

Arrays, Hashes & foreach Loops

or,
Making Your Life Easier,
One Scalar at a Time

Lists of Scalars

What if you have an unwieldy list of scalars — like our list of names from *The Simpsons*, for example? Is there a shortcut way to store them rather than the verbose and time-consuming way of creating many, many uniquely named buckets?

```
$first_person = "Lisa Simpson";
```

```
$second_person = "Bart Simpson";
```

```
$third_person = "Homer Simpson";
```

```
$fourth_person = "Marge Simpson";
```

Lists of Scalars

There is. Here's a much better way. We can use a different type of variable called an *array*. An array is, in effect, a way to manage plural scalars. Technically, it's an ordered list of scalar values and it looks like this (note the @ is a stylized "a" for "array"):

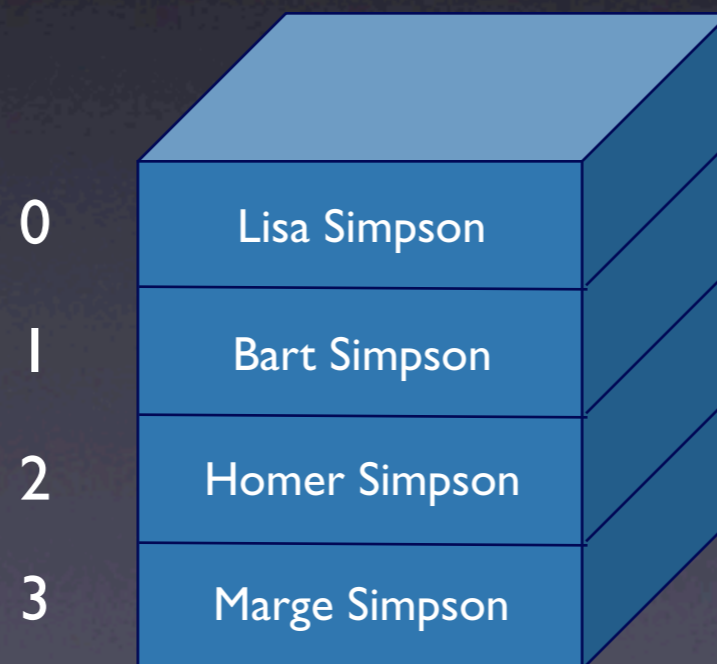
```
@simpsons_names
```

And we can assign the whole list in one fell swoop like this. The scalars (strings in our case) go into quotations and each is separated by a comma. The list itself is bracketed by parentheses.

```
@simpsons_names = ("Lisa Simpson", "Bart Simpson", "Homer Simpson", "Marge Simpson");
```

