



- [Home](#)
- [About](#)
- [Business Plan »](#)
- [Communication »](#)
- [Dieting](#)
- [Sales](#)
- [Sitemap](#)
- [Videos »](#)
- [Web Design »](#)

- [Communication »](#)
- [Diet Nutritional](#)
- [Flash Tutorial](#)
- [How To »](#)
- [Investing](#)
- [iPad »](#)
- [Marketing »](#)
- [Most Popular](#)
- [Royalty Free Photos](#)
- [Sales](#)
- [Web Design »](#)



share

Learn Lua in One Video

Posted by [Derek Banas](#) on Jun 28, 2015 in [Web Design](#) | [0 comments](#)



In todays video you will learn Lua in one video. I'll cover Data Types, Math, Conditionals, Strings, Looping, Repeat Until, getting user input, For, For In, Tables, Functions, returning multiple values, Variadic Functions, Closures, Coroutines, File I/O, Modules, Metatables, OOP, Inheritance and a lot more.

All of the code used in the video can be found below after the video. I also have videos like this for [HTML5](#), [CSS3](#), [JavaScript](#), [Java](#), [PHP](#), [OOP PHP](#), [MySQL](#), [Ruby](#), [Go](#), [C++](#), [Python](#), [Sass](#), [Susy](#), [Objective C](#) and [Swift](#).

If you like videos like this, it helps to tell Google with a click here [googleplusone]

