

Data Type Overview

- Scalars – strings and numbers \$.
- Lists (Vectors): data collections, particularly when order is important: @.
- Hashes: mapping between keys and values: %

Different types have separate name spaces.

Scope

- our *global* variable.
- local *static* variable.
- my *automatic* variable.

⇒ Watch out for my \$a, \$b;

Global Examples

```
$a = 10;  
{  
  my $a = 20;  
  print "In the block $a\n";  
  our $a = 30; print "In the block $a\n";  
}  
print "Outside the block $a\n";
```

Prints:

In the block 20

In the block 30

Outside the block 30

Static Examples

```
sub t { print @_, ' ', $a, "\n"; }  
$a = 10; t 'Global';  
{  
  local $a = 20; t 'Local';  
}  
t 'Global again';
```

Prints

Global 10

Local 20

Global again 10

Automatic Examples

```
$a = 10;  
{  
  my $a = 20; print "In the block $a\n";  
}  
print "Outside the block $a\n";
```

Prints:

In the block 20

Outside the block 10

Scalar Literals

- Numbers: 123.4
- Strings: 'aaa'. Special characters: \n, \t. Quoting.

Quote types:

- Apostrophes: 'string' (does not interpolate).
- Quote marks: "string" (interpolates).
- Command execution: `program`
- Plain format ('Mon', 'Tue', 'Wed')
- "Quote words": qw(Mon Tue Wed)

String Functions

- Interpolation replaces variable names in strings with their values – difference between "aaa \$tst" and 'aaa \$tst'.
- Special interpretation of hash keys.

Useful string functions (`perldoc -f funName`):

- Look for a string: `index 'ALALAX', 'LAX' -> 3`
- Extract a substring: `substr('1234', 1, 2) -> '23'`
- Split a string: `split`
- Maybe change a case: `lc 'Axa' -> 'axa',
uc 'Axa' -> 'AXA'`

Arithmetic operators: `+`, `-`, `*`, `/`, `**`. `perldoc perlop`.

Here Documents

Very useful when several lines have to be cited as-is.

```
print << "UNIQUE_STRING"  
BASIS  
$basisSet  
...  
UNIQUE_STRING
```

- Usual interpolation rules apply.

Arrays

```
my @a = ('alpha', 'beta', 'gamma');  
# access to a scalar element.  
my $alpha = $a[0];  
# array slice  
my @alphaBeta = @a[0,1];
```

Array functions:

- `for $_(@a){push @b, $_ if /^[A-Z]/}`
- `while(@a){ print shift @a, " and ", shift @a,"\n"}`
- `@squares = map { $_**2; } @values;`
- `@studentsByAge=map {$age{$a}<=>$age{$b}} @students`
- `@youngPuppies = grep { $dogsAge{$_} < 1 } @dogs`

pop, splice...

List Assignments and Operations

```
# swapping two variables.  
($a, $b) = ($b, $a);
```

- `join ':', (1..4) → '1:2:3:4'`
- `reverse qw(Sun Mon Tue) → ('Tue', 'Mon', 'Tue')`
- `@hoursOfRealWork = map { $_ *0.8 } @reportedHours`
- `sort { $a <=> $b } keys %aHash`
- Quickly list files:
`@outputsToProcess = glob '*.log';`

Hashes

```
my %ages = ('Hanna' => 30, 'John' => 32);  
# access to a scalar element  
my $hannasAge = $ages{'Hanna'};  
# access to hash slice:  
my $sumAge;  
map { $sumAge += $_ } %ages{'Hanna','John'};
```

Exists Function! useful for distinguishing between keys identifying an undefined value, and non-existing keys.

List vs Scalar Contexts

Some operations behave differently depending whether they are executed in a *scalar* or *list* context.

```
# Slurps entire file; lines saved as  
# elements of @input list.  
@input = <>;
```

```
# Reads single line.  
$line = <>;
```

Ending Comments

- Reading after: Chapter 2, up to “Typeglobs and Filehandles” (pp. 47-78).
- Hand in for the next session:
 - Compute the magnitude of the force applied to the atoms.
 - Play with text file processing: separate the file into a list of words and apply a text transformation. You will probably use functions `split`, `lc`, `uc` and maybe `join`.