে কোন ক্লাসের ভিতরে আমরা ঐ ট্রেইটটি ইনক্লুড করলে ঐ মেথডগুলো আমরা এ্যাক্সেস করতে পারি ঠিক যেন ঐ ক্লাসেই মেথডগুলো ডিফাইন করা হয়েছিলো ।

উদাহরন:

```php
<?php
trait CommonMethods {
    public function getName() {
        return $this->name;
    }
}

class TestClass {
    use CommonMethods;
    private $name = "test class";
}

class AnotherClass {
    use CommonMethods;
    private $name = "another class";
}

$tc = new TestClass();
var_dump($tc->getName());

$ac = new AnotherClass();
var_dump($ac->getName());
```

এখানে `TestClass` এবং `AnotherClass` সম্পূর্ন ইনডিপেন্ডেন্ট দুটি ক্লাস । তারা `CommonMethods` ট্রেইটটি ব্যবহার করে । ফলে এই ট্রেইটের মেথডটি তারা সরাসরি ব্যবহার করতে পারে ।

পিএইচপিতে যেহেতু মাল্টিপল ইনহেরিট্যান্স নেই, অর্থাৎ কোন ক্লাস শুধুমাত্র একটা ক্লাসকেই এক্সটেন্ড করতে পারে তাই এক্ষেত্রে কমন মেথডগুলো গ্রুপিং এর জন্য ট্রেইট বেশ কার্যকর ভূমিকা পালন করতে পারে ।

ম্যাজিক মেথড

পিএইচপির ক্লাসে কিছু মেথড থাকে যেগুলো দুইটি আন্ডারস্কোর দিয়ে শুরু হয়, এই মেথড গুলোকে সাধারণত ম্যাজিক মেথড বলা হয়। যদিও এই মেথডগুলো আগেথেকে ক্লাসে থাকে না, এই মেথড গুলো সাধারনত প্রোগ্রামাররাই লিখে থাকে। কিন্তু এই মেথড গুলো অন্যান্য মেথডের মত আচরন করে না।

`__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` এবং `__debugInfo()` এই মেথড গুলোকে ম্যাজিক মেথড বলা হয়ে থাকে।

construct(), destruct() এই মেথড গুলো সম্পর্কে পূর্বে আলোচনা করা হয়েছে।

গেট মেথড

এই মেথড এর মাধ্যমে ক্লাসের প্রোপার্টি এক্সেস করা যায়। নিচের উদাহরণটিতে গেট মেথড ইমপ্লিমেন্ট করে দেখানো হয়েছে।

```
public function __get($property)
{
  if (property_exists($this, $property)) {
    return $this->$property;
  }
}
```

ধরা যাক আমাদের ক্লাসের নাম Tweet যার একটি প্রোপার্টি (username) আছে । কেউ যদি username প্রোপার্টি টি এক্সেস করতে চায় তবে সে নিচের উদাহরন এর মত করে প্রোপার্টিটি এক্সেস করতে পারবে।

```
$tweet = new Tweet();
echo $tweet->username; // এটি username প্রোপার্টিটি রিটার্ন করবে, এমনকি প্রাইভেট প্রোপার্টি হলেও
```

সেট মেথড

এই মেথড এর মাধ্যমে ক্লাসের প্রোপার্টির ভ্যালু সেট করা যায়। নিচের উদাহরণটিতে সেট মেথড ইমপ্লিমেন্ট করে দেখানো হয়েছে।

```
public function __set($property, $value)
{
  if (property_exists($this, $property)) {
    $this->$property = $value;
  }
}
```

Tweet ক্লাসে username এর ভ্যালু সেট করতে চাইলে নিচের উদাহরন অনুসরন করলেই হবে।

```
$tweet = new Tweet();
$tweet->username = 'saaiful'; // এটি username প্রোপার্টির ভ্যালু সেট করবে।
```

ইজসেট মেথড

কোন প্রোপার্টি ক্লাসের মধ্যে আছে কিনা তা জানতে এই মেথড ব্যবহার করা হয়। এই মেথড ক্লাসের বাইরে থেকে এক্সেস করা যায় না।

```
public function  __isset($property)
{
  return isset($this->$property);
}
```

ক্লাসের যেকোন মেথড থেকে নিচের মত করে এই মেথডটি ব্যবহার করা যাবে

```
isset($tweet->username);
```

আনসেট মেথড

ক্লাসের কোন প্রোপার্টি সরিয়ে দিতে এই মেথড ব্যবহার করা হয়। এই মেথড ক্লাসের বাইরে থেকে এক্সেস করা যায় না।

```
public function __unset($property)
{
  unset($this->$property);
}
```

ক্লাসের যেকোন মেথড থেকে নিচের মত করে এই মেথডটি ব্যবহার করা যাবে

```
unset($tweet->username);
```

কল মেথড

যখন কোন মেথড ক্লাসের বাইরে থেকে এক্সেস করা যায় না অথবা যখন কল করা মেথডটি ক্লাসে থাকে না তখন এই মেথড কাজ শুরু করে।

```
public function __call($method, $parameters)
{
    var_dump($method);
    var_dump($parameters);
}
```

নিচের উদাহরনে post মেথড ব্যবহার করা হয়েছে , যদিও Tweet ক্লাসে এই মেথডটি নেই। কিন্তু আউটপুট লক্ষ করলে দেখা যাবে মেথডের নাম আর প্যারামিটারের var_dump করা হয়েছে। __call মেথডের মধ্যে প্রয়োজনীয় কোড লিখে এই মেথডের চমৎকার ব্যবহার করা যাবে।

```
$tweet = new Tweet();
$tweet->post("this is a test");
```

উদাহরনঃ

```php
<?php
class Tweet
{

    function __construct()
    {
        $this->username = "saaiful";
        $this->api = "https://api.twitter.com/1.1/";
        $this->param['user_timeline'] = "statuses/user_timeline.json";
        $this->param['home_timeline'] = "statuses/home_timeline.json";
        $this->param['retweets'] = "statuses/retweets";
    }

    public function fetch($url)
    {
        // send get request to $url
        var_dump($url);
    }

    public function __call($method, $parameters='')
    {
        if(array_key_exists($method, $this->param)){
            $url = $this->api . $this->param[$method];
            if(!empty($parameters)){
                $url .= "/".$parameters[0].".json";
            }
            return $this->fetch($url);
        }else{
            return false;
        }

    }
}

$tweet = new Tweet();
$tweet->retweets('abc');
$tweet->ppp('abc');
```

...চলমান

নেইমস্পেইস

আমাদের ক্লাস, ফাংশন বা কনস্ট্যান্ট নাম নিয়ে প্রায়শই সমস্যায় পড়তে হয় । দেখা যায় আমি যেই নাম ব্যবহার করেছি সেই নামে আরেকটি লাইব্রেরীতে একই নামের কিছু একটা রয়েছে । ফলাফল - নাম নিয়ে কনফ্লিক্ট । এই সমস্যা থেকে সমাধান দিতে পারে নেইমস্পেইস ।

নেইমস্পেইসের ধারনাটা খুবই সাধারন । আমরা যেমন আমাদের ফাইল পত্র গুলো নানা ফোল্ডারে সাজিয়ে রাখি, নেইমস্পেইসও এই ফোল্ডারগুলোর মত । আমাদের ক্লাস, ফাংশন, কনস্ট্যান্ট গুলো আমরা আলাদা আলাদা নেইমস্পেইসে সাজিয়ে রাখি । এতে এক নেইমস্পেইসের সাথে আরেক নেইমস্পেইসের জিনিসপত্রের নাম নিয়ে কোন কনফ্লিক্ট হয় না ।

এর আগে এই ধরনের নাম সংক্রান্ত জটিলতা এড়াতে ডেভেলপাররা আন্ডারস্কোর ব্যবহার করে নেইমস্পেস এর কাজ চালাতো । পুরোনো ফ্রেমওয়ার্কগুলোত এই ধরনের আন্ডারস্কোর বেইজড নেইমস্পেসিং এর প্রচেষ্টা দেখা যায় । পিএইপি ৫.৩ থেকে নেইমস্পেইস ল্যাঙ্গুয়েজ ফিচার হিসেবে যোগ করা হয় ।

নেইমস্পেইস তৈরি করা

নেইমস্পেইসের ভিতরে যে কোন ভ্যালিড পিএইচপি কোডই রাখা যায় । তবে নেইমস্পেইসের প্রকৃত ইফেক্ট পড়ে শুধুমাত্র ক্লাস, ইন্টারফেইস, কন্সট্যান্ট এবং ফাংশনের উপর । অর্থাৎ এগুলোকেই শুধু নেইমস্পেইসে আটকানো যায় ।

আমাদের নেইমস্পেইস ডিফাইন করতে প্রথমে `namespace` কিওয়ার্ড এবং তারপর নেইমস্পেইস এর নাম দিতে হয় । নেইমস্পেইস ডিক্লেয়ার করা শুরু হতে হবে পিএইচপি ফাইলের একেবারে উপর থেকে অর্থাৎ অন্য যে কোন কোডের আগে । একমাত্র বিকল্প শুধু `declare` কিওয়ার্ডটি, এটিই শুধু নেইমস্পেস ডিক্লেয়ারেশনের আগে আসতে পারে । একই ফাইলে একাধিক নেইমস্পেইস ডিক্লেয়ার করা সম্ভব । পরবর্তী নেইমস্পেইস এর আগ পর্যন্ত সব কোডই প্রথম নেইমস্পেইস এর অন্তর্গত ।