Cheat Sheet From the Video

```
1 -- Prints to the screen (Can end with semicolon)
2 print("Hello World")
3
4 --[[
5 Multiline comment
6 ]]
7
8 -- Variable names can't start with a number, but can contain letters, numbers
9 -- and underscores
10
11 -- Lua is dynamically typed based off of the data stored there
```

```
12 -- This is a string and it can be surrounded by ' or "
13 name = "Derek"
14
15 -- Another way to print to the screen
16 -- Escape Sequences : \n \b \t \\ \" \'
17 -- Get the string size by proceeding it with a #
18 io.write("Size of string ", #name, "\n")
19
20 -- You can store any data type in a variable even after initialization
21 name = 4
22 io.write("My name is ", name, "\n")
23
24 -- Lua only has floating point numbers and this is the max number
25 bigNum = 9223372036854775807 + 1
26 io.write("Big Number ", bigNum, "\n")
27
28 io.write("Big Number ", type(bigNum), "\n")
29
30 -- Floats are precise up to 13 digits
31 floatPrecision = 1.999999999999 + 0.0000000000005
32 io.write(floatPrecision, "\n")
33
34 -- We can create long strings and maintain white space
35 longString = [[
36 I am a very very long
37 string that goes on for
38 ever]]
39 io.write(longString, "\n")
40
41 -- Combine Strings with ..
42 longString = longString .. name
43 io.write(longString, "\n")
44
45 -- Booleans store with true or false
46 isAbleToDrive = true
47 io.write(type(isAbleToDrive), "\n")
48
49 -- Every variable gets the value of nil by default meaning it has no value
50 io.write(type(madeUpVar), "\n")
51
52 -- ----- MATH -----
53 io.write("5 + 3 = ", 5+3, "\n")
54 io.write("5 - 3 = ", 5-3, "\n")
55 io.write("5 * 3 = ", 5*3, "\n")
56 io.write("5 / 3 = ", 5/3, "\n")
57 io.write("5.2 % 3 = ", 5%3, "\n")
58
59 -- Shorthand like number++ and number += 1 aren't in Lua
60
61 -- Math Functions: floor, ceil, max, min, sin, cos, tan,
62 -- asin, acos, exp, log, log10, pow, sqrt, random, randomseed
63
64 io.write("floor(2.345) : ", math.floor(2.345), "\n")
65 io.write("ceil(2.345) : ", math.ceil(2.345), "\n")
66 io.write("max(2, 3) : ", math.max(2, 3), "\n")
67 io.write("min(2, 3) : ", math.min(2, 3), "\n")
68 io.write("pow(8, 2) : ", math.pow(8, 2), "\n")
69 io.write("sqrt(64) : ", math.sqrt(64), "\n")
70
71 -- Generate random number between 0 and 1
72 io.write("math.random() : ", math.random(), "\n")
73
74 -- Generate random number between 1 and 10
75 io.write("math.random(10) : ", math.random(10), "\n")
76
77 -- Generate random number between 1 and 100
78 io.write("math.random(1,100) : ", math.random(1,100), "\n")
79
80 -- Used to set a seed value for random
81 math.randomseed(os.time())
82
83 -- Print float to 10 decimals
84 print(string.format("Pi = %.10f", math.pi))
85
86 -- ----- CONDITIONALS -----
87 -- Relational Operators : > < >= <= == ~=
88 -- Logical Operators : and or not
89
90 age = 13
91
92 if age < 16 then
93     io.write("You can go to school", "\n")
94     local localVar = 10
95 elseif (age >= 16) and (age < 18) then
96     io.write("You can drive", "\n")
97 else
98     io.write("You can vote", "\n")
99 end
100
101 -- A variable marked local is local only to this if statement
102 -- io.write("Local Variable : ", localVar)
103
104 if (age < 14) or (age > 67) then io.write("You shouldn't work\n") end
105
106 -- Format, convert to string and place boolean value with string.format
107 print(string.format("not true = %s", tostring(not true)))
108
109 -- There is no ternary operator in Lua
110 -- canVote = age > 18 ? true : false
```

```
111
112 -- This is similar to the ternary operator
113 canVote = age > 18 and true or false
114 io.write("Can I Vote : ", tostring(canVote), "\n")
115
116 -- There is no Switch statement in Lua
117
118 -- ----- STRINGS -----
119 quote = "I changed my password everywhere to 'incorrect.' That way when I forget it,it always reminds me, 'Your password is incorrect.'"
120
121 io.write("Quote Length : ", string.len(quote), "\n")
122
123 -- Return the string after replacing
124 io.write("Replace I with me : ", string.gsub(quote, "I", "me"), "\n")
125
126 -- Find the index of a matching String
127 io.write("Index of password : ", string.find(quote, "password"), "\n")
128
129 -- Set characters to upper and lowercase
130 io.write("Quote Upper : ", string.upper(quote), "\n")
131 io.write("Quote Lower : ", string.lower(quote), "\n")
132
133 -- ----- LOOPING -----
134 i = 1
135 while (i <= 10) do
136     io.write(i)
137     i = i + 1
138
139     -- break throws you out of a loop
140     -- continue doesn't exist with Lua
141     if i == 8 then break end
142 end
143 print("\n")
144
145 -- Repeat will cycle through the loop at least once
146 repeat
147     io.write("Enter your guess : ")
148
149     -- Gets input from the user
150     guess = io.read()
