# RUBY HASHES

A Hash is a collection of key-value pairs like this: "employee" => "salary". It is similar to an Array, except that indexing is done via arbitrary keys of any object type, not an integer index.

The order in which you traverse a hash by either key or value may seem arbitrary and will generally not be in the insertion order. If you attempt to access a hash with a key that does not exist, the method will return *nil*.

## Creating Hashes:

As with arrays, there is a variety of ways to create hashes. You can create an empty hash with the *new* class method:

```
months = Hash.new
```

You can also use *new* to create a hash with a default value, which is otherwise just *nil*:

```
months = Hash.new( "month" )

or

months = Hash.new "month"
```

When you access any key in a hash that has a default value, if the key or value doesn't exist, accessing the hash will return the default value:

```
#!/usr/bin/ruby

months = Hash.new( "month" )

puts "#{months[0]}"
puts "#{months[72]}"
```

This will produce the following result:

```
month
month
```

```
#!/usr/bin/ruby

H = Hash["a" => 100, "b" => 200]

puts "#{H['a']}"
puts "#{H['b']}"
```

This will produce the following result:

```
100
200
```

You can use any Ruby object as a key or value, even an array, so following example is a valid one:

```
[1,"jan"] => "January"
```

## Hash Built-in Methods:

We need to have an instance of Hash object to call a Hash method. As we have seen, following is the way to create an instance of Hash object:

```
Hash[[key =>|, value]* ] or

Hash.new [or] Hash.new(obj) [or]

Hash.new { |hash, key| block }
```

This will return a new hash populated with the given objects. Now using created object we can call any available instance methods. For example:

```
#!/usr/bin/ruby

$, = ", "
months = Hash.new( "month" )

months = {"1" => "January", "2" => "February"}

keys = months.keys

puts "#{keys}"
```

This will produce the following result:

```
["1", "2"]
```

Following are the public hash methods (assuming *hash* is an array object ):

| SN | Methods with Description |
|----|--------------------------|
| 1 | **hash == other_hash**<br>Tests whether two hashes are equal, based on whether they have the same number of key-value pairs, and whether the key-value pairs match the corresponding pair in each hash. |
| 2 | **hash.[key]**<br>Using a key, references a value from hash. If the key is not found, returns a default value. |
| 3 | **hash.[key]=value**<br>Associates the value given by *value* with the key given by *key*. |
| 4 | **hash.clear**<br>Removes all key-value pairs from hash. |
| 5 | **hash.default**$key$ = $nil$<br>Returns the default value for *hash*, nil if not set by default=. ([] returns a default value if the key does not exist in *hash*.) |
| 6 | **hash.default = obj**<br>Sets a default value for *hash*. |
| 7 | **hash.default_proc**<br>Returns a block if *hash* was created by a block. |

| 8 | **hash.delete**_key_ **[or]** |
|---|---|
| | **array.delete**_key_ **{ \|key\| block }** |
| | Deletes a key-value pair from _hash_ by _key_. If block is used, returns the result of a block if pair is not found. Compare _delete_if_. |
| 9 | **hash.delete_if { \|key,value\| block }** |
| | Deletes a key-value pair from _hash_ for every pair the block evaluates to _true_. |
| 10 | **hash.each { \|key,value\| block }** |
| | Iterates over _hash_, calling the block once for each key, passing the key-value as a two-element array. |
| 11 | **hash.each_key { \|key\| block }** |
| | Iterates over _hash_, calling the block once for each key, passing _key_ as a parameter. |
| 12 | **hash.each_key { \|key_value_array\| block }** |
| | Iterates over _hash_, calling the block once for each _key_, passing the _key_ and _value_ as parameters. |
| 13 | **hash.each_key { \|value\| block }** |
| | Iterates over _hash_, calling the block once for each _key_, passing _value_ as a parameter. |
| 14 | **hash.empty?** |
| | Tests whether hash is empty $containsno key-value pairs$, returning _true_ or _false_. |
| 15 | **hash.fetch**_key_[, _default_] **[or]** |
| | **hash.fetch**_key_ **{ \| key \| block }** |
| | Returns a value from _hash_ for the given _key_. If the _key_ can't be found, and there are no other arguments, it raises an _IndexError_ exception; if _default_ is given, it is returned; if the optional block is specified, its result is returned. |
| 16 | **hash.has_key?**_key_ **[or] hash.include?**_key_ **[or]** |
| | **hash.key?**_key_ **[or] hash.member?**_key_ |
| | Tests whether a given _key_ is present in hash, returning _true_ or _false_. |
| 17 | **hash.has_value?**_value_ |
| | Tests whether hash contains the given _value_. |
| 18 | **hash.index**_value_ |
| | Returns the _key_ for the given _value_ in hash, _nil_ if no matching value is found. |

