1. Jelaskan masing-masing bagian dari PSR, serta berikan contohnya.

Jawab.

1. PSR-1 (Basic Coding Standard) 🡪 bagian standar yang terdiri dari apa yang harus dipertimbangkan dalam penulisan kode untuk memastikan tingkat teknis yang tinggi interoperabilitas antara kode PHP.

* File harus menggunakan tag <?php dan <?=
* File harus menggunakan UTF-8 tanpa BOM untuk kode PHP
* File harus menyatakan simbol (seperti class, fungsi, dll) atau menyebabkan efek samping tapi tidak selalu melakukan keduanya.
* Namespaces dan class harus mengikuti "autoloading" PSR: [PSR-0, PSR-4].
* Nama class dideklarasikan di StudlyCaps.
* Konstanta class dinyatakan dalam semua huruf besar dengan pemisah garis bawah.
* Nama metode dideklarasikan di camelCase.

Contoh :

**<?php**

*// side effect: change ini settings*

ini\_set('error\_reporting', **E\_ALL**);

*// side effect: loads a file*

**include** "file.php";

*// side effect: generates output*

**echo** "<html>\n";

*// declaration*

**function** **foo**()

{

*// function body*

}

1. PSR-2: Coding Style Guide 🡪 mengurangi gesekan kognitif saat memindai kode dari penulis yang berbeda. Ketika berbagai penulis berkolaborasi di beberapa proyek, ini dapat membantu agar memiliki satu set pedoman untuk digunakan di antara semua proyek tersebut.

* Kode HARUS mengikuti "panduan gaya pengkodean" PSR [PSR-1].
* Kode HARUS menggunakan 4 spasi untuk indentasi, bukan tab.
* TIDAK HARUS ada batas keras pada panjang garis; batas lunak harus 120 karakter; baris SEHARUSNYA menjadi 80 karakter atau kurang.
* Harus ada satu baris kosong setelah deklarasi ruangnama, dan di sana
* HARUS menjadi satu baris kosong setelah blok deklarasi penggunaan.
* Visibilitas HARUS dinyatakan pada semua properti dan metode; abstrak dan Final HARUS dinyatakan sebelum visibilitas;
* Statis HARUS dinyatakan setelah visibilitas.
* Kata kunci struktur kontrol HARUS memiliki satu ruang setelah mereka; metode dan fungsi panggilan TIDAK BOLEH.
* Membuka tanda kurung untuk struktur kontrol TIDAK HARUS memiliki spasi setelahnya, dan tanda kurung tutup untuk struktur kontrol TIDAK HARUS memiliki spasi sebelumnya.

Contoh :

**<?php**

**namespace** Vendor\Package;

**use** FooInterface;

**use** BarClass **as** Bar;

**use** OtherVendor\OtherPackage\BazClass;

**class** **Foo** **extends** Bar **implements** FooInterface

{

**public** **function** **sampleMethod**($a, $b **=** **null**)

{

**if** ($a **===** $b) {

bar();

} **elseif** ($a **>** $b) {

$foo**->**bar($arg1);

} **else** {

BazClass**::**bar($arg2, $arg3);

}

}

**final** **public** **static** **function** **bar**()

{

*// method body*

}

}

1. PSR-3 : Logger Interface 🡪 menjelaskan tentang antarmuka umum untuk pustaka pencatatan log. Tujuan utamanya yaitu untuk memungkinkan library menerima Psr\Log\LoggerInterface objek dan menulis log ke dalamnya dengan cara yang sederhana dan universal. Hal ini untuk memastikan bahwa library pihak ketiga yang digunakan oleh aplikasi dapat menulis ke log aplikasi terpusat.

Contoh :

**<?php**

/\*\*

\* Interpolates context values into the message placeholders.

\*/

**function** **interpolate**($message, **array** $context **=** **array**())

{

*// build a replacement array with braces around the context keys*

$replace **=** **array**();

**foreach** ($context **as** $key **=>** $val) {

*// check that the value can be casted to string*

**if** (**!**is\_array($val) **&&** (**!**is\_object($val) **||** method\_exists($val, '\_\_toString'))) {

$replace['{' **.** $key **.** '}'] **=** $val;

}

}

*// interpolate replacement values into the message and return*

**return** strtr($message, $replace);

}

*// a message with brace-delimited placeholder names*

$message **=** "User {username} created";

*// a context array of placeholder names => replacement values*

$context **=** **array**('username' **=>** 'bolivar');

*// echoes "User bolivar created"*

**echo** interpolate($message, $context);

1. PSR-4 : Autoloader 🡪 menjelaskan spesifikasi untuk kelas autoloading dari file path.

Contoh Class :

<?php

namespace Example;

/\*\*

\* An example of a general-purpose implementation that includes the optional

\* functionality of allowing multiple base directories for a single namespace

\* prefix.