উদাহরণ:

```
<?php
namespace MyProject\SubNameSpace\AnotherLevel;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

?>
```

আমরা চাইলে নেইমস্পেইস এর পর কার্লি ব্রেইস (সেকেন্ড ব্রাকেট) ব্যবহার করেও নেইমস্পেইসগুলোকে আলাদা করতে পারি । নেইমস্পেইসের নাম দেওয়া না হলে সেটি গ্লোবাল নেইমস্পেইস হিসেবে বিবেচ্য হয় । অর্থাৎ নামহীন নেইমস্পেইসে আমরা যাই ডিফাইন করি তা গ্লোবাল নেইমস্পেইস থেকেই এ্যাক্সেস করা যায় ।

নেইমস্পেইস ব্যবহার করা

প্রথমেই নিশ্চিত হতে হবে আমাদের কোড যে নেইমস্পেইসে আছে তা বর্তমান ফাইল থেকে এ্যাক্সেস করা যায় কিনা । যেমন: যদি নেইমস্পেইসটি অন্য কোন ফাইলে হয় তবে অবশ্যই সেটি ইনক্লুড করে নিতে হবে । তবে বাস্তবে বেশীরভাগ ক্ষেত্রেই আমরা অটোলোডার ব্যবহার করে নেইমস্পেইস থেকে কোড ইম্পোর্ট করতে পারবো । সেক্ষেত্রে ম্যানুয়ালি ইনক্লুড করা লাগবে না ।

এরপর আমরা `use` কিওয়ার্ডটি ব্যবহার করে তারপর নেইমস্পেইস সহ পুরো নাম উল্লেখ করবো । উদাহরণ:

```
<?php
require 'db.php';

use MyProject\DB;
use MyProject\DB\Connection as DBC;

$x = new DBC();

?>
```

এই উদাহরনে আমরা দেখছি কিভাবে কোন নেইমস্পেইস থেকে আমরা ক্লাস ইম্পোর্ট করলাম । `as` কিওয়ার্ডটি ব্যবহার করে আমরা ইম্পোর্ট করার সময় প্রয়োজনমত নাম পরিবর্তন করে দিতে পারি ।

নেইমস্পেইস থেকে গ্লোবাল কোড এ্যাক্সেস করা

আমরা কোন নেইমস্পেইস থেকে যদি কোন ক্লাস বা ফাংশন এর পুরো নেইমস্পেইসড নাম ব্যবহার না করে শুধু নাম উল্লেখ করি তাহলে পিএইচপি ধরে নেয় ঐ ক্লাস বা ফাংশনও একই নেইমস্পেইসেরই অংশ । যেমন আমরা যদি `MyProject` নেইমস্পেইসে থেকে `strlen` ফাংশনটি কল করি তাহলে পিএইচপি গ্লোবাল `strlen()` ফাংশনটি ব্যবহার না করে `MyProject\strlen()` ফাংশনটি খুজঁবে । তাই কোন নেইমস্পেইসের ভিতর থেকে গ্লোবাল নেইমস্পেইসের ক্লাস, ফাংশন ইত্যাদি এ্যাক্সেস করার সময় নামের শুরুতে একটি `\` ব্যবহার করতে হয় । যেমন:

```
<?php
namespace Foo;
$a = \strlen('hi');
```

ফাইলসিস্টেম

এই চ্যাপ্টারে আমরা পিএইচপির ডিরেক্টরী এবং ফাইল সম্পর্কিত কিছু ব্যাসিক অপারেশন দেখবো । এখানে দেখানো পদ্ধতি ছাড়াও আরো নানা পদ্ধতিতে একই কাজ করা যেতে পারে । আপাতত আমরা একটি করে পদ্ধতি দেখবো । পরবর্তীতে আমরা আরো এ্যাডভান্সড ব্যবহার দেখবো ।

ডিরেক্টরী তৈরি করা

আমরা নতুন একটি ডিরেক্টরী বা ফোল্ডার তৈরি করতে পারি `mkdir` ফাংশনটি ব্যবহার করে ।

```
<?php
$success = mkdir("test_dir", 0755);
var_dump($success);
```

এখানে আমরা আমাদের কারেন্ট লোকেশনে `test_dir` নামে একটি ডিরেক্টরী তৈরি করেছি যেটার পার্মিশন লেভেল - `0755` । পার্মিশন সম্পর্কে আরো বিস্তারিত জানতে লিনাক্স ফাইল পার্মিশন এর উপর গুগলে সার্চ করতে পারেন ।

এই ফাংশনটি ৩য় আরেকটি বুলিয়ান আর্গুমেন্ট নেয় - যদি আমরা চাই নেস্টেড ডিরেক্টরী তৈরি করতে (যেমন: "my_dir/sub_dir/another_dir") তাহলে এই প্যারামিটার এর জন্য আমরা `true` পাস করবো । অন্যথায় পিএইচপি এরর থ্রো করবে ।

```
<?php
$success = mkdir("my_dir/sub_dir/another_dir", 0755, true);
var_dump($success);
```

ডিরেক্টরী ব্রাউজ করা

আমরা কোন ডিরেক্টরী এর কন্টেন্ট ব্রাউজ করার জন্য `scandir` ফাংশনটি ব্যবহার করতে পারি ।

```
<?php
$dir    = '/tmp';
$files1 = scandir($dir);
$files2 = scandir($dir, SCANDIR_SORT_NONE);

print_r($files1);
print_r($files2);
```

এই ফাংশনটির প্রথম প্যারামিটার ফাইল পাথ এ্যাক্সেপ্ট করে । যেই ডিরেক্টরীর কন্টেন্ট আমরা ব্রাউজ করতে চাই সেটির পাথ আমরা এই আর্গুমেন্ট হিসেবে পাস করবো । ২য় প্যারামিটারটি আমরা সর্টিং এর জন্য ব্যবহার করি । এটির ভ্যালু হিসেবে আমরা এই ৩টি কন্সট্যান্ট এর যে কোনটি পাস করতে পারি:

- `SCANDIR_SORT_ASCENDING`
- `SCANDIR_SORT_DESCENDING`
- `SCANDIR_SORT_NONE`

এই প্যারামিটারটির জন্য ডিফল্ট ভ্যালু হিসেবে `SCANDIR_SORT_ASCENDING` থাকে ।

ডিরেক্টরী ডিলিট করা

`rmdir` ফাংশনটি ব্যবহার করে আমরা কোন ডিরেক্টরী ডিলিট করতে পারি:

```
<?php
rmdir('examples');
```

তবে ডিরেক্টরী ডিলিট করার আগে নিশ্চিত হতে হবে যে এই ডিরেক্টরীটি ফাকা আছে কিনা । যদি এটির মধ্যে অন্য কোন ফাইল বা সাব ডিরেক্টরী থাকে তবে আগে সেগুলো ডিলিট করে নিতে হবে ।

ফাইল তৈরি করা / ফাইলে কন্টেন্ট যোগ করা

`file_put_contents` ফাংশনটি ব্যবহার করে আমরা একটি ফাইলের কন্টেন্ট ওভাররাইট করতে পারি । যদি ঐ ফাইলটি আগে থেকে তৈরি করা না থাকে তাহলে ফাংশনটি ফাইলটি তৈরি করে নেয় ।

```
<?php
$file = 'people.txt';
$text = "Abu Ashraf Masnun\n";
file_put_contents($file, $text);
```

ডিফল্টভাবে পিএইচপি ফাইলের কন্টেন্ট ওভাররাইট করে, অর্থাৎ আগের সব কিছু মুছে ফেলে নতুন করে কন্টেন্ট যোগ করে । আমরা যদি চাই আগের কন্টেন্টের সাথে অতিরিক্ত নতুন কন্টেন্ট যোগ করতে তাহলে আমরা ৩য় আর্গুমেন্ট হিসেবে `FILE_APPEND` কন্সট্যান্টটি পাস করবো । যেমন:

```
<?php
$file = 'people.txt';
$text = "The Doctor\n";
file_put_contents($file, $text, FILE_APPEND);
```

এবার যদি আমরা ফাইলটি খুলি তাহলে দেখবো আমাদের আগের কন্টেন্ট এর সাথে এই নতুন টেক্সট যোগ হয়েছে ।

ফাইল পড়া

`file_get_contents` ফাংশনটিতে কোন ফাইল পাথ পাস করলে এটি ঐ ফাইলের কন্টেন্ট রিটার্ন করবে ।

```php
<?php
$file = file_get_contents('./people.txt');
```

মজার একটা ব্যাপার হচ্ছে এই ফাংশনটি ওয়েব এ্যাড্রেসও সাপোর্ট করে, অর্থাৎ ওয়েবে থাকা কোন কন্টেন্টও আপনি পড়তে পারবেন এভাবে:

```php
<?php
$content = file_get_contents("http://masnun.com");
var_dump($content);
```

ফাইল ডিলিট করা

ফাইল ডিলিট করার জন্য আমরা `unlink` ফাংশনটি ব্যবহার করি ।

```php
<?php
unlink("./people.txt");
```

