```lua
-- Prints to the screen (Can end with semicolon)
print("Hello World")

--[[
Multiline comment
]]

-- Variable names can't start with a number, but can contain letters, numbers
-- and underscores

-- Lua is dynamically typed based off of the data stored there
-- This is a string and it can be surrounded by ' or "
name = "Derek"

-- Another way to print to the screen
-- Escape Sequences : \n \b \t \\ \" \'
-- Get the string size by proceeding it with a #
io.write("Size of string ", #name, "\n")

-- You can store any data type in a variable even after initialization
name = 4
io.write("My name is ", name, "\n")

-- Lua only has floating point numbers and this is the max number
bigNum = 9223372036854775807 + 1
io.write("Big Number ", bigNum, "\n")

io.write("Big Number ", type(bigNum), "\n")

-- Floats are precise up to 13 digits
floatPrecision = 1.999999999999 + 0.0000000000005
io.write(floatPrecision, "\n")

-- We can create long strings and maintain white space
longString = [[
I am a very very long
string that goes on for
ever]]
io.write(longString, "\n")

-- Combine Strings with ..
longString = longString .. name
io.write(longString, "\n")

-- Booleans store with true or false
isAbleToDrive = true
io.write(type(isAbleToDrive), "\n")

-- Every variable gets the value of nil by default meaning it has no value
io.write(type(madeUpVar), "\n")

-- ---------- MATH ----------
io.write("5 + 3 = ", 5+3, "\n")
io.write("5 - 3 = ", 5-3, "\n")
io.write("5 * 3 = ", 5*3, "\n")
io.write("5 / 3 = ", 5/3, "\n")
io.write("5.2 % 3 = ", 5%3, "\n")

-- Shorthand like number++ and number += 1 aren't in Lua

-- Math Functions: floor, ceil, max, min, sin, cos, tan,
-- asin, acos, exp, log, log10, pow, sqrt, random, randomseed

io.write("floor(2.345) : ", math.floor(2.345), "\n")
io.write("ceil(2.345) : ", math.ceil(2.345), "\n")
io.write("max(2, 3) : ", math.max(2, 3), "\n")
io.write("min(2, 3) : ", math.min(2, 3), "\n")
io.write("pow(8, 2) : ", math.pow(8, 2), "\n")
io.write("sqrt(64) : ", math.sqrt(64), "\n")

-- Generate random number between 0 and 1
io.write("math.random() : ", math.random(), "\n")

-- Generate random number between 1 and 10
io.write("math.random(10) : ", math.random(10), "\n")

-- Generate random number between 1 and 100
io.write("math.random(1,100) : ", math.random(1,100), "\n")

-- Used to set a seed value for random
math.randomseed(os.time())

-- Print float to 10 decimals
print(string.format("Pi = %.10f", math.pi))

-- ---------- CONDITIONALS ----------
-- Relational Operators : > < >= <= == ~=
-- Logical Operators : and or not

age = 13

if age < 16 then
    io.write("You can go to school", "\n")
    local localVar = 10
elseif (age >= 16) and (age < 18) then
    io.write("You can drive", "\n")
else
    io.write("You can vote", "\n")
end

-- A variable marked local is local only to this if statement
-- io.write("Local Variable : ", localvar)

if (age < 14) or (age > 67) then io.write("You shouldn't work\n") end

-- Format, convert to string and place boolean value with string.format
print(string.format("not true = %s", tostring(not true)))

-- There is no ternary operator in Lua
-- canVote = age > 18 ? true : false

-- This is similar to the ternary operator
canVote = age > 18 and true or false
io.write("Can I Vote : ", tostring(canVote), "\n")

-- There is no Switch statement in Lua

-- ---------- STRINGS ----------
quote = "I changed my password everywhere to 'incorrect.' That way when I forget it,it always reminds

io.write("Quote Length : ", string.len(quote), "\n")

-- Return the string after replacing
io.write("Replace I with me : ", string.gsub(quote, "I", "me"), "\n")

-- Find the index of a matching String
io.write("Index of password : ", string.find(quote, "password"), "\n")

-- Set characters to upper and lowercase
io.write("Quote Upper : ", string.upper(quote), "\n")
io.write("Quote Lower : ", string.lower(quote), "\n")

-- ---------- LOOPING ----------
i = 1
while (i <= 10) do
  io.write(i)
  i = i + 1

  -- break throws you out of a loop
  -- continue doesn't exist with Lua
  if i == 8 then break end
end
print("\n")

-- Repeat will cycle through the loop at least once
repeat
  io.write("Enter your guess : ")

  -- Gets input from the user
  guess = io.read()

  -- Either surround the number with quotes, or convert the string into
  -- a number
until tonumber(guess) == 15

-- Value to start with, value to stop at, increment each loop
for i = 1, 10, 1 do
  io.write(i)
end

print()

-- Create a table which is a list of items like an array
months = {"January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November",
"December"}

-- Cycle through table where k is the key and v the value of each item
for k, v in pairs(months) do
  io.write(v, " ")
end

print()

-- ---------- TABLES ----------
-- Tables take the place of arrays, dictionaries, tuples, etc.

