3) Lists and arrays

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

lists and arrays

- A list is an ordered set of scalar values:

```
(1,2,3,"Matt")
```

- An array is a variable that holds a list:

```
@arr = (1,2,3,"Matt");
print @arr;
123Matt
```

index	0	1	2	3
value	1	2	3	Matt

- You can access an individual array element:

```
print $arr[1];
2

$arr[0] = "*";
print @arr;
*23Matt
```

perl starts counting at 0

lists and arrays

You can easily get a sub-array:

```
@arr = (1,2,3,"fred","bob");
print @arr;
123fredbob
print $arr[1];
2
my @sub_arr = @arr[1..3]
print @sub_arr;
23fred
```

You can extend an array as much as you like:

```
@arr2 = (1,2,3)
$arr2[5] = 6;

@arr2          # is now (1,2,3,undef,undef,6)
```

special array indices

```
$students[0] = "Doreen";
$students[1] = "Oskar";
$students[2] = "Elin";
$students[3] = "Sangeet";
$students[4] = "Malin";

$end = $#students;        # 4, the final index
print $students[$end];   # same as $students[-1]
Malin
```

another way:

```
@students = ("Doreen","Oskar","Elin","Sangeet","Malin");
```

and another way:

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

`qw` means quote with whitespace

list literals

- all these are the same:

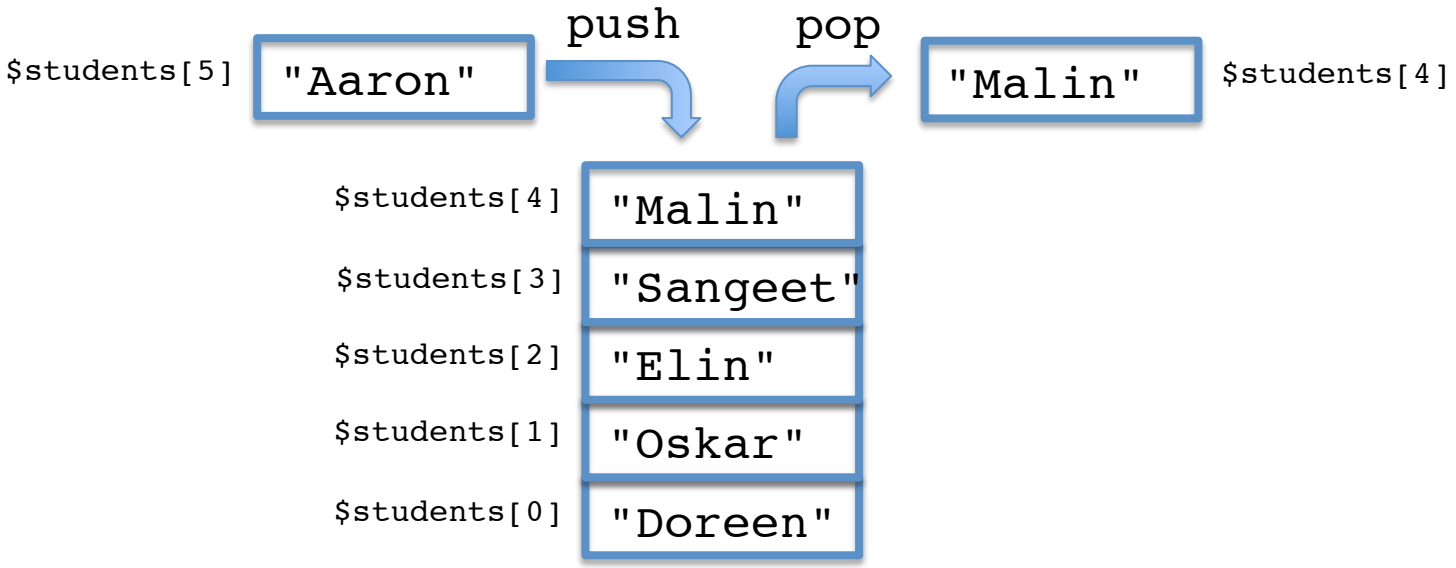
```
(1,2,3,4,5,6,7,8,9,10)
(1..10)
(1,2,3..9,10)
qw/1 2 3 4 5 6 7 8 9 10/

($i..$j)      #numbers between $i and $j
(0..$#students)  #indices of @students
```

qw can be used with a lot of characters:

