Hash, Anyone?

- The word "hash" in Perl has a specific meaning: it refers to Perl's implementation of an *associative array*
- In Perl, an associative array is like an array, but instead of indexing into the array with numbers (as in `$nums[2]` for the third element of the `@nums` list) you can index into the array with *any scalar value*
- Think of the data structure as having two columns: a *key* (or name) and a *value*
- In computer networking, we will often refer to *name/value pairs*, and Perl's hash mechanism is a direct implementation of this idea

Identifying a Hash

- Scalars are identified with a `$`
- Lists (arrays) are identified with a `@`
- Hashes are identified with a `%`
- So, if we wanted to create a hash to store information about favourite songs, we could create it like this:

```
my %favs = ();
```

- At this point, `%favs` is local to the script (or subroutine), and is currently empty (it has no name/value pairs)

Some Hashed Data

- Here's a table of favourite artists and songs, as name/value pairs:

Artist	**Song**
The Beatles	St. Pepper's Lonely Hearts Club Band
John Lennon	Cold Turkey
Paul McCartney	Maybe I'm Amazed
Led Zeppelin	Rock 'n' Roll
Crowded House	When You Come
James Taylor	Up On The Roof

- On the next three slides, we present three different ways of initialising the `%favs` hash with this data (after all, this is Perl and *There's More Than One Way To Do It!*)

Hash Method #1

```
$favs{'The Beatles'} = "St. Pepper's Lonely Hearts Club Band";
$favs{'John Lennon'} = "Cold Turkey";
$favs{'Paul McCartney'} = "Maybe I'm Amazed";
$favs{'Led Zeppelin'} = "Rock 'n' Roll";
$favs{'Crowded House'} = "When You Come";
$favs{'James Taylor'} = "Up On The Roof";
```

Hash Method #2

```
%favs = (
      'The Beatles', "St. Pepper's Lonely Hearts Club Band",
      'John Lennon', "Cold Turkey",
      'Paul McCartney', "Maybe I'm Amazed",
      'Led Zeppelin', "Rock 'n' Roll",
      'Crowded House', "When You Come",
      'James Taylor', "Up On The Roof"
    );
```

Hash Method #3

```
%favs = (
   'The Beatles' => "St. Pepper's Lonely Hearts Club Band",
   'John Lennon' => "Cold Turkey",
   'Paul McCartney' => "Maybe I'm Amazed",
   'Led Zeppelin' => "Rock 'n' Roll",
   'Crowded House' => "When You Come",
   'James Taylor' => "Up On The Roof"
);
```

Getting at a Hash Entry

- When you want to refer to a particular hash entry, simply do the following:

```
$curr_song = $favs{'John Lennon'};
```

- which will set the scalar `$curr_song` to the value associated with the `'John Lennon'` entry, which at the moment is the scalar string `'Cold Turkey'`
- Note, *just like when we worked with a list element*, the element of a hash is referred to using the $ scalar notation, not the %

Iterating Over Hashes

- We want to be able to process each of the name/values pairs associated with a particular hash
- We can use a `while` loop, together with the Perl `each` function:

```
while (($artist, $song) = each %favs)
{
    print "$artist has a song called $song\n";
}
```

- Note that the name/value pairs returned by each are *in no particular order*, so don't assume that they are

Hash Functions

- The `keys` function, when applied to a hash, returns a list containing the *name part* of the name/value pairing
- The `values` function, when applied to a hash, returns a list containing the *value part* of the name/value pairing
- Here's another example of iterating over a hash, which uses `keys` to get at the names, and uses `sort` to ensure the output is in alphabetical order:

```
foreach $artist (sort keys %favs)
{
  print "$artist has a song called $favs{$artist}\n";
}
```

Deleting from a Hash

- You can use the `undef` function to blank out an entry (the value part) of a hash entry:

```
$favs{'Paul McCartney'} = undef;
```

- Note that the hash entry for `'Paul McCartney'` is still in the hash, we have simply blanked out the value part of the name/value pairing

- To completely remove an entry from a hash, use the `delete` function:

```
delete $favs{'Paul McCartney'};
```

- Now both parts of the name/value pairing are gone, and the hash is *one entry shorter*

Checking for Existence

- To check to see if a name has an associated value, use the defined function:

```
print "$favs{'Crowded House'}\n" if defined $favs{'Crowded House'};
```

- will print the value associated with `'Crowded House'` if it is anything other than `undef`

- To see if an name/value pair already exists, use the `exists` function:

```
unless (exists $favs{'REM'})
{
  $favs{'REM'} = "It's The End Of The World As We Know It";
}
```

- will only add in the entry for `'REM'` if no entry for `'REM'` previously existed in the hash `%favs`

Limitations of Hashes

- The biggest problem with the use of hashes is that *only one value can be associated with each name*
- If we were to do this in our code:

```
$favs{'John Lennon'} = 'Imagine';
```

- we would replace `'Cold Turkey'` with `'Imagine'` - we do not create a second entry for `'John Lennon'` as hash entries are *unique*
- As the value part of the name/value pairing *has to be a scalar*, we can't assign a list to the value part (as lists are not scalars), so we are out of luck
- (Note: it's not all doom and gloom: the scalar value limitation can be overcome with the use of Perl *references*)

Another Hash Example

- Within every Perl script, the *environment variables* of the enclosing operating system are available to you through an in-built hash called %ENV
- Here's a simple script to print out the current set of environment variables:

```
#!/usr/bin/perl -w

foreach $var (sort keys %ENV)
{
 print "$var = $ENV{$var}\n";
}
```

Using Hashes with Lists

- The Linux (UNIX) `who` command lists information on the users currently logged in - here's an example of the output from my home computer:

```
root     tty1      Nov  9 22:28
barryp   tty2      Nov  9 22:28
barryp   tty3      Nov  9 22:28
root     tty4      Nov  9 23:22
mysql    tty5      Nov  9 22:10
```

- It would be nice to sort the list of user-id's and remove any duplicates (note: in the real world, this list could be very long, and would easily scroll off the screen)

- On the next page we present the Perl script to produce the list the way we want it

pwho.pl

```
#!/usr/bin/perl -w


%unique = ();   # The hash is initially empty.

# On the next line, the use of `who` allows Perl to call an
# operating system command and have the results delivered as
# lined input to the script.

for (`who`)
{
  s/\s.*\n//;        # Remove unwanted space.
  $unique{$_}++;     # Update the hash with $_ (current thing).
}

@users = sort keys %unique;  # Produce a sorted list.


print "The logged in users are: @users\n";
```