

A Few More Functions

One more quoting operator

- `qw//`
- Takes a space separated sequence of words, and returns a list of single-quoted words.
 - no interpolation done
- `@animals = qw/cat dog bird mouse/;`
- `@animals` → ('cat', 'dog', 'bird', 'mouse');
- As with `q//`, `qq//`, `qx//`, `m//`, and `s///`, you may choose any non-alphanumeric character for the delimiter.

map (EXPR|BLOCK) LIST

- evaluate EXPR (or BLOCK) for each value of LIST. Set `$_` to each value of LIST, much like a foreach loop
 - Returns a list of all results of the expression
- ```
my @words = map {split(' ', $_)} @lines;
```
- for each element of `@lines` (alias the element to `$_`), run `split(' ', $_)`, push results into `@words`.
- Equivalent:

```
my @words;
foreach (@lines) {
 push @words, split(' ', $_);
}
```

---

---

---

---

---

---

---

---

### More map examples

```
my @times = qw/morning afternoon night/;
my @greetings = map "Good $_", @times;
@greetings → ("Good morning", "Good afternoon",
"Good night")

my @nums = (1..5);
my @doubles = map {$_ * 2} @nums;
@doubles → (2, 4, 6, 8, 10);
```

- If you use a block, there is no comma separating the args. If you use an expression, there is.

---

---

---

---

---

---

---

---

### grep (EXPR|BLOCK) LIST

- Similar concept to map (and same syntax)
- returns a list of all members of the original list for which expression was true.
  - (map returns list of all return values of each evaluation)
- Pick out elements you want to keep
- `@comments = grep {/^s*#/} @all_lines;`
  - picks out all lines beginning with comments
  - Assigns `$_` to each member of `@all_lines`, then evaluates the pattern match. If pattern match is true, `$_` is added to `@comments`
- `@odds = grep { $_ % 2 == 1 } @nums;`
- `@big_words = grep { 9 < length } @words;`

---

---

---

---

---

---

---

---

### grep equivalence

```
• my @cmnts = grep {/^s*#/} @lines;

• my @cmnts;
for (@lines){
 if (/^s*#/){
 push @cmnts, $_;
 }
}
```

---

---

---

---

---

---

---

---

## each

- Pick out keys and values from a hash.
- In scalar context, just gets the next key:
  - `while (my $key = each %hash){...}`
    - `foreach my $key (keys %hash){...}`
- In list context, gets the next key and value of the hash.
  - `while (my ($key,$val)= each %hash) {...}`
    - `foreach my $key (keys %hash) {`
      - `my $val = $hash{$key};`
      - `...`
      - `}`
- If your list contains more than 2 variables, others get assigned to `undef`
- Will return key/val in same "random" order as `keys()` and `values()`

---

---

---

---

---

---

---

---

## glob EXPR

- returns EXPR after it has passed through Unix filename expansion.
  - as evaluated by the csh shell
- In Unix, `~` is a wild card that means "home directory of this user"
  - ie, my home directory is `~lallip/`
- Other wildcards:
  - `*` → 0 or more of any character
  - `?` → exactly one of any character.
- Fails: `opendir my $dh, '~lallip';`
- Works: `opendir my $dh, glob '~lallip';`

---

---

---

---

---

---

---

---

## glob returns

- In list context, returns all files/directories that fit the pattern of the wildcard expansion.
- `my @files = glob (*.pl);`
  - gets a list of all files with a .pl extension in the current directory.
- In scalar context, returns the "next" file/directory that fits the pattern.
- After last entry, will return `undef` before returning first entry again.
- `while (my $file = glob (*.txt)){`
  - `print "file = $file\n";`
  - `}`
- Special `<*>` operator uses `glob` internally
  - `while (my $file = <*.txt>) { ... }`

---

---

---

---

---

---

---

---

## eval EXPR

- evaluate perl code.
- Code is parsed and executed at run-time.
- This is considered highly dangerous by some people, and causes some to warn people away from `eval` altogether.
- ```
my $x;  
my $foo = q{$x = 'hello'};  
eval $foo;  
#$x now set to 'hello'
```

eval BLOCK

- evaluate perl code
- Code is parsed at compile time
- This is perl's method of exception handling.
- Fatal errors are trapped, errors stored in `$@`
- ```
eval { $x = $y / $z; };
warn "Exception caught: $@" if $@;
```
- Both perl errors, and your own `die()` calls are caught.

---

---

---

---

---

---

---

---

## undef & defined

- `undef` undefines its argument
- Also returns undefined value.
  - ```
$foo = undef; #works  
undef $foo; #preferred
```
 - ```
@foo = undef; #doesn't work!!
undef @foo; #works (clears @foo)
@foo = (); #preferred
```
- `undef` can be used to throw away unneeded values:  

```
($foo, undef, $bar) = fctn();
```
- `defined` returns true if argument is not `undef`. Returns false otherwise.
- ```
my ($foo, $bar) = (5);  
print "Foo is undef\n" if !defined $foo;  
print "Bar is undef\n" if !defined $bar;
```

delete & exists

- `delete` removes a key/value pair from a hash
- `my %foo = (one=>1, junk=>-999, two=>2); delete $foo{junk};`
`%foo` → (one=>1, two=>2);
– (contrast with `undef $foo{junk} ...`)
- Can be used with arrays, but doesn't do the same:
- `my @bar = ('a'..'e');`
`delete $bar[2];`
`@bar` → ('a', 'b', undef, 'd', 'e')
– will shrink array if the deleted element is last
- `exists` returns true if the key exists in the hash:
- `print "%foo has a value at 'one'\n"`
`if exists $foo{one};`
– (contrast with `defined $foo{one}`)

status of hash elements

- hash key exists → value may be false, may be undef
- hash value defined → key exists, value may be false.
- hash value true → value is defined, key exists.
- ```
if (exists $hash{$key}){
 if (defined $hash{$key}){
 if ($hash{$key}){
 print "$hash{$key} is true\n";
 } else {
 print "$hash{$key} is false\n";
 }
 } else {
 print "$key exists, value undef\n";
 }
} else {
 print "$key does not exist in hash\n";
}
```

---

---

---

---

---

---

---

---

## our

- 'declares' a global variable to be used within the smallest enclosing block.
- Within the block, the global variable does not need to be fully qualified.
- Obviously, only has any meaning if `strict` is being used.
- `our $foo;`  
– Can now use `$main::foo` just by saying `$foo`.
- Still no reason to do this - file-scoped lexicals are all you need.  
– until Object Oriented Perl

---

---

---

---

---

---

---

---