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

@students = qw{Doreen Oskar Elin Sangeet Malin};
@students = qw<Doreen Oskar Elin Sangeet Malin>;
```

push and pop

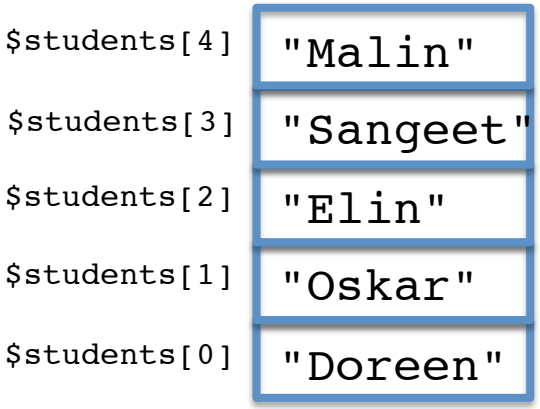
```
@students = qw/Doreen Oskar Elin Sangeet Malin/;
```




```
$final_student = pop(@students)
#"Malin"  is removed from  @students  and placed in  $final_student

push(@students,"Aaron");
#"Aaron"  is added to the end of @students

pop @students
#"Aaron"  is removed from  @students  and discarded
```

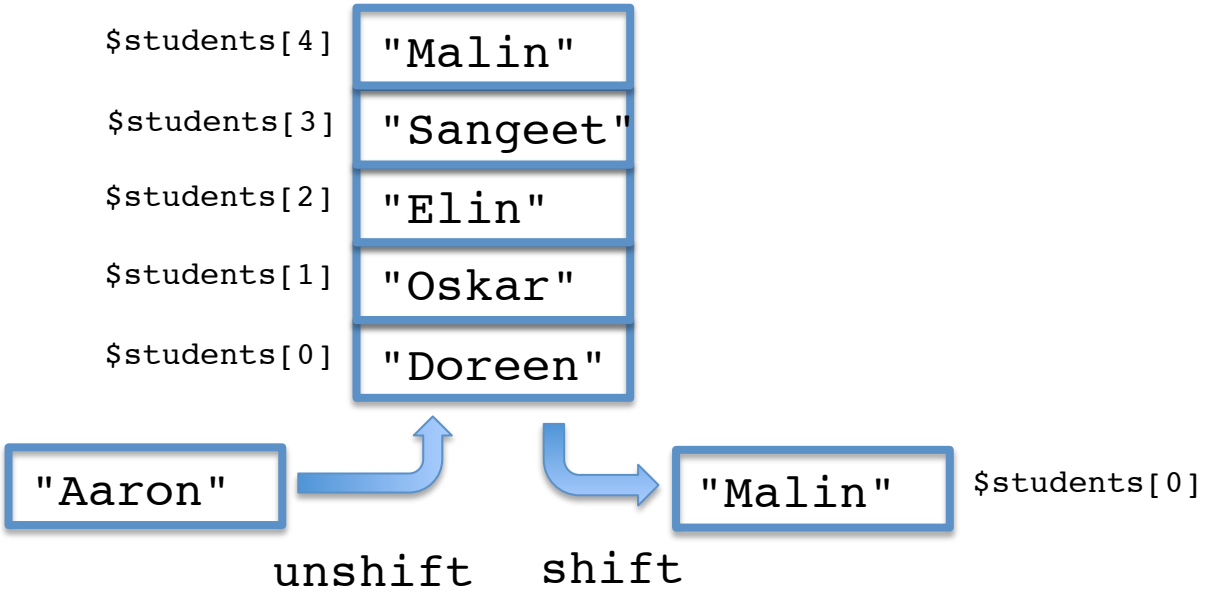
push and pop

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




```
@new_students = qw/Ingegerd Wael Daniel/;

push @students,@new_students;
# now @students has 8 elements
```

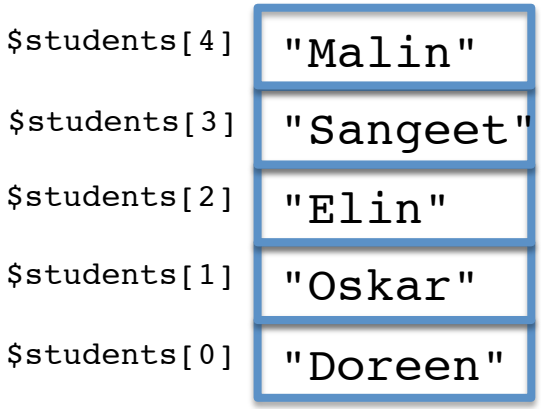
shift and unshift



Similar to push and pop, but acts on the start of the array
all of the indices are shifted when a new element is added or taken away

