6) Hashes

Matt Webster
IMBIM, BMC
matthew.webster@imbim.uu.se

Reminder: scalars

- variables in perl (e.g. `$example`) can be numbers or strings
- perl converts variable between the two depending on the operation
- backslash `\` used to indicate special characters like newline `\n` and tab `\t`
- compare numbers or strings with comparison operators and `if`
- `<STDIN>` gets input from the keyboard
- `chomp` removes `\n`
- `while` repeats a chunk of code while an expression is true

Reminder: arrays

`$array[0]`
- first element of `@array`

`$#array`
- is the final index of `@array`

`qw//`
- quoted by whitespace

`push pop`
- add or remove elements from the end

`shift unshift`
- add or remove elements from the start

`splice`
- add, remove, replace elements from the middle

`foreach`
- cycle through every element

`sort reverse`
- change order

`<STDIN>` can also be read into an array

reminder: subroutines

- subroutines
  - called by `&sub_example` or `sub_example()`
  - defined using `sub { }`
  - use `return` or it will return last evaluated value
  - arguments passed within:

  `@_`

  `$_[0], $_[1], $_[2]`….

- `my` is used to restrict a variable to a block of code
- `use strict` means all variables must be declared using `my`

reminder: input and output

```
while (<STDIN>) {  }
```

- to read from standard input

```
<>
```

- to process invocation commands and read in files

```
@ARGV
```

- contains the invocation commands (useful)

```
printf
```

- formats scalars (e.g. decimal points on numbers)

```
open FH,"input.txt";
```

- opens a filehandle for reading

```
open FH,">output.txt";
```

- opens a filehandle to write (>> to append)

```
while (<FH>) {  }
```

- reads each line of a filehandle

```
die
```

- your script dies!

scalar variables, arrays and hashes

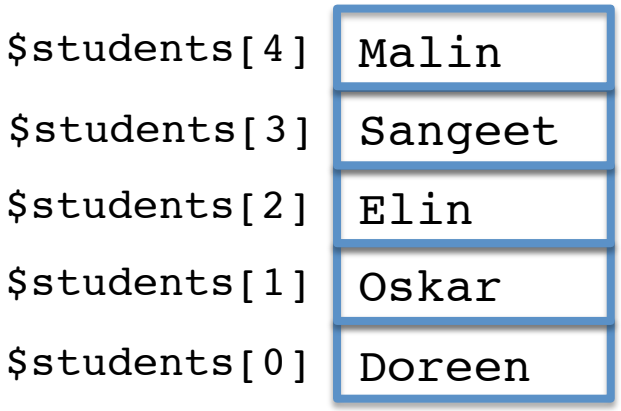
- A scalar variable contains a single string or number:

```
$month="October";
$month=10;
```

scalar variables, arrays and hashes

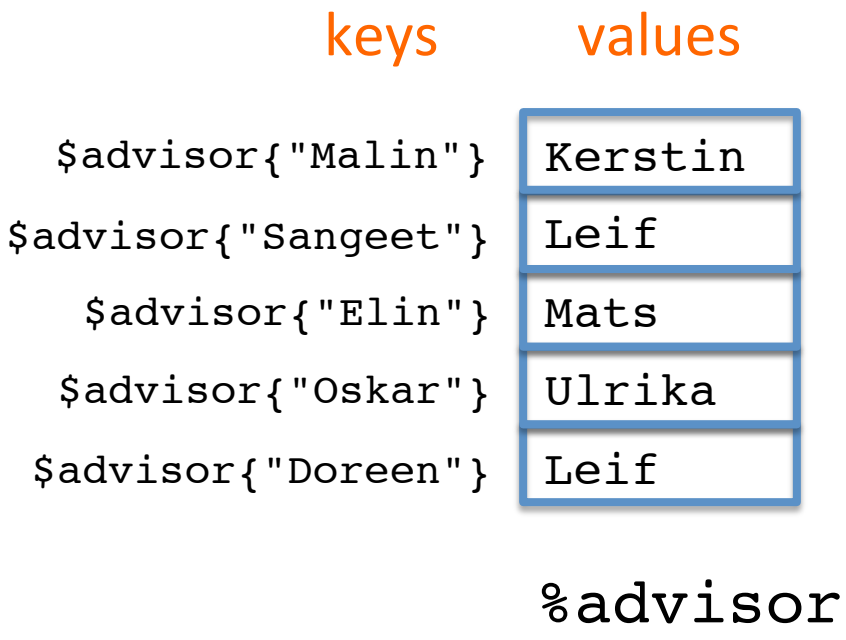
- An array contains a list of values with a *numbered* index

```
@students = qw/Doreen Oskar Elin Sangeet Malin/;
```