\*

\* Given a foo-bar package of classes in the file system at the following

\* paths ...

\*

\* /path/to/packages/foo-bar/

\* src/

\* Baz.php # Foo\Bar\Baz

\* Qux/

\* Quux.php # Foo\Bar\Qux\Quux

\* tests/

\* BazTest.php # Foo\Bar\BazTest

\* Qux/

\* QuuxTest.php # Foo\Bar\Qux\QuuxTest

\*

\* ... add the path to the class files for the \Foo\Bar\ namespace prefix

\* as follows:

\*

\* <?php

\* // instantiate the loader

\* $loader = new \Example\Psr4AutoloaderClass;

\*

\* // register the autoloader

\* $loader->register();

\*

\* // register the base directories for the namespace prefix

\* $loader->addNamespace('Foo\Bar', '/path/to/packages/foo-bar/src');

\* $loader->addNamespace('Foo\Bar', '/path/to/packages/foo-bar/tests');

\*

\* The following line would cause the autoloader to attempt to load the

\* \Foo\Bar\Qux\Quux class from /path/to/packages/foo-bar/src/Qux/Quux.php:

\*

\* <?php

\* new \Foo\Bar\Qux\Quux;

\*

\* The following line would cause the autoloader to attempt to load the

\* \Foo\Bar\Qux\QuuxTest class from /path/to/packages/foo-bar/tests/Qux/QuuxTest.php:

\*

\* <?php

\* new \Foo\Bar\Qux\QuuxTest;

\*/

class Psr4AutoloaderClass

{

/\*\*

\* An associative array where the key is a namespace prefix and the value

\* is an array of base directories for classes in that namespace.

\*

\* @var array

\*/

protected $prefixes = array();

/\*\*

\* Register loader with SPL autoloader stack.

\*

\* @return void

\*/

public function register()

{

spl\_autoload\_register(array($this, 'loadClass'));

}

/\*\*

\* Adds a base directory for a namespace prefix.

\*

\* @param string $prefix The namespace prefix.

\* @param string $base\_dir A base directory for class files in the

\* namespace.

\* @param bool $prepend If true, prepend the base directory to the stack

\* instead of appending it; this causes it to be searched first rather

\* than last.

\* @return void

\*/

public function addNamespace($prefix, $base\_dir, $prepend = false)

{

// normalize namespace prefix

$prefix = trim($prefix, '\\') . '\\';

// normalize the base directory with a trailing separator

$base\_dir = rtrim($base\_dir, DIRECTORY\_SEPARATOR) . '/';

// initialize the namespace prefix array

if (isset($this->prefixes[$prefix]) === false) {

$this->prefixes[$prefix] = array();

}

// retain the base directory for the namespace prefix

if ($prepend) {

array\_unshift($this->prefixes[$prefix], $base\_dir);

} else {

array\_push($this->prefixes[$prefix], $base\_dir);

}

}

/\*\*

\* Loads the class file for a given class name.

\*

\* @param string $class The fully-qualified class name.

\* @return mixed The mapped file name on success, or boolean false on

\* failure.

\*/

public function loadClass($class)

{

// the current namespace prefix

$prefix = $class;

// work backwards through the namespace names of the fully-qualified

// class name to find a mapped file name

while (false !== $pos = strrpos($prefix, '\\')) {

// retain the trailing namespace separator in the prefix

$prefix = substr($class, 0, $pos + 1);

// the rest is the relative class name

$relative\_class = substr($class, $pos + 1);

// try to load a mapped file for the prefix and relative class

$mapped\_file = $this->loadMappedFile($prefix, $relative\_class);

if ($mapped\_file) {

return $mapped\_file;

}

// remove the trailing namespace separator for the next iteration

// of strrpos()

$prefix = rtrim($prefix, '\\');

}

// never found a mapped file

return false;

}

/\*\*

\* Load the mapped file for a namespace prefix and relative class.

\*

\* @param string $prefix The namespace prefix.

\* @param string $relative\_class The relative class name.

\* @return mixed Boolean false if no mapped file can be loaded, or the

\* name of the mapped file that was loaded.

