

Skriptsprachenorientierte Programmietechnik

Perl Teil 5

Prof. Ilse Hartmann

5 Hashes

5.1 Einführung Hashes

5.2 Definition Hash

5.3 Zugriff auf Hashelemente

5.4 Keys und values

5.5 each

5.6 Exists und delete

5.7 Zugriff auf Hashelemente - 2

5.8 Hash an Liste zuweisen

5.9 Hash-Funktionen

- Auch „**assoziative Arrays**“ genannt
- Datenstruktur zur **Speicherung von Werten**
- Im Gegensatz zum Array erfolgt die Referenzierung nicht über Zahlen (Indizes), sondern über **eindeutige** Namen (*Schlüssel*)
- Im Gegensatz zum Array also eine **ungeordnete** Ansammlung von Werten

Hash

“alter“

35

“ab_3“

“Hallo“

“test“

1.72e30

key
value

“2.5“

12.4

“fritz“

“Hey!\n“

Zum Vergleich: Liste / Array

	0	1	2	3	4
Index					
value	35	12.4	“Hallo“	1.72e30	“Hey!\n“

- Kennzeichen für Hashvariablen: % (Prozentzeichen)
- In einer Liste werden je zwei Elemente als **Schlüssel/Wert-Paar** interpretiert
- Über den **=>-Operator** werden die Paare deutlicher dokumentiert
- Die Werte links des **=>-Operators** werden als Key und somit als String interpretiert
- Hashes und Arrays können ineinander überführt werden (*unwinding*)

Beispiel:

```
my %telefon = ('Fritz', '(01234) 5678', 'Alma', '(089) 998876',
```

```
    'Martin', '069-47110815');
```

```
my %ip_adressen = ('www.fom.de' => '62.225.8.73',
```

```
    'www.google.de' => '209.85.135.103');
```

```
my %person = (nachname => 'Mustermann',
```

```
    vorname => 'Max',
```

```
    alter => 65 );
```

```
my %etwas = @liste;
```

```
my @neueliste = %etwas;    # Achtung! Nicht zwangsläufig in derselben  
                           # Reihenfolge wie @liste!!  
                           # Paare bleiben jedoch erhalten
```

- Nutzung von Schweifklammern (`{}`)
- Wie üblich kann anstelle eines Literals ein beliebiger Ausdruck als Schlüssel dienen
- Schlüsselnamen sind stets eindeutig. Bei Zuweisungen an bestehende Hash-Elemente werden die Werte daher überschrieben
- Nicht-existente Elemente werden wie üblich als **undef** ausgegeben

Beispiel

```
my %person = (nachname => 'Mustermann',  
              vorname => 'Max',  
              alter => 65 );  
print "Nachname: $person{'nachname'}\n";  
$person{'groesse'} = 1.79;    # Neuer Eintrag wird erzeugt  
my ($v, $n) = ("vor", 'name');  
print $person{$v . $n} . "\n"; # Ausdruck für Key „vorname“  
$person{"alter"} = 66;        # Bestehender Wert wird überschrieben  
print "PLZ der Person: '$person{'plz'}'\n";    # undef
```


- Die **keys**-Funktion liefert eine Liste aller Schlüssel eines Hashes
- **values** liefert alle Werte eines Hashes
- Die Reihenfolge ist weder bei keys noch bei values vorhersagbar
- Allerdings ist die Reihenfolge jeweils dieselbe, falls der Hash nicht verändert wird
- Natürlich können die Listen **per sort** sortiert werden
- In skalarem Kontext liefern **keys und values** die Anzahl der Elemente

Beispiel:

```
my %hash = (a => 1, b => 2, 'c d' => 3, 4 => 'e');
my @schluessel = keys (%hash);
my @werte = values (%hash);
print "Alle Schlüssel: @schluessel\n";
print "Alle Werte: @werte\n";
foreach my $key (sort(keys(%hash)))
{
    print "Wert für '$key': $hash{$key}\n";
}
my $anzahl = keys(%hash);
print "Es sind $anzahl Elemente im Hash.\n";
```

Hier kein Leerzeichen!