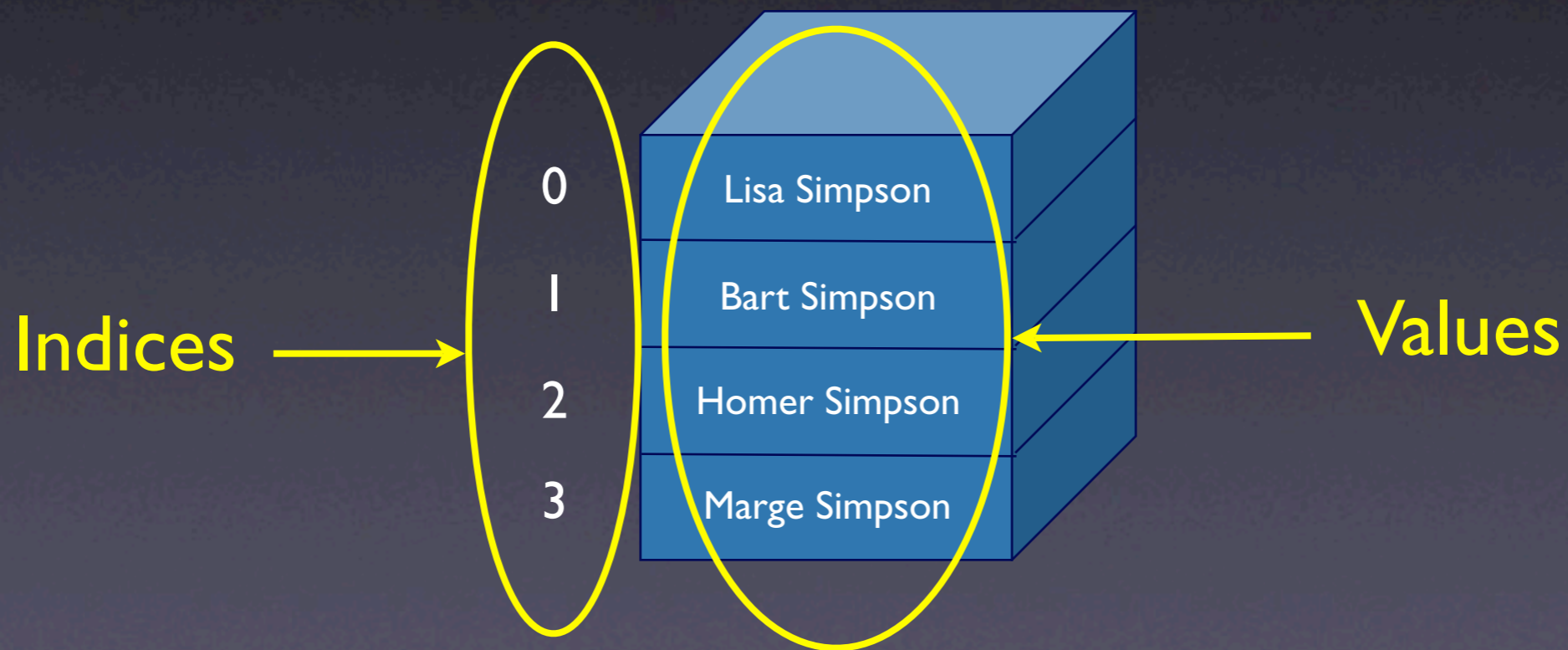
Arrays

An array is like a filing cabinet, each drawer of which can contain a scalar value. Each drawer is labeled but that label is a number called an **index** (plural is **indexes** or **indices**). You don't have a choice about how the drawers are labeled, and those labels always start at zero. You do have a choice, however, about what you put inside.