\*/

protected function loadMappedFile($prefix, $relative\_class)

{

// are there any base directories for this namespace prefix?

if (isset($this->prefixes[$prefix]) === false) {

return false;

}

// look through base directories for this namespace prefix

foreach ($this->prefixes[$prefix] as $base\_dir) {

// replace the namespace prefix with the base directory,

// replace namespace separators with directory separators

// in the relative class name, append with .php

$file = $base\_dir

. str\_replace('\\', '/', $relative\_class)

. '.php';

// if the mapped file exists, require it

if ($this->requireFile($file)) {

// yes, we're done

return $file;

}

}

// never found it

return false;

}

/\*\*

\* If a file exists, require it from the file system.

\*

\* @param string $file The file to require.

\* @return bool True if the file exists, false if not.

\*/

protected function requireFile($file)

{

if (file\_exists($file)) {

require $file;

return true;

}

return false;

}

}

1. PSR-6 : Caching Interface

Caching adalah cara umum untuk meningkatkan kinerja dari setiap project, pembuatan caching libraries salah satu fitur paling umum dari framework dan library. Tujuan PSR ini untuk memungkinkan developer untuk membuat pustaka cache-aware yang dapat diintegrasikan ke dalam kerangka kerja dan system yang ada tanpa perlu pengembangan kustom.

Contoh :

**<?php**

**namespace** Psr\Cache;

/\*\*

\* CacheItemInterface defines an interface for interacting with objects inside a cache.

\*/

**interface** CacheItemInterface

{

/\*\*

\* Returns the key for the current cache item.

\*

\* The key is loaded by the Implementing Library, but should be available to

\* the higher level callers when needed.

\*

\* @return string

\* The key string for this cache item.

\*/

**public** **function** **getKey**();

/\*\*

\* Retrieves the value of the item from the cache associated with this object's key.

\*

\* The value returned must be identical to the value originally stored by set().

\*

\* If isHit() returns false, this method MUST return null. Note that null

\* is a legitimate cached value, so the isHit() method SHOULD be used to

\* differentiate between "null value was found" and "no value was found."

\*

\* @return mixed

\* The value corresponding to this cache item's key, or null if not found.

\*/

**public** **function** **get**();

/\*\*

\* Confirms if the cache item lookup resulted in a cache hit.

\*

\* Note: This method MUST NOT have a race condition between calling isHit()

\* and calling get().

\*

\* @return bool

\* True if the request resulted in a cache hit. False otherwise.

\*/

**public** **function** **isHit**();

/\*\*

\* Sets the value represented by this cache item.

\*

\* The $value argument may be any item that can be serialized by PHP,

\* although the method of serialization is left up to the Implementing

\* Library.

\*

\* @param mixed $value

\* The serializable value to be stored.

\*

\* @return static

\* The invoked object.

\*/

**public** **function** **set**($value);

/\*\*

\* Sets the expiration time for this cache item.

\*

\* @param \DateTimeInterface|null $expiration

\* The point in time after which the item MUST be considered expired.

\* If null is passed explicitly, a default value MAY be used. If none is set,

\* the value should be stored permanently or for as long as the

\* implementation allows.

\*

\* @return static

\* The called object.

\*/

**public** **function** **expiresAt**($expiration);

/\*\*

\* Sets the expiration time for this cache item.

\*

\* @param int|\DateInterval|null $time

\* The period of time from the present after which the item MUST be considered

\* expired. An integer parameter is understood to be the time in seconds until

\* expiration. If null is passed explicitly, a default value MAY be used.

\* If none is set, the value should be stored permanently or for as long as the

\* implementation allows.

\*

\* @return static

\* The called object.

\*/

**public** **function** **expiresAfter**($time);

}

1. PSR-7: HTTP Message Interface

Dokumen ini menjelaskan antarmuka umum untuk mewakili pesan HTTP. Pesan HTTP adalah fondasi pengembangan web. Browser web dan HTTP klien seperti cURL membuat pesan permintaan HTTP yang dikirim ke server web, yang menyediakan pesan respons HTTP. Kode sisi server menerima HTTP meminta pesan, dan mengembalikan pesan tanggapan HTTP. Pesan HTTP biasanya diabstraksikan dari pengguna pengguna akhir, tetapi sebagai pengembang, kita biasanya perlu tahu bagaimana mereka terstruktur dan bagaimana caranya mengakses atau memanipulasi mereka untuk melakukan tugas kami, apakah itu mungkin membuat permintaan ke API HTTP, atau menangani permintaan masuk.

Contoh Psr\Http\Message\MessageInterface

**<?php**

**namespace** Psr\Http\Message;

/\*\*

\* HTTP messages consist of requests from a client to a server and responses

\* from a server to a client. This interface defines the methods common to

\* each.

