

Generalities

Everything in Ruby is an object.

A method always return exactly one single thing (an object).

We can use `do` and `end` in place of `{ }`.

Create a comment with `#`.

Printing Data

`print "string"` print data to the screen without adding a new line at the end

`puts "string"` print data to the screen and add an automatic new line

`p "string"` print data to the screen with a new line and give information on type of data

Special Characters

`\n` add a new line

`\t` add a tabulation

Conversions

`string.split(" ")` convert a string into an array

`array.join(" ")` convert an array into a string

`string.to_i` convert a string to an integer

`element.to_s` convert an element to a string

`element.to_a` convert an element to an array

`num.to_f` convert an integer to a float

Ranges

`(start..end).each { |ele| ... }` specify a range of numbers including end

`(start...end).each { |ele| ... }` specify a range of numbers excluding end

Iterators

`number.times { ... }` repeat a block a number of times

`array.each { |ele| ... }` iterate over element of an array

`array.each_with_index { |ele, i| ... }` iterate over elements of an array with index

`string.each_char { |char| ... }` iterate over characters of a string

Iterators (cont)

`string.each_char.with_index { |char, i| ... }` iterate over characters of a string with index

`hash.each { |key, val| ... }` iterate over elements of a hash

`hash.each_key { |key| ... }` iterate over keys of a hash

`hash.each_value { |val| ... }` iterate over values of a hash

`element.inject { |acc, el| acc + el }` return the value for the method where each element of the block is passed in an accumulator value and the current element

Arrays

`array.push` add element(s) to the end of an array

`array.unshift` add element(s) to the front of an array

`array.pop` remove the last element of an array

`array.shift` remove the first element of an array

`array.include?()` check if an element exists in an array

`array.index` find the index of an element in an array

Objects

`element.is_a? (Object)` return true if class is the class of Object, or if class is one of the superclasses of Object or modules included in Object

`element.object_id` return the memory address of some data

`prc = Proc.new { |ele| ele * 2 }` proc, an object that contains a block and allow to save blocks to variables

`p prc.call(5)` call the proc and evaluate to the last line of code executed within the block



By pixelus

cheatography.com/pixelus/

Not published yet.

Last updated 24th August, 2020.

Page 1 of 4.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopad.com>

Objects (cont)

<code>&proc</code>	convert a block into a proc or convert a proc into a block, & in the parameters for a method definition will convert a block to a proc, & in the arguments for a method call will convert a proc to a block
<code>obj.instance_of? Class</code>	return true if obj is an instance of the given class

If we don't intend to mutate a string, we can use a symbol to save some memory because a symbol value will be stored in exactly one memory location. So they are often used to act as unique identifiers.

Enumerables

<code>array.all? { ele ... }</code>	return true when all elements result in true
<code>array.any? { ele ... }</code>	return true when at least one element result in true
<code>array.none? { ele ... }</code>	return true when no element result in true
<code>array.one? { ele ... }</code>	return true when exactly one element result in true
<code>array.map { ele ... }</code>	return a new array containing the values returned by the block
<code>array.sum</code>	return the total sum of all elements
<code>array.min</code>	return the minimum element
<code>array.max</code>	return the maximum element
<code>array.flatten</code>	return the 1 dimensional version of any multidimensional array
<code>element.length</code>	return a number representing the length of the element
<code>element.count</code>	return a number representing the count of elements that result in true
<code>element.last</code>	return the last element of a string or an array

Enumerables (cont)

<code>element.last(num)</code>	return a substring from the end of the string until it reaches the num value (counting backwards), or return a copy if the given limit is greater than or equal to the string length
<code>num.even?</code>	return true if the number is even
<code>num.odd?</code>	return true if the number is odd

Input / Output

<code>require File</code>	import File when gems are involved
<code>require_relative File</code>	import File with a path to another ruby file
<code>element = gets</code>	allow a user to give input and add a newline character at the end

Scope

<code>\$message = "something"</code>	create a global variable, everywhere area in the code can access the global scope
<code>\$PROGRAM_NAME</code>	global variable, string describing the name of the program
<code>\$stdin</code>	global variable that holds a stream for the standard input
<code>\$stdout</code>	global variable which holds the standard output stream
<code>CONSTANT</code>	constant variable, cannot be reassigned and begin the name with a capital letter

Strings

<code>string.downcase</code>	return a copy of str with all downcase letters
<code>string.upcase</code>	return a copy of str with all uppercase letters
<code>string.capitalize</code>	return a copy of str with the first character converted to uppercase
<code>string.reverse</code>	return a new string with the characters in reverse order



