# GitHub

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **git init** | Initializes a new folder |
| **git add .** | Starts tracking changes for a given file name |
| **git status** | Checks status for new files, changes to be committed, etc. |
| **staging area vs. repository** | ***Staging area*** shows what files we are manipulated but do not store the data. In order to save, you must store in the ***repository***. |
| **git commit** | To store changes made, you must use the commit command. |
| **wildcard \* (ex: \*.txt)** | Used to add multiple items |
| **git log** | Shows all changes made in a journal format |
| **git remote add origin** | Creates a link between your online repository and your current file offline |
| **git push**   * **-u** * **Master** | Pushes local repo to server   * Tells git to remember the parameters * Default folder |
| **git pull** | Pulls the item back from the server to check any changes made (does a git fetch)  *There’s a shortcut that he’s going to leave to us to figure out* |
| **git diff**   * **HEAD** * **--staged** | Checks the differences in the file between when you created it and what was changed on github   * Checks most recent commit with pointer * Checks changes within files that were staged |
| **git reset** | Unstages files |
| **git checkout -- <target>** | Gets rid of changes since last commit to target |
| **git branch**   * **d** | Creates a branch (useful for making your own commits separate of what everyone else is doing)   * deletes items |
| **git rm** | Removes files |
| **git merge** | Merges files, (must switch to the folder that you want to get to using checkout) |
| **rm –rf .git** | Removes initialized git |
| **Operator** | **Function** |
| **git reflog** | Reverses everything that you have done  Gives you all the history of the changes you’ve made |

# Ruby on Rails

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **<%= %> (ERB)** | Embedded RuBy  Reads in source code html.ERB and runs anything between the lines using Ruby on Rails |
| **rails new <folder\_name>** | Create new folder in it |
| **rails server** | Generates server |
| **rails generate controller Name method1 method2** | Creates new controller with “Name” and passes two empty methods into it  NB: the controller name must always be capitalized |
| **puts** | Prints out to the console |
| **gets**   * gets.chomp | Gets the users input on the next command line   * removes extra line when inputting code |
| **eval** | Evaluates ruby code and returns result |
| **irb**  *Ex: “Kernel.loop {Kernel.puts(Kernel.eval(Kernel.gets()))} ”* | Allows you to type out ruby code in the command line   * creates a loop for you to keep entering ruby code |
| **Syntax**   * **local variables** * **classes** * **symbols** | * Use underscores between words (ex: @line\_object) * Capitalize words (ex: LineObject) * Use colon (ex: :id) |
| **‘’ vs “”** | Ruby does more work for strings in double quotations:   * Scans for \n (break) * Scans for #{expression} (expression interpolations) |
| **<<() method**  Ex: ages << person.age | Appends a value to its receiver. Most commonly used with arrays |
| **Shortcut for creating an array of words**  a = [ *'ant'*, *'bee'*, *'cat'*, *'dog'*, *'elk'* ]  *# this is the same:*  a = *%w{ ant bee cat dog elk }* |  |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **Setting up a hash:**  inst\_section = {  :cello => *'string'*,  :clarinet => *'woodwind'*,  :drum => *'percussion'*,  :violin => *'string'*  }  Can also use symbols  nst\_section = {  cello: *'string'*,  clarinet: *'woodwind'*,  drum: *'percussion'*,  oboe: *'woodwind'*,  }  Can also set the default value of a hash to be something other than nil  ex: my\_hash = Hash.new(‘Trady Blix’) | * => assigns value * :cello assigns a key |
| **Passing hashes as parameters on method calls**  redirect\_to action: *'show'*, id: product.id | * Hash has to be last parameter of the call * You can also omit () when calling functions and use colons instead |
| **Return value**  Add *return self* to make to method return  By default, puts returns nil | * Useful for chaining (ex: frank.make\_toast.make\_milkshake) |
| **Strings**   * .length * .delete() * .reverse * .upcase && .downcase * .capitalize! * .gsub(‘word1’, ‘word2’) * .split() | * Returns length of string * Deletes every instance of that character in the string * Reverses the letters in the string * Makes everything upper/lower case * Capitalizes first letter in string * Substitutes 2nd word for every instance of the 1st * Splits the string at every instance of the character passed in (delimiter) and then returns an array |
| **get ‘/’ do**  **“Hello world”**  **End** | Tells us that when the user reaches the root folder (“/”) print out Hello World |
| **“””** | Like a preformatted string |
| **Operator** | **Function** |
| **Methods**   * .include? “string” * .sort\_by {|a,b| b} * .intern * .select{|k,v| k > :c } * .each\_key and .each\_value * .respond\_to?(:method)   validates()  ex: validates :title, :description, :image\_url, presence: true  .select | * Evaluates to true if it finds what it’s looking for * Hash will be sorted by increasing order of b * Converts strings to symbols * Takes the values in a hash and compares it to an arbitrary value; any values that meet the criteria are returned * Checks to see if that method can be called on that variable (Ruby does not care about the objects contents moreso than the methods it can call) * Checks for particular values before it stores to the database   In arrays and hashes, only allows certain values to be selected |
| **Loops**  ***While loops***  while <condition is true>  code  i += 1  end  ***Until loops***  until <condition is true>  code  end  ***for loops***  for num in 1..10  puts num  end  ***loop***  loop do  code  end   * next * break   ***.each iterator***  object.each do |item| # Do something end  ***.times iterator***  10.times {print “string”} | * Use += instead because ++ and - - do not exist * Complements the while loop * Will continue running until the condition is true * Runs while condition is FALSE! * Same as old for loop * “..” includes last number “…” excludes it * Loops until terminated * Usually combined with break statements * ‘next’ is used to skip to the next iteration of the loop * ‘break’ is used to exit the loop * Variable name between | | can be anything, it is just a placeholder for each element of the object you’re using .each on * Loops a fixed number of times * A compact version of the for loop |
| **template :layout** | Sinatra has a template that you can use to apply to all of the different pages |
| **<% yield %>** | Placeholder to insert into your template |
| **Simpler If Statements**  *Expression* if *boolean*  puts “Hello world!” if true  puts “Hello world!” unless false | Used if you just want to use a one line if statement  Does not require an ‘end’ |
| **Ternary Statement**  *Boolean ? Do this if true : Do this if false*  output.empty? ? num : output  if output.empty?  output = num  end  puts output | Squishes a standard if/else statement into one line |
| **Case Statement**  *case variable*  *when <condition>*  *action*  *end*  Can also fold it up into one liners as follows:  *case variable*  *when <condition> then <action>*  *end* | Used for when you have a lot of conditions to check |
| **Conditional Assignment**  || = | Assigns a value if it hasn’t already been assigned. |
| **Implicit Return**  def method(a,b)  a + b  end | Returns the last expression evaluated in the method |
| **Splat arguments**  Indicated by a \*  Ex:  def what\_up(greeting, \*bros)  bros.each { |bro| puts "#{greeting}, #{bro}!" }  end | Useful for when the method is unsure how many arguments will be passed into it. The example on the left allows you to print out multiple ‘bros’. |
| **Combined Comparison Operator**  item1 <=> item2 | Compares two Ruby objects   * Returns 0 if they are equal * Returns -1 if item1 < item2 * Returns 1 if item1 > item2 |
| **Blocks**  They are methods which have no name  Not necessarily true! A ruby block is just a bit of code that can be executed. They are NOT objects.  *Collect Method*  Ex: my\_nums = [1, 2, 3]  my\_nums.collect { |num| num\*\*2 }  # 🡺 1, 4, 9  *Yield* | Used to execute code like a method. However, they are only called once and then disappear since they are not stored. You can also pass blocks into methods.  Takes a block and applies the expression in the block to every element in an array  Some methods can accept blocks because they have a way of transferring control from the calling method to the block and back again. Can be built into the methods using *yield*. |

