



Indonesia
Jakarta Palembang
2018



Design Scripting Fundamentals

by Mikhael Johanes

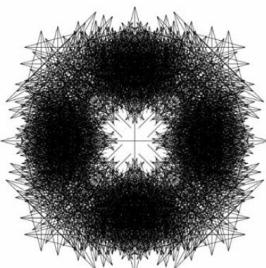
Eks Monod Diephuis & Co.
Semarang, 24 Agustus 2018



Automating routines
and repetitive activities

Why Scripting?

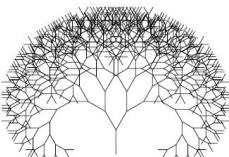
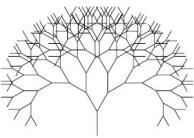
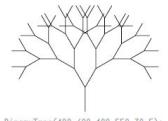
Design experimentation
by multiple versioning



Drawing with Code

The drawings shown here were created with Processing, an open-source software application. The designs are built from a binary tree, a basic data structure in which each node spawns at most two offspring. Binary trees are used to organize information hierarchies, and they often take a graphical form. The density of the final drawing depends on the angle between the "children" and the number of generations.

The larger design is created by repeating, rotating, inverting, connecting, and overlapping the tree forms. In code-based drawing, the designer varies the results by changing the inputs to the algorithm.

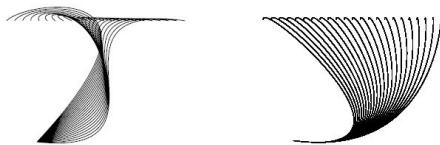


Binary Tree The drawing becomes denser with each generation. The last number in the code indicates the number of iterations. Yeohyun Ahn, MFA Studio.

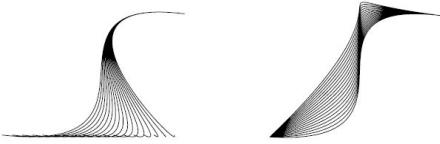
Lupton & Phillips, 2008



```
bezier(850,200,100,100,900,900,150,800);  
for(int i=0; i<900; i=i+100)  
{bezier(850,200,100,100,i,900,150,800);}
```



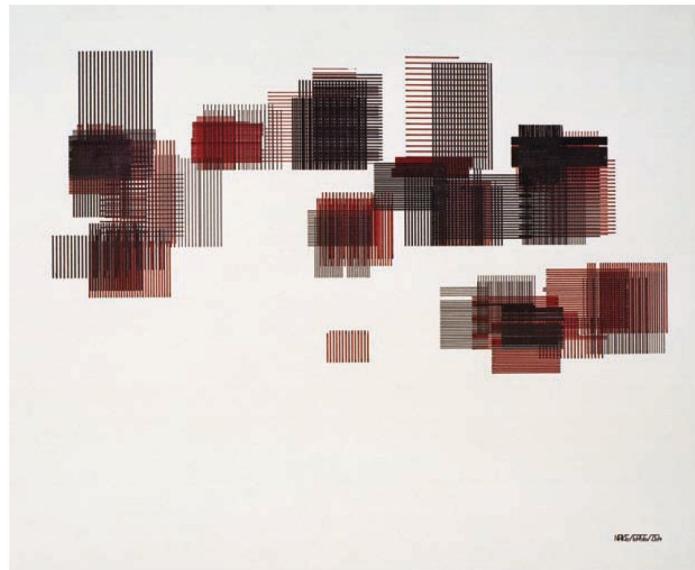
```
for(int i=0;i<900; i=i+40)  
{bezier(i,200,100,100,900,i,150,800);}  
for(int i=0;i<900; i=i+40)  
{bezier(i,200,100,100,900,i,150,800);}
```



```
for(int i=0; i<900; i=i+50)  
{bezier(900,200,100,100,900,900,i,800);}  
for(int i=0; i<900; i=i+100)  
{bezier(900,200,100,100,900,i,50,800);}
```

Repeated Bézier Curve The designer has written a function that repeats the curve in space according to a given increment (*i*). The same basic code was used to generate all the drawings shown above, with varied inputs for the anchor and control points. A variable (*i*) defines the curve. Yeohyun Ahn, MFA Studio.

