Slim
Cheat Sheet

Configuration

```
$app = new Slim(array('debug' => true));        // Set config. during instantiation
$app->config('debug', false);                   // Set config. after instantiation
$path = $app->config('templates.path');         // Get config.
```

Application settings

// config	// default value	// obs
mode	development	
debug	true	errors are converted into `ErrorException`
log.writer	\Slim\LogWriter	
log.level	\Slim\Log::DEBUG	use the constants defined in `\Slim\Log`
log.enabled	true	
templates.path	"./templates"	
view	\Slim\View	
cookies.encrypt	false	
cookies.lifetime	"20 minutes"	
cookies.path	"/"	
cookies.domain	null	
cookies.secure	false	
cookies.httponly	false	
cookies.secret_key	"CHANGE_ME"	
cookies.cipher	MCRYPT_RIJNDAEL_256	
cookies.cipher_mode	MCRYPT_MODE_CBC	
http.version	"1.1"	

Setting app mode

```
$_ENV['SLIM_MODE'] = 'production';          // Environment variable

$app = new \Slim\Slim(array(                // Application setting
    'mode' => 'production'
));
```

Configuration for specific mode

```
// Only invoked if mode is "production"
$app->configureMode('production', function () use ($app) {
    $app->config(array(
        'log.enable' => true,
        'debug' => false
    ));
});
```

Slim
Cheat Sheet

Routing

```
$app->get('/books/:id', function ($id) {            // GET
    // ...
});

$app->post('/books', function () {                  // POST
    // ...
});

$app->put('/books/:id', function ($id) {            // PUT
    // ...
});

$app->delete('/books/:id', function ($id) {         // DELETE
    // ...
});

$app->patch('/books/:id', function ($id) {          // PATCH
    // ...
});

$app->options('/books/:id', function ($id) {        // OPTIONS
    // ...
});
```

HTTP method override
(for PUT, DELETE, PATCH, OPTIONS, and custom methods)

// add a hidden field with: name = _METHOD and value = PUT | DELETE | PATCH | OPTIONS

```
<form action="/books/1" method="post">
    ...
    <input type="hidden" name="_METHOD" value="PUT"/>
    <input type="submit" value="Update Book"/>
</form>
```

Multiple HTTP methods in one route

```
$app->map('/foo/bar', function() {
    // ...
})->via('GET', 'POST');
```

Custom HTTP methods

```
$app->map('/books', function() {
    // ...
})->via('FOO');
```

Slim
Cheat Sheet

Route name

```
$app->map('/foo/bar', function() {
    // ...
})->via('GET', 'POST')->name('foo');
```

```
// and generate a URL for the named route
$url = $app->urlFor('foo', array('name' => 'Bla'));
```

Route conditions

```
$app->get('/books/:name', function ($name) {
    // ...
})->name('books')->conditions(array('name' => '\w+'));
```

Application-wide conditions

```
\Slim\Route::setDefaultConditions(array(
    'firstName' => '[a-zA-Z]{3,}'
));
```

Route parameters

```
$app->get('/books/:one/:two', function ($one, $two) {
    // ...
});
```

Wildcard route parameters

```
// A wildcard parameter is identified by a "+" suffix
$app->get('/books/:name+', function ($name) {
    // ...
});
```

E.g.:
URI "/books/Some/Parameters/Here"
$name = array('Some', 'Parameters', 'Here').

404

For not found routes Slim returns a 404 Not Found response.

Route helpers

```
$app->halt(404);
$app->halt(403, 'You shall not pass!');
$app->pass();
$app->redirect('/books');
$app->redirect('/new', 301);
$app->stop();
$app->urlFor('books', array('name' => 'Foo')); // dynamically create URL for a named route
```

```
//  immediately returns an HTTP response with
    a given status code and body
//  continue to the next matching route
// by default: 302 Temporary Redirect response
// permanent redirect
// stop the execution and send the current response
```

Slim
Cheat Sheet

Route middleware

```
function middleware1() {
    // ...
}
function middleware2() {
    // ...
}
$app = new \Slim\Slim();
$app->get('/books', 'middleware1', 'middleware2', function () {
    // ...
});
```

