

CS306: Introduction to Perl

Section #3: Hashes

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

Section 4: Hashes

Introduction
Hash functions

Slide 2

Introduction to Hashes

- A *hash* is similar to an array, but it is indexed by strings instead of numbers.
- Each element of a hash has a *key* and a *value*
- Each key is a string and is unique
- Each value is a scalar

Slide 3

Creating Hashes

- Perl uses the `%` symbol to indicate a hash
- `%profile`
- `%profile = ("name", "Fran", "gender", "male");`
 - Hash assignments expect a list consisting of name-value pairs (so the list should have an even number of elements)
- `%profile` is not related to `@profile`, which is not related to `$profile`

Slide 4

Using The => Big Arrow Notation

- Perl provides the => notation to make hash creation easier. It's called the "big arrow"
- `%profile = ("name" => "Fran",
 "gender" => "male");`
 - Easier to see the name => value pairs this way

Slide 5

Accessing Hash Elements

- Use the {key} notation
- `$profile{name}` # Is "Fran"
 - `$profile{"name"}` is fine too
- `$hash{$key}` # where \$key holds a key name

Slide 6

Modifying Hashes

- A single element
 - `$profile{name} = "George";`
- The entire hash
 - Copy a hash: `%hash2 = %hash1;`
-

Slide 7

Hash Functions

- `keys()` - returns a list of the keys in the hash
 - `@keys = keys %hash;`
- `values()` - returns a list of the values in the hash
 - `@values = values %hash;`

Slide 8

Hash Functions 2

- *each()* - pop for hashes. Pops off one key => value pair per call.
 - (*\$key*, *\$value*) = each %hash;
 - This makes more sense in loops...

```
while (($key, $value) = each %hash) {  
    # Do something with the key and value  
}
```
 - Think about what's really happening here in the while loop.
Hint: while is expecting a boolean (scalar context) inside the ().

Slide 9

Hash Functions 3

- *keys()*, *values()* and *each()* do not return their results in any well-defined order.
- Use *sort()* for that
 - @keys = sort keys %hash;
 - As always, *sort(keys(%hash))* is equivalent. Good to remind oneself every now and then.

Slide 10

Hash Functions 4

- *exists()* - returns true if particular key exists
 - if (exists \$hash{"somekey"}) ...
- *delete()* - removes a key => value pair
 - delete \$hash{\$otherkey};
 - \$hash{\$otherkey} = undef; # does not work!

Slide 11

Hash Functions 5

- *reverse()* - flips all of the keys with their values
- Why would I EVER want to do that? Weird!
- %iplookup = ('blazer1' => '138.26.65.80',
 'blazer2' => '138.26.65.81',
 'blazer3' => '138.26.65.82');
- "I have this \$ip, but what's the host name?"
 - %hostlookup = reverse %iplookup;
 - print \$hostlookup{\$ip};

Slide 12

Iterating Over Hashes

- `foreach $key (sort keys %hash) {
 print "The value of $key is $hash{$key}.\n";
}`
- `while (($key, $value) = each %hash) {
 # Do something with the key and value
}`

Slide 13

Hash Interpolation

- `print "My name is $profile{name}.";`
 - Works as you'd expect - "My name is Fran."
- `print "My hash contains %profile";`
 - prints "My hash contains %profile"
 - % has no special meaning in a double-quoted string like \$ and @ do.

Slide 14

Uses For Hashes

- Hashes are one of Perl's best features, they are extremely versatile and powerful.
- Typical uses...
 - Lookup tables like the host/ip example
 - Tracking the same data for many people/things
 - `$diskusage{fran} = 200; $diskusage{ying} = 50;`
 - Counters
 - `for $score (@scores) { $results{$score}++; }`
%results holds how many people got each specific score

Slide 15