Lupton & Phillips, 2008



Felder von Rechtecken,
Schraffuren Theklagert,
by Frieder Nake, 1965
For this image, Nake
used seven random
values to control

Reas, 2010

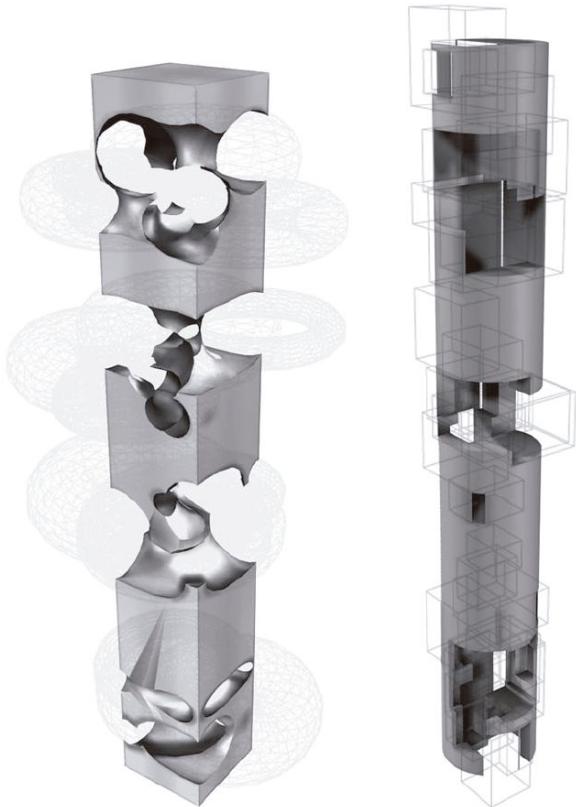


Geno Pheno Sculpture
"Fractal Dice No. 1"
by Reas, 2010

declared. "An algorithm written by the artist determines what will be shown in this new exhibition." The subsequent rolls

of a die determined the parameters of the sculpture, including its height, depth, and position of each element.

Reas, 2010



4.11

Subtracted objects are still present through their absence (class project by C. Shusta for course GSD2311 taught by Kostas Terzidis in Fall 2005 at Harvard University)

Terzidis, 2006

Script as Active Instructions

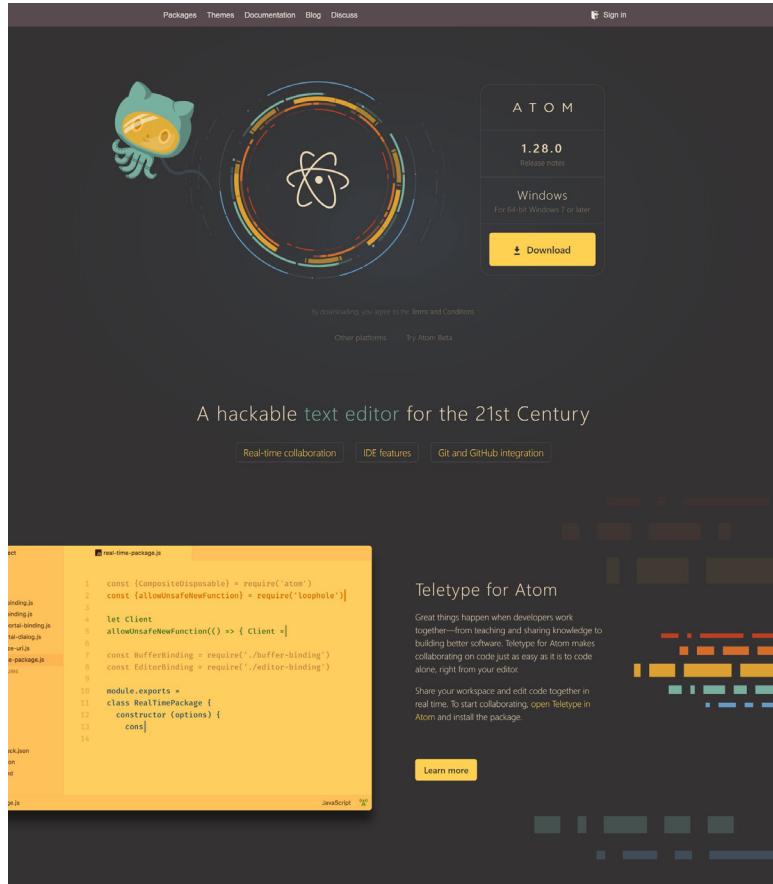
“giving specific **instructions**
to the computer with which they are
interacting”