ডিজাইন প্যাটার্ন

সফটওয়্যার ইঞ্জিনিয়ারিং এ ডিজাইন প্যাটার্ন হল কোডের সাধারণ সমস্যার রি-ইউজিবিলিটি বাড়ানোর জন্য এক প্রকারের নিয়মনীতি অথবা টেম্পলেট। যাতে করে সফটওয়্যার একটা নির্দিষ্ট আর্কিটেকচার এ তৈরি করা যায় আর কোডের পুনুরাব্রিত্তি ঠেকান যায়।

ডিজাইন প্যাটার্ন সাধারণত নিম্নলিখিত ক্যাটাগরীর হয়ে থাকেঃ

1. Creational
2. Structural
3. Behavioural

আমরা বহুল ব্যাবহ্রিত গুলো নিয়ে আলোচনা করব।

টপিকস লিস্ট

সিঙ্গেলটোন ডিজাইন প্যাটার্নঃ

সিঙ্গেলটোন ডিজাইন প্যাটার্ন ক্রিয়েশনাল ডিজাইন প্যাটার্ন ক্যাটাগরির মধ্যে পরে। এই প্যাটার্নের মুল উদ্দেশ্য হল প্রতিটি ক্লাসের শুধু মাত্র একটিই ইন্সট্যান্স/অবজেক্ট থাকা।

ধরুন, **Singleton** নামে আমাদের একটা ফাইনাল ক্লাস আছে তাহলে সিঙ্গেলটোন প্যাটার্নে এই ক্লাসকে এমনভাবে ব্যবহার করতে হবে যেন নতুন কোন ইন্সট্যান্স/অবজেক্ট তৈরি না হয়ে একটিই থাকে আর ক্লাসটিকে ইনহেরিট ও করা না যায়, যা আমরা নিচের মত করে করতে পারিঃ

```php
<?php

final class Singleton
{
    private static $instance;

    public static function getInstance()
    {
        if (null === self::$instance) {
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct()
    {

    }

    private function __clone()
    {

    }

    private function __wakeup()
    {

    }

    public function sayHi()
    {
        echo 'Hi';
    }
}

$singleton = Singleton::getInstance();

$singleton->sayHi();
```

এখানে ক্লাসটি বাইরে থেকে ইন্সট্যান্সিয়েট না করে `getInstance()` স্ট্যাটিক মেথডটি ডিক্লেয়ার করা হয়েছে যাতে ক্লাসের ইন্সট্যান্সটা রিটার্ন করে।

অর্থাৎ,

```php
$singleton = new Singleton();
```

এর পরিবর্তে

```
$singleton = Singleton::getInstance();
```

ব্যবহার করা হয়েছে।

আর ক্লাসের ইন্সট্যান্স `$instance` নামে ভ্যারিয়েবল এ রাখা হয়েছে।

যেমনঃ

```
private static $instance;

public static function getInstance()
{
    if (null === self::$instance) {
        self::$instance = new self();
    }

    return self::$instance;
}
```

আবার ক্লাসের একাধিক ইন্সস্ট্যান্স তৈরিতে বাধা দিতে আমরা `__clone()` ও `__wakeup()` ম্যাজিক মেথডগুলি ব্যাবহার করেছি।

এই লিঙ্ক থেকে কোডটি পাবেন।

অবজার্ভার ডিজাইন প্যাটার্নঃ

অবজার্ভার ডিজাইন প্যাটার্ন বিহেভিওরাল টাইপের মধ্যে পরে। এটা **pub/sub** এর নিয়মে কাজ করে অর্থাৎ কোন অবজেক্ট কিংবা সাবজেক্ট এ পরিবর্তন হলে সেটা **Publisher** তৎক্ষন্যাত **Subscriber** দেরকে জানায় দিবে কিংবা নটিফাই করবে।

পিএইসপিতে অবজার্ভার প্যাটার্নটি প্রয়োগ করতে হলে যথাক্রমে `SplSubject` ও `SplObserver` ইন্টারফেইস ইমপ্লিমেন্ট করে সাবজেক্ট ও অবজার্ভার ২ টা ক্লাস লিখতে হয়। আর সাবস্ক্রাইব করা অবজার্ভারদেরকে স্টোর করে রাখার জন্য `SplObjectStorage` এই ক্লাসটিকে ব্যাবহার করা যেতে পারে।

উপরে উল্লেখিত **SplSubject, SplObserver, SplObjectStorage** হল পিএইসপির **Standard PHP Library (SPL)**

নিচে একটি `Model` নামক ক্লাস ও দুইটি অবজার্ভার ক্লাসের উদাহরণ দেয়া হলঃ

```
<?php

class Model implements SplSubject
{
    protected $observers;

    public function __construct()
    {
        $this->observers = new SplObjectStorage();
    }

    public function attach(SplObserver $observer)
    {
        $this->observers->attach($observer);
    }

    public function detach(SplObserver $observer)
    {
        $this->observers->detach($observer);
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function __set($name, $value)
    {
        $this->data[$name] = $value;
```

```
        // notify the observers, that model has been updated
        $this->notify();
    }
}

class ModelObserver implements SplObserver
{
    public function update(SplSubject $subject)
    {
        echo get_class($subject) . ' has been updated' . '<br>';
    }
}

class Observer2 implements SplObserver
{
    public function update(SplSubject $subject)
    {
        echo get_class($subject) . ' has been updated' . '<br>';
    }
}

// Instantiate the model class for 2 different objects
$model1 = new Model();
$model2 = new Model();

// Instantiate the observers
$modelObserver = new ModelObserver();
$observer2 = new Observer2();

// Attach the observers to $model1
$model1->attach($modelObserver);
$model1->attach($observer2);

// Attach the observers to $model2
$model2->attach($observer2);

// Changing the subject properties
$model1->title = 'Hello World';
$model2->body = 'Lorem ipsum.....';
```

উপরে `Model` ক্লাসটি হল সাবজেক্ট `ModelObserver` ও `Observer2` হল অবজার্ভার।

`Model` ক্লাসটি যেহেতু `SplSubject` ইন্টারফেইস ইমপ্লিমেন্ট করে লেখা হয়েছে কাজেই `attach()`, `detach()` ও `notify()` মেথডগুলা অবশ্যই থাকতে হবে।

অপরদিকে যেহেতু `ModelObserver` ও `Observer2` ক্লাসগুলা `SplObserver` ইন্টারফেইস ইমপ্লিমেন্ট করে লেখা হয়েছে সেহেতু `update()` মেথডটি ক্লাসগুলাতে থাকতে হবে।

এবার আপনারা যদি `SplSubject` ও `SplObserver` ইন্টারফেইস ব্যাবহার না করে অবজার্ভার ডিজাইন প্যাটার্ন এর প্রয়োগ করতে চান সেটাও করতে পারবেন শুধুমাত্র আপনার বিষয় বস্তু ঠিক থাকলেই হল।

নিচে একটা উদাহরণ দেয়া হলঃ

```php
<?php

class Model
{
    protected $observers;

    public function __construct()
    {
        $this->observers = new SplObjectStorage();
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function setObservers($observers = [])
    {
        foreach ($observers as $observer) {
            $this->observers->attach($observer);
        }
    }

    public function __set($name, $value)
    {
        $this->data[$name] = $value;
        // notify the observers, that model has been updated
        $this->notify();
    }
}

class Post extends Model
{
    public function insert($data)
    {
        // Store the data
        // Notify to observers
        $this->notify();
    }

    public function update($data)
    {
        // Update the model
        // Notify to observers
        $this->notify();
    }
```

```
    public function delete($id)
    {
        // Delete the model
        // Notify to observers
        $this->notify();
    }
}

class PostModelObserver
{
    public function update($subject)
    {
        echo get_class($subject) . ' has been updated' . '<br>';
    }
}

class Observer2
{
    public function update($subject)
    {
        echo get_class($subject) . ' has been updated' . '<br>';
    }
}

$post = new Post();

$post->setObservers([new PostModelObserver, new Observer2]);

$post->title = 'Hello World';
```

এই লিঙ্ক থেকে আরও ধারনা পেতে পারেন।

অ্যাডাপ্টার ডিজাইন প্যাটার্নঃ

সফটওয়ার ইঞ্জিনিয়ারিং এ আরেকটি বহুল প্রচলিত ডিজাইন প্যাটার্ন হল অ্যাডাপ্টার ডিজাইন প্যাটার্ন। এটি স্ট্রাকচারাল প্যাটার্নের মধ্যে পরে।

আমরা বাস্তব জীবনে সবাই অ্যাডাপ্টার শব্দটির সাথে পরিচিত। যেমনঃ মোবাইলের চার্জিং অ্যাডাপ্টার, কম্পিউটারের গ্রাফিক্স অ্যাডাপ্টার।

আর অ্যাডাপ্টার ডিজাইন প্যাটার্ন অনেকটা এই অ্যাডাপ্টারের ন্যায় কাজ করে অর্থাৎ আমরা যদি কম্পিউটারের গ্রাফিক্স কিংবা ভিজিএ অ্যাডাপ্টারের কথা চিন্তা করি তাহলে বলা যায় আমরা গেইম খেলার জন্য এক বিশেষ ধরনের অ্যাডাপ্টার ব্যবহার করি আবার সাধারণ কোন কাজের জন্য সাধারণ অ্যাডাপ্টার হলেই চলে কিন্তু বিষয়বস্তু দুইটারি সমান ভিডিও আউটপুট করা দুইটিই একটা কমন প্যাটার্নে তৈরি। আর মজার বিষয় হল এই অ্যাডাপ্টার গুলো আমাদের খুশি মত আমরা পরিবর্তন করতে পারি।

