

-- A comment in Lua starts with a double-hyphen and runs to the end of the line.

*--[[Multi-line strings & comments
are adorned with double square brackets.]]*



Rensselaer

Getting Started

Python

```
1 from sys import *
2
3 def fact(n) :
4     if n < 1 :
5         return 1
6     else :
7         return n * fact(n-1)
8
9 stdout.write("Enter a number: ")
10 n = int(input())
11
12 print("Factorial is " + str(fact(n)))
```

Lua

```
? 1 function fact(n)
2     if n < 1 then
3         return 1
4     else
5         return n * fact(n-1)
6     end
7 end
8
9 io.write("Enter a number: ")
10 n = tonumber(io.read())
11
12 print("Factorial is " .. fact(n))
```



Rensselaer

Concatenation

Python

```
1 >>> 1 + 1
2 2
3 >>> "1" + "1"
4 '11'
5 >>> "1" + 1
6
7 TypeError: Can't convert 'int' object to str implicitly
8
9 >>> "1" + str(1)
10 '11'
11 >>> 1 + int("1")
12 2
```

Lua

```
? 1 > = 1 + 1
2 2
3 > = "1" + "1"
4 2
5 > = "1" + 1
6 2
7 > = 1 .. 1
8 11
9 > = "1" .. "1"
10 11
11 > = "1" .. 1
12 11
```



Math

Python

```
1 >>> 9/5
2 1.8
3 >>> math.floor(9/5)
4 1
5 >>> -1 % 5
6 4
7 >>> .1 % 5
8 0.1
```

Lua

```
1 > = 9/5
2 1.8
3 > = math.floor(9/5)
4 1
5 > = -1 % 5
6 4
7 > = .1 % 5
8 0.1
```



Comparisons

Python

```
1 >>> 5 + 6 % 7 * 2 ** 3 - 4 / 2
2 51.0
3 >>> 1 < 1
4 False
5 >>> 2 <= 2
6 True
7 >>> 3 == 3
8 True
9 >>> 4 != 4
10 False
11 >>> 5 > 5
12 False
13 >>> 6 >= 6
14 True
```

Lua

```
? 1 > = 5 + 6 % 7 * 2 ^ 3 - 4 / 2
2 51
3 > = 1 < 1
4 false
5 > = 2 <= 2
6 true
7 > = 3 == 3
8 true
9 > = 4 ~= 4
10 false
11 > = 5 > 5
12 false
13 > = 6 >= 6
14 true
```



Strings

<code>s = "there"</code>	Lua	Python
<code>t</code>	<code>string.sub(s, 1, 1)</code>	<code>s[0], s[0:1], s[:1]</code>
<code>there</code>	<code>string.sub(s, 1)</code>	<code>s[0:], s[:]</code>
<code>ther</code>	<code>string.sub(s, 1, 4)</code>	<code>s[0:4], s[:4]</code>
<code>her</code>	<code>string.sub(s, 2, -2)</code>	<code>s[1:-1]</code>
<code>here</code>	<code>string.sub(s, 2)</code>	<code>s[1:]</code>



Loops

while *exp* **do** *block* **end**

repeat *block* **until** *exp*

for *var* = *firstnum*, *lastnum* [, *step*] **do** *block* **end**



Rensselaer

Lua

```
1 > t = {1, 2, 3}
2 > a = function(x) x[1] = 10 end
3 > a(t)
4 > = t[1]
5 10
6
7 > s = "hello"
8 > a = function(x) x = "jello" end
9 > a(s)
10 > print(s)
11 Hello
12
13 > g = function() print(10) end
14 > a = function(x) x = function() print(5) end end
15 > g()
16 10
17 > a(g)
18 > g()
19 10
```



Rensselaer

Tables

Lua

```
1 > a = {1, 2, 3} -- new table created
2 > b = {1, 2, 3} -- another list-like table
3 > t[a] = 1      -- another created with one value
4 > t[b] = 2      -- 't' now has two values
5 > = t[a]        -- retrieves a value from 't'
6 1
7 > = t[b]        -- proves that 'a' and 'b' are different
8 2
9 > c = "name"
10 > d = "name"
11 > t[c] = "Andy" -- the index is the string "name"
12 > = t[d]        -- the index "name" goes with "Andy"
13 Andy
14 > = t.name      -- syntactic sugar for string indices
15 Andy
```



Rensselaer

Tables as Structure

```
point = { x = 10, y = 20 }    -- Create new table
print(point["x"])            -- Prints 10
print(point.x)               -- Has exactly the same meaning as line above
```