\*

\* Messages are considered immutable; all methods that might change state MUST

\* be implemented such that they retain the internal state of the current

\* message and return an instance that contains the changed state.

\*

\* @see http://www.ietf.org/rfc/rfc7230.txt

\* @see http://www.ietf.org/rfc/rfc7231.txt

\*/

**interface** MessageInterface

{

/\*\*

\* Retrieves the HTTP protocol version as a string.

\*

\* The string MUST contain only the HTTP version number (e.g., "1.1", "1.0").

\*

\* @return string HTTP protocol version.

\*/

**public** **function** **getProtocolVersion**();

/\*\*

\* Return an instance with the specified HTTP protocol version.

\*

\* The version string MUST contain only the HTTP version number (e.g.,

\* "1.1", "1.0").

\*

\* This method MUST be implemented in such a way as to retain the

\* immutability of the message, and MUST return an instance that has the

\* new protocol version.

\*

\* @param string $version HTTP protocol version

\* @return static

\*/

**public** **function** **withProtocolVersion**($version);

/\*\*

\* Retrieves all message header values.

\*

\* The keys represent the header name as it will be sent over the wire, and

\* each value is an array of strings associated with the header.

\*

\* // Represent the headers as a string

\* foreach ($message->getHeaders() as $name => $values) {

\* echo $name . ': ' . implode(', ', $values);

\* }

\*

\* // Emit headers iteratively:

\* foreach ($message->getHeaders() as $name => $values) {

\* foreach ($values as $value) {

\* header(sprintf('%s: %s', $name, $value), false);

\* }

\* }

\*

\* While header names are not case-sensitive, getHeaders() will preserve the

\* exact case in which headers were originally specified.

\*

\* @return string[][] Returns an associative array of the message's headers.

\* Each key MUST be a header name, and each value MUST be an array of

\* strings for that header.

\*/

**public** **function** **getHeaders**();

/\*\*

\* Checks if a header exists by the given case-insensitive name.

\*

\* @param string $name Case-insensitive header field name.

\* @return bool Returns true if any header names match the given header

\* name using a case-insensitive string comparison. Returns false if

\* no matching header name is found in the message.

\*/

**public** **function** **hasHeader**($name);

/\*\*

\* Retrieves a message header value by the given case-insensitive name.

\*

\* This method returns an array of all the header values of the given

\* case-insensitive header name.

\*

\* If the header does not appear in the message, this method MUST return an

\* empty array.

\*

\* @param string $name Case-insensitive header field name.

\* @return string[] An array of string values as provided for the given

\* header. If the header does not appear in the message, this method MUST

\* return an empty array.

\*/

**public** **function** **getHeader**($name);

/\*\*

\* Retrieves a comma-separated string of the values for a single header.

\*

\* This method returns all of the header values of the given

\* case-insensitive header name as a string concatenated together using

\* a comma.

\*

\* NOTE: Not all header values may be appropriately represented using

\* comma concatenation. For such headers, use getHeader() instead

\* and supply your own delimiter when concatenating.

\*

\* If the header does not appear in the message, this method MUST return

\* an empty string.

\*

\* @param string $name Case-insensitive header field name.

\* @return string A string of values as provided for the given header

\* concatenated together using a comma. If the header does not appear in

\* the message, this method MUST return an empty string.

\*/

**public** **function** **getHeaderLine**($name);

/\*\*

\* Return an instance with the provided value replacing the specified header.

\*

\* While header names are case-insensitive, the casing of the header will

\* be preserved by this function, and returned from getHeaders().

\*

\* This method MUST be implemented in such a way as to retain the

\* immutability of the message, and MUST return an instance that has the

\* new and/or updated header and value.

\*

\* @param string $name Case-insensitive header field name.

\* @param string|string[] $value Header value(s).

\* @return static

\* @throws \InvalidArgumentException for invalid header names or values.

\*/

**public** **function** **withHeader**($name, $value);

/\*\*

\* Return an instance with the specified header appended with the given value.

\*

\* Existing values for the specified header will be maintained. The new

\* value(s) will be appended to the existing list. If the header did not

\* exist previously, it will be added.

\*

\* This method MUST be implemented in such a way as to retain the

\* immutability of the message, and MUST return an instance that has the

\* new header and/or value.

\*

\* @param string $name Case-insensitive header field name to add.

\* @param string|string[] $value Header value(s).

\* @return static

\* @throws \InvalidArgumentException for invalid header names.