151
152     -- Either surround the number with quotes, or convert the string into
153     -- a number
154 until tonumber(guess) == 15
155
156 -- Value to start with, value to stop at, increment each loop
157 for i = 1, 10, 1 do
158     io.write(i)
159 end
160
161 print()
162
163 -- Create a table which is a list of items like an array
164 months = {"January", "February", "March", "April", "May",
165 "June", "July", "August", "September", "October", "November",
166 "December"}
167
168 -- Cycle through table where k is the key and v the value of each item
169 for k, v in pairs(months) do
170     io.write(v, " ")
171 end
172
173 print()
174
175 -- ----- TABLES -----
176 -- Tables take the place of arrays, dictionaries, tuples, etc.
177
178 -- Create a Table
179 aTable = {}
180
181 -- Add values to a table
182 for i = 1, 10 do
183     aTable[i] = i
184 end
185
186 -- Access value by index
187 io.write("First Item : ", aTable[1], "\n")
188
189 -- Items in Table
190 io.write("Number of Items : ", #aTable, "\n")
191
192 -- Insert in table, at index, item to insert
193 table.insert(aTable, 1, 0)
194
195 -- Combine a table as a String and separate with provided separator
196 print(table.concat(aTable, ", "))
197
198 -- Remove item at index
199 table.remove(aTable, 1)
200 print(table.concat(aTable, ", "))
201
202 -- Sort items in reverse
203 table.sort(aTable, function(a,b) return a>b end)
204 print(table.concat(aTable, ", "))
205
206 -- Create a multidimensional Table
207 aMultiTable = {}
208
209 for i = 0, 9 do
```

```
210     aMultiTable[i] = {}
211     for j = 0, 9 do
212         aMultiTable[i][j] = tostring(i) .. tostring(j)
213     end
214 end
215
216 -- Access value in cell
217 io.write("Table[0][0] : ", aMultiTable[1][2], "\n")
218
219 -- Cycle through and print a multidimensional Table
220 for i = 0, 9 do
221     for j = 0, 9 do
222         io.write(aMultiTable[i][j], " : ")
223     end
224     print()
225 end
226
227 -- ----- FUNCTIONS -----
228 function getSum(num1, num2)
229     return num1 + num2
230 end
231
232 print(string.format("5 + 2 = %d", getSum(5,2)))
233
234 function splitStr(theString)
235     stringTable = {}
236     local i = 1
237
238     -- Cycle through the String and store anything except for spaces
239     -- in the table
240     for str in string.gmatch(theString, "[^%s]+") do
241         stringTable[i] = str
242         i = i + 1
243     end
244
245     -- Return multiple values
246     return stringTable, i
247 end
248
249 -- Receive multiple values
250 splitStrTable, numOfStr = splitStr("The Turtle")
251
252 for j = 1, numOfStr do
253     print(string.format("%d : %s", j, splitStrTable[j]))
254 end
255
256 -- Variadic Function recieve unknown number of parameters
257 function getSumMore(...)
258     local sum = 0
259
260     for k, v in pairs{...} do
261         sum = sum + v
262     end
263     return sum
264 end
265
266 io.write("Sum : ", getSumMore(1,2,3,4,5,6), "\n")
267
268 -- A function is a variable in that we can store them under many variable
269 -- names as well as in tables and we can pass and return them though functions
270
271 -- Saving an anonymous function to a variable
272 doubleIt = function(x) return x * 2 end
273 print(doubleIt(4))
274
275 -- A Closure is a function that can access local variables of an enclosing
276 -- function
277 function outerFunc()
278     local i = 0
279     return function()
280         i = i + 1
281         return i
282     end
283 end
284
285 -- When you include an inner function in a function that inner function
286 -- will remember changes made on variables in the inner function
287 getI = outerFunc()
288 print(getI())
289 print(getI())
290
291 -- ----- COROUTINES -----
292 -- Coroutines are like threads except that they can't run in parallel
293 -- A coroutine has the status of running, susepnded, dead or normal
294
295 -- Use create to create one that performs some action
296 co = coroutine.create(function()
297     for i = 1, 10, 1 do
298         print(i)
299         print(coroutine.status(co))
300         if i == 5 then coroutine.yield() end
301     end end)
302
303 -- They start off with the status suspended
304 print(coroutine.status(co))
305
306 -- Call for it to run with resume during which the status changes to running
307 coroutine.resume(co)
```

```
309
310 -- After execution it has the status of dead
311 print(coroutine.status(co))
312
313 co2 = coroutine.create(function()
314     for i = 101, 110, 1 do
315         print(i)
316     end end)
317
318 coroutine.resume(co2)
319 coroutine.resume(co)
320
321 -- ----- FILE I/O -----
322 -- Different ways to work with files
323 -- r: Read only (default)
324 -- w: Overwrite or create a new file
325 -- a: Append or create a new file