Rensselaer

Tables as Namespace

```
Point = {}  
Point.new = function (x, y)  
  return {x = x, y = y}  
end  
Point.set_x = function (point, x)  
  point.x = x  
end
```



pairs, ipairs

Lua

```
1 > for key,value in pairs(t) do print(key,value) end
2 3      10
3 1      3
4 4      17
5 2      7
6 pi     3.14159
7 banana yellow
```

Lua

```
1 > for index,value in ipairs(t) do print(index,value) end
2 1      3
3 2      7
4 3      10
5 4      17
```

Object-Oriented Programming

```
Vector = {}          -- Create a table to hold the class methods
function Vector:new(x, y, z) -- The constructor
    local object = { x = x, y = y, z = z }
    setmetatable(object, { __index = Vector }) -- Inheritance
    return object
end
function Vector:magnitude() -- Another member function
    -- Reference the implicit object using self
    return math.sqrt(self.x^2 + self.y^2 + self.z^2)
end

vec = Vector:new(0, 1, 0) -- Create a vector
print(vec:magnitude()) -- Call a member function using ":"
print(vec.x) -- Access a member variable using "."
```



Rensselaer

Syntax Sugar

- To declare member functions inside a prototype table, one can use
 - `table:func(args)`
 - **equivalent to function** `table.func(self, args)`
- Calling class methods also makes use of the colon:
 - `object:func(args)`
 - **equivalent to** `object.func(object, args)`



Functional Programming

Lua

```
1 > infix = function(op)
2 >>   if op == "+" then return function(a, b) return a + b end
3 >>   elseif op == "-" then return function(a, b) return a - b end
4 >>   else return function() end
5 >>   end
6 >> end
7
8 > = infix("+")(10,5)
9 15
10 > = infix("-")(10,5)
11 5
12 > = infix("?")(10,5)
13 >
```





Rensselaer

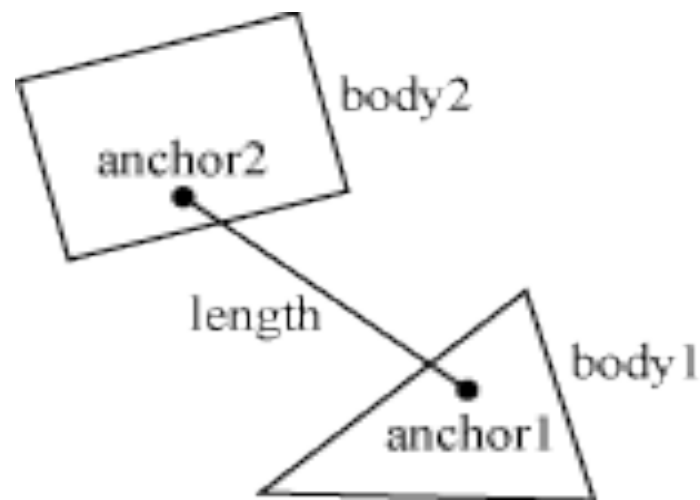
- Shape – a 2D geometrical object, such as a circle or polygon.
- (Rigid) Body – a chunk of matter that is so strong that the distance between any two bits of matter on the chunk is completely constant. They are hard like a diamond
- Fixture – binds a shape to a body and adds material properties such as density, friction, and restitution

- **Constraint** – a physical connection that removes degrees of freedom from bodies. In 2D a body has 3 degrees of freedom (two translation coordinates and one rotation coordinate). If we take a body and pin it to the wall (like a pendulum) we have constrained the body to the wall. At this point the body can only rotate about the pin, so the constraint has removed 2 degrees of freedom
- **Contact Constraint** – a special constraint designed to prevent penetration of rigid bodies and to simulate friction and restitution. You do not create contact constraints; they are created automatically by Box2D

- Joint – a constraint used to hold two or more bodies together. Box2d supports several joint types: revolute, prismatic, distance, and more. Some joints may have limits and motors
- Joint Limit – restricts the range of motion of a joint. For example, the human elbow only allows a certain range of angles
- Joint Motor – drives the motion of the connected bodies according to the joint's degrees of freedom. For example, you can use a motor to drive the rotation of an elbow

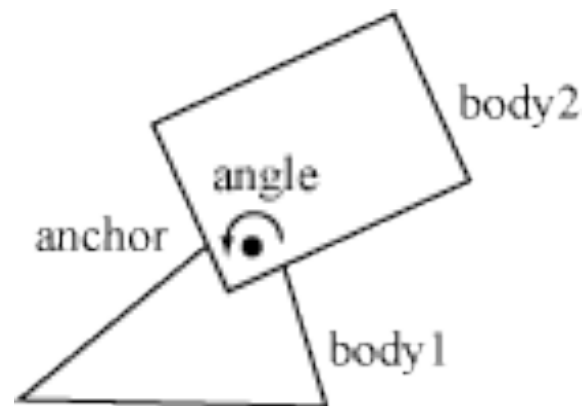
Distance Joint

Specifies that the distance between two points on two bodies must be constant. The two bodies should already be in place, then you specify the two anchor points in world coordinates. The first anchor point is connected to body 1, and the second anchor point is connected to body 2. These points imply the length of the distance constraint



