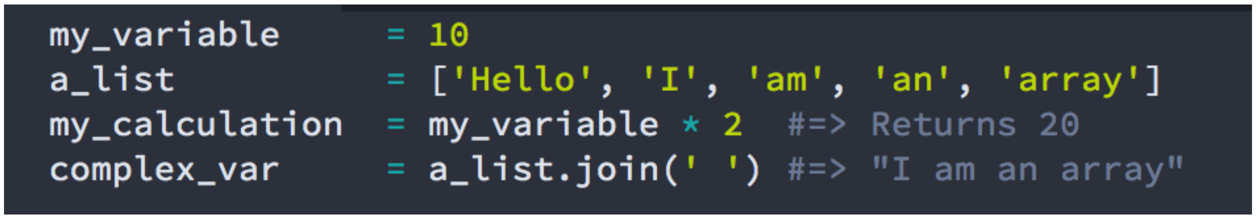
Ruby and Rails Cheat Sheet

Variables

Variables allow us to store values as a name understandable by humans. Unlike constants, the value of a variable can be set and reset/overwritten as many times as you want or need to.

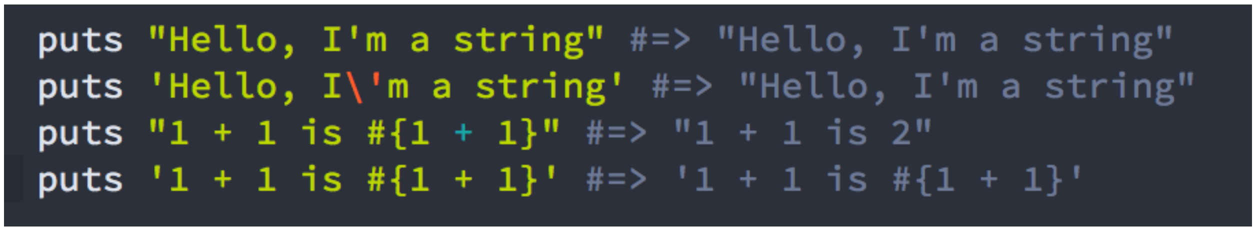
Syntax



```
my_variable     = 10
a_list          = ['Hello', 'I', 'am', 'an', 'array']
my_calculation  = my_variable * 2  #=> Returns 20
complex_var     = a_list.join(' ') #=> "I am an array"
```

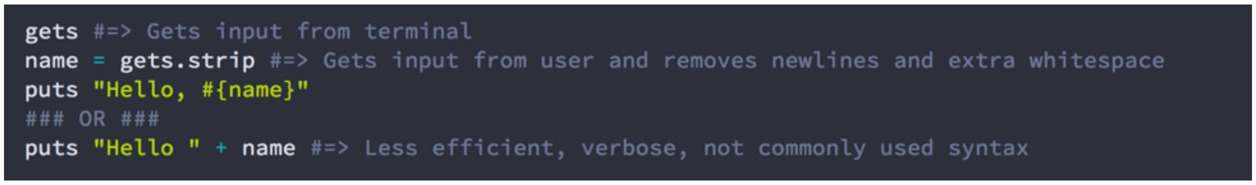
Working with Strings
Strings are just a set of characters enclosed in double or single quotes. Double quotes allow you to do things differently than single quotes. Strings, like all other built-in Ruby classes have methods on them to allow you to manipulate their value.

Escape characters and the difference between single and double quotes



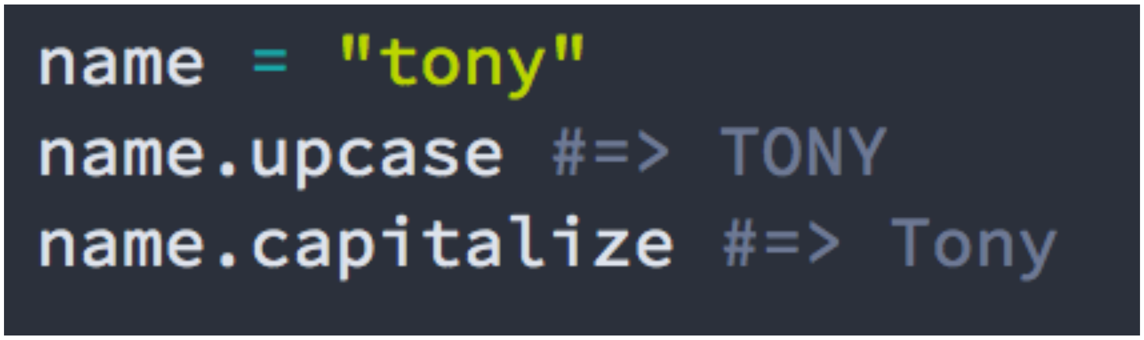
```
puts "Hello, I'm a string" #=> "Hello, I'm a string"
puts 'Hello, I\'m a string' #=> "Hello, I'm a string"
puts "1 + 1 is #{1 + 1}" #=> "1 + 1 is 2"
puts '1 + 1 is #{1 + 1}' #=> '1 + 1 is #{1 + 1}'
```