- Die **each**-Funktion liefert in einer Iteration jeweils ein Schlüssel-Wert-Paar als Liste aller Schlüssel eines Hashes
- Wenn kein weiteres Paar vorhanden ist, wird eine leere Liste zurückgegeben

```
my %hash = (a => 1, b => 2, c => 3);  
my ($schluessel, $wert);  
do  
{  
    ($schluessel, $wert) = each (%hash);  
    print "Wert für '$schluessel': $wert\n";  
} while ($schluessel);
```

- Deutlich eleganter: Wert der Listenzuweisung wird als while-Bedingung im skalaren Kontext ausgewertet, also als Anzahl der Elemente

```
while ( my ($schluessel, $wert) = each %hash )  
{  
    print "Wert für '$schluessel': $wert\n";  
}
```

- Die Funktion **exists** prüft, ob ein bestimmter Schlüssel im Hash existiert
- Mit **delete** kann ein Schlüssel/Wert-Paar aus dem Hash entfernt werden
- Dies ist nicht zu verwechseln mit der Zuweisung von **undef** an ein Element!

```
my %telefon = (Max => '(0611) 12345', Frieda => '(040) 4711');  
if (exists ($telefon{"Max"}))  
{  
    print "Max hat ein Telefon.\n";  
}  
$telefon{'Max'} = undef;      # Eintrag existiert weiterhin!  
print "Max hat zwar einen Anschluß, aber seine Nummer kenne ich  
    nicht!\n";  
delete $telefon{Max};        # Element aus Hash entfernen  
print "Der Anschluß von Max wurde zerstört!\n";
```

- Erinnerung: **Array: `$array[$index]`**, wobei Index eine Zahl ist
- Hash: **`$hash{$schluessel}`**, wobei `$schluessel` ein String ist

- **Beispiel:**

```
$AdressDaten{"Name"}
```

```
$AdressDaten{Adresse} # " optional
```

```
$AdressDaten{Telefonnummer}
```

```
$AdressDaten{"Name"} = "Roland"; $AdressDaten{Adresse} = "Linz";
```

```
$AdressDaten{Telefonnummer} = 0732;
```

oder:

```
%AdressDaten = ("Name" => "Roland", Adresse => "Linz",  
Telefonnummer => 474);
```

```
%AdressDaten = ("Name", "Roland", "Adresse", "Linz",  
"Telefonnummer" ,474)
```

Ausgabe:

```
print $AdressDaten{"Adresse"};           # 1 Element

print "@ { [ %AdressDaten ] } ";         # Alles, wie "@array,,

# $key Variable, keys Schlüsselwort
foreach $key ( keys %AdressDaten){
    print $key." = ".$AdressDaten{$key}."\n";
}
```

- **Unwinding:** Hash an Liste zuweisen, Reihenfolge muss nicht bleiben:

```
%some_hash = ( "piffi", 7, "puffi", 8, "paffi", 9);  
@some_array = %some_hash;
```
- **Wichtig beim Umkehren:** Schlüssel dienen nun als Index und müssen eindeutig sein. Wegen Unordnung des Hashs intern kann nicht entschieden werden, welcher Schlüssel benutzt wird!

Keys –Funktion zur Ausgabe aller Schlüssel.

Values-Funktion zur Ausgabe aller Werte, z.B.:

```
@keys=keys (%a);  
@values = values(%a);  
print @keys; # Anzahl Schlüssel  
print "@keys\n"; # Schlüssel selber  
print @values;  
print "@values";
```

Boole'scher Ausdruck
Solange Hash nicht leer

Funktion each zur Iteration über Schlüssel/Wert Paare:

```
while (($key, $values) = each %hash) { ...}
```

Listenkontext

Skalarer Boole'scher Kontext

- **Name eines Lesers ist Schlüssel**, Wert gibt an wie viele Bücher ausgeliehen sind, oder ob der Ausweis noch nie benutzt wurde.
- **Funktion exists**: Existiert ein Hash(index) mit diesem Wert,
- Beispiel:

```
if (exists $some_hash{"piffi"}){  
    print "Piffi hat einen Benutzerausweis\n";  
}
```

- **Funktion delete**: Löscht Schlüssel mit Wert aus Hash (gegenüber = undef, dass nur den Wert überschreibt)
- **Interpolation** nur Elementweise, nicht das Ganze Hash:

```
print "Blabla: $some_hash{$sth}\n";
```