326 -- r+: Read & write existing file
327 -- w+: Overwrite read or create a file
328 -- a+: Append read or create file
329
330 -- Create new file for reading and writing
331 file = io.open("test.lua", "w+")
332
333 -- Write text to the file
334 file:write("Random string of text\n")
335 file:write("Some more text\n")
336
337 -- Move back to the beginning of the file
338 file:seek("set", 0)
339
340 -- Read from the file
341 print(file:read("*a"))
342
343 -- Close the file
344 file:close()
345
346 -- Open file for appending and reading
347 file = io.open("test.lua", "a+")
348
349 file:write("Even more text\n")
350
351 file:seek("set", 0)
352
353 print(file:read("*a"))
354
355 file:close()
356
357 -- ----- MODULES -----
358 -- A Module is like a library full of functions and variables
359
360 -- Use require to gain access to the functions in the module
361 convertModule = require("convert")
362
363 -- Execute the function in the module
364 print(string.format("%.3f cm", convertModule.ftToCm(12)))
365
366 -- ----- METATABLES -----
367 -- Used to define how operations on tables should be carried out in regards
368 -- to adding, subtracting, multiplying, dividing, concatenating, or
369 -- comparing tables
370
371 -- Create a table and put default values in it
372 aTable = {}
373 for x = 1, 10 do
374     aTable[x] = x
375 end
376
377 mt = {
378     -- Define how table values should be added
379     -- You can also define _sub, _mul, _div, _mod, _concat (..)
380     __add = function (table1, table2)
381
382         sumTable = {}
383
384         for y = 1, #table1 do
385             if (table1[y] ~= nil) and (table2[y] ~= nil) then
386                 sumTable[y] = table1[y] + table2[y]
387             else
388                 sumTable[y] = 0
389             end
390         end
391
392         return sumTable
393     end,
394
395     -- Define how table values should be checked for equality
396     __eq = function (table1, table2)
397         return table1.value == table2.value
398     end,
399
400     -- For homework figure out how to check if less than
401     __lt = function (table1, table2)
402         return table1.value < table2.value
403     end,
404
405     -- For homework figure out how to check if less than or equal
406     __le = function (table1, table2)
407         return table1.value <= table2.value
```

```
408     end,
409 }
410
411 -- Attach the metamethods to this table
412 setmetatable(aTable, mt)
413
414 -- Check if tables are equal
415 print(aTable == aTable)
416
417 addTable = {}
418
419 -- Add values in tables
420 addTable = aTable + aTable
421
422 -- print the results of the addition
423 for z = 1, #addTable do
424     print(addTable[z])
425 end
426
427 -- ----- OBJECT ORIENTED PROGRAMMING -----
428 -- Lua is not an OOP language and it doesn't allow you to define classes
429 -- but you can fake it using tables and metatables
430
431 -- Define the defaults for our table
432 Animal = {height = 0, weight = 0, name = "No Name", sound = "No Sound"}
433
434 -- Used to initialize Animal objects
435 function Animal:new (height, weight, name, sound)
436
437     setmetatable({}, Animal)
438
439     -- Self is a reference to values for this Animal
440     self.height = height
441     self.weight = weight
442     self.name = name
443     self.sound = sound
444
445     return self
446 end
447
448 -- Outputs a string that describes the Animal
449 function Animal:toString()
450
451     animalStr = string.format("%s weighs %.1f lbs, is %.1f in tall and says %s", self.name, self.weight, self.height, self.sound)
452
453     return animalStr
454 end
455
456 -- Create an Animal
457 spot = Animal:new(10, 15, "Spot", "Roof")
458
459 -- Get variable values
460 print(spot.weight)
461
462 -- Call a function in Animal
463 print(spot:toString())
464
465 -- ----- INHERITANCE -----
466 -- Extends the properties and functions in another object
467
468 Cat = Animal:new()
469
470 function Cat:new (height, weight, name, sound, favFood)
471     setmetatable({}, Cat)
472
473     -- Self is a reference to values for this Animal
474     self.height = height
475     self.weight = weight
476     self.name = name
477     self.sound = sound
478     self.favFood = favFood
479
480     return self
481 end
482
483 -- Override an Animal function
484 function Cat:toString()
485
486     catStr = string.format("%s weighs %.1f lbs, is %.1f in tall, says %s and loves %s", self.name, self.weight, self.height, self.sound,
487
488     return catStr
489 end
490
491 -- Create a Cat
492 fluffy = Cat:new(10, 15, "Fluffy", "Meow", "Tuna")
493
494 print(fluffy:toString())
```

```
1 -- The module name and filename are the same
2 local convert = {}
3 function convert.ftToCm(feet)
4     return feet * 30.48
5 end
6 return convert
```