Route groups

```
$app->group('/api', function () use ($app) {

    $app->group('/library', function () use ($app) {

        $app->get('/books', function ($id) {
            // ...
        });

        $app->put('/books/:id', function ($id) {
            // ...
        });

        $app->delete('/books/:id', function ($id) {
            // ...
        });
    });
});
```

```
// GET    /api/library/books
// PUT    /api/library/books/:id
// DELETE  /api/library/books/:id
```

HTTP Caching

```
$app->etag('unique-id');                // send an If-None-Match header
$app->lastModified(1286139652);         // set last modified date
$app->expires('+1 week');               // set an Expires header
```

Flash Messages

```
$app->flash('error', 'Name is required');   // available in the next request's view
$app->flashNow('info', 'Expired');          // available in the current request's view
$app->flashKeep();                          // keep available to the next request
```

Slim
Cheat Sheet

Request

```
$app->request;                                  // instance of \Slim\Http\Request
$app->request->getMethod();                     // the request method, e.g. GET, POST, ...
$app->request->isGet();                         // true, if is a GET request
$app->request->isPost();                        // true, if is a POST request
$app->request->isPut();                         // true, if is a PUT request
$app->request->isDelete();                      // true, if is a DELETE request
$app->request->isHead();                        // true, if is a HEAD request
$app->request->isOptions();                     // true, if is a OPTIONS request
$app->request->isPatch();                       // true, if is a PATCH request
$app->request->isAjax();                        // true, if is a XHR/AJAX request
$app->request->headers;                         // get request headers as associative array
$app->request->headers->get('CACHE-CONTROL');   // get CACHE-CONTROL header
$app->request->params('name');                  // get a request variable value
$app->request->get('name');                     // fetch a GET variable
$app->request->post('name');                    // fetch a POST variable
$app->request->put('name');                     // fetch a PUT variable
$app->request->get();                           // all GET variables
$app->request->post();                          // all POST variables
$app->request->put();                           // all PUT variables
$app->request->getBody();                       // raw HTTP request body sent by the client
$app->request->cookies                          // complete set of HTTP request cookies
$app->request->getContentType();                // e.g. “application/json;charset=utf-8”
$app->request->getMediaType();                  // e.g.  “application/json”
$app->request->getMediaTypeParams();            //  e.g. [charset => “utf-8”]
$app->request->getContentCharset();             // e.g. “utf-8”
$app->request->getContentLength();              // fetch the content length
$app->request->getReferrer();                   // fetch the request’s referrer
$app->request->getUserAgent();                  // fetch the request’s user agent string
```

Slim
Cheat Sheet

Request - URL/URI Related

E.g.: http://localhost/site/admin/book/1

```
$app->request->getResourseUri();          //  /admin/book/1
$app->request->getPathInfo();             //  /admin/book/1
$app->request->getPath();                 //  /site/admin/book/1
$app->request->getRootUri();              //  /site
$app->request->getUrl();                  //  http://localhost
$app->request->getHost();                 //  localhost
$app->request->getHostWithPort();         //  localhost:80
$app->request->getPort();                 //  80
$app->request->getScheme();               //  http
$app->request->getIp();                   //  127.0.0.1
```

Response

```
$app->response                            // instance of \Slim\Http\Response
$app->response->setStatus(400);           // set the HTTP response's status
$app->response->getStatus();              // fetch the current HTTP status
$app->response->setBody('Foo');           // set the HTTP response's body
$app->response->write('Bar');             // append response body
$app->response->getBody();                // fetch the response object's body
$app->response->finalize();               // produce the status, header, and body
$app->response->redirect('/foo', 303);    // set response status and its Location: header
$app->response->isInformational();        // true, if is an informational response
$app->response->isOk();                   // true, if is a 200 OK response
$app->response>isSuccessful();            // true, if is a 2xx successful response
$app->response>isRedirection();           // true, if is a 3xx redirection response
$app->response->isRedirect();             // true, if is a specific redirect response
$app->response->isForbidden();            // true, if is a forbidden response
$app->response->isNotFound();             // true, if is a not found response
$app->response->isClientError();          // true, if is a client error response
$app->response->isServerError();          // true, if is a server error response
$app->response->headers->set('Content-Type', 'application/json');
$app->response->headers->get('Content-Type');
```

Slim
Cheat Sheet

Cookies

```
$app->response->cookies->set('foo', array(
    'value' => 'bar',
    'domain' => 'example.com',
    'path' => '/',
    'expires' => time() + 3600,
    'secure' => true,
    'httponly' => true
));
```

$app->setCookie($name, $value, $expiresAt, $path,$domain, $secure, $httponly);

$app->deleteCookie('foo', '/', 'foo.com', true, true);

$app->request->cookies->all();

$app->request->cookies->get('foo');

$app->response->cookies->set('foo', 'bar');

View

```
$app->render( 'myTemplate.php', array( 'name' => 'Foo' ), 404);
```

$app->view->setData('color' => 'red'); // overwrite existing view data

$app->view->appendData('foo' => 'bar'); // append data

Session

A Slim application does not presume anything about sessions.