এবার ইমপ্লিমেন্টেশনের পরিভাষায়, ধরুন আমরা একটা পিএইচপি প্রজেক্ট কিংবা অ্যাপ্লিকেশন বানাবো যেখানে আমরা ডাটাবেস অ্যাডাপ্টার হিসেবে **MySQL Adapter** আর **PDO Adapter** ব্যাবহার করব যাতে করে ক্লাইন্ট সহজেই তার পছন্দের অ্যাডাপ্টারটি ব্যাবহার করতে পারে **Database** নামে আরেকটি অ্যাডাপ্টারের মাধ্যমে। এতে করে **MySQL Adapter** আর **PDO Adapter** গুলো খুব সহজেই পরিবর্তন করা যাবে।

নিচে একটা সম্পূর্ণ উদাহরণ দেয়া হলঃ

```php
<?php

interface AdapterInterface
{
    public function query($sql);

    public function result();
}

class MySQLAdapter implements AdapterInterface
{
    protected $connection;

    protected $result;

    public function __construct($host, $username, $password, $dbname)
    {
        $this->connection = new mysqli($host, $username, $password, $dbname);
    }

    public function query($sql)
    {
        $this->result = $this->connection->query($sql);

        return $this;
```

```
    }

    public function result()
    {
        if (gettype($this->result) === 'boolean') {
            return $this->result;
        } elseif ($this->result->num_rows > 0) {
            $result = [];

            while ($row = $this->result->fetch_assoc()) {
                $result[] = $row;
            }

            return $result;
        } else {
            return [];
        }
    }
}

class PDOAdapter implements AdapterInterface
{
    protected $connection;

    protected $result;

    public function __construct($host, $username, $password, $dbname)
    {
        $this->connection = new PDO("mysql:host=$host;dbname=$dbname", $username, $passwo
    }

    public function query($sql)
    {
        $query = $this->connection->prepare($sql);
        $exec = $query->execute();

        if ($query->columnCount() == 0) {
            $this->result = $exec;
        } else {
            $this->result = $query;
        }

        return $this;
    }

    public function result()
    {
        if (gettype($this->result) === 'boolean') {
            return $this->result;
        } else {
            $data = [];

            while ($row = $this->result->fetch(PDO::FETCH_ASSOC)) {
```

```
                $data[] = $row;
            }

            return $data;
        }
    }
}

class Database
{
    protected $adapter;

    public function __construct(AdapterInterface $adapter)
    {
        $this->adapter = $adapter;
    }

    public function query($sql)
    {
        return $this->adapter->query($sql);
    }

    public function result()
    {
        return $this->adapter->result();
    }
}

$mysql = new MySQLAdapter('localhost', 'root', '1234', 'demo');
$db = new Database($mysql);

$query = $db->query("SELECT * FROM users");
$result = $query->result();
var_dump($result);
```

এখানে `AdapterInterface` ইন্টারফেইস ব্যবহার করা হয়েছে যেটিকে ইমপ্লিমেন্ট করে যথাক্রমে `MySQLAdapter` ও `PDOAdapter` ডিক্লেয়ার করা হয়েছে যাতে দুইটারি ন্যাচার কিংবা কোডবেইস একই থাকে।

আবার ডাটাবেসকে অ্যাকসেস করার জন্য ও অ্যাডাপ্টারগুলাকে ব্যবহার করার জন্য `Database` নামে একটা ক্লাস ডিফাইন করা হয়েছে। আর এর ডিপেন্ডেন্সি ইনজেকশন হিসেবে `AdapterInterface` ব্যবহার করা হয়েছে যাতে করে কেবল মাত্র `AdapterInterface` ইমপ্লিমেন্ট করা ক্লাসের ইন্সটান্সই কন্সটারক্টরে পাস করা যায়।

এখানে আমরা `MySQLAdapter` কে ব্যবহার করেছি।

```
$mysql = new MySQLAdapter('localhost', 'root', '1234', 'demo');
$db = new Database($mysql);
```

আমরা চাইলে PDOAdapter ও ব্যবহার করতে পারি নিচের মত করে।

```
$pdo = new PDOAdapter('localhost', 'root', '1234', 'demo');
$db = new Database($pdo);
```

এতে করে অ্যাডাপ্টার গুলা **Loosly Coupled/Highly Decoupled** থাকে আর বর্তমানে এই টার্মটাকে খুবই প্রাধান্য দেয়া হয় বড় কোন অ্যাপ্লিকেশন কিংবা ফ্রেমওয়ার্ক তৈরি করতে গেলে।

নিচের লিঙ্ক থেকে সোর্স কোডটি পাবেন।

ফ্যাক্টরী ডিজাইন প্যাটার্নঃ

ফ্যাক্টরী প্যাটার্ন এমন একটি প্যাটার্ন যা কম বেশি সব ধরনের অ্যাপ্লিকেশনে ব্যাবহৃত হয়ে থাকে এইটা ক্রিয়েশনাল প্যাটার্ন ক্যাটাগরীর মধ্যে পরে।

ফ্যাক্টরী প্যাটার্নের মূল উদ্দেশ্যই হল এর প্রোডাক্ট কিংবা চাইল্ড ক্লাসের অবজেক্ট তৈরি করে দেয়া। যেমন বাস্তব জীবনে যেভাবে ফ্যাক্টরীতে প্রোডাক্ট তৈরি হয়ে থাকে।

এই প্যাটার্ন ক্লাইন্টের কাছে অবজেক্ট ইনস্টানশিয়েট করার লজিক অদৃশ্যমান রাখে। আর অবজেক্টের ক্লাস গুলা একটা কমন ইন্টারফেইস কে ফলো করে বানানো থাকে।

এই প্যাটার্ন সাধারণত ৩ প্রকারেরঃ ১. সিম্পল ফ্যাক্টরী। ২. ফ্যাক্টরী মেথড। ৩. অ্যাবস্ট্রাক্ট ফ্যাক্টরী।

১. সিম্পল ফ্যাক্টরীঃ

ফ্যাক্টরী প্যাটার্নের মধ্যে সিম্পল ফ্যাক্টরী হচ্ছে সবচেয়ে সহজ প্যাটার্ন যদিও অফিশিয়ালি এই প্যাটার্ন ডিজাইন প্যাটার্ন হিসেবে স্বীকৃত না।

এই প্যাটার্নের নিয়ম মতে এর একটি ফ্যাক্টরী থাকবে আর একটি ফ্যাক্টরী একই সময় শুধুমাত্র একটাই প্রোডাক্ট তৈরি করবে অর্থাৎ একটিই ইনস্টান্স কিংবা অবজেক্ট রিটার্ন করবে।

নিচে একটা উদাহরণ দেয়া হলঃ

```
class CarFactory
{
    protected $brands = [];

    public function __construct()
    {
        $this->brands = [
            'mercedes' => 'MercedesCar',
            'toyota' => 'ToyotaCar',
        ];
    }

    public function make($brand)
    {
        if (!array_key_exists($brand, $this->brands)) {
            return new Exception('Not available this car');
        }

        $className = $this->brands[$brand];

        return new $className();
    }
```

```
}

interface CarInterface
{
    public function design();
    public function assemble();
    public function paint();
}

class MercedesCar implements CarInterface
{
    public function design()
    {
        return 'Designing Mercedes Car';
    }

    public function assemble()
    {
        return 'Assembling Mercedes Car';
    }

    public function paint()
    {
        return 'Painting Mercedes Car';
    }
}

class ToyotaCar implements CarInterface
{
    public function design()
    {
        return 'Designing Toyota Car';
    }

    public function assemble()
    {
        return 'Assembling Toyota Car';
    }

    public function paint()
    {
        return 'Painting Toyota Car';
    }
}

$carFactory = new CarFactory;

$mercedes = $carFactory->make('mercedes');
echo $mercedes->design() . '<br/>';
echo $mercedes->assemble() . '<br/>';
echo $mercedes->paint() . '<br/>';

echo '<br/>';
```

```
$toyota = $carFactory->make('toyota');
echo $toyota->design() . '<br/>';
echo $toyota->assemble() . '<br/>';
echo $toyota->paint() . '<br/>';
```

এখানে `CarFactory` নামে মূল ফ্যাক্টরী ক্লাস ডিফাইন করা হয়েছে যেটির মাধ্যমে একটা `Car` ইনস্টান্স তৈরি করা হবে। `Car` এর জন্য ২ টি ক্লাস যথাক্রমে `MercedesCar` ও `ToyotaCar` ডিফাইন করা হয়েছে যেগুলো `CarInterface` কে ফলো করেছে।

এবার চলুন `CarFactory` ক্লাসটিকে ইনস্টানশিয়েট করি।

```
$carFactory = new CarFactory;
```

এরপর ধরুন `MercedesCar` ক্লাসকে ফ্যাক্টরির মাধ্যমে ইনস্টানশিয়েট করব তাহলে টাইপ/প্যারামিটার হিসেবে `mercedes` দিতে হবে নিচের মত করে।

```
$mercedes = $carFactory->make('mercedes');
echo $mercedes->design() . '<br/>';
echo $mercedes->assemble() . '<br/>';
echo $mercedes->paint() . '<br/>';
```

