PERL HASHES

A hash is a set of **key/value** pairs. Hash variables are preceded by a percent sign. To refer to a single element of a hash, you will use the hash variable name preceded by a "$" sign and followed by the "key" associated with the value in curly brackets.

Here is a simple example of using hash variables:

```
#!/usr/bin/perl

%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

This will produce following result:

```
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Creating Hashes

Hashes are created in one of two following ways. In the first method, you assign a value to a named key on a one-by-one basis:

```
$data{'John Paul'} = 45;
$data{'Lisa'} = 30;
$data{'Kumar'} = 40;
```

In the second case, you use a list, which is converted by taking individual pairs from the list: the first element of the pair is used as the key, and the second, as the value. For example:

```
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
```

For clarity, you can use => as an alias for , to indicate the key/value pairs as follows:

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

Here is one more variant of the above form, have a look at it, here all the keys have been preceded by hyphen – and no quotation is required around them:

```
%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);
```

But it is important to note that there is a single word ie without spaces keys have been used in this form of hash formation and if you build-up your hash this way then keys will be accessed using hyphen only as shown below.

```
$val = %data{-JohnPaul}
$val = %data{-Lisa}
```

Accessing Hash Elements

When accessing individual elements from a hash, you must prefix the variable with a dollar sign $ and then append the element key within curly brackets after the name of the variable. For example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

print "$data{'John Paul'}\n";
print "$data{'Lisa'}\n";
print "$data{'Kumar'}\n";
```

This will produce following result:

```
45
30
40
```

Extracting Slices

You can extract slices of a hash just as you can extract slices from an array. You will need to use @ prefix for the variable to store returned value because they will be a list of values:

```
#!/uer/bin/perl


%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);

@array = @data{-JohnPaul, -Lisa};

print "Array : @array\n";
```

This will produce following result:

```
Array : 45 30
```

Extracting Keys and Values

You can get a list of all of the keys from a hash by using **keys** function which has the following syntax:

```
keys %HASH
```

This function returns an array of all the keys of the named hash. Following is the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@names = keys %data;

print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";
```

This will produce following result:

```
Lisa
John Paul
Kumar
```

Similarly you can use **values** function to get a list of all the values. This function has following syntax:

```
values %HASH
```

This function returns a normal array consisting of all the values of the named hash. Following is the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@ages = values %data;

print "$ages[0]\n";
print "$ages[1]\n";
print "$ages[2]\n";
```

This will produce following result:

```
30
45
40
```

Checking for Existence

If you try to access a key/value pair from a hash that doesn't exist, you'll normally get the **undefined** value, and if you have warnings switched on, then you'll get a warning generated at run time. You can get around this by using the **exists** function, which returns true if the named key exists, irrespective of what its value might be:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

if( exists($data{'Lisa'} ) ){
   print "Lisa is $data{'Lisa'} years old\n";
}
else{
   print "I don't know age of Lisa\n";
}
```

Here I introduced IF...ELSE statement which we will study in a separate chapter. For now you just assume that **if***condition* part will be executed only when given condition is true otherwise **else** part will be executed. So when we execute above program, it produces following result because here given condition *exists($data{'Lisa'}* returns true:

```
Lisa is 30 years old
```

Getting Hash Size

You can get the size - that is, the number of elements from a hash by using scalar context on either keys or values. Simply saying first you have to get an array of either the keys or values and then you can get size of array as follows:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

@values = values %data;
$size = @values;
print "2 - Hash size:  is $size\n";
```

This will produce following result:

```
1 - Hash size: is 3
2 - Hash size: is 3
```

Add & Remove Elements in Hashes

Adding a new key/value pair can be done with one line of code using simple assignment operator. But to remove an element from the hash you need to use **delete** function as shown below in the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

# adding an element to the hash;
$data{'Ali'} = 55;
@keys = keys %data;
$size = @keys;
```

```
print "2 - Hash size:  is $size\n";

# delete the same element from the hash;
delete $data{'Ali'};
@keys = keys %data;
$size = @keys;
print "3 - Hash size:  is $size\n";
```

This will produce following result:

```
1 - Hash size: is 3
2 - Hash size: is 4
3 - Hash size: is 3
```