Arrays

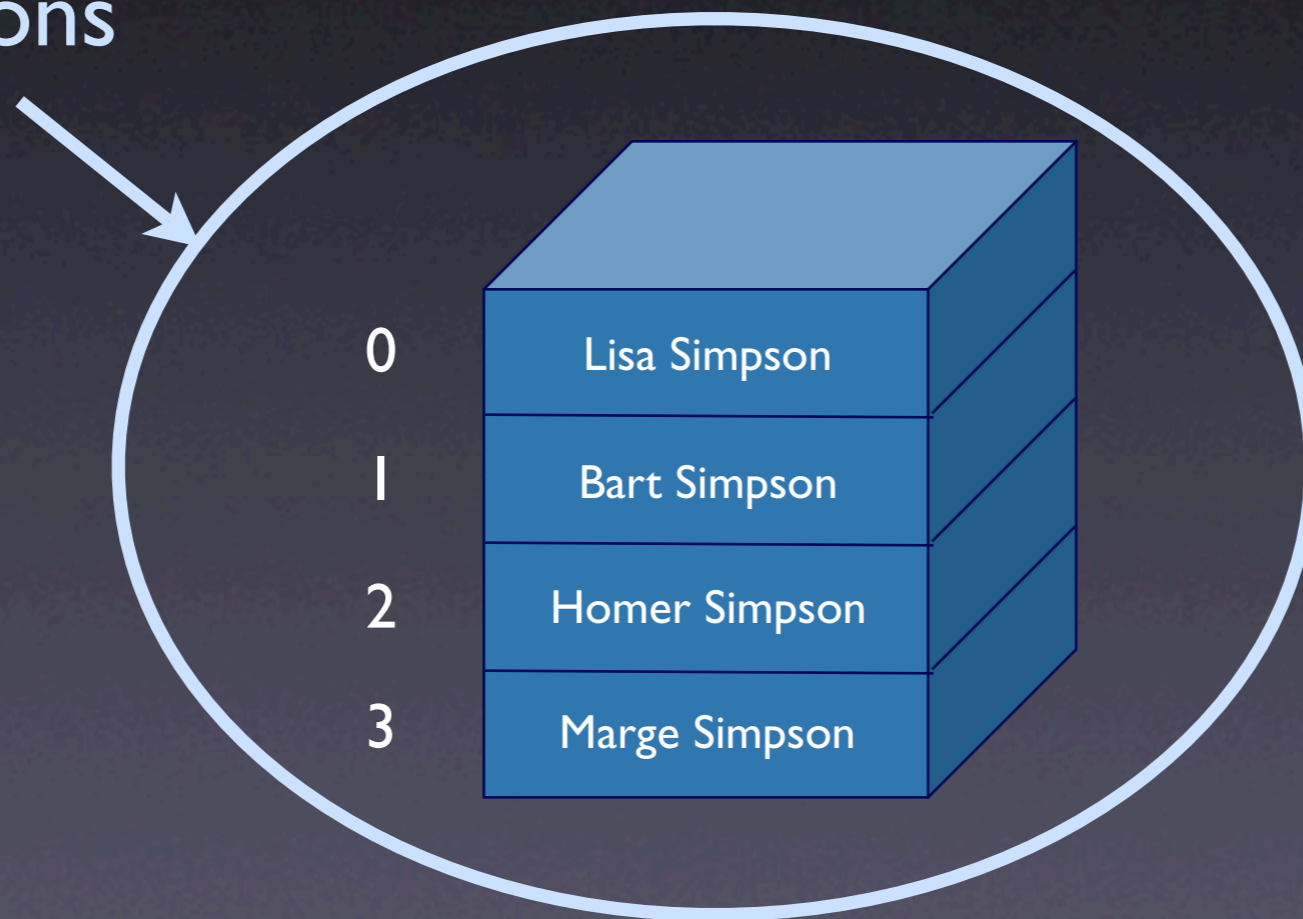
The entire array is one variable called (in this case) **@simpsons**. But the array itself is comprised of parts: **indices** and **values**. So the best way to think of an array is as an ordered list of scalar values.



Arrays

The drawer labels—the indices—are assigned automatically by Perl. Note: they start at **zero**. Not one. **Zero**. But the fact that Perl manages the indices for us automatically makes arrays very convenient and easy to use.

