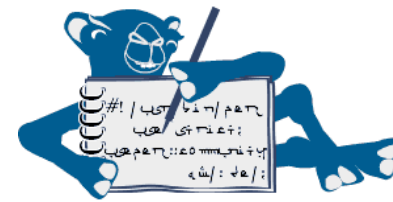


مركز  
community.de

# HTML::Template

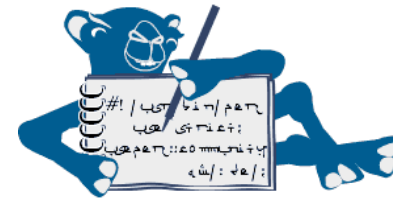
München, 17.04.2004

# Einführung



- Modul, für die Entwicklung von Perl-Webanwendungen
- Kann für jede beliebigen Text (nicht nur HTML) benutzt werden
- Vereinfacht das Design für dynamische Seiten
- z.B. neues Perl-Community.de-Board Urlaub-im-Ferienpark.de

# Einführungsbeispiel



```
<html>
  <head>
    <title>
      <!-- TEMPL_VAR NAME=WILLKOMMEN -->
    </title>
  </head>
  <body>
    <h1><!-- TEMPL_VAR NAME=WILLKOMMEN --></h1>
  </body>
</html>
```

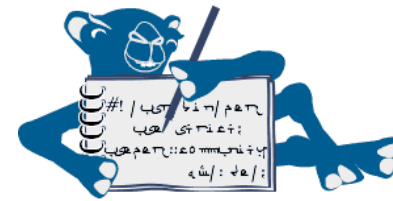


```
my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 'Willkommen bei der Perl-Community';

$template->param(WILLKOMMEN => $willkommen);

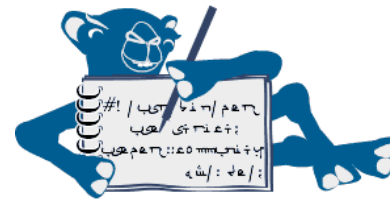
print $template->output();
```

# Einführungsbeispiel



```
<html>
  <head>
    <title>
      <!-- TMPL_VAR NAME=WILLKOMMEN -->
    </title>
  </head>
  <body>
    <h1><!-- TMPL_VAR NAME=WILLKOMMEN --></h1>
  </body>
</html>
```

# Einführungsbeispiel



```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;
use HTML::Template;

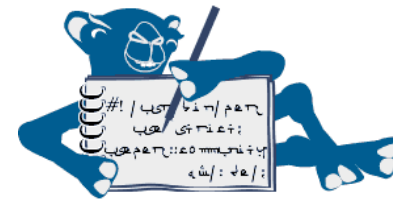
print „Content-type: text/html\n\n“;

my $template = HTML::Template->new(filename => './template.tmpl');
my $willkommen = 'Willkommen bei der Perl-Community';

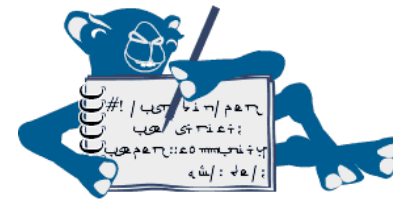
$template->param(WILLKOMMEN => $willkommen);

print $template->output();
```

# Einführungsbeispiel

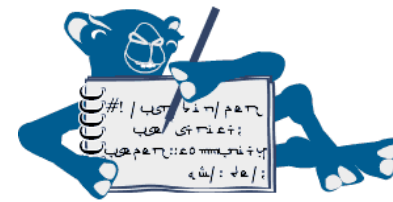


# Tags



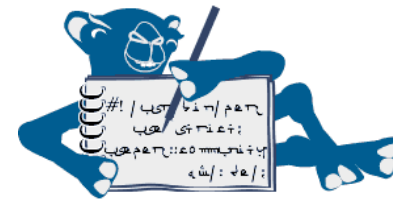
- Ähneln den „normalen“ HTML-Tags
  - `<TMPL_VAR NAME=NAME_DES_TAGS>`
- Können wahlweise auch als HTML-Kommentare geschrieben werden
  - `<TMPL_*>`
  - `<!-- TMPL_* -->`

