PHP

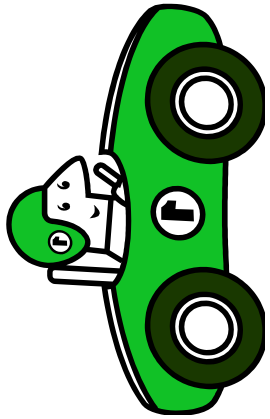
Objektorientierung, Dateizugriff, Dynamische Webseiten

Syntax
Objektorientierung

```php
class Auto {
    private $farbe = 'blau';
    private $ps;
    public static $speedLimit = 100;

    public function __construct($ps = 60) {
        $this->ps = $ps;
    }
    public function getPs() {
        return $this->ps;
    }
}

$auto = new Auto();         // Instanz der Klasse Auto
$geschw = $auto->getPs(); // 60
```



Syntax
Objektorientierung

Klassengestützte OOP

- Zugriff auf Membervariablen

```
$this->name     // Innerhalb der Klasse
$objekt->name   // Außerhalb der Klasse (falls public)
self::name      // Zugriff auf statische Variable
parent::name    // Zugriff auf Elternklasse (statisch)
```

- Konstruktor

```
function __construct() { … }
```

- Destruktor

```
function __destruct() { … }
```

Syntax
Errors & Exceptions

Exceptions

```
try {
    funcWhichMightThrowException();
} catch (Exception $e) {
    echo 'Problem calling ...: ' . $e->getMessage();
}
```

- Eigene Exception-Klassen können von Exception abgeleitet werden

```
class MyException extends Exception {...}
throw new MyException();
```

- Beim Abfangen kann der Exception-Name angegeben werden

- Die meisten PHP-Funktionen werfen im Fehlerfall keine Exception

Weitere Informationen: http://php.net/manual/language.exceptions.php

Syntax
Errors & Exceptions

Es gibt auch generische Exception-Handler-Funktionen!

```php
set_exception_handler('exception_handler');

function exception_handler($e) {
    echo 'Ups! Nicht behandelte Exception: ' .
        $e->getMessage();
}
throw new Exception('Problem aufgetreten!');
```

Hier bricht das Skript ab:

```
Ups! Nicht behandelte Exception: Problem aufgetreten!
```

Syntax
Errors & Exceptions

Unterschied: Errors & Exceptions

- Errors werden von PHP bei Fehlersituationen geworfen (und führen z.B. zum Abbruch des Skripts).

- Aber auch Errors können gefangen werden:

```
    errorHandler($errLevel, $errMsg, $errFile, $errLine) {
        throw new ErrorException($errMsg, $errLevel, $errLevel,
            $errFile, $errLine);
    }
    set_error_handler("errorHandler");
```

- Fatal-Errors können ebenfalls gefangen werden:

```
    register_shutdown_function("fatal_handler");
```

Weitere Informationen: https://www.sitepoint.com/error-handling-in-php/

Syntax
Dateizugriff

Datenhaltung meist über Datenbanken

- Dennoch manchmal Zugriff auf Dateien notwendig

```
resource fopen($filename, $mode)
```

> Öffnet Datei $filename

> $mode gibt den Zugriffstypen an: 'r' zum Lesen, 'w' zum Schreiben, 'a' zum Anhängen

> Tritt ein Fehler auf wird false zurückgegeben

```
bool fclose($handle)
```

> Schließt den Dateizeiger

> Rückgabewert true, wenn das Schließen funktioniert hat

Syntax
Dateizugriff

```
string fread($handle, $length)
```

> Liest von dem Dateizeiger bis zu $length Bytes

```
int fwrite($handle, $string)
```

> Schreibt den Inhalt von $string in die Datei, auf die der Dateizeiger zeigt
> Liefert Anzahl der geschriebenen Bytes zurück oder FALSE im Fehlerfall

```
bool file_exists($filename)
```

> Gibt TRUE zurück, wenn die Datei oder das Verzeichnis exisitert

```
int filesize($filename)
```

> Liefert die Größe der übergebenen Datei zurück.

