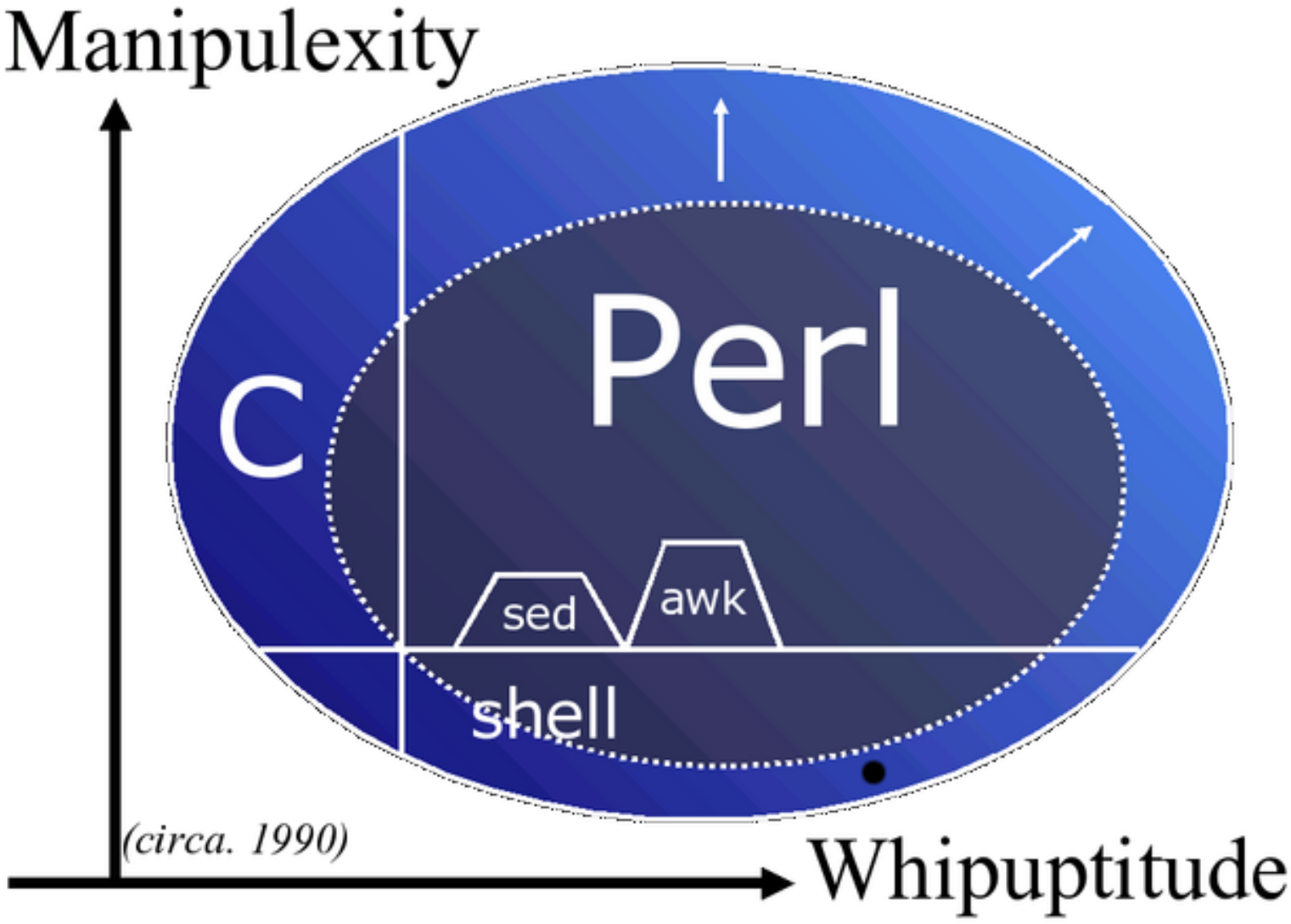
Perl 6, genau jetzt!

Ingo Blechschmidt
<iblech@web.de>

Augsburger
Linux-Infotag 2006


Manipulexity
C
Perl
sed
awk
shell
(circa. 1990)
Whipuptitude

Inhalt

Überblick über Perl 6

- Anpassungsfähigkeit von Perl 5 begrenzt; Hacks notwendig
- Ab 2000:
  Offener Neugestaltungsprozess (RFC, Mailinglisten) → *Perl 6*
- Sprachspezifikation in „Synopsen“

Ziele von Perl 6

- **Anpassungsfähigkeit, Erweiterbarkeit → Zukunftssicherheit**
- Große Manipulexity, große Whipuptitude
  - Geringe „Sprachsteuer“
  - There’s More Than One Way To Do It (TIMTOWTDI)
- Eignung für große und kleine Projekte

Ziele von Perl 6

- Anpassungsfähigkeit, Erweiterbarkeit → Zukunftssicherheit
- **Große Manipulexity, große Whipuptitude**
  - Geringe „Sprachsteuer“
  - There’s More Than One Way To Do It (TIMTOWTDI)
- Eignung für große und kleine Projekte

Ziele von Perl 6

- Anpassungsfähigkeit, Erweiterbarkeit → Zukunftssicherheit
- Große Manipulexity, große Whipuptitude
  - Geringe „Sprachsteuer“
  - There’s More Than One Way To Do It (TIMTOWTDI)
- **Eignung für große und kleine Projekte**

Perl selbst ist gar nicht so gut. . .

CPAN

macht's!

