

Ruby®

Notes for Professionals

Chapter 12: DateTime

Section 12.1: DateTime from string

`DateTime.parse` is a very useful method which constructs a `DateTime` from a string, guessing its format.

```
DateTime.parse("Jan. 6 2015")  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>  
DateTime.parse("2015-01-06T00:00:00")  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>  
# => #<DateTime: 2015-01-06T00:00:00+00:00 ((2457548), 0s, 0s), +0s, 22991617}>
```

Section 12.2: New

```
DateTime.new(2014, 10, 14)  
# => #<DateTime: 2014-10-14T00:00:00+00:00 ((2456965), 0s, 0s), +0s, 22991617}>
```

```
Current time:  
# => #<DateTime: 2016-08-04T00:43:58-03:00 ((2457905), 13488s, 667386290s)>
```

Note that it gives the current time in your timezone

Section 12.3: Add/subtract days to DateTime

```
DateTime + Fixnum (days quantity)  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>  
# => #<DateTime: 2015-12-31T23:59:59+00:00 ((2457268), 82080s, 8s), +0s, 22991617}>
```

Chapter 14: Numbers

Section 14.1: Converting a String to Integer

You can use the `Integer` method to convert a string to an integer:

```
Integer("123") # => 123  
Integer("0b1") # => 1  
Integer("0x10") # => 16  
Integer("1000") # => 1000
```

You can also pass a base parameter to the `Integer` method to convert numbers from a certain base:

```
Integer("10", 5) # => 5  
Integer("74", 8) # => 48  
Integer("NM", 36) # => 38910
```

Note that the method raises an `ArgumentError` if the parameter cannot be converted:

```
Integer("hello")  
# raises ArgumentError: invalid value for Integer(): "hello"  
Integer("23-hello")  
# raises ArgumentError: invalid value for Integer(): "23-hello"
```

You can also use the `String#to_i` method. However, this method is slightly more permissive and has a different behavior than `Integer`:

```
"23".to_i # => 23  
"23-hello".to_i # => 23  
"hello".to_i # => 0
```

`String#to_i` accepts an argument, the base to interpret the number as:

```
"10".to_i(2) # => 2  
"10".to_i(3) # => 3  
"A".to_i(16) # => 10
```

Section 14.2: Creating an Integer

```
0 # creates the Fixnum 0  
123 # creates the Fixnum 123  
1,000 # creates the Fixnum 1000. You can use _ as separator for readability
```

By default the notation is base 10. However, there are some other built-in notations for different bases:

```
0bFF # hexadecimal representation of 255, starts with 0b  
0b1000 # binary representation of 8, starts with 0b  
0x55 # Octal representation of 365, starts with 0x and digits
```

Section 14.3: Rounding Numbers

The `round` method will round a number up if the first digit after its decimal place is 5 or higher and round that digit is 4 or lower. This takes in an optional argument for the precision you're looking for.

GoalKicker.com - Ruby® Notes for Professionals

Chapter 18: Methods

Functions in Ruby provide organized, reusable code to perform a set of actions. Functions simplify the coding process, prevent redundant logic, and make code easier to follow. This topic describes the declaration and utilization of functions, arguments, parameters, yield statements and scope in Ruby.

Section 18.1: Defining a method

Methods are defined with the `def` keyword, followed by the method name and an optional list of parameter names in parentheses. The Ruby code between `def` and `end` represents the body of the method.

```
def hello(name)  
  "hello, #{name}"  
end
```

A method invocation specifies the method name, the object on which it is to be invoked (sometimes called the receiver), and zero or more argument values that are assigned to the named method parameters.

When the receiver is not explicit, it is `self`.

Parameter names can be used as variables within the method body, and the values of these named parameters come from the arguments to a method invocation.

```
hello("world")  
# => "hello, world"  
hello("All")  
# => "hello, All"
```

Section 18.2: Yielding to blocks

You can send a block to your method and it can call that block multiple times. This can be done by sending a proc/lambda or `lambda`, but is easier and faster with `yield`.

```
def simple(arg1, arg2)  
  puts "First we are here: #{arg1}"  
  puts "Finally we are here: #{arg2}"  
  yield  
end  
simple("start", "end") { puts "Now we are inside the yield!" }
```

Note that the `{ puts ... }` is not inside the parentheses, it implicitly comes after. This also means we can only have one `yield` block. We can pass arguments to the `yield`:

```
def simple(arg)  
  puts "before yield"  
  yield(arg)  
end
```