Leave a Reply

Your email address will not be published.

Comment

Name

Email

Website

Submit Comment





Search

Search

Buy me a Cup of Coffee
"Donations help me to keep the site running. One dollar is greatly appreciated." - (Pay Pal Secured)



Social Networks

-  Facebook
-  YouTube
-  Twitter
-  LinkedIn

My Facebook Page

Archives

- [April 2022](#)
- [March 2022](#)
- [February 2022](#)
- [January 2022](#)
- [June 2021](#)
- [May 2021](#)
- [April 2021](#)
- [March 2021](#)
- [February 2021](#)
- [January 2021](#)
- [December 2020](#)
- [November 2020](#)
- [October 2020](#)
- [September 2020](#)
- [August 2020](#)
- [July 2020](#)
- [June 2020](#)
- [May 2020](#)
- [April 2020](#)
- [March 2020](#)
- [February 2020](#)
- [January 2020](#)
- [December 2019](#)
- [November 2019](#)
- [October 2019](#)
- [August 2019](#)
- [July 2019](#)
- [June 2019](#)
- [May 2019](#)
- [April 2019](#)
- [March 2019](#)
- [February 2019](#)
- [January 2019](#)
- [December 2018](#)
- [October 2018](#)
- [September 2018](#)
- [August 2018](#)
- [July 2018](#)
- [June 2018](#)
- [May 2018](#)

- [April 2018](#)
- [March 2018](#)
- [February 2018](#)
- [January 2018](#)
- [December 2017](#)
- [November 2017](#)
- [October 2017](#)
- [September 2017](#)
- [August 2017](#)
- [July 2017](#)
- [June 2017](#)
- [May 2017](#)
- [April 2017](#)
- [March 2017](#)
- [February 2017](#)
- [January 2017](#)
- [December 2016](#)
- [November 2016](#)
- [October 2016](#)
- [September 2016](#)
- [August 2016](#)
- [July 2016](#)
- [June 2016](#)
- [May 2016](#)
- [April 2016](#)
- [March 2016](#)
- [February 2016](#)
- [January 2016](#)
- [December 2015](#)
- [November 2015](#)
- [October 2015](#)
- [September 2015](#)
- [August 2015](#)
- [July 2015](#)
- [June 2015](#)
- [May 2015](#)
- [April 2015](#)
- [March 2015](#)
- [February 2015](#)
- [January 2015](#)
- [December 2014](#)
- [November 2014](#)
- [October 2014](#)
- [September 2014](#)
- [August 2014](#)
- [July 2014](#)
- [June 2014](#)
- [May 2014](#)
- [April 2014](#)
- [March 2014](#)
- [February 2014](#)
- [January 2014](#)
- [December 2013](#)
- [November 2013](#)
- [October 2013](#)
- [September 2013](#)
- [August 2013](#)
- [July 2013](#)
- [June 2013](#)
- [May 2013](#)
- [April 2013](#)
- [March 2013](#)
- [February 2013](#)
- [January 2013](#)
- [December 2012](#)
- [November 2012](#)
- [October 2012](#)
- [September 2012](#)
- [August 2012](#)
- [July 2012](#)
- [June 2012](#)
- [May 2012](#)
- [April 2012](#)

- [March 2012](#)
- [February 2012](#)
- [January 2012](#)
- [December 2011](#)
- [November 2011](#)
- [October 2011](#)
- [September 2011](#)
- [August 2011](#)
- [July 2011](#)
- [June 2011](#)
- [May 2011](#)
- [April 2011](#)
- [March 2011](#)
- [February 2011](#)
- [January 2011](#)
- [December 2010](#)
- [November 2010](#)
- [October 2010](#)
- [September 2010](#)
- [August 2010](#)
- [July 2010](#)
- [June 2010](#)
- [May 2010](#)
- [April 2010](#)
- [March 2010](#)
- [February 2010](#)
- [January 2010](#)
- [December 2009](#)

Powered by [WordPress](#) | Designed by [Elegant Themes](#)
[About the Author](#) [Google+](#)