# TMPL\_VAR



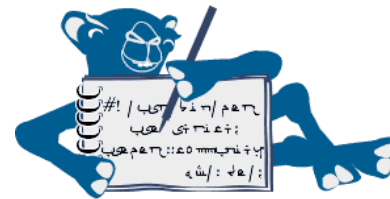
- Ist der „einfachste“ Tag
- Für jeden Tag, muss im Skript ein Parameter angegeben werden
- In der Ausgabe wird der Tag durch den Inhalt der Variable (Scalar) ersetzt
- Bei fehlendem Wert, wird der Tag einfach „gelöscht“

# TMPL\_VAR II



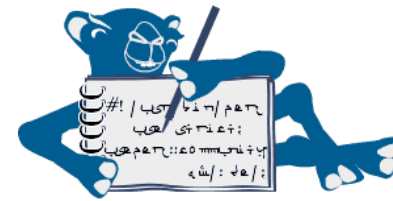
- Möglichkeit HTML zu escapen
  - `<!-- TMPL_VAR [...] ESCAPE=HTML -->`
- Möglichkeit Default-Wert anzugeben
  - `<!-- TMPL_VAR [...] DEFAULT=Wert -->`
- Besitzt *kein* schließendes Tag

# TMPL\_LOOP



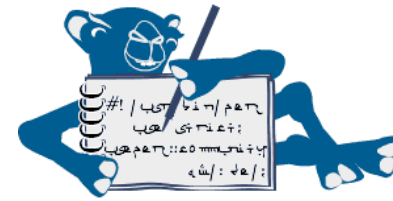
- Zur Darstellung von Tabellen geeignet
- Innerhalb der Schleife Verwendung von `TMPL_VARS`
- Zum Ersetzen Übergabe einer Arrayreferenz
- Arrayreferenz enthält Hashreferenzen

# TMPL\_LOOP II



- Schleife läuft über die Liste und ersetzt bei jedem Durchlauf die Variablen
- Schachtelung von mehreren Schleifen möglich
- hat ein schließendes Tag

# Beispiel TMPL\_LOOP



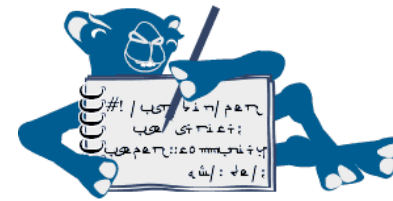
```
<html>
<head>
  <title>
    MITGLIEDER
  </title>
</head>
<body>
  <table border=1>
    <!-- TMPL_LOOP NAME=WILLKOMMEN --><tr>
      <td><!-- TMPL_VAR NAME=VORNAME --></td>
      <td><!-- TMPL_VAR NAME=NACHNAME --></td>
    </tr>
    <!-- /TMPL_LOOP -->
  </table>
</body>
</html>
```

Max	Mueller
Gabi	Kastenhuber

```
print $template->output();
```

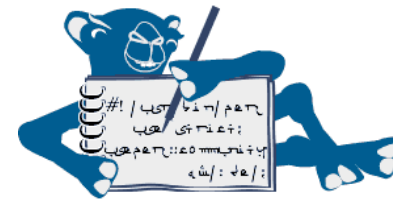
```
h\n";
template->new(filename => './template.tpl');
=> 'Max',
=> 'Mueller'},
=> 'Gabi',
=> 'Kastenhuber'}];
MEN => $willkommen);
```

# Beispiel TMPL\_LOOP



```
<html>
<head>
  <title>
    MITGLIEDER
  </title>
</head>
<body>
  <table border=1>
    <!-- TMPL_LOOP NAME=WILLKOMMEN --><tr>
      <td><!-- TMPL_VAR NAME=VORNAME --></td>
      <td><!-- TMPL_VAR NAME=NACHNAME --></td>
    </tr>
    <!-- /TMPL_LOOP -->
  </table>
</body>
</html>
```

