# PROGRAMMING CHEAT SHEET

## HTML SYNTAX

### Starting an HTML Document

**<!DOCTYPE HTML>** - Identifying the document as an HTML document

**<html></html>** - Enter in code

**<head></head>** - Used for title

**<title></title>** - Change webpage title (shows up on tab)

**<body></body>** - For webpage content

### Paragraphs

**<p></p> -** Paragraph

**<br></br> -** Break

### Images

**<img src=“src”/>** - Self-closing

### Links

**<a href=“src”></a>**

### Tables

**<table></table>** - Start table

**<tr></tr>** - Table row

**<td></td>** - Table data (columns in rows, must be nested in tr)

### Lists

**<ol></ol>** - Ordered list (numbered)

**<ul></ul>** - Unordered list (bullets)

**<li></li>** - List Item

* Can incorporate ‘style’ tags

### Formatting Font

**font-family: “ ”;** - Changes font

**color: “ ”;** - Changes color

**<em></em>** **-** Italicize

**<strong></strong>** **-** Bold

**<span style=“ ”></span> -** Apply formatting to a section

# Javascript

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **“Name”.length** | Returns length of a string |
| **confirm(“”);** | Returns a message with a pop-up box |
| **prompt(“”);** | Pop-up box with input |
| **Math Operators**   * + * - * / * \* * % * ( ) | * Add * Subtract * Divide * Multiply * Mod (returns remainder) * Controls order |
| **Boolean Operators**   * **<** * **>** * **<=** * **>=** * **===** * **!==** | * Less than * Greater than * Less than or equal to * Greater than or equal to * Equal to * Not equal to |
| **If Statements**  If (condition) {  Codetocarryout  } | Processes code inside brackets if condition is met |
| **Else Statements** | Processes else if the ‘if’ condition is not met |
| **.substring (x, y)** | Takes a substring of original string  ‘x’ is where it begins  ‘y’ is where it ends  0 is the first letter |
| **Initiating variables**  var myVariable = itemtype; | Declares variable to desired item type |
| **Creating Functions**  var itemName = function(inputs) {  }; | Declares a function for repeating codes |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **For Loops**  for (var i = 2 ; i < 13; i++) {  console.log(i);  } | Loops from initialized integer until a condition is true and will repeat by a specified rule. |
| **Loop Operators**   * ++ * -- * += * -= | * Increment by 1 * Decrement by 1 * Increment by specified number * Decrement by specified number |
| **Arrays**  var junkData = ["Eddie Murphy", 49, "peanuts", 31]; | Stores multiple items into an array  Note: Arrays start counting from 0 |
| **While Loops**  While (condition) {  Some action;  } | Continues to loop until a certain condition is met. Good if the number of iterations is unknown. |
| **Do While Loops**  Do {  Some actions;  } While (condition); | Does one iteration before the condition is evaluated to be true or false. |
| **Switch**  Switch(variable) {  Case(‘text’):  Function;  Break;  Default:  Function;  Break;  } | Searches a variable for a particular case and carries out the appropriate functions. |
| **Logical Operators**   * And: && * Or: || * Not: ! |  |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **Objects**  *Literal Notation*  var object = {  key: value,  key: value,  key: value  };  *Constructor Notation*  var object = new Object();  object[“name”] = value; OR object.name = value; | Creates an object that has different properties to be accessed. Is different than an array in that the items can be called separately with no integer value. |
| **Accessing Object Properties**  Ex: var object1 = {  name: ‘Ronald’,  age: ‘ 24’  };  *Dot Notation:*  object1.name = ‘Ronathan’;  *Bracket Notation:*  var name = object1["Ronathan"]; | Different ways to access an object’s properties. |
| **Methods**  var object1 = {  age: 24  };  object1.setAge = function (x) {  object1.age = x;  }; | Methods are functions that are associated with objects. Just like a property is associated with an object.  Functions can only use parameters input, but a method can change an object’s properties or make calculations using their properties. |
| **For/In Loops** | Searches an object for all of its items. Similar to an array but looks all elements inside an object and terminates when it has run through all elements. |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **Making methods work for several objects**  Use the word ‘this’  Ex:  var setAge = function(newAge) {  this.age = newAge;  };  var bob = {};  bob.age = 30;  bob.setAge = setAge; | It will refer to the object that called the method.  Will assign ‘this’ to Bob. Needs to be done. |
| **Custom Constructors**  ***function*** Person(name,age) {  this.name = name;  this.age = age;  }  var george = ***new*** Person("George Washington", 275); | Initiates a custom creator from which we can use to quickly assign our objects properties very easily (in one line).  No longer need to make empty objects. Can assign values as soon as an object is created.  Can also create constructors with methods. |
| **Methods – Literal Notation**  speak: function () {  } | Can create functions in literal notation. |
| **Object Properties – Bracket Notation**  var aProperty = “job”;  james[aProperty]; | Can assign strings to a variable and use that same variable to call a property.  Useful for For-In loops. |
| **Classifying Types**  Typeof | Identifies whether a variable is a string, number, function, or object. |
| **Finding out Properties**  object.hasOwnProperty(‘property’); | Identifies whether an object has a particular property.  Returns a Boolean value |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **Object Oriented Programming (OOP)**  ***Class*** | A new constructor is defining a new class. |
| ***Prototype***  Dog.prototype.bark = function () {  Code here;  }; | The prototype keeps track of what a class can / cannot do.  Using prototype will teach all new objects of the same class a new method. |
| ***Inheritance***  newClass.prototype = new Class(); | Other objects can use the methods and properties of another’s through inheritance.  You can also chain prototypes up, creating a first class, second class, third class and so on. |
| ***Public vs. Private*** | By default, all properties of an object are automatically public. This means that they can be accessed outside of that class.  To make a property or method private, make it a local variable to the constructor. |
| ***Using Typeof*** | Can use typeof combined with an If statement to ensure that only certain types of variables are printed (ex: strings) |

# GitHub

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **git init** | Initializes a new folder |
| **git status** | Checks status for new files, changes to be committed, etc. |
| **git add .** | Starts tracking changes for a given file name |
| **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***. |
| **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** | To remotely push your local repository to a server |
| **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 }* |  |

|  |  |
| --- | --- |
| **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 |
| **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 |
| **Ternary Statement**  output.empty? ? num : output  if output.empty?  output = num  end  puts output | Squishes a standard if statement into one line  Only works for if else  ***Ask question about this syntax*** |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **Methods**   * .include? “string” * .sort\_by {|a,b| b} * .intern * .select{|k,v| k > :c } * .each\_key and .each\_value   validates()  ex: validates :title, :description, :image\_url, presence: true  lul | * 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 for particular values before it stores to the database |
| **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 |

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **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 | 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. |
| **rake**  ex: rake db:migration  rake test | Rake looks for all migrations not applied to the database and applies them  Preset tests for the scaffolding |
| **…** | Alternative to string literals |

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

# 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 |