Syntax
Dateizugriff

Vereinfachter Zugriff auf Dateien:

```
string file_get_contents($filename)
```

> Liefert den Inhalt der Datei $filename zurück

```
int file_put_contents($filename, $data)
```

> Schreibt $data in die Datei mit dem Namen $filename
> Wenn die Datei nicht exisitert, wird sie erstellt

Beispiel

```php
<?php
    $fileName    = 'title_form.html';
    $fileContent = file_get_contents($fileName);

    echo $fileContent; // Ausgabe des Dateiinhaltes
                       // (auch mit HTML-Tags)

    $fileContent = htmlspecialchars($fileContent);
    echo $fileContent; // Ausgabe mit „entschärften“
                       // HTML-Tags

?>
```

Dynamische Webseiten

Dynamische Webseiten

Statische Webseiten

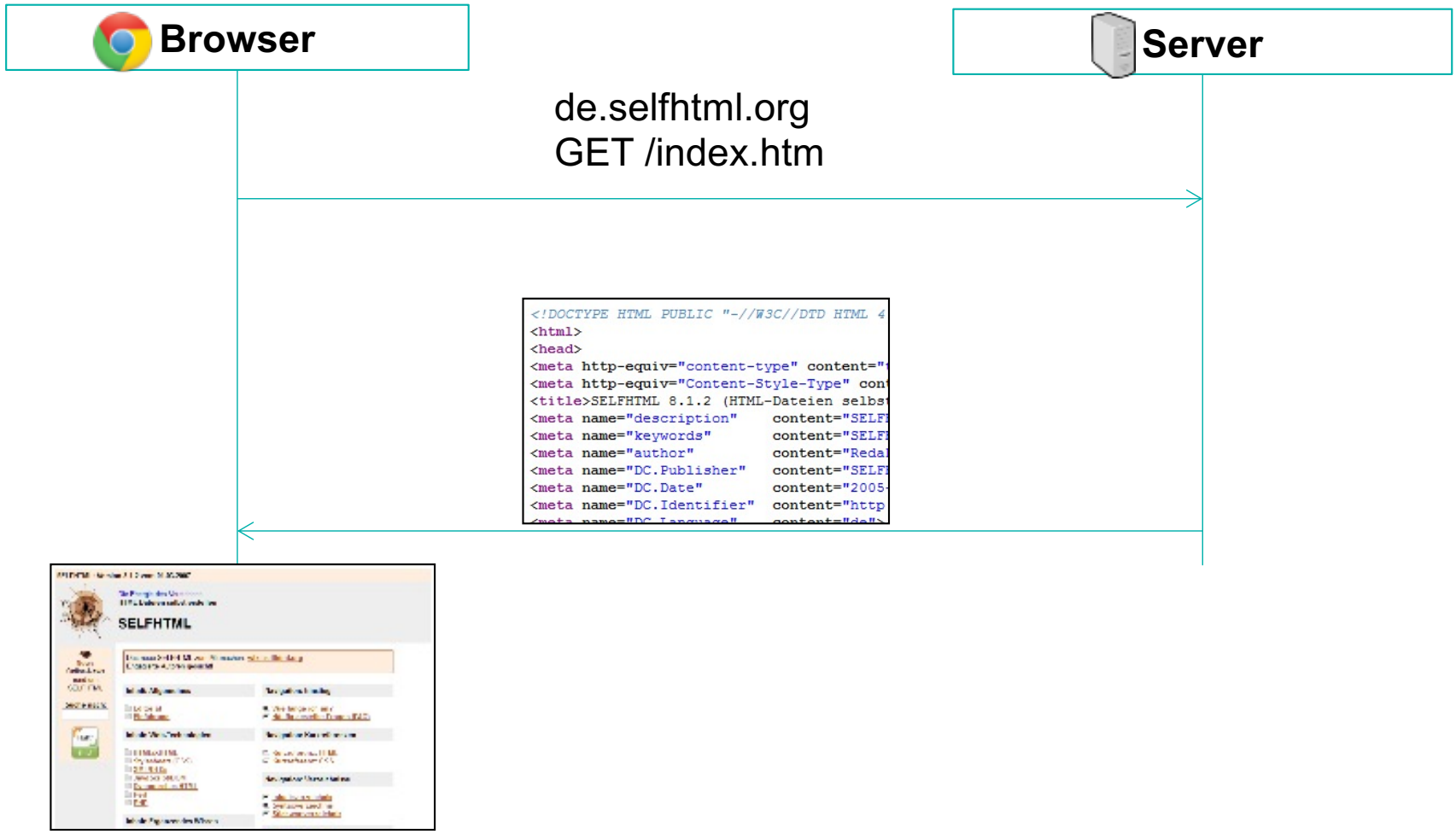
- HTML/CSS-Dateien auf dem Server
  - > Auslieferung an den Browser
  - > Stellen immer den gleichen Inhalt dar
  - > Beispiel: Webvisitenkarte

