RUBY LOOPS - WHILE, FOR, UNTIL, BREAK, REDO AND RETRY

Loops in Ruby are used to execute the same block of code a specified number of times. This chapter details all the loop statements supported by Ruby.

Ruby *while* Statement:

Syntax:

```
while conditional [do]
   code
end
```

Executes *code* while *conditional* is true. A *while* loop's *conditional* is separated from *code* by the reserved word do, a newline, backslash \, or a semicolon ;.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5

while $i < $num  do
   puts("Inside the loop i = #$i" )
   $i +=1
end
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
```

Ruby *while* modifier:

Syntax:

```
code while condition

OR

begin
  code
end while conditional
```

Executes *code* while *conditional* is true.

If a *while* modifier follows a *begin* statement with no *rescue* or ensure clauses, *code* is executed once before conditional is evaluated.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5
begin
   puts("Inside the loop i = #$i" )
```

```
   $i +=1
end while $i < $num
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
```

Ruby *until* Statement:

```
until conditional [do]
   code
end
```

Executes *code* while *conditional* is false. An *until* statement's conditional is separated from *code* by the reserved word *do*, a newline, or a semicolon.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5

until $i > $num  do
   puts("Inside the loop i = #$i" )
   $i +=1;
end
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
Inside the loop i = 5
```

Ruby *until* modifier:

Syntax:

```
code until conditional

OR

begin
   code
end until conditional
```

Executes *code* while *conditional* is false.

If an *until* modifier follows a *begin* statement with no *rescue* or ensure clauses, *code* is executed once before *conditional* is evaluated.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5
```

```
begin
   puts("Inside the loop i = #$i" )
   $i +=1;
end until $i > $num
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
Inside the loop i = 5
```

Ruby *for* Statement:

Syntax:

```
for variable [, variable ...] in expression [do]
   code
end
```

Executes *code* once for each element in *expression*.

Example:

```
#!/usr/bin/ruby

for i in 0..5
   puts "Value of local variable is #{i}"
end
```

Here, we have defined the range 0..5. The statement for i in 0..5 will allow i to take values in the range from 0 to 5 *including*5. This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
Value of local variable is 3
Value of local variable is 4
Value of local variable is 5
```

A *for...in* loop is almost exactly equivalent to:

```
(expression).each do |variable[, variable...]| code end
```

except that a for loop doesn't create a new scope for local variables. A for loop's *expression* is separated from *code* by the reserved word do, a newline, or a semicolon.

Example:

```
#!/usr/bin/ruby

(0..5).each do |i|
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
Value of local variable is 3
```

```
Value of local variable is 4
Value of local variable is 5
```

Ruby *break* Statement:

Syntax:

```
break
```

Terminates the most internal loop. Terminates a method with an associated block if called within the block *withthemethodreturningnil*.

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i > 2 then
      break
   end
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
```

Ruby *next* Statement:

Syntax:

```
next
```

Jumps to next iteration of the most internal loop. Terminates execution of a block if called within a block (with *yield* or call returning nil).

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i < 2 then
      next
   end
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 2
Value of local variable is 3
Value of local variable is 4
Value of local variable is 5
```

Ruby *redo* Statement:

Syntax:

```
redo
```

Restarts this iteration of the most internal loop, without checking loop condition. Restarts *yield* or *call* if called within a block.

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i < 2 then
      puts "Value of local variable is #{i}"
      redo
   end
end
```

This will produce the following result and will go in an infinite loop:

```
Value of local variable is 0
Value of local variable is 0
..........................
```

Ruby *retry* Statement:

Syntax:

```
retry
```

If *retry* appears in rescue clause of begin expression, restart from the beginning of the 1begin body.

```
begin
   do_something # exception raised
rescue
   # handles error
   retry  # restart from beginning
end
```

If retry appears in the iterator, the block, or the body of the for expression, restarts the invocation of the iterator call. Arguments to the iterator is re-evaluated.

```
for i in 1..5
   retry if some_condition # restart from i == 1
end
```

Example:

```
#!/usr/bin/ruby

for i in 1..5
   retry if  i > 2
   puts "Value of local variable is #{i}"
end
```

This will produce the following result and will go in an infinite loop:

```
Value of local variable is 1
Value of local variable is 2
Value of local variable is 1
Value of local variable is 2
Value of local variable is 1
Value of local variable is 2
..........................
```