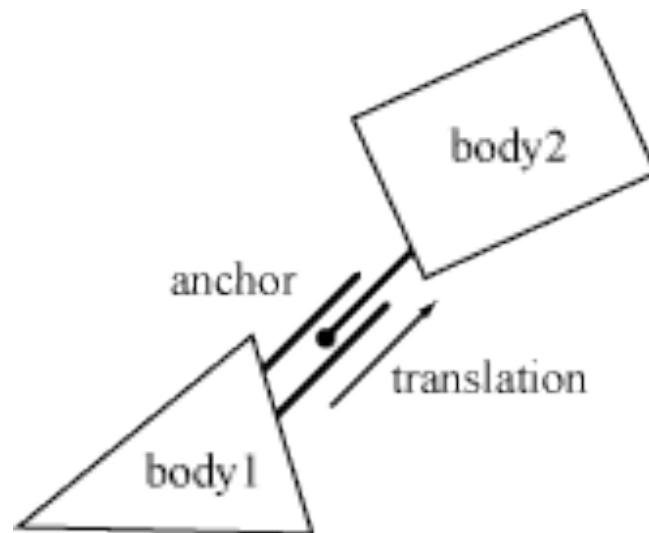
Revolute Joint

Causes two bodies to share a common anchor point, often called a hinge point. The revolute joint has a single degree of freedom: the relative rotation of the two bodies. This is called the joint angle



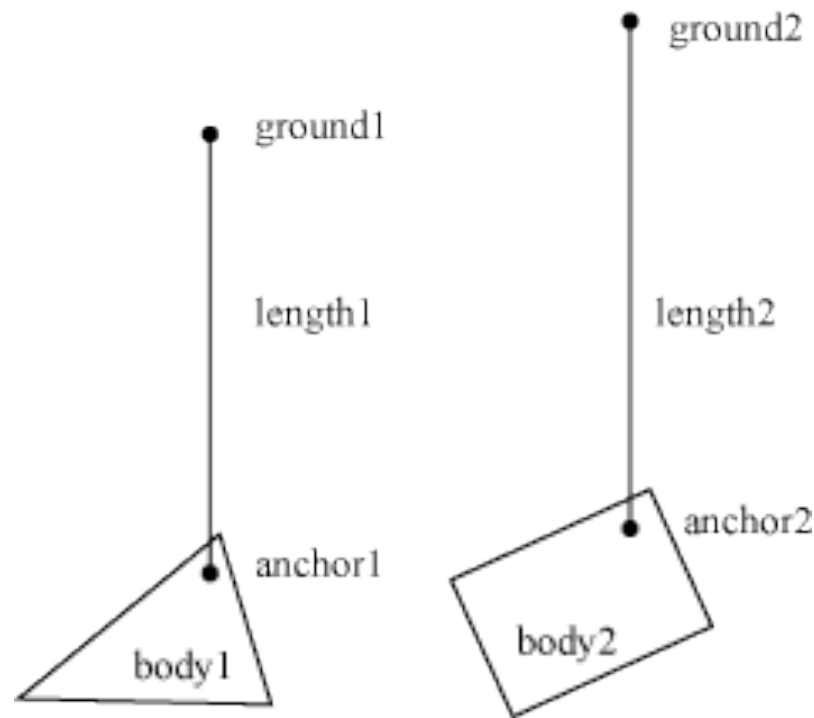
Prismatic Joint

Allows for relative translation of two bodies along a specified axis. A prismatic joint prevents relative rotation. Therefore, a prismatic joint has a single degree of freedom



Pulley Joint

A pulley joint creates an idealized pulley. The pulley connects two bodies to ground and to each other. As one body goes up, the other goes down. The total length of the pulley rope is conserved according to the initial configuration



Limitations

- Stacking heavy bodies on top of much lighter bodies is not stable. Stability degrades as the mass ratio passes 10:1.
- Polygons may not slide smoothly over chains of edge shapes or other polygon shapes that are aligned. For this reason, tile-based environments may not have smooth collision with box-like characters.
- Chains of bodies connected by joints may stretch if a lighter body is supporting a heavier body. For example, a wrecking ball connect to a chain of light weight bodies may not be stable. Stability degrades as the mass ratio passes 10:1.
- Continuous collision is processed sequentially. In a time of impact event, the body is moved back and held there for the rest of the time step. This may make fast moving objects appear to move in a non-smooth manner.

