Ruby Objects & Common Methods

Ruby comes out of the box with tons of methods ready to use. So what can all these methods do??

OMG so many awesome things!

These methods belong to specific objects, also known as classes, in Ruby. Below you will find a list of common Ruby methods organized by what object they belong to. Some methods belong to many objects.

Array

clear
Clears the contents of an array.

```
my_array = [1, 2, 3]
my_array.clear # my_array = []
```

concat
Appends one array to another.

```
my_array_1 = ['d', 'e', 'f']
my_array_2 = ['g', 'h']
my_array_1.concat(my_array_2) # my_array_1 = ['d', 'e', 'f', 'g', 'h']
```

count
Counts the number of elements in an array.

```
my_array = ['d', 'e', 'f']
my_array.count # 3
```

Ruby Objects & Common Methods Cheatsheet

---

delete

Deletes the specified value from an array.

```
my_array = ['a', 'b', 'c']
my_array.delete('a') # returns 'a'; my_array = ['b', 'c']
```

---

drop

Drops the specified number of elements from an array and returns a new array with the elements left over.

```
my_array = ['a', 'b', 'c']
my_new_array = my_array.drop(2) # my_new_array = ['c']
```

---

empty?

Returns true or false depending on whether the array has any elements.

```
my_array = ['a', 'b', 'c']
my_array.empty? # false
my_other_array = []
my_other_array.empty? # true
```

---

eql?

Returns true or false depending on whether two arrays have the same content.

```
my_array_1 = ['d', 'e', 'f']
my_array_2 = ['g', 'h']
my_array_1.eql?(my_array_2) # false
my_array_3 = ['x', 'y']
my_array_4 = ['x', 'y']
my_array_3.eql?(my_array_4) # true
```

You can also compare parts of arrays and strings:

```
my_array_1[0].eql?('d') # true
```

Ruby Objects & Common Methods Cheatsheet

fetch

Attempts to return the value of the specified position (index). Ruby is zero-indexed!

```
my_array = ['a', 'b', 'c']
my_array.fetch(0) # returns 'a'
my_array.fetch(5) # throws an error because the 5th position does not exist in my_array
```

first

Returns the first element, or the number of elements specified starting at the beginning of the array.

```
my_array = ['a', 'b', 'c']
my_array.first # returns 'a'
my_array.first(2) # returns ['a', 'b']
```

index

Returns the index of the specified element.

```
my_array = ['a', 'b', 'c']
my_array.index('b') # returns 1
```

join

Returns a string containing each array element separated by the specified symbol.

```
my_array = ['a', 'b', 'c']
my_array.join('-') # returns 'a-b-c'
```

Ruby Objects & Common Methods Cheatsheet

---

last
Returns the last element, or the number of elements specified starting at the end of the array and moving backward.

```
my_array = ['a', 'b', 'c']

my_array.last # returns 'c'

my_array.last(2) # returns ['b', 'c']
```

---

length
Returns the number of elements in an array.

```
my_array = ['a', 'b', 'c']

my_array.length # 3

[].length # 0
```

---

new
Returns a new empty array.

```
my_array = Array.new # my_array = []
```

---

push
Pushes the specified element(s) to an array.

```
my_array = ['a', 'b', 'c']

my_array.push('d', 'e') # my_array = ['a', 'b', 'c', 'd', 'e']
```

---

replace
Replaces the contents of an array with a new array.

```
my_array = ['a', 'b', 'c']

my_array.replace([1, 2, 3]) # my_array = [1, 2, 3]
```

Ruby Objects & Common Methods Cheatsheet

sort
Sorts an array and returns the sorted array.

```
my_array = [2, 3, 1, 5, 12, 6]
my_array.sort # returns [1, 2, 3, 5, 6, 12]
```

uniq
Removes duplicate values from an array and returns a new array with the elements left over.

```
my_array = ['a', 'b', 'a', 'c', 'b', 'd', 'a']
my_array.uniq # returns['a', 'b', 'c', 'd']
```

Hash

assoc
Searches a hash for the specified key. Returns an array including the specified key and the found value.

```
my_hash = {'colors' => ['red', 'green'], 'size' => 'large'}
my_hash.assoc('size') # returns ['size', 'large']
my_hash.assoc('colors') # returns ['colors', ['red', 'green']]
```

clear
Clears the contents of a hash.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.clear # my_hash = {}
```

delete
Deletes the specified key from the hash and returns the value being deleted.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.delete('color') # returns 'blue'; my_hash = {'size' => 'small'}
```

Ruby Objects & Common Methods Cheatsheet

empty?
Returns true or false depending on whether the hash has any key / value pairs.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.empty? # false
my_other_hash = Hash.new
my_other_hash.empty? # true
```

fetch
Returns the value of the specified key.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.fetch('size') # returns 'small'
```

has_key?
Returns true or false depending on whether the hash contains the specified key.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.has_key?('size') # returns true
my_hash.has_key?('location') # returns false
```

keys
Returns a new array containing the keys of the hash.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.keys # returns ['color', 'size']
```

length
Returns the number of key / value pairs in the hash.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.length # returns 2
```

Ruby Objects & Common Methods Cheatsheet

new
Returns a new empty hash.

```
my_hash = Hash.new # my_hash = {}
```

replace
Replaces the key / value pairs of a hash with a new hash.