```
$first_student = shift @students;
# $first_student equal to "Doreen", @students loses an element

unshift(@students,$first_student);
# @students is back to normal
```

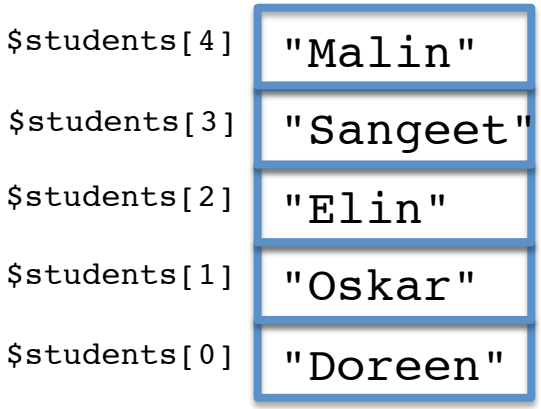
splice



usage: `splice (@array, start, length, replacement)`

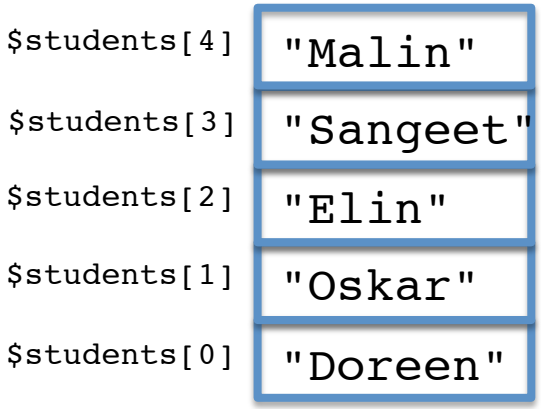
```
@removed = splice (@students,3);
# "Sangeet" and "Malin" placed in @removed

@removed = splice (@students,2,2);
# "Elin" and "Sangeet" placed in @removed

@removed = splice (@students,2,1,"Maria");
# "Elin" is placed in @removed and "Maria" in $students[2]
```

foreach




```
foreach $person (@students) {
    print "$person is taking a perl course\n";
}
```

foreach

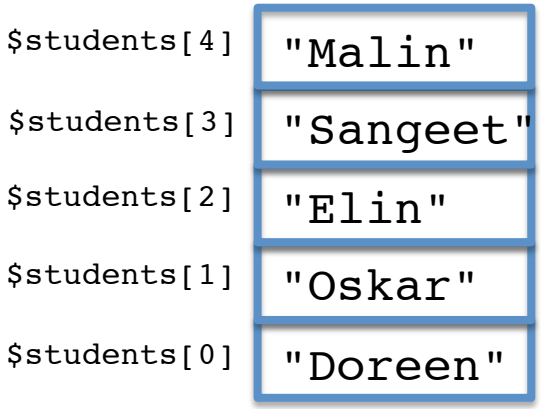



```
foreach $person (@students) {
    print "$person is taking a perl course\n";
}

foreach (@students) {
    print "$_ is taking a perl course\n";
}
```

both do the same thing

$_ is a default variable that stores the last value

reverse and sort




```
@reversed_students = reverse (@students);
# in the opposite order

@sorted_students = sort (@students);
# Doreen, Elin, Malin, Oskar, Sangeet

@reverse_sorted_students = reverse (@sorted_students);
# Sangeet, Oskar, Malin, Elin, Doreen

sort @students;
# doesn't do anything to @students
```

scalar and list context

The way perl treats lists and scalars depends on the context

```
42 + something     # something must be scalar
sort something     # something must be a list

@sorted_students = sort (@students);
# Doreen, Elin, Malin, Oskar, Sangeet

42 + @students;
# gives 47,  ( 42 + 5 )
```

`@students` is equal to 5 in scalar context

scalar and list context

```
@numbers = 2 * 6;
```
gives `$numbers[0] = 12`

scalar context:

```
if, while, +, -, $scalar
```

list context:

```
push, foreach, sort, reverse, @array
```

use scalar to force scalar context

```
scalar(@students)
```
gives 5

<STDIN> in list context

```
@lines = <STDIN>;
```
this will read each input line as a new list element until you press control-D

```
chomp(@lines);
```
this will remove \n from all the elements

```
chomp(@lines=<STDIN>);
```
will read from keyboard and remove the \n

summary

`$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

exercises

13.15 in room A6:003 under bikupan

remember UU username and password C