Dynamische Webseiten

- Erzeugung der Seite im Moment der Anforderung
  - > Darstellung aktueller Daten möglich
  - > Inhalt abhängig von Aktionen des Benutzers
  - > Beispiel: Suchformular

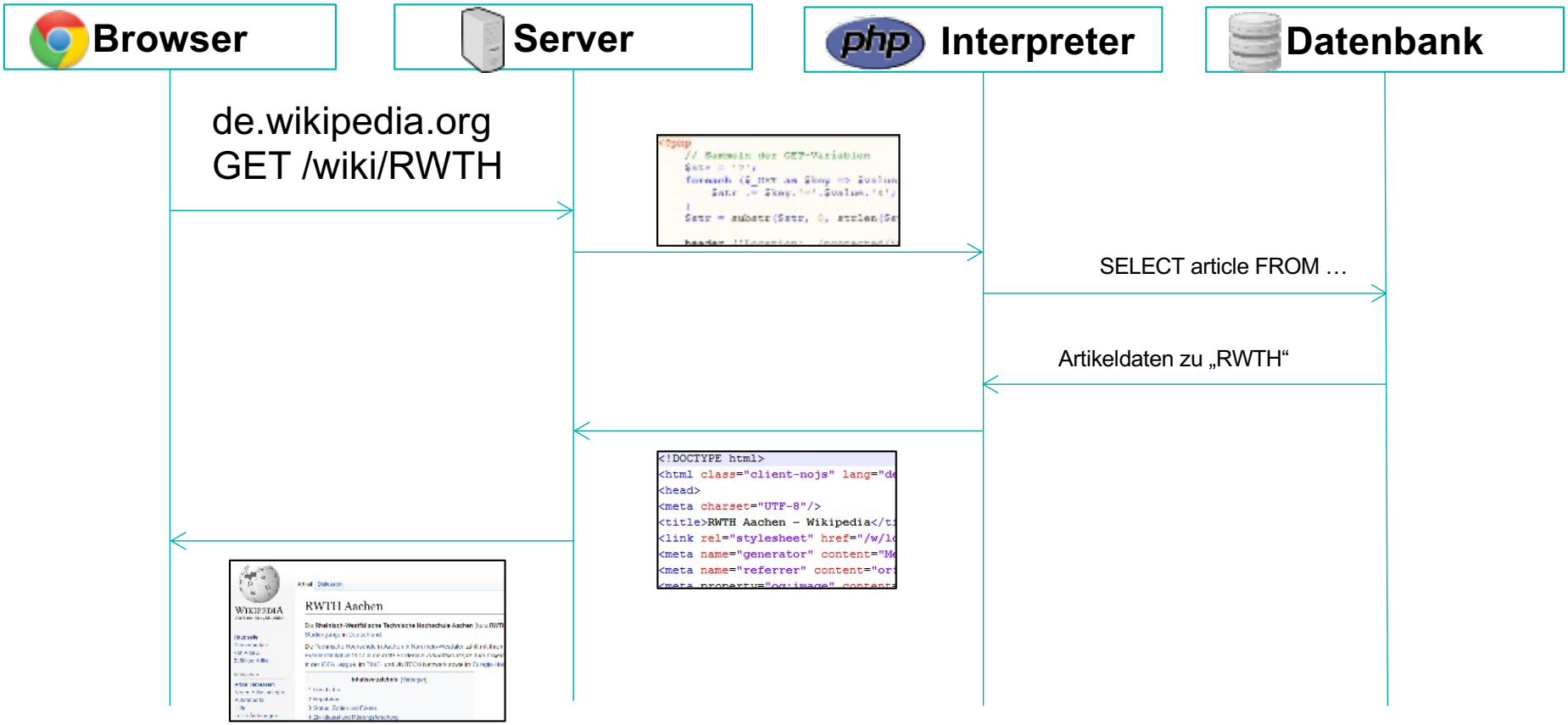
Dynamische Webseiten

Aufruf einer statischen Webseite (vereinfacht)