GoalKicker.com - Ruby® Notes for Professionals

200+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Ruby Language	2
Section 1.1: Hello World	2
Section 1.2: Hello World as a Self-Executable File—using Shebang (Unix-like operating systems only)	2
Section 1.3: Hello World from IRB	3
Section 1.4: Hello World without source files	3
Section 1.5: Hello World with tk	3
Section 1.6: My First Method	4
Chapter 2: Casting (type conversion)	6
Section 2.1: Casting to a Float	6
Section 2.2: Casting to a String	6
Section 2.3: Casting to an Integer	6
Section 2.4: Floats and Integers	6
Chapter 3: Operators	8
Section 3.1: Operator Precedence and Methods	8
Section 3.2: Case equality operator (===)	10
Section 3.3: Safe Navigation Operator	11
Section 3.4: Assignment Operators	11
Section 3.5: Comparison Operators	12
Chapter 4: Variable Scope and Visibility	13
Section 4.1: Class Variables	13
Section 4.2: Local Variables	14
Section 4.3: Global Variables	15
Section 4.4: Instance Variables	16
Chapter 5: Environment Variables	18
Section 5.1: Sample to get user profile path	18
Chapter 6: Constants	19
Section 6.1: Define a constant	19
Section 6.2: Modify a Constant	19
Section 6.3: Constants cannot be defined in methods	19
Section 6.4: Define and change constants in a class	19
Chapter 7: Special Constants in Ruby	20
Section 7.1: <code>__FILE__</code>	20
Section 7.2: <code>__dir__</code>	20
Section 7.3: <code>\$PROGRAM_NAME</code> or <code>\$0</code>	20
Section 7.4: <code>\$\$</code>	20
Section 7.5: <code>\$1</code> , <code>\$2</code> , etc	20
Section 7.6: <code>ARGV</code> or <code>\$*</code>	20
Section 7.7: <code>STDIN</code>	20
Section 7.8: <code>STDOUT</code>	20
Section 7.9: <code>STDERR</code>	20
Section 7.10: <code>\$stderr</code>	21
Section 7.11: <code>\$stdout</code>	21
Section 7.12: <code>\$stdin</code>	21
Section 7.13: <code>ENV</code>	21

Chapter 8: Comments	22
Section 8.1: Single & Multiple line comments	22
Chapter 9: Arrays	23
Section 9.1: Create Array of Strings	23
Section 9.2: Create Array with Array::new	23
Section 9.3: Create Array of Symbols	24
Section 9.4: Manipulating Array Elements	24
Section 9.5: Accessing elements	25
Section 9.6: Creating an Array with the literal constructor []	26
Section 9.7: Decomposition	26
Section 9.8: Arrays union, intersection and difference	27
Section 9.9: Remove all nil elements from an array with #compact	28
Section 9.10: Get all combinations / permutations of an array	28
Section 9.11: Inject, reduce	29
Section 9.12: Filtering arrays	30
Section 9.13: #map	30
Section 9.14: Arrays and the splat (*) operator	31
Section 9.15: Two-dimensional array	31
Section 9.16: Turn multi-dimensional array into a one-dimensional (flattened) array	32
Section 9.17: Get unique array elements	32
Section 9.18: Create Array of numbers	32
Section 9.19: Create an Array of consecutive numbers or letters	33
Section 9.20: Cast to Array from any object	33
Chapter 10: Multidimensional Arrays	35
Section 10.1: Initializing a 2D array	35
Section 10.2: Initializing a 3D array	35
Section 10.3: Accessing a nested array	35
Section 10.4: Array flattening	35
Chapter 11: Strings	37
Section 11.1: Difference between single-quoted and double-quoted String literals	37
Section 11.2: Creating a String	37
Section 11.3: Case manipulation	38
Section 11.4: String concatenation	38
Section 11.5: Positioning strings	39
Section 11.6: Splitting a String	40
Section 11.7: String starts with	40
Section 11.8: Joining Strings	40
Section 11.9: String interpolation	41
Section 11.10: String ends with	41
Section 11.11: Formatted strings	41
Section 11.12: String Substitution	41
Section 11.13: Multiline strings	41
Section 11.14: String character replacements	42
Section 11.15: Understanding the data in a string	43
Chapter 12: DateTime	44
Section 12.1: DateTime from string	44
Section 12.2: New	44
Section 12.3: Add/subtract days to DateTime	44
Chapter 13: Time	46
Section 13.1: How to use the strftime method	46