\* @throws \InvalidArgumentException for invalid header values.

\*/

**public** **function** **withAddedHeader**($name, $value);

/\*\*

\* Return an instance without the specified header.

\*

\* Header resolution MUST be done without case-sensitivity.

\*

\* This method MUST be implemented in such a way as to retain the

\* immutability of the message, and MUST return an instance that removes

\* the named header.

\*

\* @param string $name Case-insensitive header field name to remove.

\* @return static

\*/

**public** **function** **withoutHeader**($name);

/\*\*

\* Gets the body of the message.

\*

\* @return StreamInterface Returns the body as a stream.

\*/

**public** **function** **getBody**();

/\*\*

\* Return an instance with the specified message body.

\*

\* The body MUST be a StreamInterface object.

\*

\* This method MUST be implemented in such a way as to retain the

\* immutability of the message, and MUST return a new instance that has the

\* new body stream.

\*

\* @param StreamInterface $body Body.

\* @return static

\* @throws \InvalidArgumentException When the body is not valid.

\*/

**public** **function** **withBody**(StreamInterface $body);

}

1. PSR-11 : Container Interface

Dokumen ini menjelaskan antarmuka umum untuk wajah injeksi ketergantungan. Tujuannya untuk membakukan bagaimana kerangka kerja dan pustaka memanfaatkan container untuk mendapatkan objek dan parameter.

Contoh Psr\Container\ContainerInterface :

**<?php**

**namespace** Psr\Container;

/\*\*

\* Describes the interface of a container that exposes methods to read its entries.

\*/

**interface** ContainerInterface

{

/\*\*

\* Finds an entry of the container by its identifier and returns it.

\*

\* @param string $id Identifier of the entry to look for.

\*

\* @throws NotFoundExceptionInterface No entry was found for \*\*this\*\* identifier.

\* @throws ContainerExceptionInterface Error while retrieving the entry.

\*

\* @return mixed Entry.

\*/

**public** **function** **get**($id);

/\*\*

\* Returns true if the container can return an entry for the given identifier.

\* Returns false otherwise.

\*

\* `has($id)` returning true does not mean that `get($id)` will not throw an exception.

\* It does however mean that `get($id)` will not throw a `NotFoundExceptionInterface`.

\*

\* @param string $id Identifier of the entry to look for.

\*

\* @return bool

\*/

**public** **function** **has**($id);

}

1. PSR-13 : Hypermedia Links

Tautan hypermedia menjadi bagian yang semakin penting dari web, baik dalam konteks HTML dan berbagai konteks format API. Namun, tidak ada satu pun format hypermedia yang umum, dan juga cara umum untuk merepresentasikan tautan antar format. Spesifikasi ini bertujuan untuk menyediakan pengembang PHP dengan cara umum yang sederhana untuk merepresentasikan sebuah hypermedia link secara terpisah dari format serialisasi yang digunakan. Sehingga memungkinkan sebuah sistem untuk membuat serial tanggapan dengan hypermedia link menjadi satu atau lebih format kawat secara mandiri dari proses memutuskan apa yang seharusnya menjadi tautan tersebut.

Contoh Psr\Link\LinkInterface :

**<?php**

**namespace** Psr\Link;

/\*\*

\* A readable link object.

\*/

**interface** LinkInterface

{

/\*\*

\* Returns the target of the link.

\*

\* The target link must be one of:

\* - An absolute URI, as defined by RFC 5988.

\* - A relative URI, as defined by RFC 5988. The base of the relative link

\* is assumed to be known based on context by the client.

\* - A URI template as defined by RFC 6570.

\*

\* If a URI template is returned, isTemplated() MUST return True.

\*

\* @return string

\*/

**public** **function** **getHref**();

/\*\*

\* Returns whether or not this is a templated link.

\*

\* @return bool

\* True if this link object is templated, False otherwise.

\*/

**public** **function** **isTemplated**();

/\*\*

\* Returns the relationship type(s) of the link.

\*

\* This method returns 0 or more relationship types for a link, expressed

\* as an array of strings.

\*

\* @return string[]

\*/

**public** **function** **getRels**();

/\*\*

\* Returns a list of attributes that describe the target URI.

\*

\* @return array

\* A key-value list of attributes, where the key is a string and the value

\* is either a PHP primitive or an array of PHP strings. If no values are

\* found an empty array MUST be returned.