-- Create a Table
aTable = {}

-- Add values to a table
for i = 1, 10 do
  aTable[i] = i
end

-- Access value by index
io.write("First Item : ", aTable[1], "\n")

-- Items in Table
io.write("Number of Items : ", #aTable, "\n")

-- Insert in table, at index, item to insert
table.insert(aTable, 1, 0)

-- Combine a table as a String and seperate with provided seperator
```

```
print(table.concat(aTable, ", "))

-- Remove item at index
table.remove(aTable, 1)
print(table.concat(aTable, ", "))

-- Sort items in reverse
table.sort(aTable, function(a,b) return a>b end)
print(table.concat(aTable, ", "))

-- Create a multidimensional Table
aMultiTable = {}

for i = 0, 9 do
  aMultiTable[i] = {}
  for j = 0, 9 do
    aMultiTable[i][j] = tostring(i) .. tostring(j)
  end
end

-- Access value in cell
io.write("Table[0][0] : ", aMultiTable[1][2], "\n")

-- Cycle through and print a multidimensional Table
for i = 0, 9 do
  for j = 0, 9 do
    io.write(aMultiTable[i][j], " : ")
  end
  print()
end

-- ---------- FUNCTIONS ----------
function getSum(num1, num2)
  return num1 + num2
end

print(string.format("5 + 2 = %d", getSum(5,2)))

function splitStr(theString)

  stringTable = {}
  local i = 1

  -- Cycle through the String and store anything except for spaces
  -- in the table
  for str in string.gmatch(theString, "[^%s]+") do
    stringTable[i] = str
    i = i + 1
  end

  -- Return multiple values
  return stringTable, i
end

-- Receive multiple values
splitStrTable, numOfStr = splitStr("The Turtle")

for j = 1, numOfStr do
  print(string.format("%d : %s", j, splitStrTable[j]))
end

-- Variadic Function recieve unknown number of parameters
function getSumMore(...)
  local sum = 0

```

```lua
  for k, v in pairs{...} do
    sum = sum + v
  end
  return sum
end

io.write("Sum : ", getSumMore(1,2,3,4,5,6), "\n")

-- A function is a variable in that we can store them under many variable
-- names as well as in tables and we can pass and return them though functions

-- Saving an anonymous function to a variable
doubleIt = function(x) return x * 2 end
print(doubleIt(4))

-- A Closure is a function that can access local variables of an enclosing
-- function
function outerFunc()
  local i = 0
  return function()
    i = i + 1
    return i
  end
end

-- When you include an inner function in a function that inner function
-- will remember changes made on variables in the inner function
getI = outerFunc()
print(getI())
print(getI())

-- ---------- COROUTINES ----------
-- Coroutines are like threads except that they can't run in parallel
-- A coroutine has the status of running, susepnded, dead or normal

-- Use create to create one that performs some action
co = coroutine.create(function()
  for i = 1, 10, 1 do
  print(i)
  print(coroutine.status(co))
  if i == 5 then coroutine.yield() end
  end end)

-- They start off with the status suspended
print(coroutine.status(co))

-- Call for it to run with resume during which the status changes to running
coroutine.resume(co)

-- After execution it has the status of dead
print(coroutine.status(co))

co2 = coroutine.create(function()
  for i = 101, 110, 1 do
  print(i)
  end end)

coroutine.resume(co2)
coroutine.resume(co)

-- ---------- FILE I/O ----------
-- Different ways to work with files
-- r: Read only (default)
-- w: Overwrite or create a new file
-- a: Append or create a new file
-- r+: Read & write existing file
-- w+: Overwrite read or create a file
-- a+: Append read or create file

-- Create new file for reading and writing
file = io.open("test.lua", "w+")

-- Write text to the file
file:write("Random string of text\n")