Questions: for syntax, is it required to have the equals?

|  |  |
| --- | --- |
| **Procs**  Blocks that you store in a variable  Ex: multiples\_of\_3 = Proc.new do |n|  n % 3 == 0  end  (1..100).to\_a.select(&multiples\_of\_3)  *Call Method*  *.call*  *Symbols and Procs* | Blocks cannot be saved. However, instead of having to define a method, you can define a proc instead. To do so, assign a variable to the block and now you can access that variable like a method. Good for DRY principles.  & operator used to convert variable into a block  Can call a proc by using the call method.  Can convert symbols to procs using the & sign |
| **Lambdas**  Lambda {block code here} | Like procs, they are objects.  Lambdas check the number of arguments passed to it, a proc does not.  When a lambda returns, it gives control back to the method. Procs return immediately. |
| **rake**  ex: rake db:migration  rake test  rake db:rollback | Rake looks for all migrations not applied to the database and applies them  Preset tests for the scaffolding  Rollbacks the previous migration (could be useful!) |
| **…** | Alternative to string literals |
| **Concatenation Operator**  << | Used to push |
| **Scope Resolution Operator**  :: | Fancy way of referencing a class within a namespace |
| **Rails console** | Great way of playing with code in your command line |

|  |  |
| --- | --- |
| **Variables**  Variables have a defined scope depending on how you initialize them. You can do so in the following manner  *Local variables*  Ex: name = “Ronald”  *Global Variables*  $name  *Class Variables*  @@name  *Instance Variables*  @name | When defining a variable inside something it will be local to that code only. Ex: methods  Global variables are accessible everywhere.  Class variables are similar to instance variables except that it belongs to the actual class and can be accessed by all instances of that class.  Instance variables belong to an instance of a class. For example, if you create me = Something.new, the instance variable belongs to “me”. |
| **Inheritance**  *Syntax: <*  *super*  *one class at a time* | Allows one class to inherit everything from another class  Allows you to directly access the attributes or methods of a superclass within the keyword.  You can only inherit from one class, otherwise you will get a superclass error (get around by mixins). |
| **Instance Level Method**  # | Used to define instance level methods |

# Command Line Operators

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **ls** | List stuff |
| **cd** | Opens up a folder down |
| **which <function>** | Finds where that function is located |
| **<function> $PATH** | Finds exact path |
| **Flags**   * **-e** * **-m** | * Execute * Message |
| **man <function>**  **Use q to get out of program**  **Or quit or exit** | Lists a summary of all the items in that function |
| **gem**   * gem install sinatra | Can be used to access packages of data for rails   * installs the package |
| **require** | Loads the package |