\*/

**public** **function** **getAttributes**();

}

1. PSR-15 : HTTP Server Request Handlers

Dokumen ini menjelaskan antarmuka umum untuk penangan permintaan server HTTP ("Permintaan penangan") dan komponen middleware server HTTP ("middleware") yang menggunakan pesan HTTP. Penangan permintaan HTTP adalah bagian mendasar dari aplikasi web apa pun. Sisi server kode menerima pesan permintaan, memprosesnya, dan menghasilkan pesan respons. HTTP middleware adalah cara untuk memindahkan permintaan umum dan pemrosesan respons dari lapisan aplikasi. Antarmuka yang dijelaskan dalam dokumen ini adalah abstraksi untuk penangan permintaan dan middleware.

Contoh Psr\Http\Server\RequestHandlerInterface :

**namespace** Psr\Http\Server;

**use** Psr\Http\Message\ResponseInterface;

**use** Psr\Http\Message\ServerRequestInterface;

/\*\*

\* Handles a server request and produces a response

\*

\* An HTTP request handler process an HTTP request in order to produce an

\* HTTP response.

\*/

**interface** RequestHandlerInterface

{

/\*\*

\* Handles a request and produces a response

\*

\* May call other collaborating code to generate the response.

\*/

**public** **function** **handle**(ServerRequestInterface $request)**:** ResponseInterface;

}

1. PSR-16 : Common Interface for Caching Libraries

Dokumen ini menjelaskan antarmuka yang sederhana namun dapat diperluas untuk item cache dan cache driver. Caching adalah cara umum untuk meningkatkan kinerja dari setiap proyek, pembuatan caching libraries salah satu fitur paling umum dari banyak framework dan library.

Contoh :

**<?php**

**namespace** Psr\SimpleCache;

**interface** CacheInterface

{

/\*\*

\* Fetches a value from the cache.

\*

\* @param string $key The unique key of this item in the cache.

\* @param mixed $default Default value to return if the key does not exist.

\*

\* @return mixed The value of the item from the cache, or $default in case of cache miss.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

**public** **function** **get**($key, $default **=** **null**);

/\*\*

\* Persists data in the cache, uniquely referenced by a key with an optional expiration TTL time.

\*

\* @param string $key The key of the item to store.

\* @param mixed $value The value of the item to store, must be serializable.

\* @param null|int|\DateInterval $ttl Optional. The TTL value of this item. If no value is sent and

\* the driver supports TTL then the library may set a default value

\* for it or let the driver take care of that.

\*

\* @return bool True on success and false on failure.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

**public** **function** **set**($key, $value, $ttl **=** **null**);

/\*\*

\* Delete an item from the cache by its unique key.

\*

\* @param string $key The unique cache key of the item to delete.

\*

\* @return bool True if the item was successfully removed. False if there was an error.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

**public** **function** **delete**($key);

/\*\*

\* Wipes clean the entire cache's keys.

\*

\* @return bool True on success and false on failure.

\*/

**public** **function** **clear**();

/\*\*

\* Obtains multiple cache items by their unique keys.

\*

\* @param iterable $keys A list of keys that can obtained in a single operation.

\* @param mixed $default Default value to return for keys that do not exist.

\*

\* @return iterable A list of key => value pairs. Cache keys that do not exist or are stale will have $default as value.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $keys is neither an array nor a Traversable,

\* or if any of the $keys are not a legal value.

\*/

**public** **function** **getMultiple**($keys, $default **=** **null**);

/\*\*

\* Persists a set of key => value pairs in the cache, with an optional TTL.

\*

\* @param iterable $values A list of key => value pairs for a multiple-set operation.

\* @param null|int|\DateInterval $ttl Optional. The TTL value of this item. If no value is sent and

\* the driver supports TTL then the library may set a default value

\* for it or let the driver take care of that.

\*

\* @return bool True on success and false on failure.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $values is neither an array nor a Traversable,

\* or if any of the $values are not a legal value.

\*/

**public** **function** **setMultiple**($values, $ttl **=** **null**);

/\*\*

\* Deletes multiple cache items in a single operation.

\*

\* @param iterable $keys A list of string-based keys to be deleted.

\*

\* @return bool True if the items were successfully removed. False if there was an error.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $keys is neither an array nor a Traversable,

\* or if any of the $keys are not a legal value.

\*/

**public** **function** **deleteMultiple**($keys);

/\*\*

\* Determines whether an item is present in the cache.

\*

\* NOTE: It is recommended that has() is only to be used for cache warming type purposes

\* and not to be used within your live applications operations for get/set, as this method

\* is subject to a race condition where your has() will return true and immediately after,

\* another script can remove it making the state of your app out of date.