# Beispiel TMPL\_LOOP



```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;
use HTML::Template;

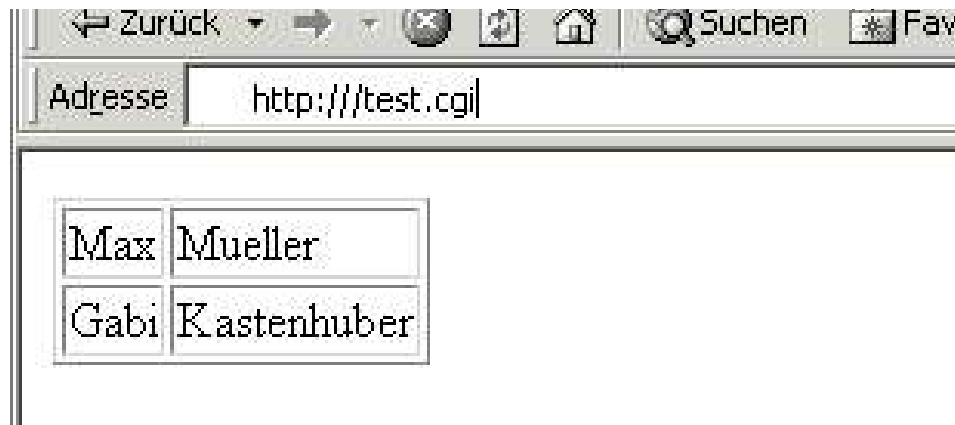
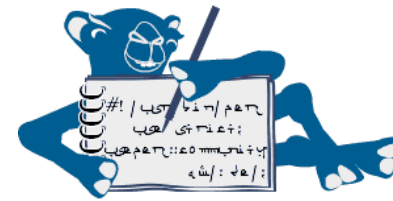
print „Content-type: text/html\n\n“;

my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = [{ Vorname => 'Max',
                    Nachname => 'Mueller' },
                  { Vorname => 'Gabi',
                    Nachname => 'Kastenhuber' }];

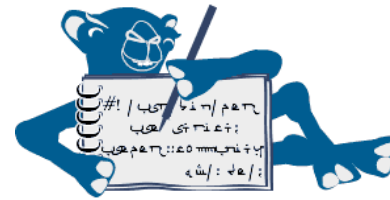
$template->param(WILLKOMMEN => $willkommen);

print $template->output();
```

# Beispiel TMPL\_LOOP

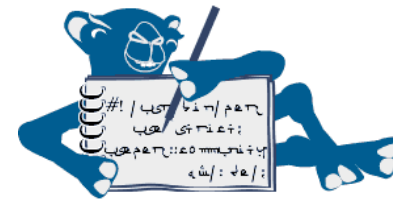


# TMPL\_INCLUDE



- Zum Einfügen eines weiteren Templates
- Tags des eingefügten Templates wie eigene Tags
- Maximale Tiefe von includes: 10
- max\_include einstellbar
- Kein abschließendes Tag

# Beispiel TMPL\_INCLUDE



```
<html>
<head>
  <title>
    TMPL_INCLUDE - Beispiel
  </title>
</head>
<body>
  <!-- TMPL_INCLUDE NAME=Template -->
</body>
</html>
```

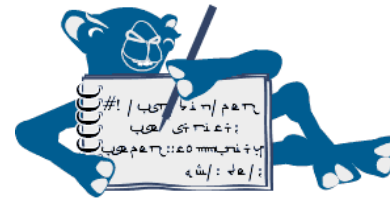
```
/html\n\n“;
```

```
my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 'Hallo';
my $scalar = 'Ersetzung';

$template->param(WILLKOMMEN => $willkommen,
                VARIABLE => $scalar);

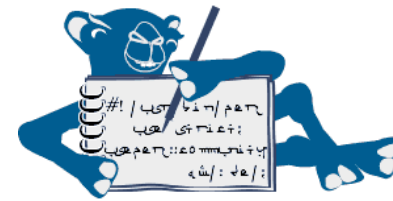
print $template->output();
```

# Beispiel TMPL\_INCLUDE



```
<html>
  <head>
    <title>
      TMPL_INCLUDE - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_INCLUDE NAME=Template -->
  </body>
</html>
```