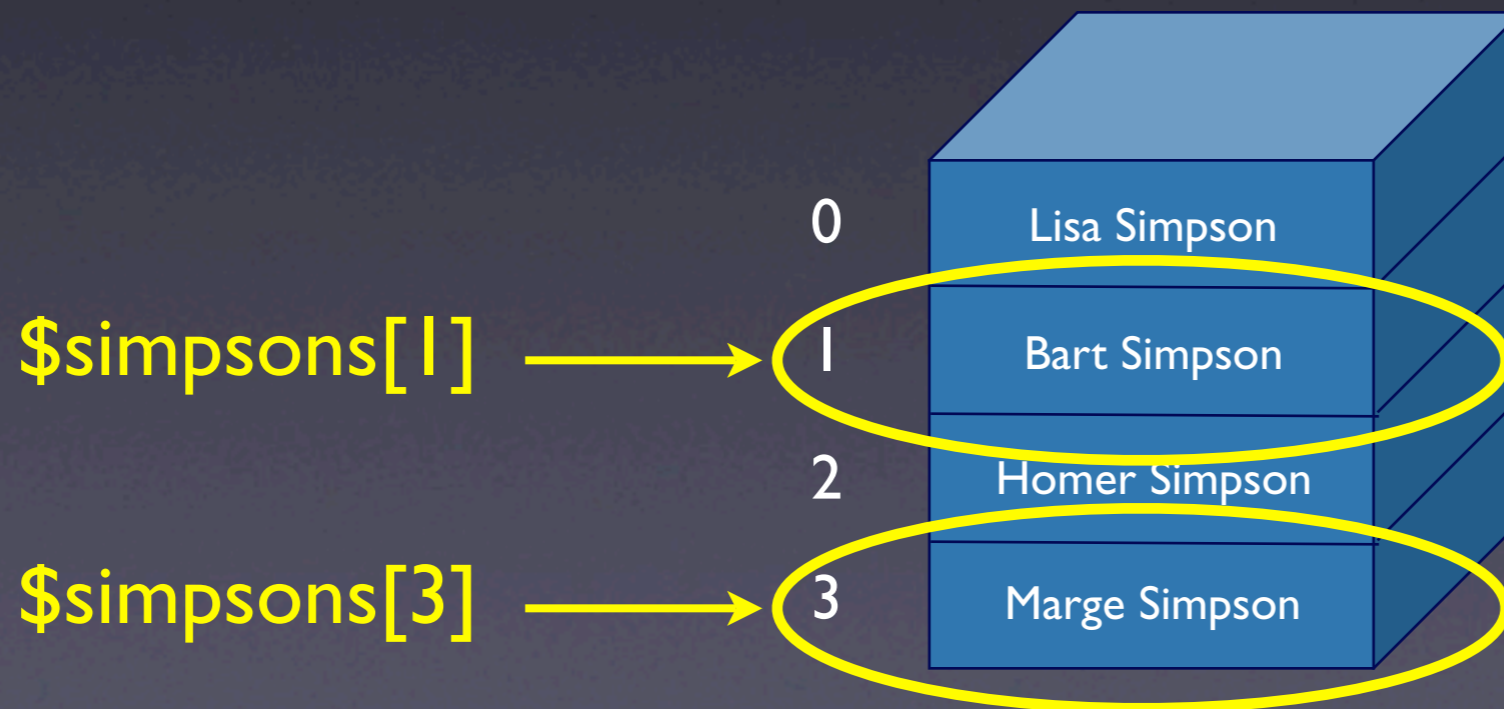
@simpsons



Arrays

Each individual item in our array is accessible by means of its numeric index which is placed inside square brackets like this: **[3]**.

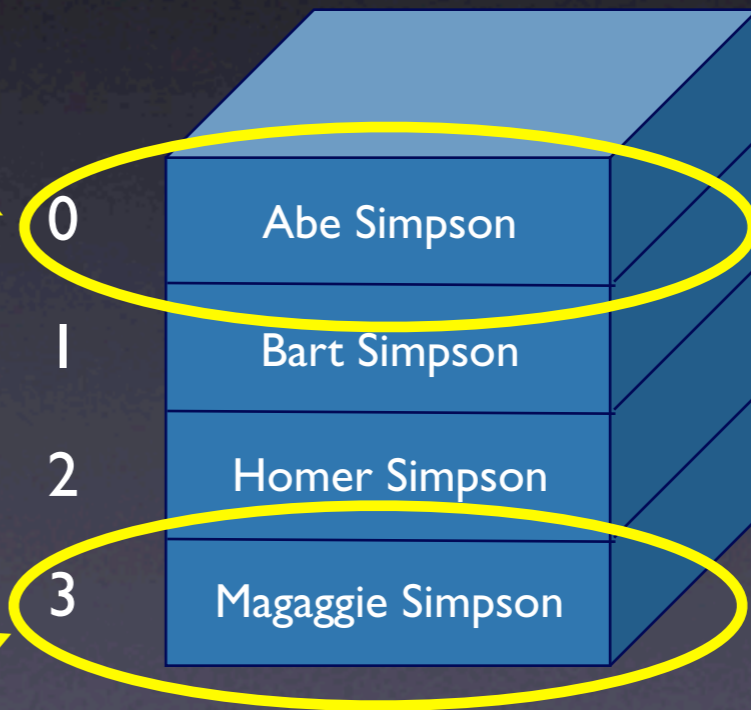
One confusing thing about Perl is that when we refer to the entire array at once, we use the **@** symbol, but when we refer to individual elements of the list, we treat them as scalars (because we're referring to only one of them) and so we go back to using the **\$** sign. The square brackets tell you it's an array reference.