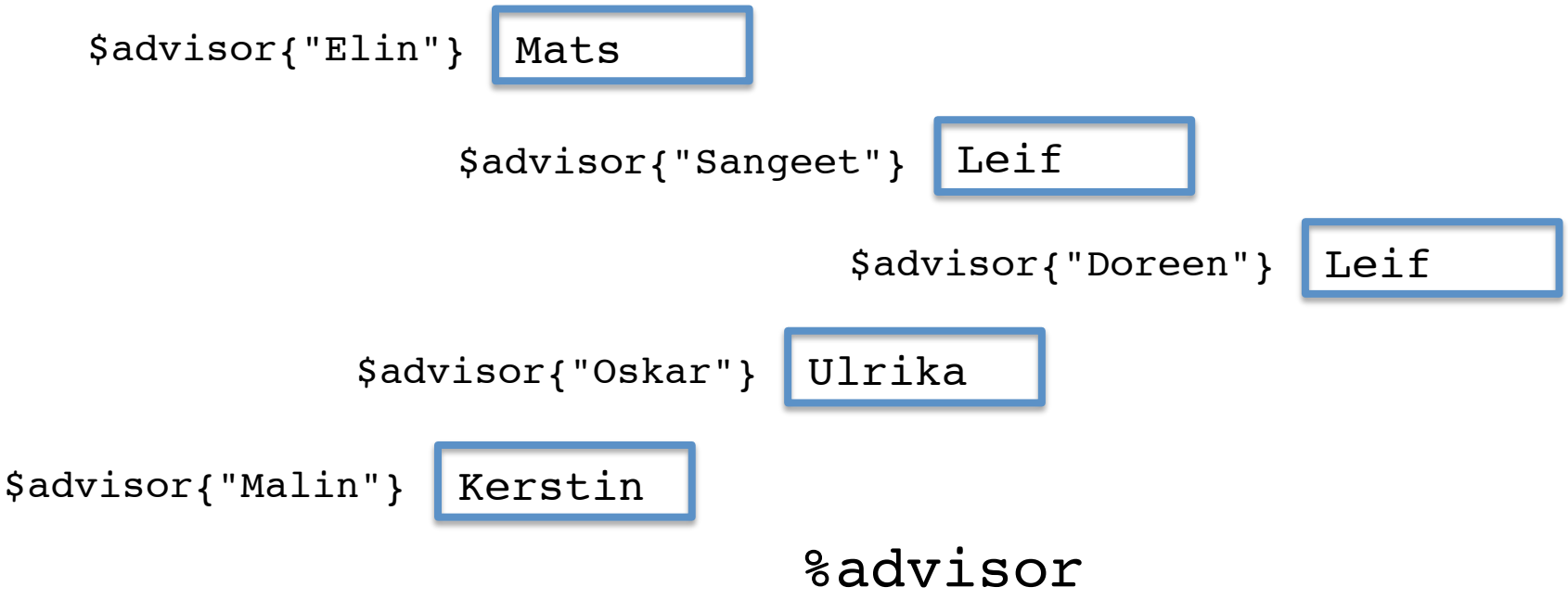
scalar variables, arrays and hashes

- An hash contains a set of values with an index of *names*



scalar variables, arrays and hashes

- unlike arrays, hashes are not in order



hashes

- hashes have *keys* and *values*
- perl can very quickly find the value that matches a key, even if the hash is huge
- similar to looking up something in a database

hashes have many uses

- there are a huge number of reasons why you might want to store a value with a key
- how many times does something occur?
  - how many times does A occur in a sequence?
  - how many times does a SNP occur in a file?
  - how many different SNPs are there in a file?
- matching keys and values
  - storing the sequence of an individual ID
  - translating two gene IDs
  - barcode of sample ID
  - sex of sample

hashes are used in alignment

build an index of the reference sequence for fast access

seed length 7

```
0     5     10    15

GACCTCATCGATCCCACTG
GACCTCA              →  chromosome 1, pos 0
 ACCTCAT             →  chromosome 1, pos 1
  CCTCATC            →  chromosome 1, pos 2
   CTCATCG           →  chromosome 1, pos 3
    TCATCGA          →  chromosome 1, pos 4
     CATCGAT         →  chromosome 1, pos 5
      ATCGATC        →  chromosome 1, pos 6
       TCGATCC       →  chromosome 1, pos 7
        CGATCCC      →  chromosome 1, pos 8
         GATCCCA     →  chromosome 1, pos 9
```