অনুরূপ ভাবে `ToyotaCar` ক্লাসকে ইনস্টানশিয়েট করতে হলে

```
$toyota = $carFactory->make('toyota');
echo $toyota->design() . '<br/>';
echo $toyota->assemble() . '<br/>';
echo $toyota->paint() . ‘<br/>';
```

আর ডিফাইন না করা কোন ক্লাসের টাইপ দিলে সেটি এরর দেখাবে।

২. ফ্যাক্টরী মেথডঃ

ফ্যাক্টরী মেথড প্যাটার্ন অনেকখানি সিম্পল ফ্যাক্টরী প্যাটার্নের মতই শুধুমাত্র এর মূল পার্থক্য হল এটি তার সাব ক্লাস গুলোকে ক্লাস ইনস্টানশিয়েট করার স্বাধীনতা দিয়ে দেয়। আর এর একাধিক ফ্যাক্টরী থাকতে পারে।

নিচে একটা উদাহরণ দেয়া হলঃ

```
abstract class VehicleFactoryMethod
{
    abstract public function make($brand);
}

class CarFactory extends VehicleFactoryMethod
```

```
{
    public function make($brand)
    {
        $car = null;

        switch ($brand) {
            case "mercedes":
                $car = new MercedesCar;
                break;
            case "toyota":
                $car = new ToyotaCar;
                break;
        }

        return $car;
    }
}

class BikeFactory extends VehicleFactoryMethod
{
    public function make($brand)
    {
        $bike = null;

        switch ($brand) {
            case "yamaha":
                $bike = new YamahaBike;
                break;
            case "ducati":
                $bike = new DucatiBike;
                break;
        }

        return $bike;
    }
}

interface CarInterface
{
    public function design();

    public function assemble();

    public function paint();
}

interface BikeInterface
{
    public function design();

    public function assemble();

    public function paint();
```

```
}

class MercedesCar implements CarInterface
{
    public function design()
    {
        return 'Designing Mercedes Car';
    }

    public function assemble()
    {
        return 'Assembling Mercedes Car';
    }

    public function paint()
    {
        return 'Painting Mercedes Car';
    }
}

class ToyotaCar implements CarInterface
{
    public function design()
    {
        return 'Designing Toyota Car';
    }

    public function assemble()
    {
        return 'Assembling Toyota Car';
    }

    public function paint()
    {
        return 'Painting Toyota Car';
    }
}

class YamahaBike implements BikeInterface
{
    public function design()
    {
        return 'Designing Yamaha Bike';
    }

    public function assemble()
    {
        return 'Assembling Yamaha Bike';
    }

    public function paint()
    {
        return 'Painting Yamaha Bike';
```

```
    }
}

class DucatiBike implements BikeInterface
{
    public function design()
    {
        return 'Designing Ducati Bike';
    }

    public function assemble()
    {
        return 'Assembling Ducati Bike';
    }

    public function paint()
    {
        return 'Painting Ducati Bike';
    }
}

$carFactoryInstance = new CarFactory;

$mercedes = $carFactoryInstance->make('mercedes');
echo $mercedes->design() . '<br/>';
echo $mercedes->assemble() . '<br/>';
echo $mercedes->paint() . '<br/>';

echo '<br/>';

$toyota = $carFactoryInstance->make('toyota');
echo $toyota->design() . '<br/>';
echo $toyota->assemble() . '<br/>';
echo $toyota->paint() . '<br/>';

echo '<br/>';

$bikeFactoryInstance = new BikeFactory;

$yamaha = $bikeFactoryInstance->make('yamaha');
echo $yamaha->design() . '<br/>';
echo $yamaha->assemble() . '<br/>';
echo $yamaha->paint() . '<br/>';

echo '<br/>';

$ducati = $bikeFactoryInstance->make('ducati');
echo $ducati->design() . '<br/>';
echo $ducati->assemble() . '<br/>';
echo $ducati->paint() . '<br/>';
```

এখানে ফ্যাক্টরী মেথডের জন্য `VehicleFactoryMethod` নামে একটা অ্যাবস্ট্রাক্ট ক্লাস ডিফাইন করা হয়েছে যেটির সাব ক্লাস যথাক্রমে `CarFactory` ও `BikeFactory` আছে যেগুলা ভিন্ন ভিন্ন একক ফ্যাক্টরী। আবার প্রতিটি ফ্যাক্টরীর জন্য সিম্পল ফ্যাক্টরী প্যাটার্নের ন্যায় ইন্টারফেইস `CarInterface` ও `BikeInterface` ডিফাইন করা হয়েছে যেগুলোকে ইমপ্লিমেন্ট করে কংক্রিট ক্লাস অর্থাৎ ইনস্টানশিয়েট যোগ্য ক্লাস যথাক্রমে `CarFactory` এর আওতায় `MercedesCar` ও `ToyotaCar` এবং `BikeFactory` এর আওতায় `YamahaBike` ও `DucatiBike` ডিফাইন করা হয়েছে।

সুতরাং `CarFactory` ও `BikeFactory` ক্লাসগুলো নির্ধারণ করতে পারবে সে কোন ক্লাসকে ইনস্টানশিয়েট করবে।

নিচের কোডটি খেয়াল করলে বুঝতে পারবেন ২ টি আলাদা ফ্যাক্টরীর মাধ্যমে প্যারামিটার কিংবা `Car` এর ব্র্যান্ড পাস করে কাঙ্ক্ষিত অবজেক্ট কে পাওয়া যায়।

```
$carFactoryInstance = new CarFactory;

$mercedes = $carFactoryInstance->make('mercedes');
echo $mercedes->design() . '<br/>';
echo $mercedes->assemble() . '<br/>';
echo $mercedes->paint() . '<br/>';

echo '<br/>';

$toyota = $carFactoryInstance->make('toyota');
echo $toyota->design() . '<br/>';
echo $toyota->assemble() . '<br/>';
echo $toyota->paint() . '<br/>';

echo '<br/>';

$bikeFactoryInstance = new BikeFactory;

$yamaha = $bikeFactoryInstance->make('yamaha');
echo $yamaha->design() . '<br/>';
echo $yamaha->assemble() . '<br/>';
echo $yamaha->paint() . '<br/>';

echo '<br/>';

$ducati = $bikeFactoryInstance->make('ducati');
echo $ducati->design() . '<br/>';
echo $ducati->assemble() . '<br/>';
echo $ducati->paint() . '<br/>';
```

৩. অ্যাবস্ট্রাক্ট ফ্যাক্টরীঃ

অ্যাবস্ট্রাক্ট ফ্যাক্টরী এমন একটি পদ্ধতি প্রদান করে যেখানে একটি মূল (অ্যাবস্ট্রাক্ট) ফ্যাক্টরী অনেকগুলো একক ফ্যাক্টরীকে একত্রিত করে রাখে।

এক কথায়, প্রথমে একটি অ্যাবস্ট্রাক্ট ফ্যাক্টরী অনেকগুলো প্রোডাক্ট ফ্যাক্টরী তৈরি করে এরপর প্রতিটি ফ্যাক্টরী একাধিক প্রোডাক্ট কিংবা অবজেক্ট তৈরি করে।

উল্লেখ্য, প্রতিটি প্রোডাক্ট ফ্যাক্টরী ক্লাসকে একটা কমন অ্যাবস্ট্রাক্ট ক্লাসকে এক্সটেন্ড করতে হবে অথবা একটা কমন ইন্টারফেইসকে ইমপ্লিমেন্ট করতে হবে। আবার প্রতিটি প্রোডাক্ট ফ্যাক্টরী ক্লাসে একাধিক আর একই মেথড থাকতে হবে।

নিচে একটা উদাহরণ দেয়া হলঃ

```
abstract class AbstractVehicleFactory
{
    abstract public function makeCar();
    abstract public function makeBike();
}

class BangladeshiFactory extends AbstractVehicleFactory
{
    public function makeCar()
    {
        return new ToyotaCar();
    }

    public function makeBike()
    {
        return new YamahaBike();
    }
}

class USAFactory extends AbstractVehicleFactory
{
    public function makeCar()
    {
        return new MercedesCar();
    }

    public function makeBike()
    {
        return new DucatiBike();
    }
}

abstract class AbstractVehicle
{
    abstract public function design();
    abstract public function assemble();
    abstract public function paint();
}

abstract class AbstractCarVehicle extends AbstractVehicle
{

}
```

```
abstract class AbstractBikeVehicle extends AbstractVehicle
{

}

class MercedesCar extends AbstractCarVehicle
{
    public function design()
    {
        return 'Designing Mercedes Car';
    }

    public function assemble()
    {
        return 'Assembling Mercedes Car';
    }

    public function paint()
    {
        return 'Painting Mercedes Car';
    }
}

class ToyotaCar extends AbstractCarVehicle
{
    public function design()
    {
        return 'Designing Toyota Car';
    }

    public function assemble()
    {
        return 'Assembling Toyota Car';
    }

    public function paint()
    {
        return 'Painting Toyota Car';
    }
}

class YamahaBike extends AbstractBikeVehicle
{
    public function design()
    {
        return 'Designing Yamaha Bike';
    }

    public function assemble()
    {
        return 'Assembling Yamaha Bike';
    }
```

```
    public function paint()
    {
        return 'Painting Yamaha Bike';
    }
}

class DucatiBike extends AbstractBikeVehicle
{
    public function design()
    {
        return 'Designing Ducati Bike';
    }

    public function assemble()
    {
        return 'Assembling Ducati Bike';
    }

    public function paint()
    {
        return 'Painting Ducati Bike';
    }
}

$bangladeshiFactoryInstance = new BangladeshiFactory;
$car = $bangladeshiFactoryInstance->makeCar();
echo $car->design() . '<br/>';
echo $car->assemble() . '<br/>';
echo $car->paint() . '<br/>';

echo '<br/>';

$bike = $bangladeshiFactoryInstance->makeBike();
echo $bike->design() . '<br/>';
echo $bike->assemble() . '<br/>';
echo $bike->paint() . '<br/>';

echo '<br/>';

$usaFactoryInstance = new USAFactory;
$car = $usaFactoryInstance->makeCar();
echo $car->design() . '<br/>';
echo $car->assemble() . '<br/>';
echo $car->paint() . '<br/>';

echo '<br/>';

$bike = $usaFactoryInstance->makeBike();
echo $bike->design() . '<br/>';
echo $bike->assemble() . '<br/>';
echo $bike->paint() . ‘<br/>';
```