Arrays

You can assign new values to each “drawer” by doing what looks like a regular scalar assignment.

```
$simpsons[0] = "Abe Simpson";
```

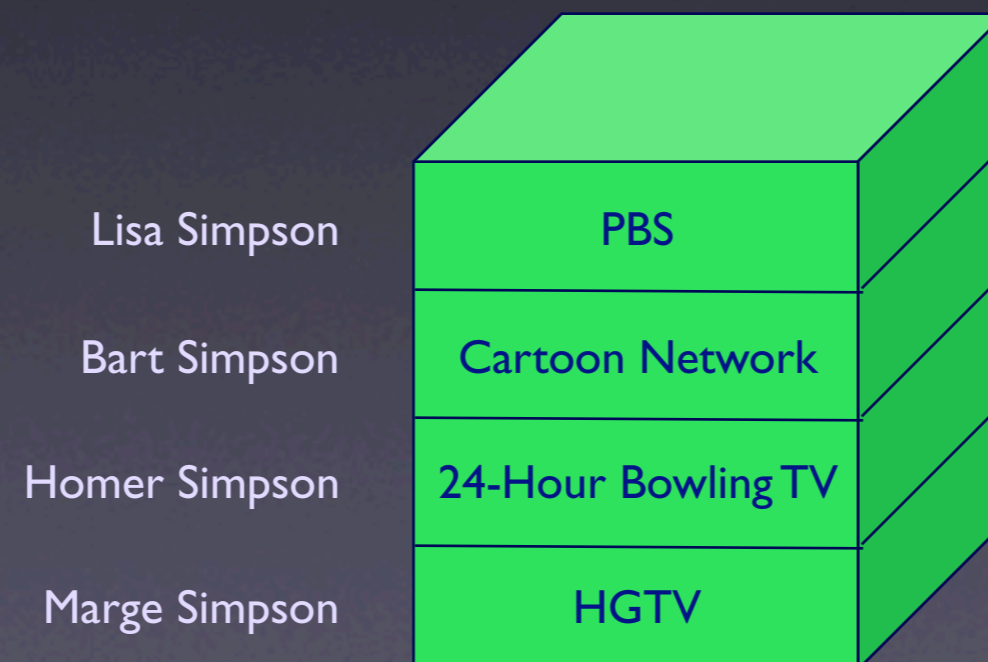


```
$simpsons[3] = "Magaggie Simpson";
```