# Beispiel TMPL\_INCLUDE



```
#!/usr/bin/perl

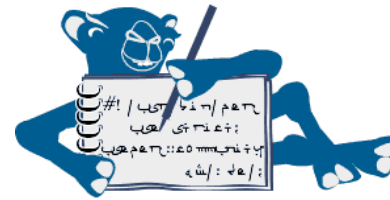
use strict;
use warnings;
use diagnostics;
use HTML::Template;

print „Content-type: text/html\n\n“;

my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 'Hallo';
my $scalar = 'Ersetzung';

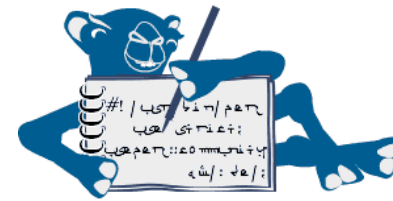
$template->param(WILLKOMMEN => $willkommen,
                VARIABLE => $scalar);
print $template->output();
```

# TMPL\_IF



- Teil zwischen den Tags wird nur ausgegeben, wenn die Variable einen Wahrheitswert liefert.
- Sollte nur benutzt werden, wenn im Skript kein if() benutzt wird
- abschließendes /TMPL\_IF

# Beispiel TMPL\_IF



```
<html>
<head>
  <title>
    TMPL_IF - Beispiel
  </title>
</head>
<body>
  <!-- TMPL_IF NAME=BOOL -->
  Text zwischen den If-Tags
  <!-- /TMPL_IF -->
</body>
</html>
```

```
/perl
```

```
ags;
ostics;
::Template;
```

```
Content-type: text/html\n\n“;
```

```
template = HTML::Template->new(filename => './template.tpl');
```

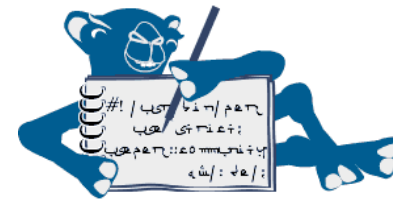
```
my $willkommen = 1;
```

```
$template->param(BOOL => $willkommen);
```

```
print $template->output();
```

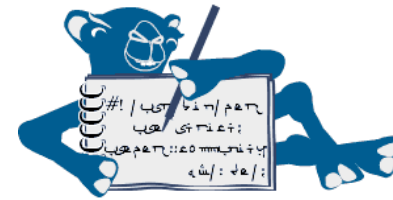


# Beispiel TMPL\_IF



```
<html>
  <head>
    <title>
      TMPL_IF - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_IF NAME=BOOL -->
      Text zwischen den If-Tags
    <!-- /TMPL_IF -->
  </body>
</html>
```

# Beispiel TMPL\_IF



```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;
use HTML::Template;

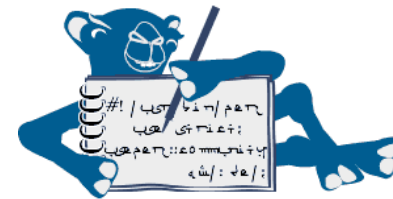
print „Content-type: text/html\\n\\n“;

my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 1;

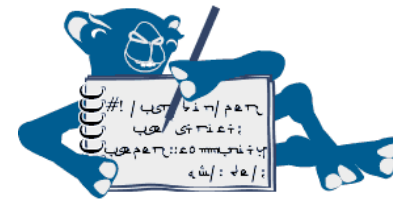
$template->param(BOOL => $willkommen);

print $template->output();
```

# Beispiel TMPL\_IF

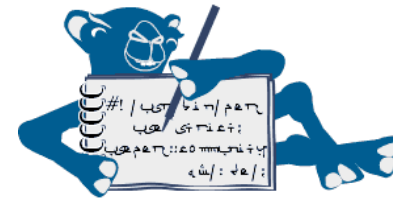


# TMPL\_ELSE



- Alternativer Block zu TMPL\_IF und TMPL\_UNLESS
- *Kein* /TMPL\_ELSE

# Beispiel TMPL\_ELSE



```
<html>
<head>
  <title>
    TMPL_IF - Beispiel
  </title>
</head>
<body>
  <!-- TMPL_IF NAME=BOOL -->
    Text zwischen den If-Tags
  <!-- TMPL_ELSE -->
    Sonst kommt dieser Text
  <!-- /TMPL_IF -->
</body>
</html>
```

```
perl
```

```
;
```

```
ics;
```

```
Template;
```

```
ent-type: text/html\n\n“;
```

```
e = HTML::Template->new(filename => './template.tmpl');
```