উপরের কোডে `AbstractVehicleFactory` নামে একটা অ্যাবস্ট্রাক্ট ফ্যাক্টরী ক্লাস ডিফাইন করা হয়েছে যেখানে `makeCar()` ও `makeBike()` ২টা মেথড দেয়া আছে যাতে সাব ফ্যাক্টরী গুলো ওই মেথড গুলো ডিফাইন করে।

এখানে একটি মেথড `Car` অবজেক্ট তৈরি করতে আরেকটি `Bike` অবজেক্ট তৈরি করতে ব্যবহৃত হয়েছে যা সব গুলা ফ্যাক্টরীকেই করতে হবে। আর মূল বিষয় হল একেক ফ্যাক্টরী একেক ব্র্যান্ডের `Car` ও `Bike` অবজেক্ট তৈরি করবে।

যেমন এখানে আমরা `BangladeshiFactory` ফ্যাক্টরী ব্যাবহার করেছি যেটি `ToyotaCar` ও `YamahaBike` ক্লাসের অবজেক্ট তৈরি করবে। অনুরূপ ভাবে, `USAFactory` ফ্যাক্টরী `MercedesCar` ও `DucatiBike` ক্লাসের অবজেক্ট তৈরি করবে।

আবার আপনি চাইলে একটা ফ্যাক্টরীতে একাধিক ব্রান্ডের `Car` কিংবা `Bike` এর অবজেক্ট তৈরি করতে পারেন সেক্ষেত্রে র‍্যান্ডমলি কিংবা লজিক্যালি করতে হবে।

এই লিঙ্ক থেকে সোর্স কোডটি পাবেন।

ডিপেন্ডেন্সি ইনজেকশন ডিজাইন প্যাটার্নঃ

ডিপেন্ডেন্সি ইনজেকশন স্ট্রাকচারাল ডিজাইন প্যাটার্নের মধ্যে পরে। এইটা এমন একটা ডিজাইন প্যাটার্ন যা কোন ক্লাসের ডিপেন্ডেন্সি অর্থাৎ প্রয়োজনীয় অবজেক্ট গুলোকে রান টাইম কিংবা কম্পাইল টাইমে সহজে পরিবর্তনে সহায়তা করে।

এই প্যাটার্নের মূল উদ্দেশ্যই হল লুজলি কাপল আর্কিটেকচার ইমপ্লিমেন্ট করা যাতে করে একটা ভাল মানের অ্যাপ তৈরি করা যায়।

ডিপেন্ডেন্সি ইনজেকশন ডিজাইন প্যাটার্ন হল `S.O.L.I.D Principle` এর `D` যার পূর্ণ অর্থ `Dependency Inversion Principle (DIP)` যেটি `Inversion of Control (IoC)` কে অনুসরণ করে।

এখানে `Dependency Inversion Principle` বলতে `Decoupling` করাকে বুঝানো হয় আর `Inversion of Control` বলতে কিভাবে ডিপেন্ডেন্সি রিজল্ভ করা হবে সেটিকে বুঝায়। ডিপেন্ডেন্সি রিজল্ভ করতে `Dependency Injection (DI) Container` বা `Inversion of Control (IoC) Container` ব্যাবহৃত হয়ে থাকে।

আমরা সাধারণত একটি ক্লাসে অন্য ক্লাসের অবজেক্ট ব্যাবহার করলে নিচের মত করে হার্ডকোড করি যা হাইলি কাপল্ড থাকে।

```
class Database
{
    protected $adapter;

    public function __construct()
    {
        $this->adapter = new MySqlAdapter;
    }
}

class MysqlAdapter
{

}
```

আর ডিপেন্ডেন্সি ইনজেকশন ডিজাইন প্যাটার্নে কোন ক্লাস কিংবা ইন্টারফেইসকে টাইপ হিন্ট করে কনস্ট্রাক্টর কিংবা মেথডে ইঞ্জেক্ট করতে হয় নিচের মত করে।

```
class Database
{
    protected $adapter;

    public function __construct(MySqlAdapter $adapter)
    {
        $this->adapter = $adapter;
    }
}

class MysqlAdapter
{

}
```

উপরে `MySqlAdapter` ক্লাসকে ডিপেন্ডেন্সি হিসেবে রাখা হয়েছে। আর এই ডিপেন্ডেন্সি রিজল্ভ করতে হলে অবশ্যই `MySqlAdapter` এর অবজেক্ট কন্সট্রাক্টরের প্যারামিটারে দিতে হবে।

যেমনঃ

```
$mysqlAdapter = new MysqlAdapter;
$database = new Database($mysqlAdapter);
```

ডিপেন্ডেন্সি প্রধানত তিন ভাবে ইনজেক্ট করা যায়।

১. কন্সট্রাক্টর ইনজেকশনঃ

যা কনস্ট্রাক্টরের মাধ্যমে ইঞ্জেক্ট করা হয়।

```
public function __construct(MySqlAdapter $adapter)
{
    $this->adapter = $adapter;
}
```

২. সেটার ইনজেকশনঃ

যা কোন মেথডের প্যারামিটারে ইঞ্জেক্ট করা হয়।

```
public function setterMethod(MySqlAdapter $adapter)
{
    $this->adapter = $adapter;
}
```

৩. ইন্টারফেইস ইনজেকশনঃ

ইন্টারফেইসকে কোন কনস্ট্রাক্টরে অথবা সেটার মেথডে ইঞ্জেক্ট করা হয়।

```
public function __construct(AdapterInterface $adapter)
{
    $this->adapter = $adapter;
}
```

আমরা আমাদের প্রজেক্টে ডিপেন্ডেন্সি গুলোকে স্বয়ংক্রিয় ভাবে ইঞ্জেক্ট কিংবা রিজল্ভ করতে ডিপেন্ডেন্সি ইনজেকশন কন্টেইনার ব্যাবহার করব যা আগেই উল্লেখ করেছি। অনেক ফ্রেমওয়ার্কে এই কন্টেইনার সাধারণত বিল্ট-ইন দেয়া থাকে যেমনঃ `Symfony`, `Laravel`, `Yii`

আমরা সাধারণ প্রজেক্টের ক্ষেত্রে `Pimple` নামে কন্টেইনারটি ব্যাবহার করতে পারি। আবার আমি আমার কাজের জন্য খুব সহজ এবং অপ্টিমাইজ একটা কন্টেইনার বানিয়েছিলাম আপনারা চাইলে সেটি দেখতে পারেন এই লিঙ্ক থেকে। আশাকরি সোর্স কোড ও ডকুমেন্টেশন থেকে আপনারা ভাল ধারণা পাবেন।

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।

ফ্যাসাড ডিজাইন প্যাটার্নঃ

`Facade` ডিজাইন প্যাটার্ন স্ট্রাকচারাল ডিজাইন প্যাটার্নের মধ্যে পরে। ফ্যাসাড ডিজাইন প্যাটার্ন এর কাজ হল ক্লায়েন্ট এর কাছে একটা কমপ্লেক্স সিস্টেম বা ইন্টারফেইস হতে একটা সহজ ইন্টারফেইস প্রদান করা যাতে কমপ্লেক্স কিংবা আগলি কোড গুলো হিডেন অবস্থায় থাকে।

লেগাসি কিংবা কমপ্লেক্স কোন সিস্টেম এর কোডকে সহজ ভাবে উপস্থাপন করার দরকার হলে এই প্যাটার্ন ব্যাবহার করা হয়।

ধরুন আপনার সিস্টেম কিংবা অ্যাপ্লিকেশনে একটা কমপ্লেক্স লাইব্রেরি ব্যাবহার করার দরকার পরতেছে আর আপনি উক্ত কমপ্লেক্স পার্টকে সহজ করে তোলার জন্য একটা `Wrapper` বানিয়ে সেইটা করতে পারেন।

