Samenvatting Perl: Perl, Joris' way

Algemeen

Scalair / Array / Hash

```
$scalair = "";
@array = ();
%hash = (1 => "waarde", 2 => "tweedewaarde");

# Speciale zaken bij hashes
@keys = keys %hash;
@values = values %hash;

# Checken of een sleutel bestaat
if exists $hash{key};
if defined $hash{key};

# Referenties naar arrays en hashes
$array = [];
$hash = {};
```

Lijst cyclisch overlopen

Basic idee: Wanneer je een element pop/shift moet je het terug op de omgekeerde plaats in de lijst zetten. Als je `pop` doet (achteraan verwijderen), doe dan `unshift` van die waarde (vooraan toevoegen). Als je `shift` (vooraan verwijderen) doe dan `push` (vooraan toevoegen)

```
# Uitgeschreven manier
@array = (1, 2, 3);

while(1){
    $value = pop @array; # of hier shift
    print $value, "\n";
    unshift(@array, $value); # of hier push
    sleep 1;
}
```

Variabele ($a) instellen op default waarde als een andere ($b) geen waarde had

```
$c = "default";
$a = $b || $c;
```

Cyclisch wisselen

```
($a, $b) = ($b , $a);
```

Omzetten naar ascii waarden en omgekeerd

```
$char = "k";
$ascii = ord($char);
chr($ascii);
```

reverse op een string

```
# scalar is belangrijk, omgekeerde letters:
print scalar reverse $string

# Omgekeerde woorden zo
$omgekeerdeWoorden = join " ", reverse split(" ", $string);
```

Upper- en lowercase

```
uc()
lc()
```

Afronden

```
$floating = 0.5666996969;

# afronden
my $afgerond = sprintf("%0.2f", $floating);

# Vergelijken
print "groter" if sprintf("%0.2f", 3.147728) > sprintf("%0.2f", 3.1415167349);

# vergelijkingsoperatoren voor vergelijken als strings. De gewone is volgens nummer
# Als je dus alfabetisch wilt moet je volgende gebruiken:
gt
lt
ge
le
```

Random getal genereren

```
srand($getal) # seeden
rand() # getal genereren tussen 0 en 1
rand(5) # random getal tussen 0 en 5, nul inbegrepen, 5 niet

# Wachtwoord genereren tussen bepaald bereik:
$begin = 6;
$einde = 9;

$lengte = 10;
```

```
$password .= int(rand($einde - $begin + 1) + $begin ), "\n" foreach (1..$lengte);
print($password);
```

Hexadecimaal / Octaal / Binair

```
# oct kan alles naar integers omzetten
oct("0x2f");
oct("01010101");
oct("081");
```

Bepaalde elementen uit een array halen

```
# undef skipt een element in een lijst
($eerste, $tweede, undef, \$derde) = @array;
```

Array initialiseren

```
@array = ("Ik", "ben", "een", "zin");
@array = qw(Ik ben een zin);
```

Elementen van een array uitschrijven met komma's tussen elk element, behalve het laatste

We gebruiken hier **slicing**

```
@array = (1, 2, 3, 4, 5, 6, 7);
print join(", ", @array[0..scalar @array - 2]), " en ",  $array[-1];
```

Lijst vergroten

Gewoon op een index waar nog niets stond toevoegen.

```
@array = (1, 2, 3, 4, 5, 6, 7);
$array[10] = 3;
```

Array: toevoegen en verwijderen

```
push @array, "een waarde";
push @array, ("een", "lijst", "van", "waarden");

# Achteraan verwijderen
$var = pop @array;

# Vooraan verwijderen
$var = shift @array;
```

```
# Meerdere elementen verwijderen
@elementen = splice(@array, 0, 2); # Eerste twee verwijderen
@elemnten = splice(@array, -2); # Laatste twee verwijderen
```

Array circulair overlopen

Sorteren

```
sort @array; # Sorteert gewoon
reverse sort @array; # Omgekeerd sorteren (meestal alfabetisch)
sort {$a operator $b} @array; # Uitgebreid sorteren
sort {$a <=> $b} @array; # Spaceship operator, voor numeriek ordenen.
```

Duplicaten verwijderen

Met behulp van een hash

```
@array = (1,1,1,2,2,2,3,3,3);
$hash{$_} = undef foreach @array;
@array = keys %hash;

# In 1 lijn en met grep
%seen; # Moet er zelfs niet staan
@array = grep { !$seen{$_} ++ } @array; # Doet meteen ook ++ als het elemnent er
niet inzat.
```

Elmenten bepalen die in 1 lijst voorkomen maar niet in een andere

```
@a1 = (1, 2, 3, 4, 5, 6);
@a2 = (1, 3, 4, 5);

# In een hash steken en alle waarden van een bepaalde lijst verwijderen
$seen{$_} = undef foreach @a1;
delete @seen{@a2};

@result = keys %seen;
```

Hash overlopen

```
while(($key, $value) = each(%hash)){ # Doe iets met $key en $value}
print "$_ => $hash{$_} \n" foreach keys %hash; # Met forloop
```

Hash gesorteerd overlopen

```
print "$_ => $hash{$_}" foreach sort { $a <=> $b } keys %hash; # indices
print "$_ => $hash{$_}" foreach sort { $hash{$a} <=> $hash{$b} } keys %hash; #
waarden
```

Verwerken van bestanden

Bestand inlezen

<> zal bestanden vanaf de command line inlezen. Ge kunt ook gewoon @ARGV aanpassen en er gewoon de naam van het bestand insteken.

```
# Lijn per lijn inlezen
while(<>){
    $line = chomp($_); # End of line weghalen uit line
}

# Bulk inlezen
@lines = <>;

# Bulk inlezen en in 1 string zetten
$file = join "", @lines;

# Omgekeerd verwerken
@lines = reverse @lines;
```