Strings (cont)

<code>string.chomp</code>	remove the last character if it's a newline or carriage return
<code>string.index(position)</code>	return the character in the specified position

Hashes

<code>hash = Hash.new</code> OR <code>hash = {}</code>	create a new hash
<code>hash = Hash.new(...)</code>	create a new hash with a default value
<code>hash[key]</code>	return value of the hash key
<code>hash.has_key?()</code>	check if a key exists in a hash
<code>hash.has_value?()</code>	check if a value exists in a hash

Symbols

<code>&:symbol</code>	turn the symbol into a simple proc, equivalent to: <code>element.method { ele ele.symbol_method }</code>
<code>hash = {:k1 => "v1", :k2 => "v2"}</code>	initializing a hash with symbol keys allows to drop the rocket (<code>=></code>) and move the colon (<code>:</code>) to the right of the symbol

Symbols are immutable and can never be changed.

If we don't intend to mutate a string, we can use a symbol to save some memory because a symbol value will be stored in exactly one memory location. So they are often used to act as unique identifiers.

Operators

<code>def method(arg_1, arg_2, *other_args)</code>	accept additional arguments and stock them into an array
<code>method(*array)</code>	pass an array into a function expecting multiple arguments
<code>method(**hash)</code>	pass a hash into a function expecting multiple arguments

Operators (cont)

<code>array = [*arr_1, element, *arr_2]</code>	decompose an array into individual items where each individual element becomes an argument
<code>hash = [*some_hash, symbol: value]</code>	decompose a hash into individual items where each individual element becomes an argument, only work with hashes where the keys are symbols
<code>element_1 <=> element_2</code>	compare two values and return -1, 0, or 1
<code>a = b</code>	assign b to a if a is nil or false
<code>a &&= b</code>	assign b to a if a is true or not nil

Class

<code>initialize</code>	put define default argument
<code>@variable</code>	inside <code>#initialize</code> instance variable or attribute of class, typically assigned inside <code>#initialize</code> , changing the variable will only effect that one instance
<code>@@variable</code>	class variable, typically assigned inside of the class, but not inside of <code>#initialize</code> , changing the variable will effect all instances because all instances of the class
<code>CLASS_CONSTANT</code>	class constant, will be shared among all instances of a class, but cannot be changed
<code>attr_reader</code>	instance variable getter
<code>attr_writer</code>	instance variable setter
<code>attr_accessor</code>	instance variable getter and setter
<code>Class.new</code>	create a new anonymous (unnamed) class

Class (cont)

def method	instance method we can only call it on a Class instance we initialized using Class.new, instance method depends on the attributes or instance variables of an instance
def self.method	class method called directly on the class, self refers to the Class itself and cannot refer to any instance attributes like @variable (Class::method)
Queue.new	create a queue, process work in FIFO (first-in-first-out) order
Class::CONSTANT	access to the constant inside the class

To create a class we use the class keyword.

The name of a class must begin with a capital letter.

We can define methods within a class.

Syntactic Sugar

el_1==(el_2)	equivalent to: el_1 == el_2
element.[](num)	equivalent to: element[num]
el.[]=(num, string)	equivalent to: el[num] = string

Debugging with Byebug

require "byebug"	add to the top of your file to gain access to the gem
debugger	place this line at a point in your file where you want to begin debugger mode
l <start line>-<end line>	list the line numbers in the specified range
step or s	step into the method call on the current line, once execution is paused on a line containing a method call
next or n	move to the next line of executed code
break <line num> or b <line num>	place a breakpoint at the specified line number, this will pause execution
continue or c	resume normal execution of the code until a breakpoint
display <variable>	automatically show the current value of a variable

Testing with RSpec

describe	name the method being tested
it	expresse the expected behavior of the method being tested
expect	show how that behavior is tested
context	additional blocks to outline different scenarios that code is expected to satisfy
Class#method	refers to the instance method in the class
Class.method or Class::method	refers to the class method in the class

To use RSpec, we need to separate our implementation code files from the testing files using a /lib and /spec folder respectively.

```
/example_project
├── lib
│   ├── add.rb
│   └── prime.rb
└── spec
    ├── add_spec.rb
    └── prime_spec.rb
```

Exceptions

begin...rescue...end	react to an exception, the code in the begin block will execute until an exception is reached, once an exception is reached, the execution will immediately jump to rescue
raise	bring up an exception, flag an exceptional scenario that should be handled in a specific way