Section 13.2: Creating time objects	46
Chapter 14: Numbers	47
Section 14.1: Converting a String to Integer	47
Section 14.2: Creating an Integer	47
Section 14.3: Rounding Numbers	47
Section 14.4: Even and Odd Numbers	48
Section 14.5: Rational Numbers	48
Section 14.6: Complex Numbers	48
Section 14.7: Converting a number to a string	49
Section 14.8: Dividing two numbers	49
Chapter 15: Symbols	50
Section 15.1: Creating a Symbol	50
Section 15.2: Converting a String to Symbol	50
Section 15.3: Converting a Symbol to String	51
Chapter 16: Comparable	52
Section 16.1: Rectangle comparable by area	52
Chapter 17: Control Flow	53
Section 17.1: if, elsif, else and end	53
Section 17.2: Case statement	53
Section 17.3: Truthy and Falsy values	55
Section 17.4: Inline if/unless	56
Section 17.5: while, until	56
Section 17.6: Flip-Flop operator	57
Section 17.7: Or-Equals/Conditional assignment operator (=)	57
Section 17.8: unless	58
Section 17.9: throw, catch	58
Section 17.10: Ternary operator	58
Section 17.11: Loop control with break, next, and redo	59
Section 17.12: return vs. next: non-local return in a block	61
Section 17.13: begin, end	61
Section 17.14: Control flow with logic statements	62
Chapter 18: Methods	63
Section 18.1: Defining a method	63
Section 18.2: Yielding to blocks	63
Section 18.3: Default parameters	64
Section 18.4: Optional parameter(s) (splat operator)	65
Section 18.5: Required default optional parameter mix	65
Section 18.6: Use a function as a block	66
Section 18.7: Single required parameter	66
Section 18.8: Tuple Arguments	66
Section 18.9: Capturing undeclared keyword arguments (double splat)	67
Section 18.10: Multiple required parameters	67
Section 18.11: Method Definitions are Expressions	67
Chapter 19: Hashes	69
Section 19.1: Creating a hash	69
Section 19.2: Setting Default Values	70
Section 19.3: Accessing Values	71
Section 19.4: Automatically creating a Deep Hash	72
Section 19.5: Iterating Over a Hash	73
Section 19.6: Filtering hashes	74

Section 19.7: Conversion to and from Arrays	74
Section 19.8: Overriding hash function	74
Section 19.9: Getting all keys or values of hash	75
Section 19.10: Modifying keys and values	75
Section 19.11: Set Operations on Hashes	76
Chapter 20: Blocks and Procs and Lambdas	77
Section 20.1: Lambdas	77
Section 20.2: Partial Application and Currying	78
Section 20.3: Objects as block arguments to methods	80
Section 20.4: Converting to Proc	80
Section 20.5: Blocks	81
Chapter 21: Iteration	83
Section 21.1: Each	83
Section 21.2: Implementation in a class	84
Section 21.3: Iterating over complex objects	84
Section 21.4: For iterator	85
Section 21.5: Iteration with index	85
Section 21.6: Map	86
Chapter 22: Exceptions	87
Section 22.1: Creating a custom exception type	87
Section 22.2: Handling multiple exceptions	87
Section 22.3: Handling an exception	88
Section 22.4: Raising an exception	90
Section 22.5: Adding information to (custom) exceptions	90
Chapter 23: Enumerators	91
Section 23.1: Custom enumerators	91
Section 23.2: Existing methods	91
Section 23.3: Rewinding	91
Chapter 24: Enumerable in Ruby	93
Section 24.1: Enumerable module	93
Chapter 25: Classes	96
Section 25.1: Constructor	96
Section 25.2: Creating a class	96
Section 25.3: Access Levels	96
Section 25.4: Class Methods types	98
Section 25.5: Accessing instance variables with getters and setters	100
Section 25.6: New, allocate, and initialize	101
Section 25.7: Dynamic class creation	101
Section 25.8: Class and instance variables	102
Chapter 26: Inheritance	104
Section 26.1: Subclasses	104
Section 26.2: What is inherited?	104
Section 26.3: Multiple Inheritance	106
Section 26.4: Mixins	106
Section 26.5: Refactoring existing classes to use Inheritance	107
Chapter 27: method_missing	109
Section 27.1: Catching calls to an undefined method	109
Section 27.2: Use with block	109
Section 27.3: Use with parameter	109

[Click here to download full PDF material](#)