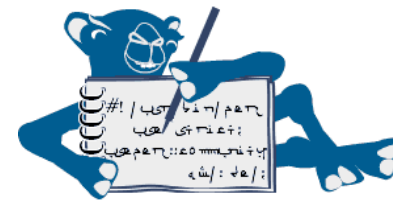
```
namen = 0;
```

```
$template->param(BOOL => $willkommen);
```

```
print $template->output();
```

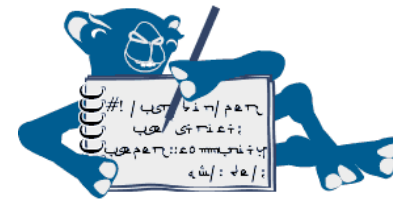
Sonst kommt dieser

# Beispiel TMPL\_ELSE



```
<html>
  <head>
    <title>
      TMPL_IF - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_IF NAME=BOOL -->
      Text zwischen den If-Tags
    <!-- TMPL_ELSE -->
      Sonst kommt dieser Text
    <!-- /TMPL_IF -->
  </body>
</html>
```

# Beispiel TMPL\_ELSE



```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;
use HTML::Template;

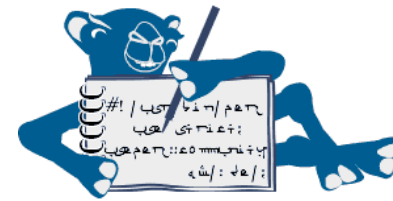
print „Content-type: text/html\n\n“;

my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 0;

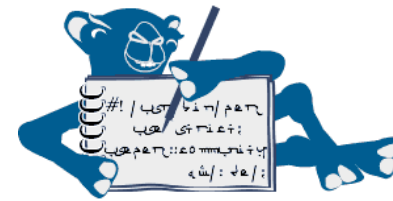
$template->param(BOOL => $willkommen);

print $template->output();
```

# Beispiel TMPL\_ELSE

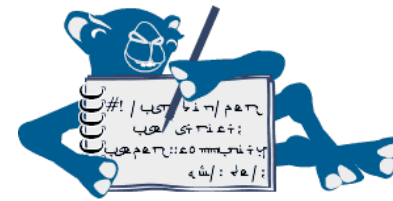


# TMPL\_UNLESS



- Auch mit TMPL\_ELSE kombinierbar
- Text zwischen Tags wird ausgegeben, wenn Parameter *false* zurückgibt oder nicht definiert ist

# Beispiel TMPL\_UNLESS



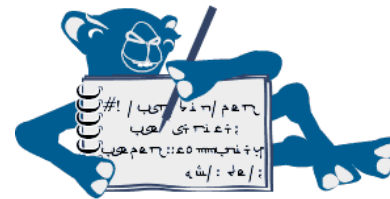
```
<html>
<head>
  <title>
    TMPL_UNLESS - Beispiel
  </title>
</head>
<body>
  <!-- TMPL_UNLESS NAME=BOOL -->
  Text zwischen den Unless-Tags
  <!-- /TMPL_UNLESS -->
</body>
</html>
```

```
template;
e: text/html\n\n“;
HTML::Template->new(filename => ‘./template.tmpl’);
= 0;
```



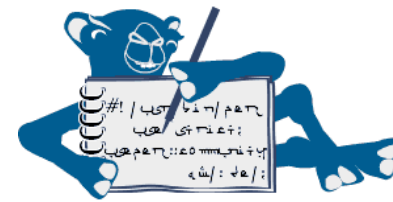
```
$template->param(BOOL => $willkommen);
print $template->output();
```

# Beispiel TMPL\_UNLESS



```
<html>
  <head>
    <title>
      TMPL_UNLESS - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_UNLESS NAME=BOOL -->
    Text zwischen den Unless-Tags
    <!-- /TMPL_UNLESS -->
  </body>
</html>
```