// to persist session data in encrypted, hashed HTTP cookies:

```
$app->add(new \Slim\Middleware\SessionCookie(array(
    'expires' => '20 minutes',
    'path' => '/',
    'domain' => null,
    'secure' => false,
    'httponly' => false,
    'name' => 'slim_session',
    'secret' => 'CHANGE_ME',
    'cipher' => MCRYPT_RIJNDAEL_256,
    'cipher_mode' => MCRYPT_MODE_CBC
)));
```

Slim
Cheat Sheet

Hooks

```
$app->hook('the.hook.name', function () use ($app) {
    // Do something
});
```

```
slim.before              // before the Slim application is run and before output buffering is turned on
slim.before.router       // after output buffering is turned on and before the router is dispatched
slim.before.dispatch     // before the current matching route is dispatched
slim.after.dispatch      // after the current matching route is dispatched
slim.after.router        // after the router is dispatched, before the Response is sent to the client,
                            and after output buffering is turned off
slim.after               // after output buffering is turned off and after the Response is sent to
                            the client
```

Middlewares

```
// Middlewares can inspect, analyze, or modify the application environment, request, and
// response before and/or after the Slim application is invoked.
// They are called in the inverse order that they are definied.

class AllCapsMiddleware extends \Slim\Middleware
{
    public function call()
    {
        // Get reference to application
        $app = $this->app;

        // Run inner middleware and application
        $this->next->call();

        // Capitalize response body
        $res = $app->response;
        $body = $res->getBody();
        $res->setBody(strtoupper($body));
    }
}

// Add (enable) the middleware:
$app->add(new \AllCapsMiddleware());
```

Slim
Cheat Sheet

Logging

// Enable/disable during instantiation

```
$app = new Slim(array(
    'log.enabled' => true
));
```

or

```
$app->log->setEnabled(true);                    // Enable logging
$app->log->setEnabled(false);                   // Disable logging
```

Log data

```
$app->log->debug(mixed $object);
$app->log->info(mixed $object);
$app->log->notice(mixed $object);
$app->log->warning(mixed $object);
$app->log->error(mixed $object);
$app->log->critical(mixed $object);
$app->log->alert(mixed $object);
$app->log->emergency(mixed $object);
```

Log levels

```
\Slim\Log::EMERGENCY (Level 1)
\Slim\Log::ALERT (Level 2)
\Slim\Log::CRITICAL (Level 3)
\Slim\Log::ERROR (Level 4)
\Slim\Log::WARN (Level 5)
\Slim\Log::NOTICE (Level 6)
\Slim\Log::INFO (Level 7)
\Slim\Log::DEBUG (Level 8)
```

```
$app->log->setLevel(\Slim\Log::WARN);           // Set the log level
```

Slim
Cheat Sheet

Custom Error Handling

```
$app->error(function (\Exception $e) use ($app) {
    $app->render('error.php');
});
```

```
// invoking
$app->error();
```

Custom Not Found Handling

```
$app->notFound(function () use ($app) {
    $app->render('404.html');
});
```

```
// invoking
$app->notFound();
```

Dependency Injection

Simple values

```
$app->foo = 'bar';
```

Resource locator

```
// Set
$app->now = function () {
    return time();
};
```

```
// Get
$now = $app->now;
```

Singleton

```
// resource definitions stay the same each time they are requested

// Define
$app->container->singleton('db', function () {
    return new PDO('sqlite:database.db');
});
```

```
// Get
$pdo = $app->db;
```

Closure resources

```
// store closure as raw value and not have it invoked

// Define closure
$app->myClosure = $app->container->protect(function () {});

// Return raw closure without invoking it
$myClosure = $app->myClosure;
```