Comprehensive Perl Archive Network

- CPAN: Sammlung von Perl-Modulen
- Motto: Kein Coding = bestes Coding
- *Vokabular* > *Syntax!*

- 10^{+} Jahre, $2\,500^{+}$ Entwickler, $8\,000^{+}$ Module
- Automatisierte Tests, Bugtracking, Paketverwaltung

Architektur

- Nutzungsmöglichkeit bereits vorhandener CPAN-Module *Muss*
- *Multilingualität*
  - Einbindung von Modulen anderer Sprachen

```
use perl5:DBI;
use jsan:DOM;
use c:fftw;
```

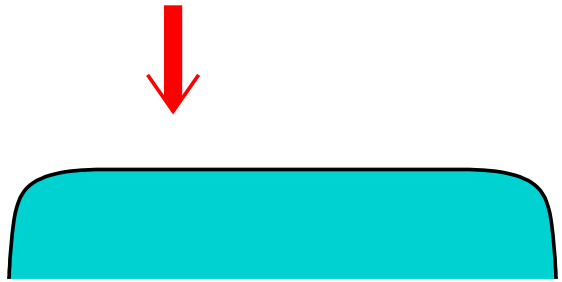
  - Nutzung von Perl 6-Modulen aus anderen Sprachen heraus

```
JSAN.use('Perl6');
#include <perl6.h>
import Perl6
```

- ▸ Typsystem ▸ Self-hosting ▸ Rules ▸ Objektorientierung ▸ Parrot

„Die Wasserbetttheorie“

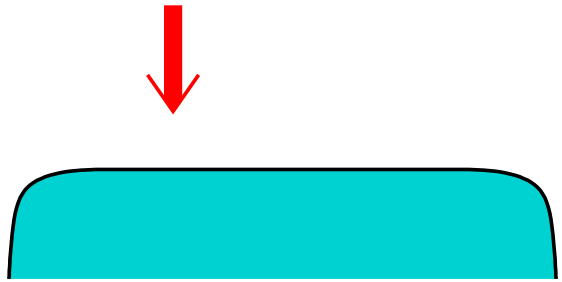
- Wasserbetthöhe als Maß für die Komplexität eines Features
- Eindrücken an einer Stelle (Vereinfachung) . . .
- . . . Hochkommen an anderen Stellen (Komplizierung)



- „Komplexitätserhaltung“

„Die Wasserbetttheorie“

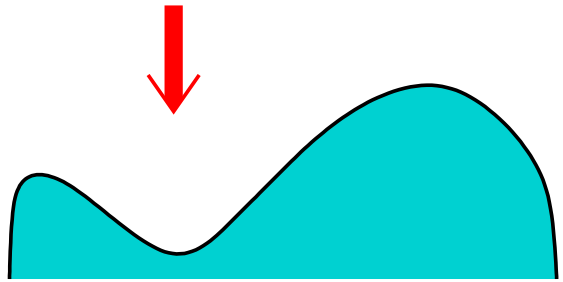
- Wasserbetthöhe als Maß für die Komplexität eines Features
- Eindrücken an einer Stelle (Vereinfachung)...
- ...Hochkommen an anderen Stellen (Komplizierung)



- „Komplexitätserhaltung“

„Die Wasserbetttheorie“

- Wasserbetthöhe als Maß für die Komplexität eines Features
- Eindrücken an einer Stelle (Vereinfachung). . .
- . . . Hochkommen an anderen Stellen (Komplizierung)



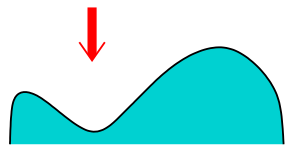
- „Komplexitätserhaltung“

„Die Wasserbetttheorie“ – Beispiele

- Sehr viele Operatoren: `+@= /%^ !@=`
- Sehr wenig Operatoren: `set(x, add(5, 3))`

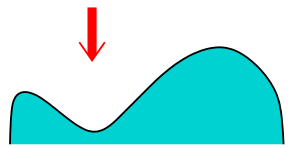
- Sehr viele Datentypen:
  `Number::Real::Positive::Prime`
  `Array::OptimizedForMemory[String::Unicode]`
- Sehr wenig Datentypen:
  `Scalar Array`

- Ziel: Balance –
  einfach zu verstehen, einfach zu schreiben



„Die Wasserbetttheorie“ – Beispiele

- Sehr viele Operatoren: `+@= /%^ !@=`
- Sehr wenig Operatoren: `set(x, add(5, 3))`
- Sehr viele Datentypen:
  `Number::Real::Positive::Prime`
  `Array::OptimizedForMemory[String::Unicode]`
- Sehr wenig Datentypen:
  `Scalar Array`
- Ziel: Balance –
  einfach zu verstehen, einfach zu schreiben

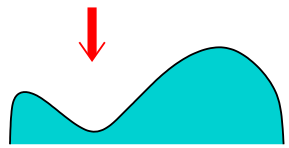


„Die Wasserbetttheorie“ – Beispiele

- Sehr viele Operatoren: `+@= /%^ !@=`
- Sehr wenig Operatoren: `set(x, add(5, 3))`

- Sehr viele Datentypen:
  `Number::Real::Positive::Prime`
  `Array::OptimizedForMemory[String::Unicode]`
- Sehr wenig Datentypen:
  `Scalar Array`

- Ziel: Balance –
  einfach zu verstehen, einfach zu schreiben



Huffmannkodierung

- Datenkomprimierung durch Zuweisung. . .

  . . . *kurzer* Sequenzen an *häufige* Inputs
  . . . *langer* Sequenzen an *seltene* Inputs

Natürliche Huffmannkodierung:

- „Haus“ „Markt“ „Jagd“ „Geld“
- „Industrieroboter“ „Elektromagnetismus“ ⟶ „EM“

Huffmannkodierung bei Unix:

- `cat` `ls` `cp` `sed` `grep`
- `svscan-add-to-inittab` `mysql_fix_privilege_tables`

Huffmannkodierung

- Datenkomprimierung durch Zuweisung. . .

  . . . *kurzer* Sequenzen an *häufige* Inputs
  . . . *langer* Sequenzen an *seltene* Inputs

Natürliche Huffmannkodierung:

- „Haus“ „Markt“ „Jagd“ „Geld“
- „Industrieroboter“ „Elektromagnetismus“ $\longrightarrow$ „EM“

Huffmannkodierung bei Unix:

- `cat` `ls` `cp` `sed` `grep`
- `svscan-add-to-inittab` `mysql_fix_privilege_tables`

Huffmannkodierung

- Datenkomprimierung durch Zuweisung. . .