| 19 | **hash.indexes***keys* |
|---|---|
| | Returns a new array consisting of values for the given key*s*. Will insert the default value for keys that are not found. This method is deprecated. Use select. |

| 20 | **hash.indices***keys* |
|---|---|
| | Returns a new array consisting of values for the given key*s*. Will insert the default value for keys that are not found. This method is deprecated. Use select. |

| 21 | **hash.inspect** |
|---|---|
| | Returns a pretty print string version of hash. |

| 22 | **hash.invert** |
|---|---|
| | Creates a new *hash*, inverting *keys* and *values* from *hash*; that is, in the new hash, the keys from *hash* become values and values become keys. |

| 23 | **hash.keys** |
|---|---|
| | Creates a new array with keys from *hash*. |

| 24 | **hash.length** |
|---|---|
| | Returns the size or length of *hash* as an integer. |

| 25 | **hash.merge***other_hash* **[or]** |
|---|---|
| | **hash.merge***other_hash* **{ \|key, oldval, newval\| block }** |
| | Returns a new hash containing the contents of *hash* and *other_hash*, overwriting pairs in hash with duplicate keys with those from *other_hash*. |

| 26 | **hash.merge!***other_hash* **[or]** |
|---|---|
| | **hash.merge!***other_hash* **{ \|key, oldval, newval\| block }** |
| | Same as merge, but changes are done in place. |

| 27 | **hash.rehash** |
|---|---|
| | Rebuilds *hash* based on the current values for each *key*. If values have changed since they were inserted, this method reindexes *hash*. |

| 28 | **hash.reject { \|key, value\| block }** |
|---|---|
| | Creates a new *hash* for every pair the *block* evaluates to *true* |

| 29 | **hash.reject! { \|key, value\| block }** |
|---|---|
| | Same as *reject*, but changes are made in place. |

| 30 | **hash.replace***other_hash* |
|---|---|

Replaces the contents of *hash* with the contents of *other_hash*.

| 31 | **hash.select { \|key, value\| block }**<br><br>Returns a new array consisting of key-value pairs from *hash* for which the *block* returns *true*. |
| --- | --- |
| 32 | **hash.shift**<br><br>Removes a key-value pair from *hash*, returning it as a two-element array. |
| 33 | **hash.size**<br><br>Returns the *size* or length of *hash* as an integer. |
| 34 | **hash.sort**<br><br>Converts *hash* to a two-dimensional array containing arrays of key-value pairs, then sorts it as an array. |
| 35 | **hash.store***key, value*<br><br>Stores a key-value pair in *hash*. |
| 36 | **hash.to_a**<br><br>Creates a two-dimensional array from hash. Each key/value pair is converted to an array, and all these arrays are stored in a containing array. |
| 37 | **hash.to_hash**<br><br>Returns *hash self*. |
| 38 | **hash.to_s**<br><br>Converts *hash* to an array, then converts that array to a string. |
| 39 | **hash.update***other_hash* **[or]**<br><br>**hash.update***other_hash* **{\|key, oldval, newval\| block}**<br><br>Returns a new hash containing the contents of *hash* and *other_hash*, overwriting pairs in *hash* with duplicate keys with those from *other_hash*. |
| 40 | **hash.value?***value*<br><br>Tests whether *hash* contains the given *value*. |
| 41 | **hash.values**<br><br>Returns a new array containing all the values of *hash*. |
| 42 | **hash.values_at***obj, . . .*<br><br>Returns a new array containing the values from *hash* that are associated with the given |

key or keys.