Hashes

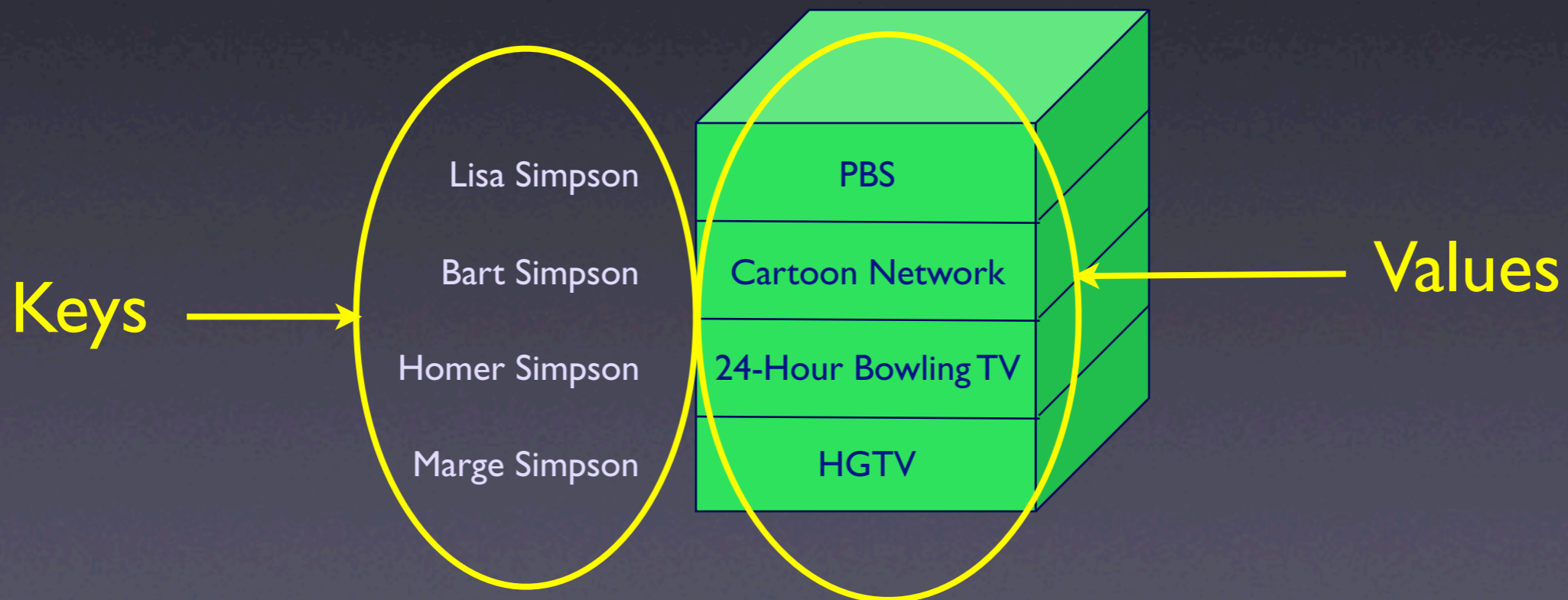
A hash is also like a filing cabinet, each drawer of which can contain a scalar value. However, each drawer is labeled with a string called a **key**. With a hash, you do have a choice about how the drawers are labeled, so you can make the keys work for you.