  . . . *kurzer* Sequenzen an *häufige* Inputs
  . . . *langer* Sequenzen an *seltene* Inputs

Natürliche Huffmannkodierung:

- „Haus“ „Markt“ „Jagd“ „Geld“
- „Industrieroboter“ „Elektromagnetismus“ $\longrightarrow$ „EM“

Huffmannkodierung bei Unix:

- `cat` `ls` `cp` `sed` `grep`
- `svscan-add-to-inittab` `mysql_fix_privilege_tables`

Huffmannkodierung

- Datenkomprimierung durch Zuweisung. . .

  . . . *kurzer* Sequenzen an *häufige* Inputs
  . . . *langer* Sequenzen an *seltene* Inputs

Natürliche Huffmannkodierung:

- „Haus“ „Markt“ „Jagd“ „Geld“
- „Industrieroboter“ „Elektromagnetismus“ $\longrightarrow$ „EM“

Huffmannkodierung bei Unix:

- `cat` `ls` `cp` `sed` `grep`
- `svscan-add-to-inittab` `mysql_fix_privilege_tables`

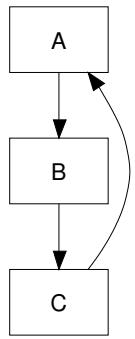
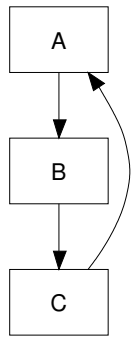
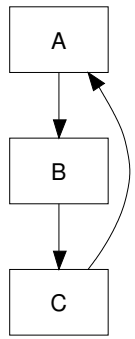
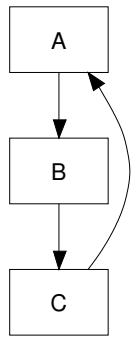
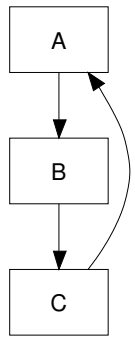
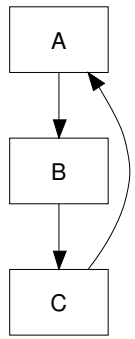
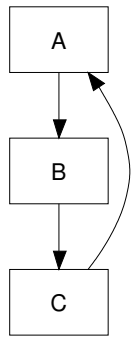
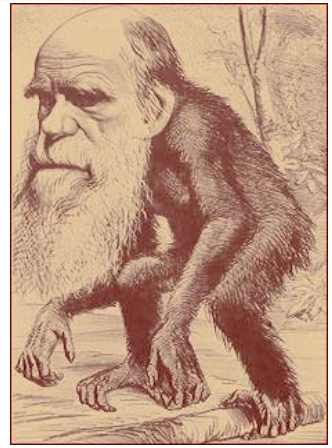
Huffmannkodierung – Beispiele

- ```
  System.out.print("Hallo, Welt!");
  ```
  ```
  ⟶ say "Hallo, Welt!";
  ```
- ```
  public Foo getFoo() { return foo; }
  public void setFoo(Foo newFoo) {
      foo = newFoo;
  }
  ```
  ```
  ⟶ has Foo $foo is rw;
  ```
- ```
  require_once("Foo.php");
  ```
  ```
  ⟶ use Foo;
  ```

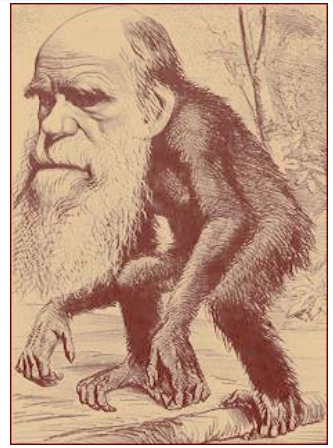

  ```

Huffmannkodierung – Beispiele

- ```
  System.out.print("Hallo, Welt!");
  ```
  ```
  ⟶ say "Hallo, Welt!";
  ```
- ```
  public Foo getFoo() { return foo; }
  public void setFoo(Foo newFoo) {
      foo = newFoo;
  }
  ```
  ```
  ⟶ has Foo $foo is rw;
  ```
- ```
  require_once("Foo.php");
  ```
  ```
  ⟶ use Foo;
  ```


  ```

Huffmannkodierung – Beispiele

- ```
  System.out.print("Hallo, Welt!");
  ```
  ```
  ⟶ say "Hallo, Welt!";
  ```
- ```
  public Foo getFoo() { return foo; }
  public void setFoo(Foo newFoo) {
      foo = newFoo;
  }
  ```
  ```
  ⟶ has Foo $foo is rw;
  ```
- ```
  require_once("Foo.php");
  ```
  ```
  ⟶ use Foo;
  ```


  ```

Perl 6-Besonderheiten

- „Topicalization“
- Schleifen
- Rules
- Anpassungsmöglichkeiten

Weiter. . .

“Topicalization”

- Verfolgen des aktuellen Themas als Mittel gegen Wiederholungen

Im Deutschen:

- Relativpronomen („die Sprache, *die* cool ist“)
- Personalpronomen („*du* hier links vorne“)

Unklar?

Nein!

“Topicalization”

```
say Person.search("Grtz Baka").get_tel();
Person.search("Grtz Baka").set_gehalt(
    Person.search("Grtz Baka").get_gehalt()
    + 1000
);
```

“Topicalization”

```
say Person.search("Grtz Baka").get_tel();
Person.search("Grtz Baka").set_gehalt(
    Person.search("Grtz Baka").get_gehalt()
    + 1000
);
```


FINALE
SAY
NO TO
Wieder-
holungen!

“Topicalization”

```
say Person.search("Grtz Baka").get_tel();
Person.search("Grtz Baka").set_gehalt(
    Person.search("Grtz Baka").get_gehalt()
    + 1000
);
```

“Topicalization”

```
say $_.get_tel();
$_.set_gehalt(
    $_.get_gehalt()
    + 1000
);
```

“Topicalization”