এবার নিচে একটা উদাহরণের মাধ্যমে প্যাটার্নটি বোঝানোর চেষ্টা করা হলঃ

```php
<?php

class Cart
{
    public function addProducts($products)
    {
        // Product adding codes goes here
    }

    public function getProducts()
    {
        // Product retrieval codes goes here
    }
}

class Order
{
    public function process($products)
    {
        // Order processing codes goes here
    }
}

class Payment
{
    public function charge($charge)
    {
        // Additional charge codes goes here
    }

    public function makePayment()
    {
        // Payment method verify & payment codes goes here
    }
```

```
}

class Shipping
{
    public function calculateCharge()
    {
        // Calculation codes goes here
    }

    public function shipProducts()
    {
        // Ship process codes goes here
    }
}

class CustomerFacade
{
    public function __construct()
    {
        $this->cart = new Cart;
        $this->order = new Order;
        $this->payment = new Payment;
        $this->shipping = new Shipping;
    }

    public function addToCart($products)
    {
        $this->cart->addProducts($products);
    }

    public function checkout()
    {
        $products = $this->cart->getProducts();

        $this->totalAmount = $this->order->process($products);
    }

    public function makePayment()
    {
        $charge = $this->shipping->calculateCharge();
        $this->payment->charge($charge);

        $isCompleted = $this->payment->makePayment();

        if ($isCompleted) {
            $this->shipping->shipProducts();
        }
    }
}

$customer = new CustomerFacade;

$products = [
```

```
    [
        'name' => 'Polo T-Shirt',
        'price' => 40,
    ],
    [
        'name' => 'Smart Watch',
        'price' => 400,
    ],
];

$customer->addToCart($products);
$customer->checkout();
$customer->makePayment();
```

উপরের কোডটি খেয়াল করলে দেখতে পারবেন এখানে একটা ই-কমার্স অ্যাপ্লিকেশনের প্রসেস দেখানো হয়েছে। এর জন্য আমরা যথাক্রমে `Cart` , `Order` , `Payment` , `Shipping` ক্লাসগুলো ব্যাবহার করেছি আর ফ্যাসাড হিসেবে `CustomerFacade` ক্লাস ব্যাবহার করেছি। এখানে কোন প্রোডাক্টকে কার্টে যুক্ত করার জন্য `Cart` ক্লাসটি, অর্ডার প্রসেস করার জন্য `Order` ক্লাসটি, পেমেন্ট প্রসেস করার জন্য `Payment` ক্লাসটি আর প্রডাক্ট এর শিপিং হ্যান্ডল করার জন্য `Shipping` ক্লাসটি ব্যাবহার করেছি।

এখন মুল কথা হল আমরা যদি এসব কাজের জন্য প্রতিবার উক্ত ক্লাস গুলোকে বার বার কল করি তাহলে অনেক সময় সাপেক্ষ বেপার হয়ে পরবে আর স্ট্রাকচারটিও ভাল হবেনা। আর তাই এখানে ফ্যাসাড প্যাটার্নটি ব্যাবহার করা হয়েছে। যাতে ডেভেলপার কিংবা ক্লায়েন্ট হিসেবে সুধু মাত্র `CustomerFacade` ক্লাসটিকে ব্যাবহার করে উপরে উল্লেখিত সবগুলো কাজ অনায়াসে করা সম্ভব।

অতিরিক্ত বিষয় (লারাভেল ফ্যাসাড) ঃ

আমরা যারা লারাভেল ব্যাবহার করি তারা কম বেশি সবাই জানি লারাভেল এ অনেকগুলো বিল্ড-ইন ফ্যাসাড আছে কিংবা ব্যাবহার হয়। যেমনঃ `DB` , `View` , `Event` , `Queue` , `Mail` ইত্যাদি।

আমরা মূলত যেটা জানি তা হল লারাভেল এ ফ্যাসাড `Statically` কোন ক্লাসকে কল করার জন্য ব্যাবহার হয়। আসলে বিষয়টি ঠিক তেমন নয়। এখানে অনেক কমপ্লেক্স সিস্টেমকে হাইড করে আমাদের কাছে সহজ ভাবে উপস্থাপন করা হয়েছে তার সাথে লারাভেল ফ্যাসাড প্যাটার্নের সাথে `__callStatic` ম্যাজিক মেথডটি ব্যাবহার করা হয়েছে। যাতে ডেভেলপার কিংবা ক্লায়েন্টকে আলাদাভাবে ক্লাস ইন্সট্যানশিয়েট করতে না হয়।

নিচে একটা উদাহরণ দেয়া হলঃ

```php
<?php

class Person
{
    public function getFullName()
    {
        return 'Sohel Amin';
    }
}

class PersonFacade
{
    static $instance;

    public static function __callStatic($method, $args)
    {
        if (null === static::$instance) {
            static::$instance = new Person;
        }

        $instance = static::$instance;

        switch (count($args)) {
            case 0:
                return $instance->$method();
            case 1:
                return $instance->$method($args[0]);
            case 2:
                return $instance->$method($args[0], $args[1]);
            case 3:
                return $instance->$method($args[0], $args[1], $args[2]);
            case 4:
                return $instance->$method($args[0], $args[1], $args[2], $args[3]);
            default:
                return call_user_func_array([$instance, $method], $args);
        }

    }
}

var_dump(PersonFacade::getFullName());
```

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।

স্ট্রাটেজি ডিজাইন প্যাটার্নঃ

`Strategy` ডিজাইন প্যাটার্ন বিহেভিওরাল ডিজাইন প্যাটার্নের মধ্যে পরে। `Strategy` এর অর্থ হল কৌশল, কোন কিছু করতে গেলে তার জন্য কৌশল কিংবা এক গুচ্ছ পদক্ষেপ গ্রহণ করাই হল স্ট্রাটেজি।

প্রোগ্রামিং এর পরিভাষায়, একটি নির্দিষ্ট কাজ সম্পন্ন করতে ভিন্ন ভিন্ন অ্যালগরিদম নির্ধারণ করার স্বাধীনতা থাকাই স্ট্রাটেজি প্যাটার্ন। এই প্যটার্নকে আবার পলিসি প্যাটার্নও বলা হয়ে থাকে।

ধরুন, আপনি ঢাকা থেকে চট্টগ্রাম যাইতে চাচ্ছেন এরজন্য আপনি চাইলে বাস, ট্রেন কিংবা প্লেন এ করে যাইতে পারেন। এইক্ষেত্রে গন্তব্যস্থল একটিই কিন্তু এটা সম্পন্ন করতে ভিন্ন ভিন্ন স্ট্রাটেজি অনুসরণ করা যায়।

এবার চলুন একটা বাস্তব ভিত্তিক উদাহরণের মাধ্যমে প্যাটার্নটি বুঝা যাক। প্রথমে চলুন একটা ইন্টারফেইস বানিয়ে ভিন্ন ভিন্ন স্ট্রাটেজি ইমপ্লিমেন্ট করি নিচের মত করে।

```
interface TravelStrategy
{
    public function travel();
}

class BusTravelStrategy implements TravelStrategy
{
    public function travel()
    {
        // Bus travel strategy will goes here
    }
}

class TrainTravelStrategy implements TravelStrategy
{
    public function travel()
    {
        // Train travel strategy will goes here
    }
}

class PlaneTravelStrategy implements TravelStrategy
{
    public function travel()
    {
        // Plane travel strategy will goes here
    }
}
```

এবার মেইন কনটেক্সট ক্লাস হিসেবে `Traveler` নামক একটা ক্লাস ডিফাইন করি।

```
class Traveler
{
    protected $traveler;

    public function __construct(TravelStrategy $traveler)
    {
        $this->traveler = $traveler;
    }

    public function travel()
    {
        $this->traveler->travel();
    }
}
```

পরিশেষে, স্ট্রাটেজি পরিবর্তন করে সহজে আমরা আমাদের কার্য সম্পন্ন করতে পারি।

```
$traveler = new Traveler(new BusTravelStrategy());
$traveler->travel();

$traveler1 = new Traveler(new PlaneTravelStrategy());
$traveler1->travel();
```

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।

ইটারেটর ডিজাইন প্যাটার্নঃ

ইটারেটর ডিজাইন প্যাটার্ন বিহেভিওরাল টাইপের মধ্যে পরে। এই প্যাটার্ন এর মুল উদ্দেশ্যই হচ্ছে ইটারেটরের ব্যাবহার করা। ইটারেটর একটা কন্টেইনার কিংবা অবজেক্ট এর ইলিমেন্টকে ট্রাভার্স করার জন্য সহায়তা করে আর এতে ভিতরের লজিক গুলো লুকানো অবস্থায় থাকে। যারফলে, আমরা কন্টেইনারে আমাদের পছন্দের মত ডাটা স্ট্রাকচার ব্যাবহার করতে পারি।

এবার চলুন আমরা কিভাবে এই প্যাটার্নটি ইমপ্লিমেন্ট করতে পারি। পিএইচপির একটা বিল্ড-ইন `Iterator` ইন্টারফেইস আছে আমরা সেটি ব্যাবহার করব।

সর্বপ্রথমে, আমরা ইলিমেন্ট বা আইটেম এর জন্য `Book` নামে একটা ক্লাস ডিফাইন করব।

```
class Book
{
    private $title;

    public function __construct($title)
    {
        $this->title = $title;
    }

    public function getTitle()
    {
        return $this->title;
    }
}
```

এবার কন্টেইনার এর জন্য `BookList` নামে একটা ক্লাস ডিফাইন করব।

```
class BookList implements Iterator, Countable
{
    private $books = [];

    private $currentIndex = 0;

    public function current()
    {
        return $this->books[$this->currentIndex];
    }

    public function key()
    {
        return $this->currentIndex;
    }

    public function next()
```

```
    {
        $this->currentIndex++;
    }

    public function rewind()
    {
        $this->currentIndex = 0;
    }

    public function valid()
    {
        return isset($this->books[$this->currentIndex]);
    }

    public function count()
    {
        return count($this->books);
    }

    public function addBook(Book $book)
    {
        $this->books[] = $book;
    }

    public function removeBook(Book $bookToRemove)
    {
        foreach ($this->books as $key => $book) {
            if ($book->getTitle() === $bookToRemove->getTitle()) {
                unset($this->books[$key]);
            }
        }

        $this->books = array_values($this->books);
    }
}
```

