Cheatography

# Ruby Programming Cheat Sheet

by Frank Perez (frankperez) via cheatography.com/15494/cs/1602/

## Getting Help

ruby-lang.org

> Primary resource for Ruby "Ruby Website"

ruby-doc.org

> Official documentation for Ruby.

## Running Single Commands

```
ruby -e 'puts 123'
ruby -e 'print 123'
ruby -e 'puts "Hello".reverse'
```

## Creating a Comment

```
# Single line comment
```

## Accessing IRB

```
irb
irb --simple-prompt
```

## Comparison/Logic Operators

| | |
|---|---|
| Equals | == |
| Less than | < |
| Greater than | > |
| Less than, equal to | <= |
| Greater than, equal to | >= |
| Not | ! |
| Not Equal | != |
| And | && |
| Or | \|\| |

Returns true/false

## Conditionals: if, else, elsif

```
x = 1
if x == 1
    puts "Text here"
end
if x == 2
    puts "Text here"
else
    puts "Text here"
```

## Conditionals: if, else, elsif (cont)

```
end
if x == 2
    puts "Text here"
elsif x == 1
    puts "Text here"
else
    puts "Text here"
end
```

## Conditionals: unless, case

```
x = 1
unless x == 2
    puts "This runs if the above boolean is
false."
end
case
    when boolean
        puts "Text here"
    when boolean
        puts "Text here"
    else
        puts "Text here"
    end
case test_value
    when value
        puts "Text here"
    when value
        puts "Text here"
    else
        puts "Text here"
    end
```

## Inline Conditional

```
puts "test" if name == "Frank"
```

## Ternary Operator

```
x = boolean ? "test 1" : "test 2"
```

> This will assign one of the 2 values based on the boolean result.

## OR/OR-EQUALS Operator

```
x = y || z
```

> If y has a value set x equal to y else set it equal to z.

```
x ||= y
```

> If x has a value, nothing happens. If it does not then set x to the value of y.

## Loops

```
x = 0
loop do
    x+=2
    break if x >= 20
    puts x
end
```

You can use the following within a loop

| | |
|---|---|
| break | Terminate the whole loop |
| next | Jump to the next loop |
| redo | Redo this loop |
| retry | Start the whole loop over again |

## Loops: while

```
x = 0
while x < 20
    x += 2
    puts x
end
# You can also use the inline version
x = 0
puts x += 2 while x < 100
```

## Loops: until

```
y = 23245
until y <= 1
    puts y /=2
end
# You can also use the inline version
y = 23245
puts y /= 2 until y <= 1
```

## Loops: for

```
fruits = ['banana', 'apple', 'pear']
for fruits in fruits
    puts fruit.captialize
end
```

## Iterators

```
5.times { puts "Hello" }
1.upto(5) { puts "Hello" }
5.downto(1) { puts "Hello" }
(1..5).each { puts "Hello" }
array.each { |num| puts num * 20 }
```

You can use do and end inplace of { }

## Variable Scopes

| | |
|---|---|
| Global Variable | $variable = "Test" |
| Class Variable | @@variable = "Test" |
| Instance Variable | @variable = "Test" |
| Local Variable | variable = "Test" |
| Block Variable | variable = "Test" |

## Integers

| | |
|---|---|
| 1234.class | This will tell you what class the Integer object belongs to. |
| 10.2.to_i | Will convert number to integer. |

Stored in 2 ways: Fixnum and Bignum

## Floats

| | |
|---|---|
| 12345.10.round | Rounds the float to integer. |
| 12345.to_f | Converts a integer to a float. |
| 12345.10.floor | Rounds down to whole number. |
| 12345.10.ceil | Rounds up to whole number. |

## String Methods

```
"Hello".reverse
"Hello".capitalize
"Hello".downcase
"Hello".upcase
"Hello".length
"Hello".upcase.reverse
```

Strings can be in single or double quotes. Ruby will always return them in double quotes.

## Constants

Similar to variables, not true objects.
A constant should not change unlike a variable.
Define constants in all CAPS
TEST = 2
Anything that begins with a capital letter at the beginning is considered a constant.
If you try to change the value of a constant, it will display a warning, but will still change the value.