```
given %Person<Grtz Baka> {
    say $_.get_tel();
    $_.set_gehalt(
        $_.get_gehalt()
        + 1000
    );
}
```

“Topicalization”

```
given %Person<Grtz Baka> {
    say $_.get_tel();
    $_.set_gehalt(
        $_.get_gehalt()
        + 1000
    );
}
```

“Topicalization”

```
given %Person<Grtz Baka> {
    say .get_tel();
    .set_gehalt(
        .get_gehalt()
        + 1000
    );
}
```

Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <  42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..41 -> $i {
    say $i;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..41 -> $i {
    say $i;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..41 -> $i {
    say $i;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..42 -> $i {
    say $i;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..42 -> $i {
    say $_;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..42 {
    say $_;
    ...;
}
```



Schleifen

- Schleifen in C, Java, ...:

```
for(int i = 17; i <= 42; i++) {
    printf("%d\n", i);
    ...;
}
```

- Schleifen in Perl 6:

```
for 17..42 {
    say;
    ...;
}
```



Anpassungsmöglichkeiten



- Überschreiben/Erweitern von „Builtins“, Operatoren und Klassen
- Keine qualitativen Unterschiede zwischen Erweiterungen und dem Kern

- ```
  use Numbers::Surreal;
  say 1/ω;  # ε
  ```
- ```
  use Operators::Quantum;
  say |0⟩ + |1⟩;
  ```
- ```
  use Process::Remote;
  sub foo is remote(google.de) {...}
  foo(...);
  ```
  ```

Anpassungsmöglichkeiten



- Codeausführung zur Compile-Zeit – „mit dem Compiler reden“

```
say "Ich wurde vor ",
    time - BEGIN { time },
    " Sekunden kompiliert.";
```

▸ Erklärung

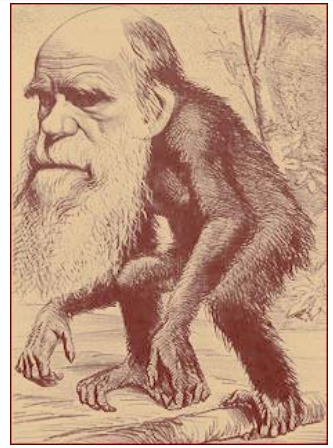
- Makros ala C (instabil)...

```
#define foo bar        // C, C++
macro foo { 'bar' }    # Perl 6
```

- ...und Lisp (hygienisch)

▸ Beispiele

Anpassungsmöglichkeiten



- Codeausführung zur Compile-Zeit – „mit dem Compiler reden“

```
say "Ich wurde vor ",
    time - BEGIN { time },
    " Sekunden kompiliert.";
```

▸ Erklärung

- Makros ala C (instabil)...

```
#define foo bar          // C, C++
macro foo { 'bar' }      # Perl 6
```

- ...und Lisp (hygienisch)

▸ Beispiele

Zeitlicher Verlauf

ab 2000 „Ideensammlung“, Spezifikationen
aber: keine Implementierung!

1.2.2005 Pugs! ☺

- Experimentelle Compiler- und Interpreterplattform
- „Übungsprojekt“ von Audrey Tang
- Heute: 150^+ Entwickler, $9\,000^+$ Commits

Tag 6 Einfacher Interpreter:

```
say "Hallo, Welt!";
```

Zeitlicher Verlauf (Forts.)

Tag 23 `Test.pm` – Test-driven Development:

```
use Test;
is 42 - 19, 23, 'Subtraktion';
```

Heute: 11 000$^+$ Tests auf 35 000$^+$ Zeilen; Smokeserver

Tag 96 `Net::IRC` – svnbot

Tag 117 `Net::IRC` – evalbot

```
<iblech>  ?eval 42 - 19
<evalbot> 23
```

Juni, Juli Kompilation zu Parrot, Perl 5 und JavaScript

. . . (vieles mehr)

Zeitlicher Verlauf (Forts.)

Tag 23 `Test.pm` – Test-driven Development:

```
use Test;
is 42 - 19, 23, 'Subtraktion';
```

Heute: 11 000$^+$ Tests auf 35 000$^+$ Zeilen; Smokeserver

Tag 96 `Net::IRC` – svnbot

Tag 117 `Net::IRC` – evalbot

```
<iblech>  ?eval 42 - 19
<evalbot> 23
```

Juni, Juli Kompilation zu Parrot, Perl 5 und JavaScript

. . . (vieles mehr)

Zeitlicher Verlauf (Forts.)

Tag 23 `Test.pm` – Test-driven Development:

```
use Test;
is 42 - 19, 23, 'Subtraktion';
```

Heute: $11\,000^+$ Tests auf $35\,000^+$ Zeilen; Smokeserver

Tag 96 `Net::IRC` – svnbot

Tag 117 `Net::IRC` – evalbot

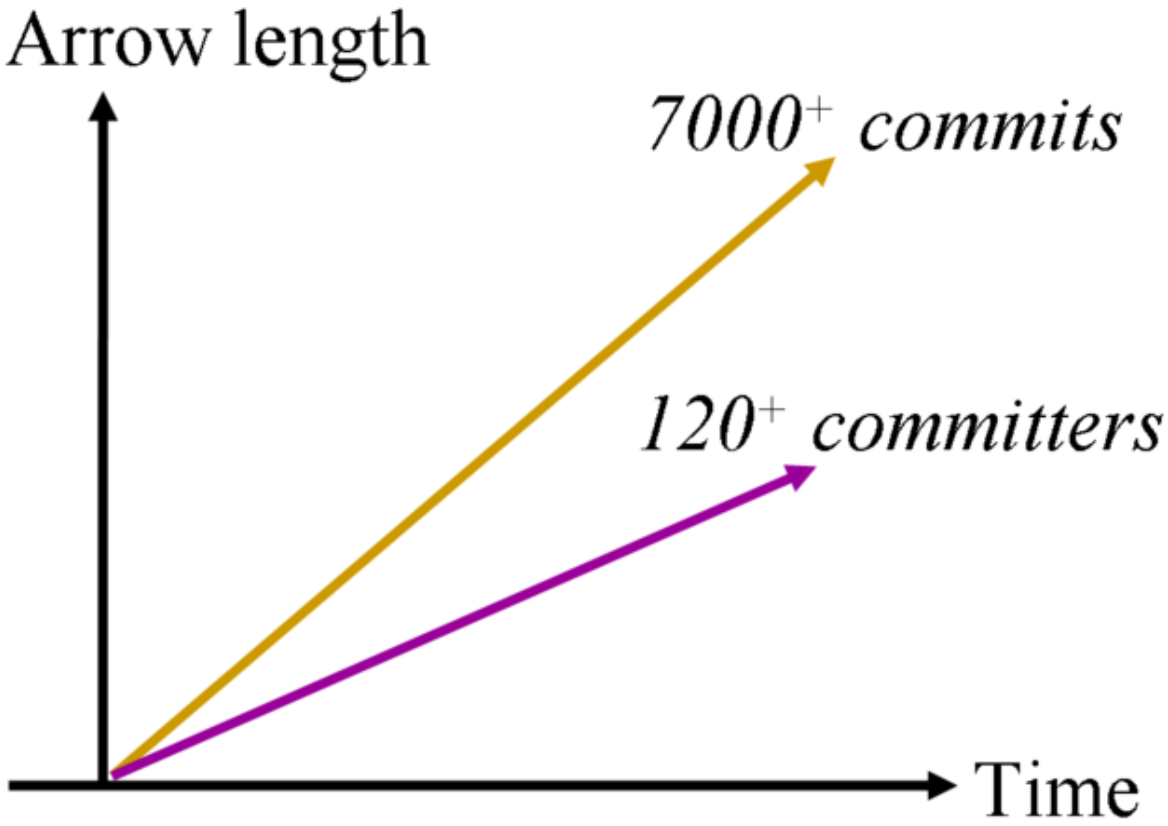
```
<iblech>  ?eval 42 - 19
<evalbot> 23
```

Juni, Juli Kompilation zu Parrot, Perl 5 und JavaScript

... (vieles mehr)

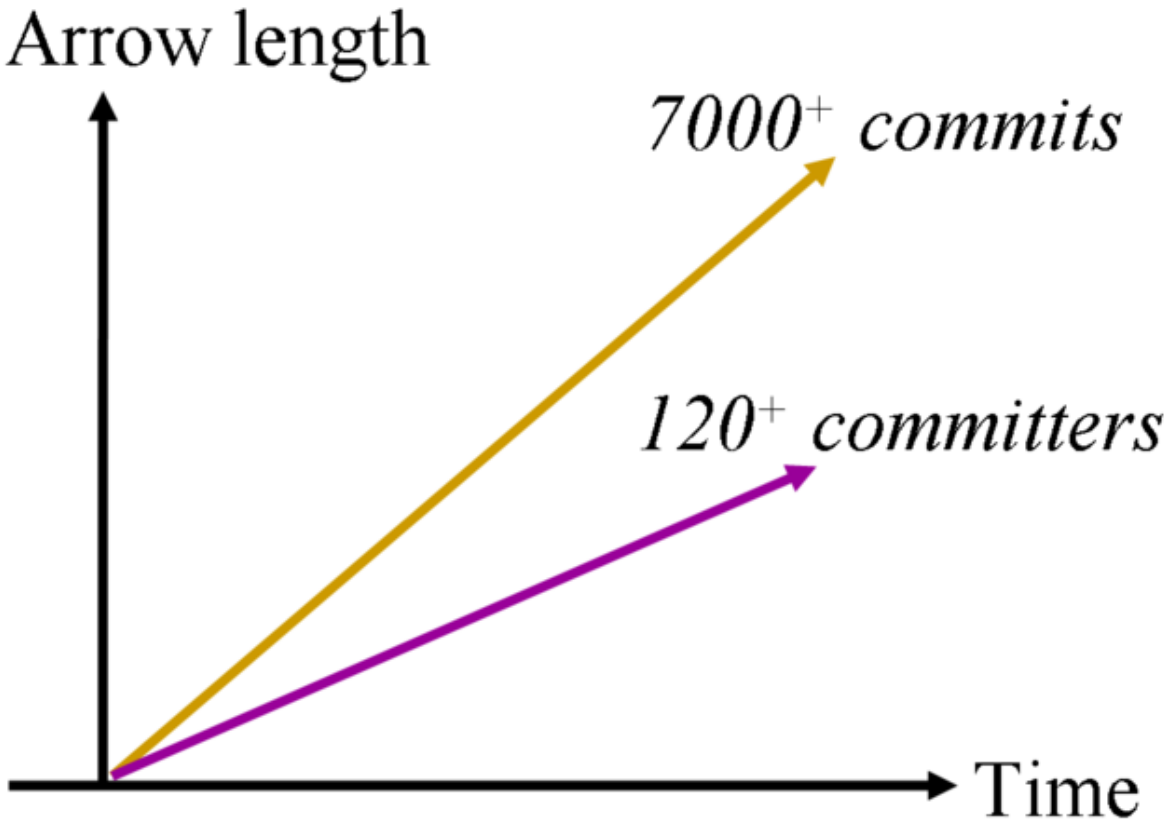
Entwicklung

- Test-driven Development
- Perl 5-, Perl 6-, Haskell-, JavaScript-Leute, . . .
- Fokus auf. . . pugs −03?



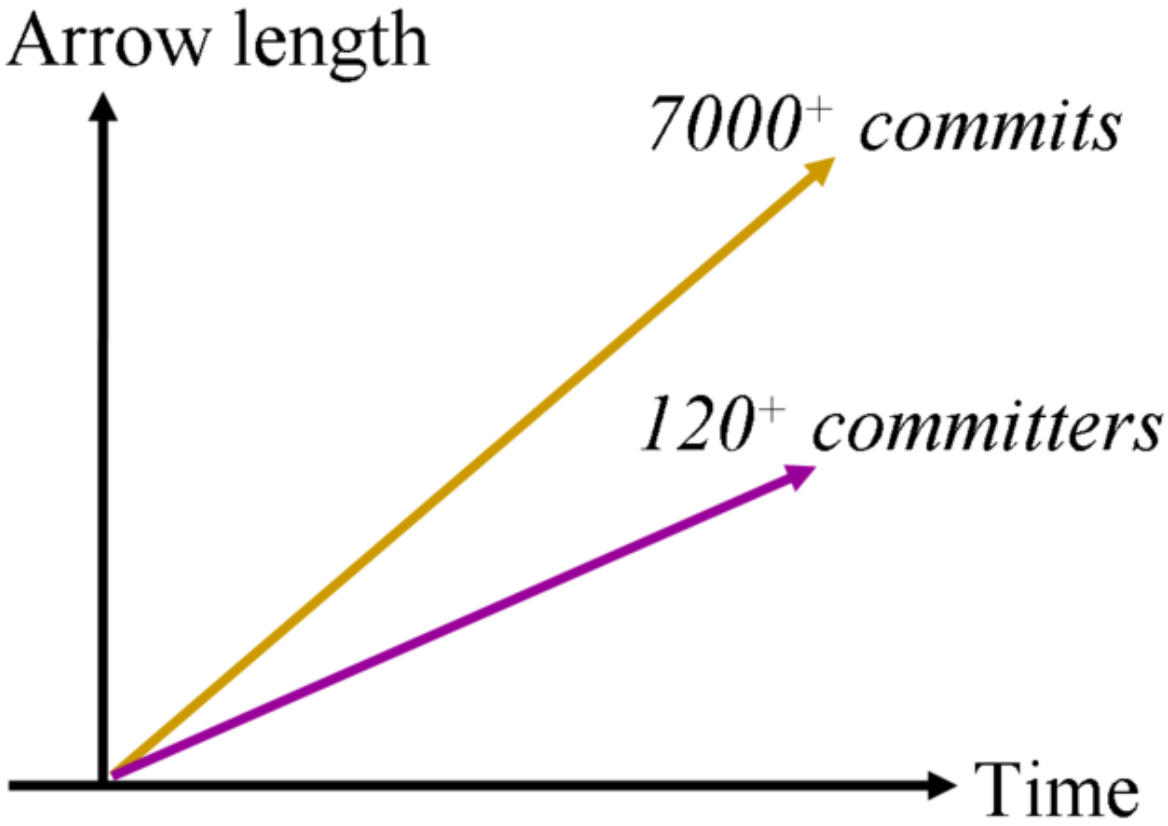
Entwicklung

- Test-driven Development
- Perl 5-, Perl 6-, Haskell-, JavaScript-Leute, . . .
- Fokus auf. . . `pugs -O3`?



Entwicklung

- Test-driven Development
- Perl 5-, Perl 6-, Haskell-, JavaScript-Leute, . . .
- Fokus auf. . . `pugs -Os`? (Größe?)



-O*fun*!

- “Imagineering”
- Code statt bloßer Ideen
- Vermeidung von Deadlocks