Get input from user in the terminal and concatenate strings



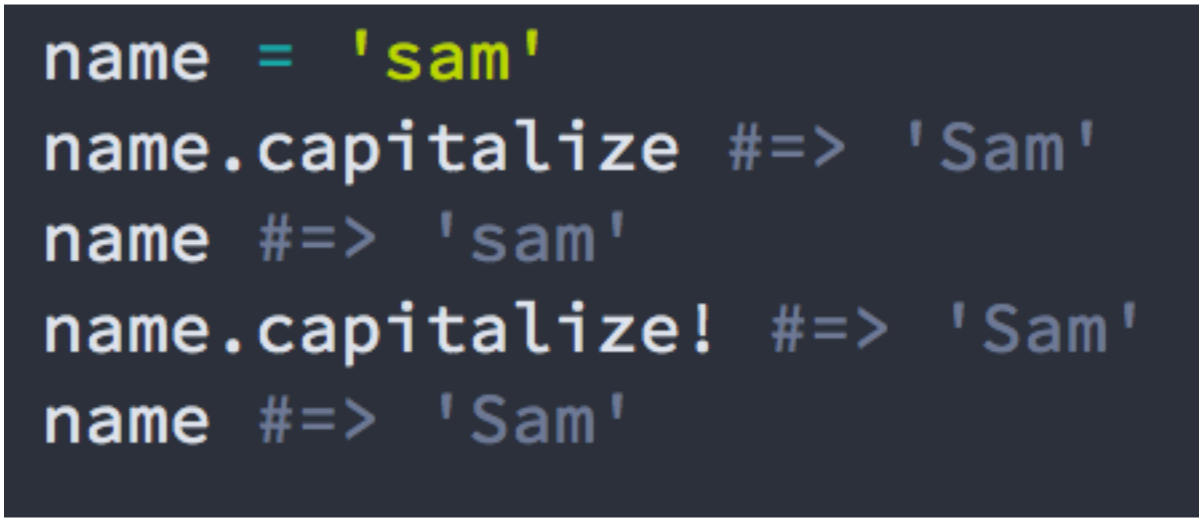
```
gets #=> Gets input from terminal
name = gets.strip #=> Gets input from user and removes newlines and extra whitespace
puts "Hello, #{name}"
### OR ###
puts "Hello " + name #=> Less efficient, verbose, not commonly used syntax
```

Built in string methods



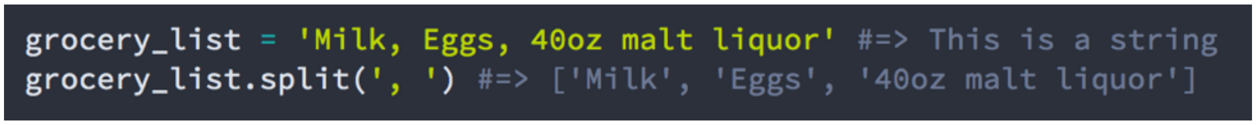
```
name = "tony"
name.upcase #=> TONY
name.capitalize #=> Tony
```

Update a variable *in-place*



```
name = 'sam'
name.capitalize #=> 'Sam'
name #=> 'sam'
name.capitalize! #=> 'Sam'
name #=> 'Sam'
```

More built in methods

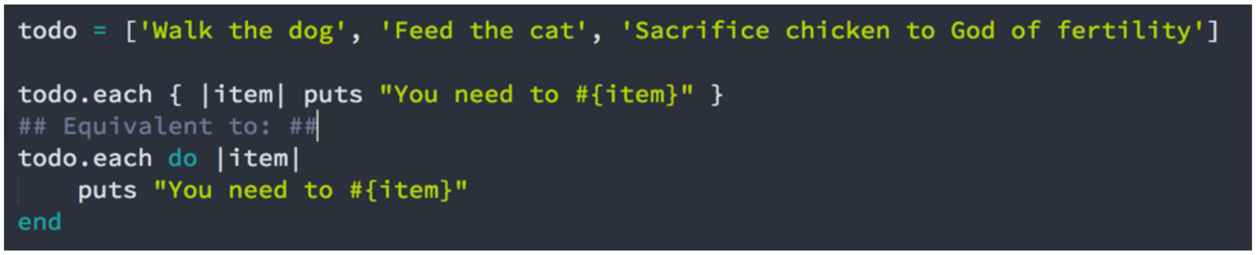


```
grocery_list = 'Milk, Eggs, 40oz malt liquor' #=> This is a string
grocery_list.split(', ') #=> ['Milk', 'Eggs', '40oz malt liquor']
```

Arrays

Arrays are basically lists of values. Arrays can hold an infinite number of values all of which can be a different type.They're useful for storing related information. Anything you'd use a list to keep track of in real life would be useful in an array.