## Boolean Methods

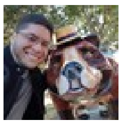
| | |
|---|---|
| z.nil? | Will check if the variable z is == to nil |
| 2.between?(1,4) | Will check if the number 2 is between 1 and 4 |
| [1,2,3].empty? | Will return true/false if its empty. |
| [1,2,3].include?(2) | Returns true/false if the number 2 exists in array. |
| {‘a’ => 1, ‘b’ => 2}.has_key?(‘a’) | Returns true/false if the key exits. |
| {‘a’ => 1, ‘b’ => 2}.has_value?(2) | Will return true/false if the value exits. |

## Arrays

| | |
|---|---|
| data_set = [] | Sets an empty array, and also clears out existing array |
| data_set = [“a”, “b”, “c”] | Sets an array with data |
| data_set[1] | Returns data from the defined positioned. |
| data_set[0] = “d” | Sets the value of the element with key 0 to d |
| data_set << “e” | Appends the data to the array |
| data_set[1] = nil | Removes data from an array |
| data_set.clear | Clears out an array |

## Array Methods

| | |
|---|---|
| array.inspect | Will return a string representation of the array. |



# Ruby Programming Cheat Sheet

by Frank Perez (frankperez) via cheatography.com/15494/cs/1602/

## Array Methods (cont)

| | |
|---|---|
| array.to_s | Joins array elements together. |
| array.join(",") | Will implode the array by comma. |
| x = “1,2,3,4,5”; y = x.split(‘,’) | This will return an array, separating each value by comma. |
| array.sort | Will sort your array asc order |
| array.uniq | Will return an array with no duplicates |
| array.uniq! | Will update the array with the new version in place. |
| array.delete_at(2) | Will delete the element based on key and return the value that it it deleted. |
| array.delete(4) | Will delete the element based on value |
| array.push(4) | Will add to an array - last position |
| array.pop | Will remove the last element from the array |
| array.shift | Will remove the first element from the array |
| array.unshift(1) | Will put the value to the front of the array |
| array + [9,10,11,12] | Will take first array and add these other values from the second array to it. |
| array - [9,10] | Willl search and remove the values 9,10 |

## Hashes

| | |
|---|---|
| mixed = {1 => [‘a’, ‘b’, ‘c’], ‘hello’ => ‘world’, [10,20] => ‘top’} | You can have mixed values in hashes. |
| mixed.keys | Returns all of the keys |
| mixed.values | Returns all of the values |
| mixed.length | Returns the length of the hash |
| mixed.size | Returns the length of the hash |
| mixed.to_a | Converts the hash to an array |
| mixed.clear | Will return an empty hash |
| mixed = {} | Will return an empty hash |
| mixed.key('world') | Will return the key of the hash value. |
| mixed['test'] = 'value' | Will add/set value to hash. |
| mixed[[10,20]] | Returns the value for the hash key which is [10,20] |

## Symbols

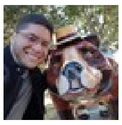
| | |
|---|---|
| :test | Prefixed with a colon and stored in memory once where as a string is stored in memory each time. |
| hash = {:first_name => “Frank”, :last_name => “Perez”} | Works well with hashes. |
| hash[:first_name] | You will need to reference the symbol from the hash like such. |

A label used to identify a piece of data.

## Ranges

| | |
|---|---|
| 1..10 | Inclusive Range |
| 1...10 | Exclusive Range |
| (1..10).to_a | Converts the range to an array. |
| (1..10).class | Will let you know that its a range. |
| x = 1..10 | Sets a range to the variable x. |
| x.begin | Returns the first number. |
| x.first | Returns the first number. |
| x.end | Returns the last number. |
| x.last | Returns the last number. |
| z = [*x] | Using the splat operator *, you can assign the range as an array. |
| y = 1...10; y.include?(10) | Returns false, as it is not included in the range. |
| alpha = ‘a’..’m’ | Creates an inclusive range of letters. |
| alpha.include?(‘g’) | Will return true, as it exists in the range. |
| [*alpha] | Will return all the letters in the range, array format, shorthand. |

Inclusive ranges include all numbers in a range where exclusive ranges excludes the last number.



By **Frank Perez** (frankperez)
cheatography.com/frankperez/
www.frankperez.net

Not published yet.
Last updated 13th May, 2016.