- Moderne Versionskontrolle
- Liberale Ausgabe von Commit-Berechtigungen

“Frivolous toy interpreter”

“~~Frivolous~~ toy interpreter”

“~~Frivolous~~ toy ~~interpreter~~”

“toy”

Live-Demo

Siehe auch

- `http://dev.perl.org/perl6/`
  Perl 6-Projektseite
- `http://www.pugscode.org/`
  Pugs-Projektseite, mit Vorträgen
- `perl6-language@perl.org`,
  `gmane.comp.lang.perl.perl6.language`,
  `#perl6` auf Freenode
- `http://www.oreillynet.com/onlamp/blog/2005/10/ofun.html`
  Geoff Broadwell über *-Ofun*
- `http://developers.slashdot.org/article.pl?sid=05/10/09/1831219`
  “frivolous toy interpreter”

linux user group e.v.
luga
augsburg

Join the fun!

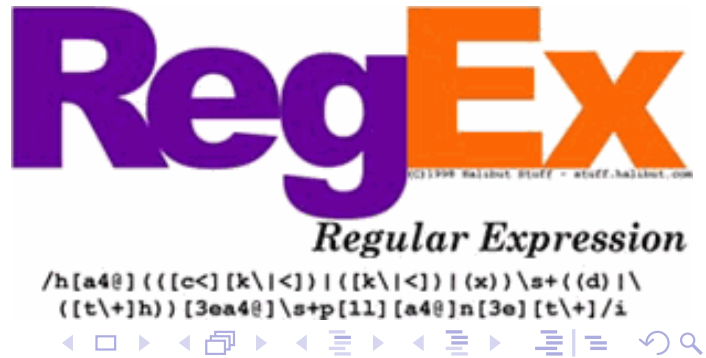
`http://xrl.us/lit06perl`
`#perl6` auf Freenode

Bonus-Slides

Rules

- Reguläre Ausdrücke zu undurchsichtig
- Reguläre Ausdrücke nicht mächtig genug
- In Perl 6: Rules!

