Perl

Boek: R.L. Schwartz & T. Phoenix & b d foy,
Learning Perl, fourth edition, O'Reilly, 2005

`www.cs.tut.fi/~jkorpela/perl/intro.html`

`www.perl.org`

Edit een file `hallo.pl`, met daarin:

```
#!/usr/bin/perl -w
# dit is een regel met commentaar
print "Goedenavond.\n";
```

Perl *interpreteert* (een kleine broer van compileren) dit "script". De regel met `#!` onthult het pad naar de binary, op niet-Unix systemen: `#!perl`; de `-w` zet waarschuwingen aan. Let er op dat de file `hallo.pl` executable moet zijn.

Perl is in (vanaf) 1987 door Larry Wall bedacht. Huidige versie: 5.zoveel, we wachten op Perl 6.

Een ingewikkelder programma:

```
#!/usr/bin/perl -w
@regels = `perldoc -u -f atan2`;
foreach (@regels) {
  s/\w<([^>]+)>/\U$1/g;
  print;
} # foreach
```

Dit stopt de uitvoer van het Unix-commando `perldoc ...` (let op de `...`, backquote's) in aparte regels, vervangt in ieder van die regels (bijvoorbeeld) `C<acDEf%%>` door `ACDEF%%`, en drukt dat af. Door `\U` wordt "geheugen" `$1` in hoofdletters omgezet, zie verderop.

Variabelen in Perl zijn niet getypeerd, en hoeven niet gedeclareerd te worden:

```
#!/usr/bin/perl -w
$een = "aap";
$twee = "Ik ben een $een\n";
print $twee;
$een = 42;
$twee = "Maar nog " . "geen $een\n";
print $twee;
```

Let op de automatische "interpolatie" in de variabele `$twee` (gebeurt niet als er enkele quotes om de string staan) en op de stringconcatenatie (met een punt).

Array's in Perl:

```
#!/usr/bin/perl -w
@vanalles = ("tja",42,"Okee  ",-1.34);
for ( $i = 0; $i <= $#vanalles; $i++ ) {
  print "$i-de element: $vanalles[$i]\n";
} # for
```

Array's beginnen bij `0`. Ze groeien/krimpen in zekere zin "automatisch"; `$#vanalles` is de index van het laatste array-element.

Let ook op het overvloedig gebruik van accolade's. Bij een `if`-statement *moet* het:

```
#!/usr/bin/perl -w
if ( ! defined ($i) ) {
  print "Ongedefinieerde variabele\n";
} # if
```

array's — vervolg

Het doorlopen van array's kan ook zo:

```
#!/usr/bin/perl -w
@steden = ("Leiden","Den Haag","Utrecht");
foreach $stad (@steden) {
  print "We zijn nu in $stad\n";
} # foreach
```

Of nog korter met verborgen variabele `$_`:

```
#!/usr/bin/perl -w
@steden = ("Leiden","Den Haag","Utrecht");
foreach (@steden) {
  print "We zijn nu in $_\n";
} # foreach
```

Er zijn meer verborgen variabelen beschikbaar. En voor liefhebbers de *hash*, `%info`, met:

```
$info{"feest"} = "23 juni, 14.00 uur";
```

Lezen vanuit een file:

```
#!/usr/bin/perl -w
$tel = 0;
open (DEFILE, "<index.html");
while ( <DEFILE> ) {
  if ( /<a/i ) {  # komt <a of <A voor?
    print $_;
    $tel++;
  } # if
} # while
close (DEFILE);
print $tel, " speciale regels\n";
```

De < opent de file voor input.
En weer die verborgen variabele $_!

Zoeken en vervangen gaat aan de hand van *reguliere expressies*. Enkele voorbeelden:

```
$iets =~ s/abc/defg/g;
# vervang elk (dankzij /g) voorkomen
# van abc in de variabele $iets door defg
if ( $iets =~ /ab?c*d+[A-D]/ ) { ...
# zit in $iets naast elkaar een a,
# dan NUL of EEN b, NUL of MEER c's,
# en EEN of MEER d's, gevolgd door
# een hoofdletter A of B of C of D?
```

Ze kunnen zeer complex worden! Zo matcht `\w` één letter of underscore, `\d` één cijfer, is `|` of, zijn `^` en `$` begin en eind van de regel (maar `[^x]` betekent "niet `x`"), en forceren ronde haakjes niet alleen een prioriteitsvolgorde maar ook een "geheugen" in `$1`, `$2`, ...