# Beispiel TMPL\_UNLESS



```
#!/usr/bin/perl

use strict;
use warnings;
use diagnostics;
use HTML::Template;

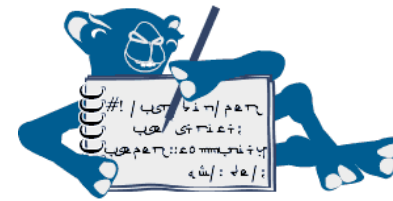
print „Content-type: text/html\\n\\n“;

my $template = HTML::Template->new(filename => './template.tpl');
my $willkommen = 0;

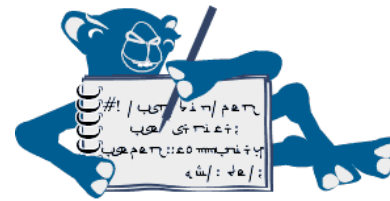
$template->param(BOOL => $willkommen);

print $template->output();
```

# Beispiel TMPL\_UNLESS

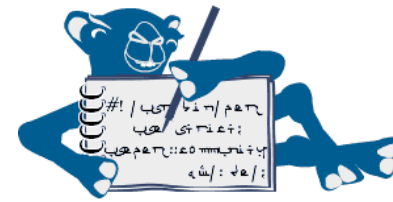


# Methoden



- `new()` erzeugt neues Template-Objekt
- `param()` ersetzt die Tags
- `clear_params()` setzt alle Parameter auf undef
- `output()` gibt den Ausgabertext zurück
- `query()` gibt Informationen über die Struktur des Templates zurück

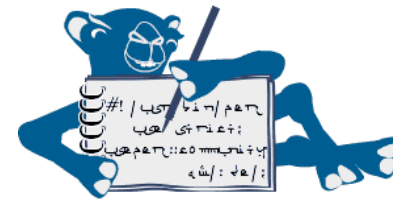
# new()



perl  
community.de

- Erzeugt neues Objekt
  - Dateiname
  - Referenz auf Scalar
  - Referenz auf Array
  - Filehandle

# new() - Syntax



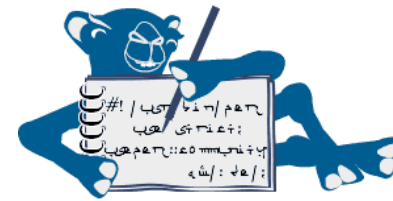
```
my $template = HTML::Template->new(filename => './template.tmpl', options => 'value');
```

```
my $template = HTML::Template->new(scalarref => $ref_to_text, options => 'value');
```

```
my $template = HTML::Template->new(arrayref => $ref_to_array, options => 'value');
```

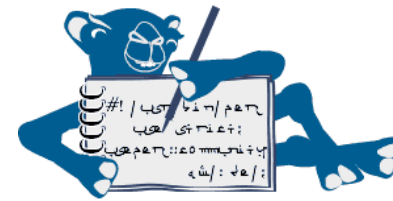
```
my $template = HTML::Template->new(filehandle => *FH, options => 'value');
```

# new() - Optionen



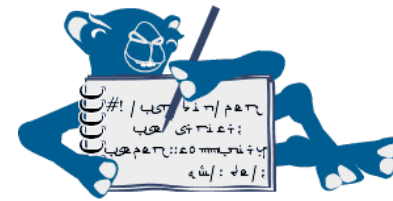
- Fehlererkennung
  - die\_on\_bad\_params
    - 0: beendet nicht das Programm bei Fehlerhaften Parametern
    - 1: beendet das Programm
  - strict
    - 0: ignoriert falsche TMPL\_\*-Tags (z.B. TMPL\_FOO)
    - 1: beendet Programm bei falschen Tags
  - vanguard\_compatibility\_mode
    - 1: erlaubt auch Parameter in der Form %NAME%

# new() - Optionen



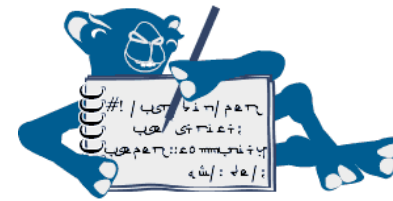
- Caching
  - cache
    - 1: geparstes Template wird gecached
      - Hauptsächlich bei Apache/mod\_perl
      - nur bei filename-Parameter
  - shared\_cache
    - 1: benutzt IPC::SharedCache, um cache im SharedMemory zu speichern
      - Signifikante Reduktion der Speicherbelegung
      - Läuft auch unter CGI
  - double\_cache
    - 1: benutzt cache und shared\_cache