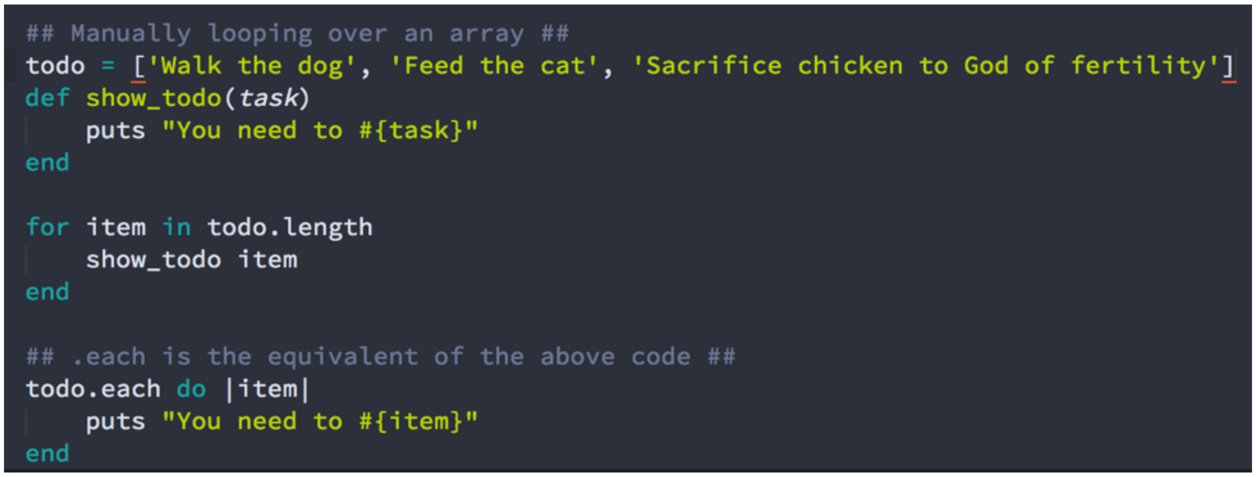
Syntax for defining and looping over arrays



```
todo = ['Walk the dog', 'Feed the cat', 'Sacrifice chicken to God of fertility']

todo.each { |item| puts "You need to #{item}" }
## Equivalent to: ##
todo.each do |item|
    puts "You need to #{item}"
end
```

Blocks are unnamed functions. In this example we show how we would loop over an array manually and then show the same function being run by passing a block to the ".each" method of the array class.



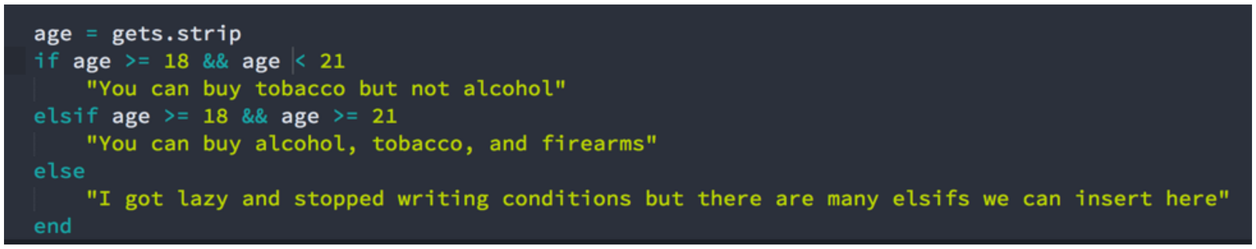
```
## Manually looping over an array ##
todo = ['Walk the dog', 'Feed the cat', 'Sacrifice chicken to God of fertility']
def show_todo(task)
    puts "You need to #{task}"
end

for item in todo.length
    show_todo item
end

## .each is the equivalent of the above code ##
todo.each do |item|
    puts "You need to #{item}"
end
```

Conditionals

Conditionals allow us to compare values. Conditionals can have an infinite number of conditions to check but its smart to just keep them to a handful of checks.



```
age = gets.strip
if age >= 18 && age < 21
    "You can buy tobacco but not alcohol"
elsif age >= 18 && age >= 21
    "You can buy alcohol, tobacco, and firearms"
else
    "I got lazy and stopped writing conditions but there are many elsifs we can insert here"
end
```

Sublime Text Shortcuts

See http://billpatrianakos.me/blog/2014/09/11/sublime-text-keyboard-shortcuts/ for a table of useful shortcuts for Sublime Text. Remember, on Windows and Linux, just replace the Command key with the Control key.