file:write("Some more text\n")

-- Move back to the beginning of the file
file:seek("set", 0)

-- Read from the file
print(file:read("*a"))

-- Close the file
file:close()

-- Open file for appending and reading
file = io.open("test.lua", "a+")

file:write("Even more text\n")

file:seek("set", 0)

print(file:read("*a"))

file:close()

-- ---------- MODULES ----------
-- A Module is like a library full of functions and variables

-- Use require to gain access to the functions in the module
convertModule = require("convert")

-- Execute the function in the module
print(string.format("%.3f cm", convertModule.ftToCm(12)))

-- ---------- METATABLES ----------
-- Used to define how operations on tables should be carried out in regards
-- to adding, subtracting, multiplying, dividing, concatenating, or
-- comparing tables

-- Create a table and put default values in it
aTable = {}
for x = 1, 10 do
  aTable[x] = x
end

mt = {
  -- Define how table values should be added
  -- You can also define _sub, _mul, _div, _mod, _concat (..)
  __add = function (table1, table2)

    sumTable = {}

    for y = 1, #table1 do
      if (table1[y] ~= nil) and (table2[y] ~= nil) then
        sumTable[y] = table1[y] + table2[y]
      else
        sumTable[y] = 0
      end
    end
    return sumTable
  end,

  -- Define how table values should be checked for equality
  __eq = function (table1, table2)
    return table1.value == table2.value
  end,

  -- For homework figure out how to check if less then
  __lt = function (table1, table2)
    return table1.value < table2.value
  end,

  -- For homework figure out how to check if less then or equal
  __le = function (table1, table2)
    return table1.value <= table2.value
  end,
}

-- Attach the metamethods to this table
setmetatable(aTable, mt)

-- Check if tables are equal
print(aTable == aTable)

addTable = {}

-- Add values in tables
addTable = aTable + aTable

-- print the results of the addition
for z = 1, #addTable do
  print(addTable[z])
end

-- ---------- OBJECT ORIENTED PROGRAMMING ----------
-- Lua is not an OOP language and it doesn't allow you to define classes
-- but you can fake it using tables and metatables

-- Define the defaults for our table
Animal = {height = 0, weight = 0, name = "No Name", sound = "No Sound"}

-- Used to initialize Animal objects
function Animal:new (height, weight, name, sound)

  setmetatable({}, Animal)

  -- Self is a reference to values for this Animal
  self.height = height
  self.weight = weight
  self.name = name
  self.sound = sound

  return self
end

-- Outputs a string that describes the Animal
function Animal:toString()

  animalStr = string.format("%s weighs %.1f lbs, is %.1f in tall and says %s", self.name, self.weigh

  return animalStr
end

```

```
-- Create an Animal
spot = Animal:new(10, 15, "Spot", "Roof")

-- Get variable values
print(spot.weight)

-- Call a function in Animal
print(spot:toString())

-- ---------- INHERITANCE ----------
-- Extends the properties and functions in another object

Cat = Animal:new()

function Cat:new (height, weight, name, sound, favFood)
  setmetatable({}, Cat)

  -- Self is a reference to values for this Animal
  self.height = height
  self.weight = weight
  self.name = name
  self.sound = sound
  self.favFood = favFood

  return self
end

-- Overide an Animal function
function Cat:toString()

  catStr = string.format("%s weighs %.1f lbs, is %.1f in tall, says %s and loves %s", self.name, sel

  return catStr
end

-- Create a Cat
fluffy = Cat:new(10, 15, "Fluffy", "Meow", "Tuna")

print(fluffy:toString())
```