এখানে `Iterator` ইন্টারফেসের জন্য যথাক্রমে `current()`, `key()`, `next()`, `rewind()` ও `valid()` মেথডগুলি ইমপ্লিমেন্ট করা হয়েছে আর `Countable` ইন্টারফেইসের এর জন্য `count()` মেথডটি ইমপ্লিমেন্ট করা হয়েছে যা ইলেমেন্ট কাউন্ট করতে সাহায্য করবে। আর ইলিমেন্ট অ্যাড আর রিমুভ করার জন্য `addBook()` ও `removeBook()` কাস্টম মেথডগুলি ব্যাবহার করা হয়েছে।

এবার কন্টেইনার ক্লাসটি ইন্সটানশিয়েট করে কিছু ইলিমেন্ট অ্যাড করে আমরা নিচের ন্যায় লুপের মাধ্যমে ইলিমেন্ট ট্রাভার্স করে অ্যাকসেস করতে পারি।

```php
$bookList = new BookList();
$bookList->addBook(new Book('Design Pattern'));
$bookList->addBook(new Book('Head First Design Pattern'));
$bookList->addBook(new Book('Clean Code'));
$bookList->addBook(new Book('The Pragmatic Programmer'));

$bookList->removeBook(new Book('Design Pattern'));

foreach ($bookList as $book) {
    echo $book->getTitle() . PHP_EOL;
}
```

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।

প্রক্সি ডিজাইন প্যাটার্নঃ

`Proxy` ডিজাইন প্যাটার্ন স্ট্রাকচারাল ডিজাইন প্যাটার্নের মধ্যে পরে। এই প্যাটার্ন শুরুর আগে আসুন আমরা “প্রক্সি” শব্দের অর্থ জেনে নেই। প্রক্সি এমন একটি প্রতিনিধি বা বস্তু যা অন্য বিষয় বস্তুর হয়ে কাজ করে।

অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং এ প্রক্সি হলঃ একটি অবজেক্ট অন্য কোন অবজেক্টের হয়ে কাজ করা বা তাকে কন্ট্রোল করা।

প্রক্সি সাধারণত ৩ প্রকারেরঃ

1. Virtual Proxy: এই প্রক্সি মুল অবজেক্টকে ইন্সটানশিয়েট বা ইনিশিয়ালাইজ করতে বিলম্ব করে যতক্ষণ না দরকার পরে।
2. Remote Proxy: এই প্রক্সি কোন রিমুট লোকেশনে অবস্থিত কোন অবজেক্টকে রিপ্রেজেন্ট করে। যেমনঃ সার্ভার থেকে কোন অবজেক্টকে অ্যাকসেস করা।
3. Protection Proxy: এই প্রক্সি মুল অবজেক্টকে অ্যাকসেস করার আগে সেকুরিটি চেক করে।
4. Smart Proxy: এই প্রক্সি মুল অবজেক্টের রেফারেন্স নাম্বার ট্রাক করে এবং প্রয়োজন মত মেমোরি থেকে লোডিং অথবা ফ্রি করতে সহয়তা করে।

এখানে আমরা `Virtual Proxy` এর একটি উদাহরণ দেখব।

```php
interface FileInterface
{
    public function content();
}

class RealFile implements FileInterface
{
    private $fileName;

    private $fileContent;

    public function __construct($fileName)
    {
        $this->fileName = $fileName;

        $this->readFile();
    }

    private function readFile()
    {
        $this->fileContent = file_get_contents($this->fileName);
    }

    public function content()
    {
        return $this->fileContent;
    }
}

class ProxyFile implements FileInterface
{
    private $fileName;

    private $realFileObject;

    public function __construct($fileName)
    {
        $this->fileName = $fileName;
    }

    public function content()
    {
        // Lazy load the file using the RealFile class
        if (!$this->realFileObject) {
            $this->realFileObject = new RealFile($this->fileName);
        }

        return $this->realFileObject->content();
    }
}
```

উপরের কোডটি খেয়াল করলে আমরা দেখতে পাব একই ইন্টারফেইস `FileInterface` ব্যাবহার করে রিয়েল অবজেক্ট এর জন্য `RealFile` ও প্রক্সি অবজেক্টের জন্য `ProxyFile` নামক ক্লাস ইমপ্লিমেন্ট করা হয়েছে।

`ProxyFile` এর `content()` মেথডটি দেখলে বুঝতে পাব যে এর মাধ্যমে মুল `RealFile` ক্লাস এর ইন্সটানশিয়েট করা হয়েছে লেজিলোডিং পদ্ধতির মাধ্যমে যাতে অ্যাকসেস না করা পর্যন্ত ইন্সটানশিয়েট না করা হয়।

```
public function content()
{
    // Lazy load the file using the RealFile class
    if (!$this->realFileObject) {
        $this->realFileObject = new RealFile($this->fileName);
    }

    return $this->realFileObject->content();
}
```

এবার নিচের মত করে উভয় ক্লাসকে ইন্সটানশিয়েট করে কল করা হলে প্রথমে ভিন্ন ভিন্ন মেমোরি দখল করবে।

```
$realFile = new RealFile('/path/to/file.jpg');
var_dump(memory_get_usage()); // ~5Mb
$realFile->content();
var_dump(memory_get_usage()); // ~5Mb

$realFile->content();
var_dump(memory_get_usage()); // ~5Mb

$proxyFile = new ProxyFile('/path/to/file.jpg');
var_dump(memory_get_usage()); // ~350Kb
$proxyFile->content();
var_dump(memory_get_usage()); // ~5Mb

$proxyFile->content();
var_dump(memory_get_usage()); // ~5Mb
```

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।

ডেকোরেটর ডিজাইন প্যাটার্নঃ

`Decorator` ডিজাইন প্যাটার্ন স্ট্রাকচারাল ডিজাইন প্যাটার্নের মধ্যে পরে। `Decorator` শব্দটি শুনলেই আমরা বুঝতে পারছি যে এটি কোন কিছুর প্রসাধক হিসেবে কাজ করে থাকে।

অবজেক্ট ওরিয়েন্টেডের ক্ষেত্রে ডেকোরেটর একটি নির্দিষ্ট অবজেক্টকে স্টাটিক্যালি অথবা ডায়নামিক্যালি সংযুক্তি বা পরিবর্তন করে থাকে।

এখন প্রশ্ন আসতে পারে আমরা কোন ক্লাসকে ইনহেরিট করেই তো এই কাজটি করতে পারি তাহলে কেন ডেকোরেটর ব্যাবহার করবো? ইনহেরিট্যান্স এর মাধ্যমে আমরা একটা ক্লাসকে পরিবর্তন করে থাকি তার মানে সাবক্লাস দিয়ে আমরা যতগুলো অবজেক্ট তৈরি করবো সবগুলাই সেইম হবে। অন্যদিকে ডেকোরেটর আমাদেরকে এই ক্ষেত্রে শুধুমাত্র কোন নির্দিষ্ট অবজেক্টে পরিবর্তন করতে ফ্লেক্সিবিলিটি দিয়ে থাকে।

এবার চলুন একটা উদাহরণ দেখা যাক।

```php
interface EmailInterface
{
    public function body();
}

class Email implements EmailInterface
{
    public function body()
    {
        return 'Simple email body.';
    }
}

abstract class EmailDecorator implements EmailInterface
{
    public $email;

    public function __construct(EmailInterface $email)
    {
        $this->email = $email;
    }

    abstract public function body();
}

class NewYearEmailDecorator extends EmailDecorator
{
    public function body()
    {
        return $this->email->body() . ' Additional text from deocorator.';
    }
}
```

উপরের কোডে খেয়াল করলে দেখতে পাবেন ইমেইল পাঠানোর জন্য একটি মুল ইন্টারফেইস `EmailInterface` আর এর কনক্রিট ক্লাস `Email` ইমপ্লিমেন্ট করা হয়েছে যা দিয়ে আমরা সিম্পল ইমেইল করতে পারি।

এবার ডেকোরেটর এর জন্য `EmailDecorator` অ্যাবস্ট্রাক্ট ক্লাস ডিফাইন করেছি আর এইটা কে ইনহেরিট করে `NewYearEmailDecorator` কনক্রিট ক্লাস ডিফাইন করেছি যার মাধ্যমে আমরা খুব সহজেই ইমেইলের অবজেক্টকে পরিবর্তন/সংযুক্তি করতে পারবো।

আমরা চাইলে `EmailDecorator` অ্যাবস্ট্রাক্ট ক্লাসটি ইনহেরিট করে আরও ক্লাস ডিফাইন করতে পারি।

এবার নিচের কোডটি দেখলে বুঝতে পারবেন কিভাবে অবজেক্টকে ডেকোরেট করা হয়েছে।

```php
// Simple Email
$email = new Email();
var_dump($email->body());

// Decorated Email
$emailNewYearDecorator = new NewYearEmailDecorator($email);
var_dump($emailNewYearDecorator->body());
```

এই চ্যাপ্টারের সোর্স কোডটি এই লিঙ্ক থেকে পাবেন।