RUBY BLOCKS

You have seen how Ruby defines methods where you can put number of statements and then you call that method. Similarly, Ruby has a concept of Block.

- A block consists of chunks of code.
- You assign a name to a block.
- The code in the block is always enclosed within braces .
- A block is always invoked from a function with the same name as that of the block. This means that if you have a block with the name *test*, then you use the function *test* to invoke this block.
- You invoke a block by using the *yield* statement.

Syntax:

```
block_name{
   statement1
   statement2
   ..........
}
```

Here, you will learn to invoke a block by using a simple *yield* statement. You will also learn to use a *yield* statement with parameters for invoking a block. You will check the sample code with both types of *yield* statements.

The *yield* Statement:

Let's look at an example of the yield statement:

```
#!/usr/bin/ruby

def test
   puts "You are in the method"
   yield
   puts "You are again back to the method"
   yield
end
test {puts "You are in the block"}
```

This will produce the following result:

```
You are in the method
You are in the block
You are again back to the method
You are in the block
```

You also can pass parameters with the yield statement. Here is an example:

```
#!/usr/bin/ruby

def test
   yield 5
   puts "You are in the method test"
   yield 100
end
test {|i| puts "You are in the block #{i}"}
```

This will produce the following result:

```
You are in the block 5
You are in the method test
You are in the block 100
```

Here, the *yield* statement is written followed by parameters. You can even pass more than one parameter. In the block, you place a variable between two vertical lines || to accept the parameters. Therefore, in the preceding code, the yield 5 statement passes the value 5 as a parameter to the test block.

Now, look at the following statement:

```
test {|i| puts "You are in the block #{i}"}
```

Here, the value 5 is received in the variable i. Now, observe the following puts statement:

```
puts "You are in the block #{i}"
```

The output of this puts statement is:

```
You are in the block 5
```

If you want to pass more than one parameters, then the *yield* statement becomes:

```
yield a, b
```

and the block is:

```
test {|a, b| statement}
```

The parameters will be separated by commas.

Blocks and Methods:

You have seen how a block and a method can be associated with each other. You normally invoke a block by using the yield statement from a method that has the same name as that of the block. Therefore, you write:

```
#!/usr/bin/ruby

def test
  yield
end
test{ puts "Hello world"}
```

This example is the simplest way to implement a block. You call the test block by using the *yield* statement.

But if the last argument of a method is preceded by &, then you can pass a block to this method and this block will be assigned to the last parameter. In case both * and & are present in the argument list, & should come later.

```
#!/usr/bin/ruby

def test(&block)
   block.call
end
test { puts "Hello World!"}
```

This will produce the following result:

```
Hello World!
```

BEGIN and END Blocks

Every Ruby source file can declare blocks of code to be run as the file is being loaded *theBEGINblocks* and after the program has finished executing *theENDblocks*.

```
#!/usr/bin/ruby

BEGIN {
  # BEGIN block code
  puts "BEGIN code block"
}

END {
  # END block code
  puts "END code block"
}
  # MAIN block code
puts "MAIN code block"
```

A program may include multiple BEGIN and END blocks. BEGIN blocks are executed in the order they are encountered. END blocks are executed in reverse order. When executed, above program produces the following result:

```
BEGIN code block
MAIN code block
END code block
```