Dynamische Webseiten

Aufruf einer dynamischen Webseite (vereinfacht)



Dynamische Webseiten
Verarbeitung von Formularen

Beispiel: Login-Formular

```
<form action="do.php?q=login" method="post">
    <input type="text" name="username" />
    <input type="password" name="pw" />
    <input type="submit" value="Login" />
</form>
```

- Übermittelt Parameter „username“ und „pw“ per POST-Request

- Auswertung des GET-Parameters (q=login)
  > Bringt Variabilität in die URL, wir könnten beispielsweise über ein Script mehrere Methoden ansprechen
- Auswertung des Formulars
  > Ist „username“ und „pw“ korrekt?
  > Evtl. per Datenbankabfrage herausfinden?

Dynamische Webseiten
Verarbeitung von Formularen

GET-Request

- Übertragung der Daten über die Adresszeile
- Parameter werden durch das Zeichen ? In der URL eingeleitet
- Parameter werden in PHP im Array $_GET gespeichert
- Nicht geeignet zur Übertragung großer Datenmengen oder sensibler Daten

- Beispiel: `index.php?id=5&foo=bar`
  > Zugriff im PHP-Skript:

```php
$_GET['id']  // 5
$_GET['foo'] // bar
```

Dynamische Webseiten
Verarbeitung von Formularen

POST-Request

- Übertragung der Daten im HTTP-Body
  > Daten sind nicht in der URL sichtbar

- Zugriff auf POST-Daten in PHP:

```php
$_POST['text']   // liefert POST-Parameter text
```

Validierung der Nutzereingaben

- Beschränkungen des Clients können umgangen werden
- Beispiel:

```
<input type="text" name="plz" maxlength="5" />
```

- Parameter kann trotzdem mehr als 5 Zeichen haben
  - > User umgeht aktiv die Beschränkung
  - > Client unterstützt/versteht die Beschränkung nicht
- Serverseitige Prüfung von Nutzerdaten ist zwingend erforderlich

- **„Never trust the user“**

Filter-Funktionen

- Vereinfacht Validierung der Eingaben
- Liefern *null*, *false* oder eine nach *$filter* gültige Variable zurück
  - > *FILTER_VALIDATE_** validiert Eingaben (alles oder nichts!)
  - > *FILTER_SANITIZE_** korrigiert Eingaben

```
mixed filter_input($type, $variable_name, $filter
      [, $options])
```

> Filtert GET/POST-Eingaben

```
mixed filter_var($variable, $filter [, $options])
```

> Filtert Variablen oder Werte

Weitere Informationen: http://php.net/manual/en/filter.filters.php

Dynamische Webseiten
Filter-Funktionen

filter_input-Beispiel:

```
if (! filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL) ) {
        echo "E-Mail is not valid";
} else {
        echo "E-Mail is valid";
}
```

Hinweis: Es wird nur auf die syntaktische Korrektheit der eMail geprüft, nicht ob der Benutzer oder der Mail-Server existieren!

Dynamische Webseiten
Filter-Funktionen

filter_input-Beispiel:

```php
$heightInCm = filter_input(INPUT_GET, 'height', FILTER_VALIDATE_FLOAT);
if ($heightInCm === null) {
    // Parameter 'height' nicht gesetzt
} else if ($heightInCm === false) {
    // Parameter 'height' ist kein gültiger Float-Wert
} else {
    // Gültiger Float-Wert, $heightInCm ist ein Double!
}

$range = array (
    'options' => array ('min_range' => 1, 'max_range' => 100)
);
$choice = filter_input(INPUT_POST, 'choice', FILTER_VALIDATE_INT, $range);
// $choice ist nun null, false oder ein Integer im Bereich von 1 bis 100

$data = filter_input(INPUT_POST, 'data', FILTER_UNSAFE_RAW);
// Parameter 'data' so lassen, kann aber optional Zeichen speziell codieren
```