Bestand aanpassen

```
@ARGV_COPY = @ARGV;


# Eerst lijnen inlezen en dan iets mee doen.
while(<>){
    chomp($_);
    push @lines, $_ . " Vervolg";
}

$^I = ".bak"; # Backup maken
@ARGV = @ARGV_COPY;
$ln = 0;

# Hier begint het wegschrijven
while(<>){
    print $lines[$ln], "\n";
    $ln++;
}
```

Lijnnummers

$. bevat het rijnummer.

```
while (<>){
    print "$. $_";
}
```

Paragraafmode

`$/` bevat de paragraafmode

```
$/ = "";
```

Regexen in Perl

Basics

```
# Gebruiken van algemene variabelen
$string = "1 2 3 4";
$string =~ m/(\d) (\d) (\d) (\d)/;
print "$1 $2 $3 $4";

# Direct in variabelen steken
($a, $b, $c, $d) = ($string =~ m/(\d) (\d) (\d) (\d)/);
```

Regex vlaggen

```
g; # global:  Don't return after first match
i; # insensitive: case insensitive matching
s; # single line: dot matches newline
m; # multi-line: ^ matches beginning of line and $ matches end of line
   # ==> HANDIG om samen met s te gebruiken

# Standaard zijn identifiers greedy

# globaal aanpassen:
U; # ungreedy: identifiers are nongreedy

# Per identifier met ?
"/d*?"
```

chop en chomp

`chop` verwijdert het laatste teken van een zin.

`chomp` verwijdert de laatste newlines (enkel `"\n"`).

substr

```
$string = "Cedric";
substr($string, 0, 3); # Ced
```

```
substr($string, -2); # ic

substr($string, 0, 6) = "Anthony"; # De string is vervangen door Anthony
substr($string, 0, -2) = "oc" # Cedroc;
```

File verwerken waarbij \ een continuation karakter is

Continuation karakter wil zeggen dat de lijn nog niet klaar is.

```
@string = <DATA>;
$string = join "", @string;

while($string =~ m/(.*?)[^\\]$/gms){
    print $1;
    print "\n\n";
}


__DATA__
DISTFILES = $(DIST_COMMON) $(SOURCES) $(HEADERS) \
        $(TEXINFOS) $(INFOS) $(MANS) $(DATA)
DEP_DISTFILES = $(DIST_COMMON) $(SOURCES) $(HEADERS) \
        $(TEXINFOS) $(INFO_DEPS) $(MANS) $(DATA) \
        $(EXTRA_DIST)
```

Bespreking:

```
$string =~ m/(.*?)[^\\]$/gms
```

- `(.*?)` matchet non greedy alle tekens. Dit is nodig om ervoor te zorgen dat niet het hele bestand in 1 keer wordt ingelezen
- `[^\\]` matchet niet `\`. Omdat `\`een speciaal karakter is moet het escaped worden
- `$` matchet het einde van een lijn
- `g` : global, doe dit voor alle voorkomens. Dit is nodig om niet in een oneindige loop te blijven zitten met de while
- `m` : multiline, maakt het mogenlijk om `^` en `$` als respectievelijk begin en einde van de lijn te nemen
- `s` : single line. `.` wordt nu ook gezien als een newline. Dit is nodig om over meerdere lijnen te kunnen lezen

Split

Vaak kunnen regexen makkelijker geschreven worden door middel van `split`. Bv om een zin in woorden te verdelen met een bepaalde delimeter:

```
$string = "Ik ben een zin met woorden";
$delimeter = " ";
@woorden = split($delimeter, $string);
```

Een deelstring vervangen door een variabele

```
$string = "Ik ben \$naam";
$naam = "Cedric";

# Let op de ${}, $ is hier de globale hash
$string =~ s/\$naam/${naam}/g;
```

Enkel lijnen inlezen tussen specifieke codes

Dit is handig voor het filteren van een bestand zodat we direct kunnen skippen tot een bepaalde code.

Bemerk het verschil tussen `..` en `...`

- `..` : codes inclusief
- `...`: codes exclusief

Je kan dit ook met lijnnummers doen ipv met regexen.

```
while(<DATA>){
    if(/begin/ .. /end/){ # of ... als je de codes niet inclusief wilt
        print $_;
    }
}

__DATA__
begin
Ik ben een lijn die moet genomen worden
Ik ook
end
Ik niet
begin
Ik wel
end
Ik niet
```

Backreferences

Om bijvoorbeeld dubbele woorden te vinden in een tekst:

```
@lines = <DATA>;
$string = join "", @lines;

while($string =~ /(\w+)\s\1/gs){
    print $1, "\n";
}

__DATA__
tekst tekst ik ben een woord woord
cedric ik ben een tekst woord cedric
cedric lol lol
```

Bespreking:

- `\w`matchet een karakter

- `\s` matchet alle spaces, in combinatie met de s identifier matcht deze ook newlines
- `\1` verwijst terug naar de het eerste match die tussen haken staat.

Referenties

== pointers in Perl

```
$string = "hallo";
$ref = \$string; # maakt pointer naar $string

# Je kan dat ook doen met arrays en hashes
@array = ();
$ref = \@array;

%hash = ();
$ref = \%hash;

# Dereferen
$ref->[0] # element uit een lijst ophalen
$ref->{key} # element uit een hash ophalen
```

Het handige aan referenties is dat je nu structuren kunt combineren. Op het examen bv een raster met daarin allemaal hashes kan je dan zo uitpakken:

```
$speelveld; # ref naar een tweedimensionale array gevuld met hashes
$speelveld->[0][0]{key} # Pijlen tussen opeenvolgende [] en {} moet je niet
schrijven
```