\*

\* @param string $key The cache item key.

\*

\* @return bool

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

**public** **function** **has**($key);

}

1. PSR-17 : HTTP Factories

Dokumen ini menjelaskan standar umum untuk pabrik yang membuat PSR-7 objek HTTP yang sesuai. Pabrik HTTP adalah metode yang digunakan untuk objek HTTP baru, seperti yang didefinisikan oleh PSR-7. Pabrik HTTP harus mengimplementasikan antarmuka ini untuk setiap jenis objek yang disediakan oleh paket.

Contoh (RequestFactoryInterface ) :

**namespace** Psr\Http\Message;

**use** Psr\Http\Message\RequestInterface;

**use** Psr\Http\Message\UriInterface;

**interface** RequestFactoryInterface

{

/\*\*

\* Create a new request.

\*

\* @param string $method The HTTP method associated with the request.

\* @param UriInterface|string $uri The URI associated with the request.

\*/

**public** **function** **createRequest**(string $method, $uri)**:** RequestInterface;

}

1. Sebutkan dan jelaskan tipe data dalam PHP, serta berikan contohnya

Jawab.

1. Integer 🡪 tipe bilangan bulat atau bilangan yang tidak memiliki nilai di belakang koma.

Contoh :

<?php

$a = 12345;

$b = 5000;

?>

1. Float 🡪 tipe data pada php yang memiliki bagian desimal di akhir angka.

Contoh :

<?php

$nilai1=4.5;

$nilai2=3.5;

$hasil=$nilai1+$nilai2;

echo $hasil;

?>

1. String 🡪 type data pada php yang berisi text dan karakter dimana bentuknya bisa kata atau kalimat. Cara penulisan string dalam PHP :

* Single quoted 🡪 dengan memberi tanda single quoted (‘) di awal dan di akhir kalimat yang kita buat.

Contoh :

<?php

$string1 = ‘Komsi’;

$string2 = ‘Sekolah Vokasi’;

$string 3 = ‘Prodi $string1 fakultas $string2’;

echo $string1; // output : Komsi

echo $string2; // output : Sekolah Vokasi

echo $string3; // output : Prodi $string1 fakultas $string2

?>

* Double quoted 🡪 dengan memberi tanda single quoted (“) di awal dan di akhir kalimat yang kita buat. Jika penulisan string menggunakan double quoted dan didalamnya terdapat variable maka php akan memprosesnya tapi jika menggunakan single quoted tidak. Apabila di dalam double quoted ada kalimat yang menggunakan tanda (“”) harus menggunakan backslash di awal dan akhir kalimat.

Contoh :

<?php

$string1 = “Komsi”;

$string2 = “Sekolah Vokasi”;

$string 3 = “Prodi $string1 fakultas $string2”;

echo $string1; // output : Komsi

echo $string2; // output : Sekolah Vokasi

echo $string3; // output : Prodi Komsi fakultas Sekolah Vokasi

?>

1. Boolean 🡪 hanya memiliki dua nilai yaitu true atau false.

Contoh :

<?php

$x = true;

$y = false;

echo $x; // output : 1

echo $y; // tidak ada output yg ditampilkan karena jika ditampilkan menggunakan echo, tipe data boolean “dipaksa” berganti dengan tipe data string.

?>

1. Array 🡪 tipe data yang didalamnya terdiri dari kumpulan tipe data.

Contoh :

<?php

$nama = array("Andri", "Joko", "Sukma", "Rina", "Sari");

echo $nama[1]; //Andri

echo "<br />";

echo $nama[2]; //Joko

?>

1. Null 🡪 Tipe Data yang satu ini hanya memiliki satu nilai yaitu NULL atau kosong.

Contoh :

<?php

$a = null;

?>

1. Object 🡪 tipe data yang menyimpan data dan informasi tentang cara mengolah data tersebut.

Contoh :

<?php

class Codepolitan {

function Codepolitan() {

$this->wijay = "Saya seorang Coder";

}

}

// mendefinisikan penggunaan objek

$soffi = new Codepolitan();

// mengeluarkan nilai objek

echo $soffi->wijay;

?>

1. Jelaskan masing-masing operator dan berikan contohnya

Jawab.

