In this lesson take each keyword and first try to write out what it does from memory. Next, search online for it and see what it really does. This may be difficult because some of these are difficult to search for, but try anyway.

If you get one of these wrong from memory, make an index card with the correct definition and try to "correct" your memory.

Finally, use each of these in a small Ruby program, or as many as you can get done. The goal is to find out what the symbol does, make sure you got it right, correct it if you do not, then use it to lock it in.

# **Keywords**

| Keyword | Description | Example |
| --- | --- | --- |
| BEGIN | Run this block when the script starts. | BEGIN { puts "hi" } |
| END | Run this block when the script is done. | END { puts "hi" } |
| alias | Create another name for a function. | alias X Y |
| and | Logical and, but lower priority than &&. | puts "Hello" and "Goodbye" |
| begin | Start a block, usually for exceptions. | begin end |
| break | Break out of a loop right now. | while true; break; end |
| case | Case style conditional, like an if. | case X; when Y; else; end |
| class | Define a new class. | class X; end |
| def | Define a new function. | def X(); end |
| defined? | Is this class/function/etc. defined already? | defined? Class == "constant" |
| do | Create a block that maybe takes a parameter. | (0..5).each do |x| puts x end |
| else | Else conditional. | if X; else; end |
| elsif | Else if conditional | if X; elsif Y; else; end |
| end | Ends blocks, functions, classes, everything. | begin end # many others |
| ensure | Run this code whether an exception happens or not. | begin ensure end |
| for | For loop syntax. The .each syntax is preferred. | for X in Y; end |
| if | If conditional. | if X; end |
| in | In part of for-loops. | for X in Y; end |
| include | The include statement "mixes in" a module into a class. It’s a method. | include module\_name or include?(module\_name) |
| module | Define a new module. | module X; end |
| next | Skip to the next element of a .each iterator. | (0..5).each {|y| next } |
| not | Logical not. But use ! instead. | not true == false |
| or | Logical or. | puts "Hello" or "Goodbye" |
| redo | Rerun a code block exactly the same. | (0..5).each {|i| redo if i > 2} |
| require | require is similar to C include. It’s a method. | require(name), require “something” |
| rescue | Run this code if an exception happens. | begin rescue X; end |
| retry | In a rescue clause, says to try the block again. | (0..5).each {|i| retry if i > 2} |
| return | Returns a value from a function. Mostly optional. | return X |
| self | The current object, class, or module. | defined? self == "self" |
| super | The parent class of this class. | super |
| then | Can be used with if optionally. | if true then puts "hi" end |
| undef | Remove a function definition from a class. | undef X |
| unless | Inverse of if. | unless false then puts "not" end |
| until | Inverse of while, execute block as long as false. | until false; end |
| when | Part of case conditionals. | case X; when Y; else; end |
| while | While loop. | while true; end |
| yield | Pause and transfer control to the code block. | yield |

# **Data Types**

For data types, write out what makes up each one. For example, with strings write out how you create a string. For numbers write out a few numbers.

| Type | Description | Example |
| --- | --- | --- |
| true | True boolean value. | true or false == true |
| false | False boolean value. | false and true == false |
| nil | Represents "nothing" or "no value". | x = nil |
| strings | Stores textual information. | x = "hello" |
| numbers | Stores integers. | i = 100 |
| floats | Stores decimals. | i = 10.389 |
| arrays | Stores a list of things. | j = [1,2,3,4] |
| hashes | Stores a key=value mapping of things. | e = {'x' => 1, 'y' => 2} |

# **String Escape Sequences**

For string escape sequences, use them in strings to make sure they do what you think they do.

| Escape | Description |
| --- | --- |
| \\ | Backslash |
| \' | Single-quote |
| \" | Double-quote |
| \a | Bell |
| \b | Backspace |
| \f | Formfeed |
| \n | Newline |
| \r | Carriage |
| \t | Tab |
| \v | Vertical tab |

# **Operators**

Some of these may be unfamiliar to you, but look them up anyway. Find out what they do, and if you still can't figure it out, save it for later.

| Operator | Description | Example |
| --- | --- | --- |
| + | Add | 2 + 4 == 6 |
| - | Subtract | 2 - 4 == -2 |
| \* | Multiply | 2 \* 4 == 8 |
| \*\* | Power of | 2 \*\* 4 == 16 |
| / | Divide | 2 / 4.0 == 0.5 |
| % | Modulus | 2 % 4 == 2 |
| > | Greater than | 4 > 4 == false |
| . | Dot access | "1".to\_i == 1 |
| :: | Colon access | Module::Class |
| [] | List brackets | [1,2,3,4] |
| ! | Not | !true == false |
| < | Less than | 4 < 4 == false |
| > | Greater than | 4 < 4 == false |
| >= | Greater than equal | 4 >= 4 == true |
| <= | Less than equal | 4 <= 4 == true |
| <=> | Comparison | 4 <=> 4 == 0 |
| == | Equal | 4 == 4 == true |
| === | Equality | 4 === 4 == true |
| != | Not equal | 4 != 4 == false |
| && | Logical and (higher precedence) | true && false == false |
| || | Logical or (higher precedence) | true || false == true |
| .. | Range inclusive | (0..3).to\_a == [0, 1, 2, 3] |
| ... | Range non-inclusive | (0...3).to\_a == [0, 1, 2] |
| @ | Object scope | @var ; @@classvar |
| @@ | Class scope | @var ; @@classvar |
| $ | Global scope | $stdin |