```
my_hash = {'fruit' => 'apple', 'vegetable' => 'spinach'}
my_hash.replace({'starch' => 'potato', 'drink' => 'water'}) # my_hash =
{'starch' => 'potato', 'drink' => 'water'}
```

values
Returns a new array containing the values of the hash.

```
my_hash = {'color' => 'blue', 'size' => 'small'}
my_hash.values # returns ['blue', 'small']
```

Integer

even?
Returns true or false depending on whether the number is even.

```
2.even? # true
7.even? # false
```

next
Returns one number greater.

```
10.next # returns 11
```

odd?
Returns true or false depending on whether the number is odd.

```
3.odd? # true
10.odd? # false
```

Ruby Objects & Common Methods Cheatsheet

pred
Returns one number lesser.

```
10.pred # returns 9
```

round
Rounds a number to the specified number of decimal points. Defaults to 0. Will not add extra zeros if number is whole.

```
100.round # 100
100.round(3) # 100.0
100.92857985703458034.round(5) # 100.92858
```

String

capitalize
Replaces the first letter of a string with uppercase, and the rest of the letters with lowercase.

```
'BrOoKlYn'.capitalize # 'Brooklyn'
'NEW yoRK'.capitalize # 'New york'
```

Ruby Objects & Common Methods Cheatsheet

chomp

Removes the new line from the end of a string, or a specified portion of a string, and returns a new string.

"\n" adds a new line to the end of a string and must be inside double quotes:

```
"hello there\n"
```

chomp removes the new line:

```
"hello there\n".chomp # returns "hello there"
```

chomp can also remove portions of a string and return what's left:

```
'Hello World!'.chomp(' World!') # returns "Hello"
```

chop

Removes the last character from a string and returns a new string. Also removes the new line from the end of a string.

"\n" adds a new line to the end of a string and must be inside double quotes:

```
"hello there\n"
```

chop removes the new line:

```
"hello there\n".chop # returns "hello there"
```

chop also removes the last character from a string:

```
'hello there'.chop # returns 'hello ther'
```

clear

Makes an empty string.

```
'Skillcrush'.clear # ''
```

Ruby Objects & Common Methods Cheatsheet

downcase

Replaces all uppercase letters in a string with lowercase.

```
'BrOoKlYn'.downcase # returns 'brooklyn'
```

---

empty?

Returns true or false depending on whether a string is empty or has contents.

```
'Skillcrush'.empty? # returns false
"".empty? # returns true
```

---

eql?

Returns true or false depending on how two strings compare. The strings are considered equal if their content and length are the same.

```
'apple'.eql?('orange') # returns false
'banana'.eql?('banana') # returns true
```

---

insert

Inserts the given character(s) into a string at the given index. Ruby is zero-indexed!

```
'Brooklyn'.insert(0, 'X') # 'XBrooklyn' - X is inserted at the 0 position of the string 'Brooklyn', which is the beginning
'New York'.insert(8, 'XYZ') # 'New YorkXYZ' - X is inserted at the 8th position of the string 'New York', which is the end
```

---

length

Returns the length of the string.

```
'hello'.length # returns 5
```

Ruby Objects & Common Methods Cheatsheet

lstrip

Removes leading whitespace from a string.

```
'     hello'.lstrip # 'hello'
```

---

replace

Replace the existing string with a new string.

```
'hello'.replace('world') # 'world'
```

---

rstrip

Removes trailing whitespace from a string.

```
'hello     '.rstrip # 'hello'
```

---

strip

Removes leading and trailing whitespace from a string.

```
'     hello     '.strip # 'hello'
```

---

to_i

Attempts to convert a string to a number, starting at the beginning of the string. Any characters including and after the first letter are ignored. 0 is returned if the string cannot be converted to a number.

```
'123abc'.to_i # 123
'Skillcrush'.to_i # 0
```

---

upcase

Replaces all lowercase letters in a string with uppercase.

```
city = 'BrOoKlYn'
city.upcase # returns 'BROOKLYN'
```

Ruby Objects & Common Methods Cheatsheet

Time

Time is a little special because its methods must be run against an instance of the Time class, meaning you have to first create a variable by calling the Time class, like so:

```
time = Time
```

now

Returns the current date and time.

```
time = Time
time.now # 2014-05-15 17:45:24 -0400
```

---

year

Returns the current year. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.year # 2014
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.year # 2014
```

month

Returns the current month. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.month # 5
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.month # 5
```

---

day

Returns the current day. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.day # 15
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.day # 15
```

monday? (tuesday?, wednesday?, etc.)

Returns true or false depending on whether the day is Monday (Tuesday, Wednesday, etc.). You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.monday? # false
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.thursday? # true
```

---

hour

Returns the current hour, in 24-hour format. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.hour # 17
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.hour # 17
```

Ruby Objects & Common Methods Cheatsheet

---

min

Returns the current minute. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.min # 45
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.min # 45
```

---

sec

Returns the current second. You can either use it with the method now (so Ruby knows the current date & time):

```
time = Time
time.now.sec # 24
```

Or by first creating a variable with the Time object and now method:

```
time = Time.now
time.sec # 24
```

---

to_a

Returns an array containing all the elements of the specified time.

```
time = Time
time_array = time.now.to_a # [24, 45, 17, 15, 5, 2014, 5, 136, true,
"EDT"]
```

More Information About Ruby

For more information about Ruby, check out the Ruby Documentation.