1. Operator perbandingan 🡪 digunakan untuk membandingkan dua buah nilai pada PHP dan hasilnya berupa boolen (true atau false).

|  |  |
| --- | --- |
| Operator | Keterangan |
| $a == $b | TRUE jika nilai $a dan $b sama tanpa membedakan tipe data nya |
| $a === $b | TRUE jika nilai dan tipe data $a dan $b sama |
| $a != $b | TRUE jika nilai $a dan $b tidak sama tanpa membedakan tipe data nya |
| $a <> $b | TRUE jika nilai $a dan $b tidak sama tanpa membedakan tipe data nya |
| $a !== $b | TRUE jika nilai dan tipe data $a dan $b tidak sama |
| $a < $b | TRUE jika nilai $a lebih kecil daripada $b |
| $a > $b | TRUE jika nilai $a lebih besar daripada $b |
| $a <= $b | TRUE jika nilai $a lebih kecil atau sama dengan $b |
| $a >= $b | TRUE jika nilai $a lebih besar atau sama dengan $b |
| $a <=> $b | TRUE jika nilai $a lebih besar, sama dengan, atau lebih besar dari $b . Mulai tersedia pada PHP 7 |

Contoh :

<?php

if ($\_POST['password'] == '1234')

{

echo 'Login sukses';

}

1. Operator aritmatika 🡪  operator matematis yang terdiri dari operator**penambahan, pengurangan, perkalian, pembagian, modulus, plus,**dan**minus.**

| **Operator** | **Keterangan** |
| --- | --- |
| $a + $b | Penjumlahan |
| $a – $b | Pengurangan |
| $a \* $b | Perkalian |
| $a / $b | Pembagian |
| $a % $b | *Modulus*. Sisa bagi antara $a dan $b |
| $a \*\* $b | *Exponentiation*. Diperkenalkan mulai PHP 5.6 |

Contoh :

<?php

for ($i = 1; $i <= 10, $i++) {

if ($i % 2) {

echo '<div class="baris-ganjil"></div>';

} else {

echo '<div></div>';

}

}

1. Operator prioritas 🡪 operator yang memiliki prioritas tertentu, misalnya operator perkalian dan pembagian lebih diutamakan daripada penjumlahan dan pengurangan.

Contoh :

<?php

$a = 2 + 10 / 2;

echo $a; // Hasil 7

$b = 2 + 10 \* 2;

echo $a; // Hasil 22

1. Operator logika 🡪 untuk membandingkan dua buah nilai boolean (true atau false) dengan hasil nilai juga berupa boolean (true atau false). Daftar operator logika :

| **Operator** | **Keterangan** |
| --- | --- |
| $a AND $b | Bernilai TRUE jika $a dan $b bernilai TRUE |
| $a OR $b | Bernilai TRUE jika $a atau $b bernilai TRUE |
| $a && $b | Bernilai TRUE jika $a dan $b bernilai TRUE, $a memiliki prioritas |
| $a || $b | Bernilai TRUE jika $a atau $b bernilai TRUE, $a memiliki prioritas |
| $a XOR $b | Bernilai TRUE jika $a atau $b bernilai TRUE, tapi tidak keduanya |
| !$a | Not. Bernilai TRUE jika $a bernilai FALSE |

Contoh :

<?php

$level = 1;

if ($level == 1 || $level == 2) {

echo 'Selamat datang admin';

}

1. Increment dan decrement 🡪 untuk menambahkan atau mengurangkan nilai sebanyak 1 pada suatu variable.

| **Operator** | **Nama** | **Keterangan** |
| --- | --- | --- |
| ++$a | Pre Increment | Tambahkan 1 ke $a, kemudian tampilkan nilai $a |
| $a++ | Post Increment | Tampilkan nilai $a, kemudian tambahkan 1 ke $a |
| –$a | Pre Decrement | Kurangkan 1 dari $a, kemudian tampilkan nilai $a |
| $a– | Post Decrement | Tampilkan $a, kemudian kurangkan 1 dari $a |

Contoh :

<?php

$nomor = 1;

while($nomor <= 5) {

echo $nomor++;

} // hasil 12345

1. Operator assignment 🡪 untuk memberikan nilai pada suatu variable.

| **Operator** | **Alias** | **Contoh** |
| --- | --- | --- |
| $a += $b | $a = $a + $b | $a += 2 |
| $a -= $b | $a = $a – $b | $a -= 2 |
| $a \*= $b | $a = $a \* $b | $a \*= 2 |
| $a /= $b | $a = $a / $b | $a /= 2 |
| $a %= $b | $a = $a % $b | $a %= 2 |

Contoh :

<?php

$sql = 'SELECT \* FROM sales';

$query = mysqli\_query($sql);

$total = 0;

while($row = mysqli\_fetch\_array($query))

{

$total += $row['jml\_bayar'];

}