hashes are used in alignment

build an index of the reference sequence for fast access

```
TCGATCC ?

                0    5    10   15

                GACCTCATCGATCCCACTG
                GACCTCA             →  chromosome 1, pos 0
                 ACCTCAT            →  chromosome 1, pos 1
                  CCTCATC           →  chromosome 1, pos 2
                   CTCATCG          →  chromosome 1, pos 3
                    TCATCGA         →  chromosome 1, pos 4
                     CATCGAT        →  chromosome 1, pos 5
                      ATCGATC       →  chromosome 1, pos 6
                       TCGATCC      →  chromosome 1, pos 7
                        CGATCCC     →  chromosome 1, pos 8
                         GATCCCA    →  chromosome 1, pos 9
```

hashes are used in alignment

build an index of the reference sequence for fast access

```
TCGATCC = chromosome 1, pos 7

          0    5    10   15

          GACCTCATCGATCCCACTG
          GACCTCA            →  chromosome 1, pos 0
           ACCTCAT           →  chromosome 1, pos 1
            CCTCATC          →  chromosome 1, pos 2
             CTCATCG         →  chromosome 1, pos 3
              TCATCGA        →  chromosome 1, pos 4
               CATCGAT       →  chromosome 1, pos 5
                ATCGATC      →  chromosome 1, pos 6
                 TCGATCC     →  chromosome 1, pos 7
                  CGATCCC    →  chromosome 1, pos 8
                   GATCCCA   →  chromosome 1, pos 9
```

Accessing hashes

```
@students=qw/Elin Oskar/;
$advisor{"Oskar"} = "Ulrika";
$advisor{"Elin"} = "Mats";

foreach my $person (@students) {
   print "advisor of $person is $advisor{$person}\n";
}
```

```
advisor of Elin is Mats
advisor of Oskar is Ulrika
```

Another way to do it:

```
@students=qw/Elin Oskar/;
$advisor{"Oskar"} = "Ulrika";
$advisor{"Elin"} = "Mats";

foreach (@students) {
   print "advisor of $_ is $advisor{$_}\n";
}
```

```
advisor of Elin is Mats
advisor of Oskar is Ulrika
```

hash assignment

```
$advisor{"Oskar"} = "Ulrika";
$advisor{"Elin"} = "Mats";
```

the same is this:

```
%advisor = ("Oskar","Ulrika","Elin","Mats");
```

but this is better:

```
%advisor = (
    "Oskar"   => "Ulrika",
    "Elin"    => "Mats",
)
```

then you can see which are the key=>value pairs

unwinding a hash

```
%advisor = ("Oskar","Ulrika","Elin","Mats");

@names=%advisor;

# @names will have all the keys-value pairs, although in any order

my %student = reverse %advisor;

# this exchanges keys and values
# so now the names of advisors are the keys
```

keys and values

This part is very important!

```
%advisor = (
    "Oskar"   => "Ulrika",
    "Elin" => "Mats",
)
foreach $student (sort keys %advisor) {
    print "advisor of $student is $advisor{$student}\n";
}
```

```
advisor of Elin is Mats
advisor of Oskar is Ulrika
```

keys and values

```
%advisor = (
    "Oskar"  => "Ulrika",
    "Elin"   => "Mats",
)

@student_list = keys %advisor;
@advisor_list = values %advisor;



$number_of_students = keys %advisor;
# gives the number of keys (scalar context)
```

counting occurrences with hashes

```
foreach $student (@full_student_list) {
   $name_count{$student}++;
}
foreach (keys %name_count) {
   print "$name_count{$_} person called $_\n";
}
```

```
1 person called Maria
2 person called Oskar
1 person called Camille
1 person called Anders
.....
```

`exists` and `delete`

```
if (exists $advisor{"Maria"}) {
   print "Maria has an advisor\n";
}

delete $advisor{"Maria"};
```

this removes this value

there is a difference between a value that doesn't exist and one that is undefined (`undef`)

%ENV

`%ENV` contains environmental variables

$ENV{PATH} contains the PATH (where the operating system looks for commands such as perl and ls):

```
PATH = /usr/bin:/usr/local/bin:
```

possible to access this (but unlikely you will need to)

summary: hashes

- a hash is a group of key-value pairs
- you can quickly retrieve the value for each key
- this data structure is extremely useful e.g. for
  - counting occurrences
  - associating data (such as different IDs)
- summary of syntax:

```
%hash
$hash{$key}=$value
keys %hash
values %hash
exists $hash{$key}
delete $hash{$key}
```