Let's say that you wanted to store everyone's favourite TV network:



Hashes

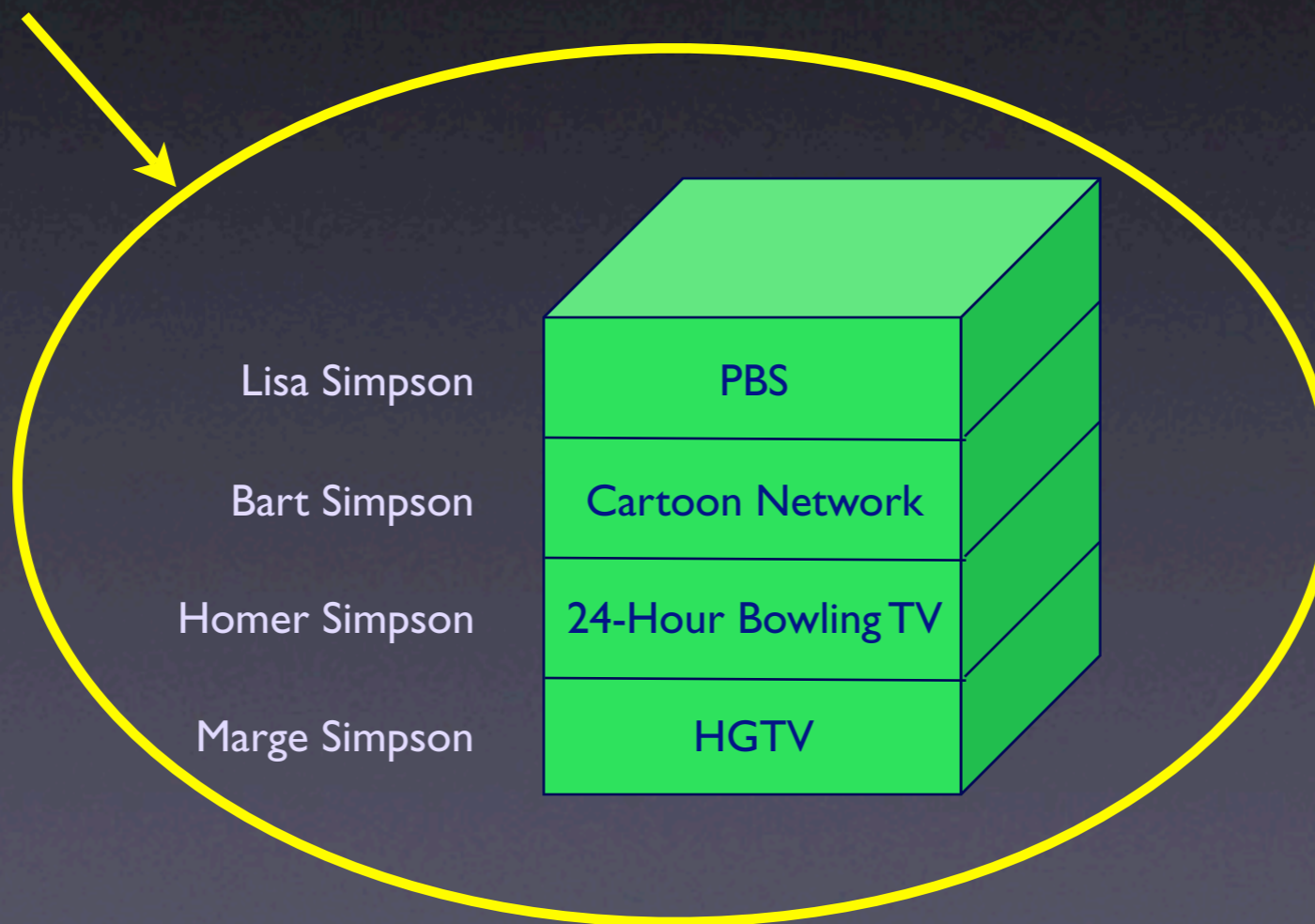
The entire hash might be referred to as `%favourite_tv_stations`, (note the hash prefix: `%`—it looks supposedly like an “h”) but the hash is composed of two parts: **keys** and **values**.