```
grammar Grammar::URL {
    rule url  { <protocol> \:// <host> <path> }
    rule path { [ / <filename> ]+ }

    rule protocol { http | ftp }
    rule filename {...}
}
```



Self-hosting

- Ziel: Perl 6-Compiler *in Perl 6*! (self-hosting)

1. P6→P5-Compiler in Perl 5 (A)
2. Portieren von (A) nach Perl 6 (B)
3. Kompilation von (B) durch (A)
   → Fertigstellung des Bootstrappings

- Damit:
  Funktionsfähiger P6→P5-Compiler
  in Perl 6
- Weitere Backends:
  JavaScript, Parrot, Haskell, ...

Self-hosting

- Ziel: Perl 6-Compiler *in Perl 6*! (self-hosting)

1. P6→P5-Compiler in Perl 5 (A)
2. Portieren von (A) nach Perl 6 (B)
3. Kompilation von (B) durch (A)
   → Fertigstellung des Bootstrappings

- Damit:
  Funktionsfähiger P6→P5-Compiler
  in Perl 6
- Weitere Backends:
  JavaScript, Parrot, Haskell, ...

Self-hosting

- Ziel: Perl 6-Compiler *in Perl 6*! (self-hosting)

1. P6→P5-Compiler in Perl 5 (A)
2. Portieren von (A) nach Perl 6 (B)
3. Kompilation von (B) durch (A)
   → Fertigstellung des Bootstrappings

- Damit:
  Funktionsfähiger P6→P5-Compiler
  in Perl 6
- Weitere Backends:
  JavaScript, Parrot, Haskell, ...

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^
    time - BEGIN { time },

    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^
    time - BEGIN { time },

    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^
    time - BEGIN { time },

    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^

    time - BEGIN { time },

    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von `BEGIN`-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
    time - BEGIN { time },
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^^^^^^^^^^^^^^^^^
    time - BEGIN { time },
^^^^^^^^^^^^^^^^^^^^^^^^^^
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^^^^^^^^^^^^^^^^^
    time - BEGIN { time },
^^^^^^^^^^^^^^^^^^^^^^^^^^
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^^^^^^^^^^^^^^^^^
    time - BEGIN { time },
^^^^^^^^^^^^^^^^^^^^^^^^^^
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time -
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^^^^^^^^^^^^^^^^^
    time - BEGIN { time },
^^^^^^^^^^^^^^^^^^^^^^^^^^
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time -
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
    time - BEGIN { time },
" Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time - 1142182282
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
    time - BEGIN { time },
" Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time - 1142182282,
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
^^^^^^^^^^^^^^^^^^^^^
    time - BEGIN { time },
^^^^^^^^^^^^^^^^^^^^^^^^^^
    " Sekunden kompiliert.";
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time - 1142182282,
```

Anpassungsmöglichkeiten

BEGIN-Blöcke

- Ausführung von Code innerhalb von BEGIN-Blöcken *schon zur Compile-Zeit*
- Eingabecode:

```
say "Ich wurde vor ",
    time - BEGIN { time },
    " Sekunden kompiliert.";
```

- Was wirklich kompiliert wird:

```
say "Ich wurde vor ",
    time - 1142182282,
    " Sekunden kompiliert.";
```

Anpassungsmöglichkeiten

Hygienische Makros

- Ausführung von Makros zur Compile-Zeit

```
macro compiletime_say (Str $text) {
    say $text;
}

compiletime_say("Hallo!");
```

- Quasiquoting

```
macro plus_42 (AST $ast) {
    return q:code { 42 + {{{ $ast }}} };
}

say plus_42(23);  # 65
```

Anpassungsmöglichkeiten

Hygienische Makros

- Ausführung von Makros zur Compile-Zeit

```
macro compiletime_say (Str $text) {
    say $text;
}

compiletime_say("Hallo!");
```

- Quasiquoting

```
macro plus_42 (AST $ast) {
    return q:code { 42 + {{{ $ast }}} };
}

say plus_42(23);  # 65
```

◂ Anpassungsmöglichkeiten

Typsystem

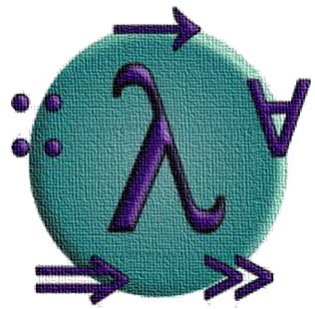
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $zahl = 7;  # :-(
my $fido;  # :-(
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

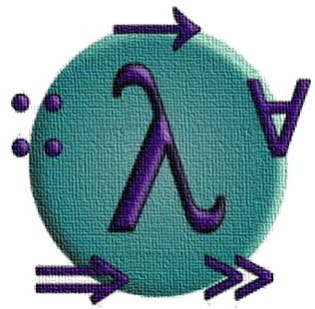
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $zahl = 7;  # :-(
my $fido;  # :-(
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

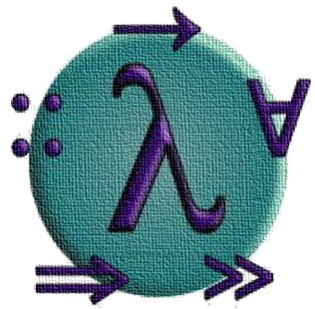
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $zahl = 7;  # :-(
my $fido;  # :-(
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

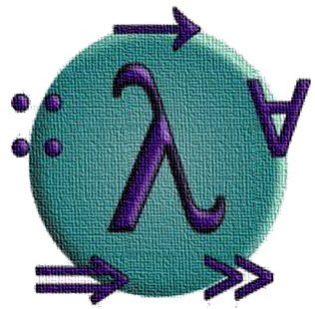
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $pinguinalter = 7;  # :)
my $fido;  # :-(
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

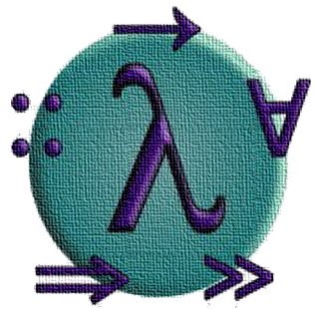
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $pinguinalter = 7;  # :)
my $fido;  # :-(
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

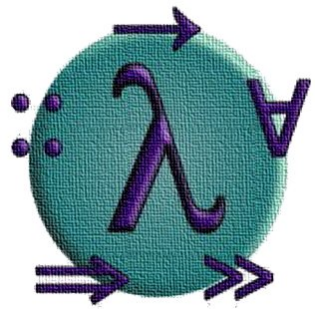
- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

```
my $pinguinalter = 7;   # :)
my Dog::Dackel $fido;   # :)
```

- Problem: „*Typen Tippen* weniger gut“
- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