# new() - Optionen



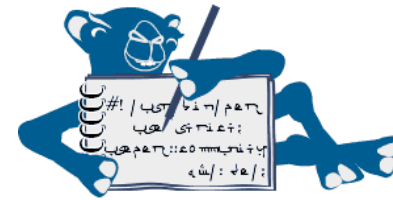
- Debugging
  - debug
    - 1: schreibt debug-Informationen auf STDERR
  - cache\_debug
    - Schreibt Informationen über Cache auf STDERR

# new() - Optionen



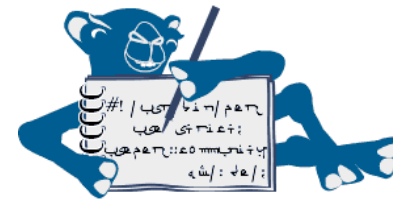
- **Verschiedene**
  - **associate**
    - Anderes Objekt muss eine param()-Methode besitzen
    - Häufig ein CGI-Objekt
    - „erbt“ Parameter-Werte des anderen Objekts
  - **case\_sensitive**
    - 1: Beachtet die Schreibweise der Parameter (Name ne NAME)
  - **loop\_context\_vars**
    - `__inner__`
    - `__first__`
    - `__last__`

# new()-Optionen



- Verschiedene
  - no\_includes
    - 1: erlaubt keine Template-Includes
  - max\_includes
    - Maximale „Tiefe“ der Includes
  - global\_vars
    - Parameter außerhalb einer Schleife, werden der Schleife „bekanntgemacht“

# param()



- Abfrage welche Parameter im Template

```
my @parameters = $template->param();
```

- Abfrage, welchen Wert ein Parameter hat

```
my $value = $template->param('NAME');
```

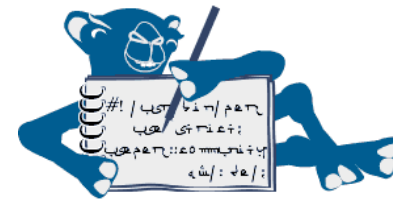
- Setzen eines Wertes für einen Parameter

```
$template->param(NAME => 'Wert');
```

- Setzen von Werten für mehrere Parameter

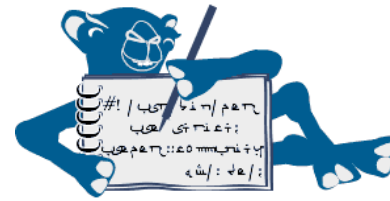
```
$template->param(NAME => 'Wert',  
                NAME2 => 'Wert2');
```

# clear\_params()



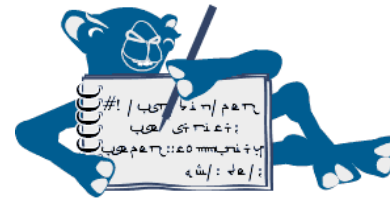
- setzt alle Parameter auf undef
- Eignet sich zum erstellen mehrerer HTML-Seiten in einem Programm

# output()



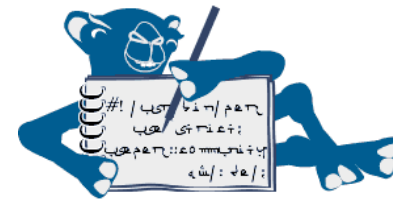
- Gibt den gesamten Template-Text zurück
- Parameter wurde ersetzt

# query()

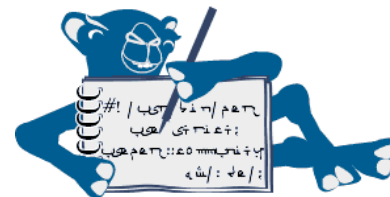


- Informationen über die Struktur des Templates
  - Ist ein Parameter enthalten?
  - Welcher Typ ist der Parameter?
- Namen in Kleinbuchstaben
- Typen in Großbuchstaben

# Zusammenfassung



- HTML::Template eignet sich sehr gut für Portale oder Communities
- Durch „Logik“-Tags variable Templates möglich
- Trennung von Design und Programm möglich
- Mehr Optionen zu den Methoden in der Doku auf [cpan.org](http://cpan.org)



# HTML::Template

München, 17.04.2004