Hashes

Perl treats `%favourite_tv_stations` as one variable, even though it is composed of smaller parts that can be manipulated individually.

`%favourite_tv_stations`



Hashes are really useful for data that occur in pairs. Perhaps surprisingly, that happens a lot.

%latin_to_english_dictionary

arbor	tree
mens	mind
scientia	knowledge
vestigium	footstep

%nfl_standings

Green Bay Packers	10-6
Minnesota Vikings	8-8
Detroit Lions	5-11
Chicago Bears	4-12

%wine_inventory

Chardonnay	8
Merlot	4
Pinot Noir	7
Zinfandel	5

%oscar_winners

Best Actress	Hilary Swank
Best Actor	Jamie Foxx
Best Director	Clint Eastwood
Best Picture	Million Dollar Baby

Hash Syntax

Although a hash is very similar to an array, there are a few important differences. One is the syntax. Hashes use the % sign, which is a stylized “h”:

```
%favourite_tv_shows
```

And we can assign the whole hash using the “big arrow” operator:

```
%favourite_tv_shows =
```

```
  (“Lisa Simpson” => “PBS”,
```

```
  “Bart Simpson” => “Cartoon Network”,
```

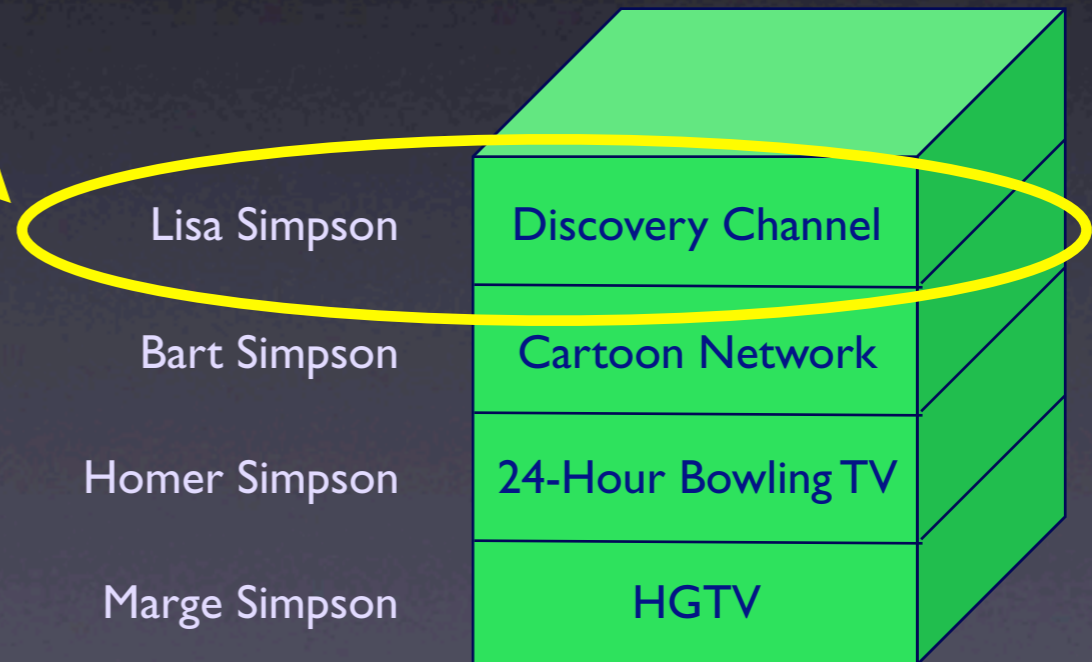
```
  “Homer Simpson” => “24-Hour Bowling TV”,
```

```
  “Marge Simpson” => “HGTV”);
```

Hash Assignment

You can also assign one “drawer” of a hash by treating it as a scalar. As with arrays, the % prefix changes to \$ when you do so. Note that the brackets are curly, not square.

```
$favourite_tv_stations{“Lisa Simpson”} = “Discovery Channel”;
```



Array vs. Hash

	Array	Hash
prefix	@	%
scalar syntax	<code>\$array[3]</code>	<code>\$hash{"Homer"}</code>
access via...	indices (unique numbers starting at 0)	keys (unique strings)
organization	ordered	unordered
good for...	lists, queues	pairs of data (inventory, tests)

Iterating thru an Array

Here's a common problem: you need to examine every element in an array or hash and do some processing on it. How to do that?

The solution is to use a programming structure called a **loop**. Perl's **foreach** loop is just what you want:

```
foreach $person (@simpsons) {  
    print "$person\n"  
}
```

This tells Perl to visit each value in the array **@simpsons**, put the value into the scalar variable **\$person**, and print it to the screen. The closing curly bracket says to loop back and get the next value.

Perl's Magical Variable: \$_

If you don't want to specify a new scalar variable into which the values should go, you can use an automatic Perl variable with the magical name `$_`:

```
foreach (@simpsons) {  
    print "$_\n"  
}
```

`$_` is called a "default variable." Perl will use it automatically if you don't tell Perl specifically where to put those values. As you can tell by its prefix, it's a scalar.

(Perl also has a default `@_` variable for arrays but we won't be learning enough Perl to try it out.)

Iterating thru a Hash

The foreach loop can iterate through a hash too. Note that we have to tell Perl to look specifically at the keys, though. Since Perl gives us more control over a hash, we have more responsibility to tell Perl explicitly what we want:

```
foreach $person (keys %tv_stations) {  
    print "$person likes $tv_stations{$person}\n"  
}
```

This tells Perl to visit each key in the hash `%tv_stations`, to put the key into the scalar variable `$person`, and to print out the key and the matching value. The closing curly tells Perl to loop back and get the next hash key.

Using `$_` with a Hash

You can use the magical default scalar with hashes too:

```
foreach (keys %tv_stations) {  
    print "$_ likes $tv_stations{$_}\n"  
}
```

As in the last loop, this tells Perl to visit each key in the hash `%tv_stations`, to put the key into the magical scalar variable `$_`, and to print out the key and the matching value. The closing curly tells Perl to repeat the process until all the hash's keys have been visited.

Arrays, Hashes & foreach Loops

or,
Making Your Life Easier,
One Scalar at a Time