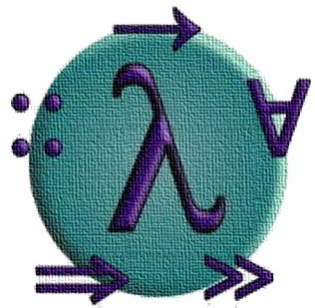
```
my $pinguinalter = 7;   # :)
my Dog::Dackel $fido;   # :)
```

- Problem: „*Typen Tippen* weniger gut“

- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my Num $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

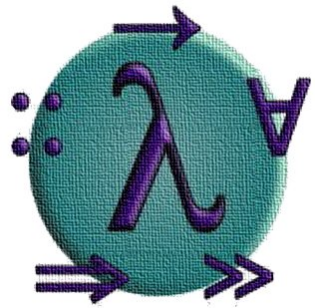
```
my $pinguinalter = 7;   # :)
my Dog::Dackel $fido;   # :)
```

- Problem: „*Typen Tippen* weniger gut“

- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my ~~Num~~ $sinn = 42;
```

- (Details noch im Wandel)



Typsystem

- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

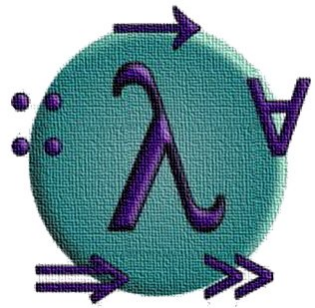
```
my $pinguinalter = 7;  # :)
my Dog::Dackel $fido;  # :)
```

- Problem: „*Typen Tippen* weniger gut“

- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my $sinn = 42;  # $sinn automatisch Num
```

- (Details noch im Wandel)



Typsystem

- „*Typisierung* gut“ –
  - Fehlervermeidung schon zur Compile-Zeit
  - Typen als implizite Dokumentation

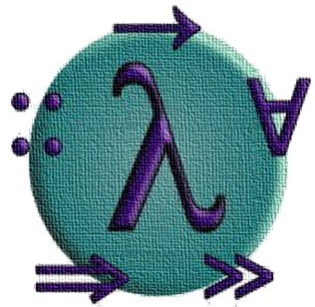
```
my $pinguinalter = 7;  # :)
my Dog::Dackel $fido;  # :)
```

- Problem: „*Typen Tippen* weniger gut“

- Abhilfe: Typerschließung durch den Compiler (type inference)

```
my $sinn = 42;  # $sinn automatisch Num
```

- (Details noch im Wandel)



Klassen

- Huffmannkodierung:
  Automatische Accessorgenerierung,
  Standard-Konstruktor (new)
- Mehrfachvererbung und Mixins (Rollen)
- Erweiterbarkeit zur Laufzeit

Klassen – Beispieldefinition

```
class Dog::Dackel {
    is Dog;
    has Str $name;
    has Person $owner is rw;

    method bark {
        say "Hallo. Ich bin $.name.";
    }
}

my $fido = Dog::Dackel.new(:name<Grtz>);
$fido.bark();
```

Mixins durch Rollen

- Mixins durch Rollen (in Ruby: Module)
- Einbindung von Methoden, Attributen, Eltern
- „Zusammenbasteln“ von Klassen
  (`$normales_objekt but Log::STDERR`)

Mixins durch Rollen – Beispiel

```
role Sager::A { method sag_a { say "A" } }
role Sager::B { method sag_b { say "B" } }

class Sager::AundB {
    does Sager::A;
    does Sager::B;
}

my $sager = Sager::AundB.new;
$sager.sag_a();  # A
$sager.sag_b();  # B
```

Parrot

- JIT-fähige virtuelle Maschine
- Plattformunabhäniger Bytecode
- “One bytecode to rule them all”

Perl 5	Perl 6		Perl 5	Perl 6
BASIC	JavaScript		BASIC	JavaScript
Lisp	PHP	↔ Parrot ↔	Lisp	PHP
Python	Ruby		Python	Ruby
TCL	. . .		TCL	. . .

Verbreitung von Perl 6-Modulen

1. Schreiben von Modulen in Perl 6
2. (Automatische) Kompilation P6→P5
3. Verteilung des „Perl 5“-Moduls

- Kein Perl 6-Zwang für Modulnutzer
- Kein Mehraufwand für Modulprogrammierer
- Damit Ermöglichung einer langen Übergangszeit

Perl 6 in Perl 5

```
#!/usr/bin/perl5

use warnings;
use strict;

# Hier normales Perl 5...

{
    use v6-pugs;
    # In diesem Block Perl 6...
}

# Ab hier wieder Perl 5...
```

Bildquellen

- http://gnosislivre.org/twiki/pub/PerlMongersSSA/WebHome/camel.gif
- http://packages.gentoo.org/images/app-arch/bzip2.jpg
- http://www.sweetiebag.com/product_images/details/Topic.jpg
- http://www.ksta.de/ks/images/mdsBild/1119107183084l.jpg
- http://perlcabal.org/~autrijus/osdc/hatching.png
  http://perlcabal.org/~autrijus/osdc/line.png
  http://perlcabal.org/~autrijus/osdc/logo.jpg
- http://whyfiles.org/095evolution/images/Darape.jpg
- http://www.rr19.de/rr19/zielscheibe.gif
- http://upload.wikimedia.org/wikipedia/fr/2/2e/Ubuntu.gif
- http://www.fancy365.com/art/mini/jenga/images/jenga_03.jpg
- http://cpan.org/misc/jpg/cpan.jpg
- http://perl.plover.com/yak/presentation/samples/happy-baby.JPG
- http://www.puppydogweb.com/kennels/images/pugs_pineycreek7.jpg
- http://www.luga.de/LUGA_Logo
- http:
  //www.lakehousecreations.com/images/ThankYou/Thank%20You%202003%20(12).jpg
- http://perl.plover.com/yak/presentation/samples/present.gif
- http://stuff.halibut.com/images/shirts/img_regex_big.gif
- http:
  //lewis.up.edu/efl/mclary/German_406_Spring_2003/406%20Images/muenchhausen.png
- http://www.szabgab.com/talks/parrot/img0.jpg