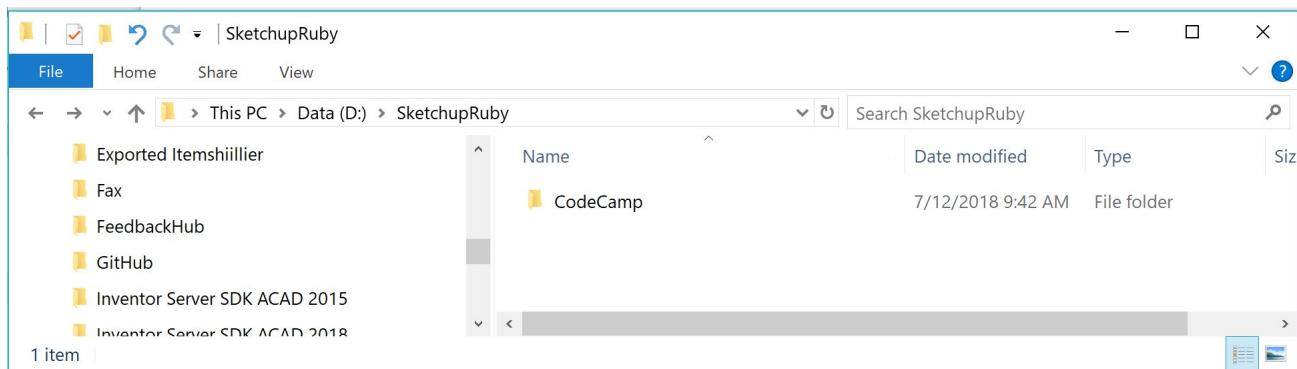
Burry, 2011 – *Scripting Cultures*

interacting

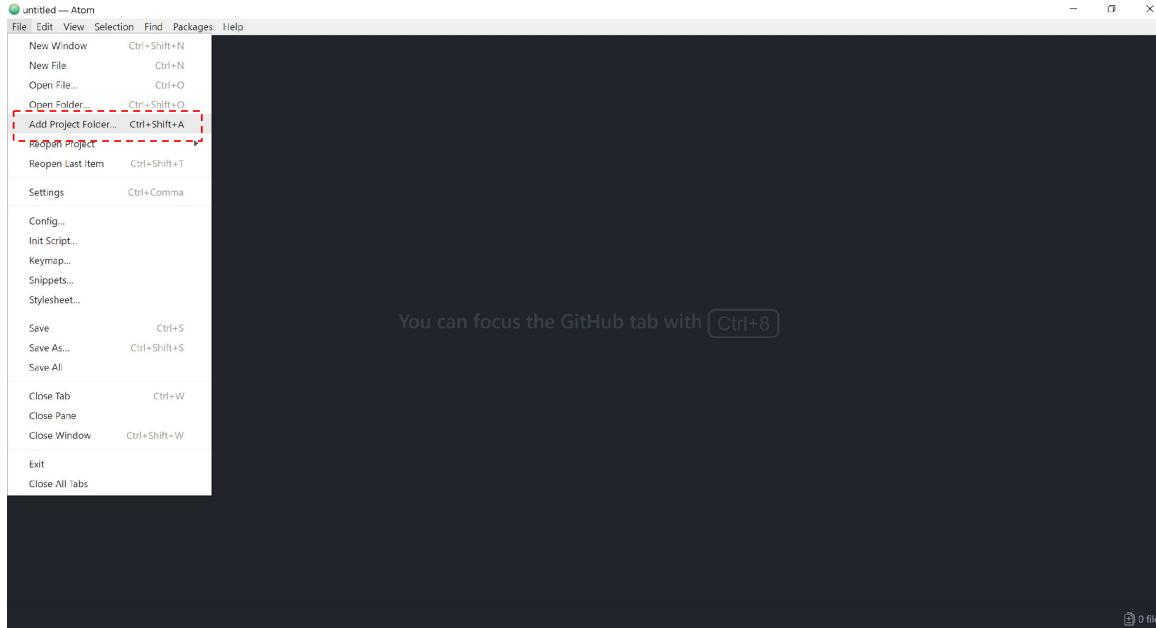
Installing Atom Code Editor

<https://atom.io/>

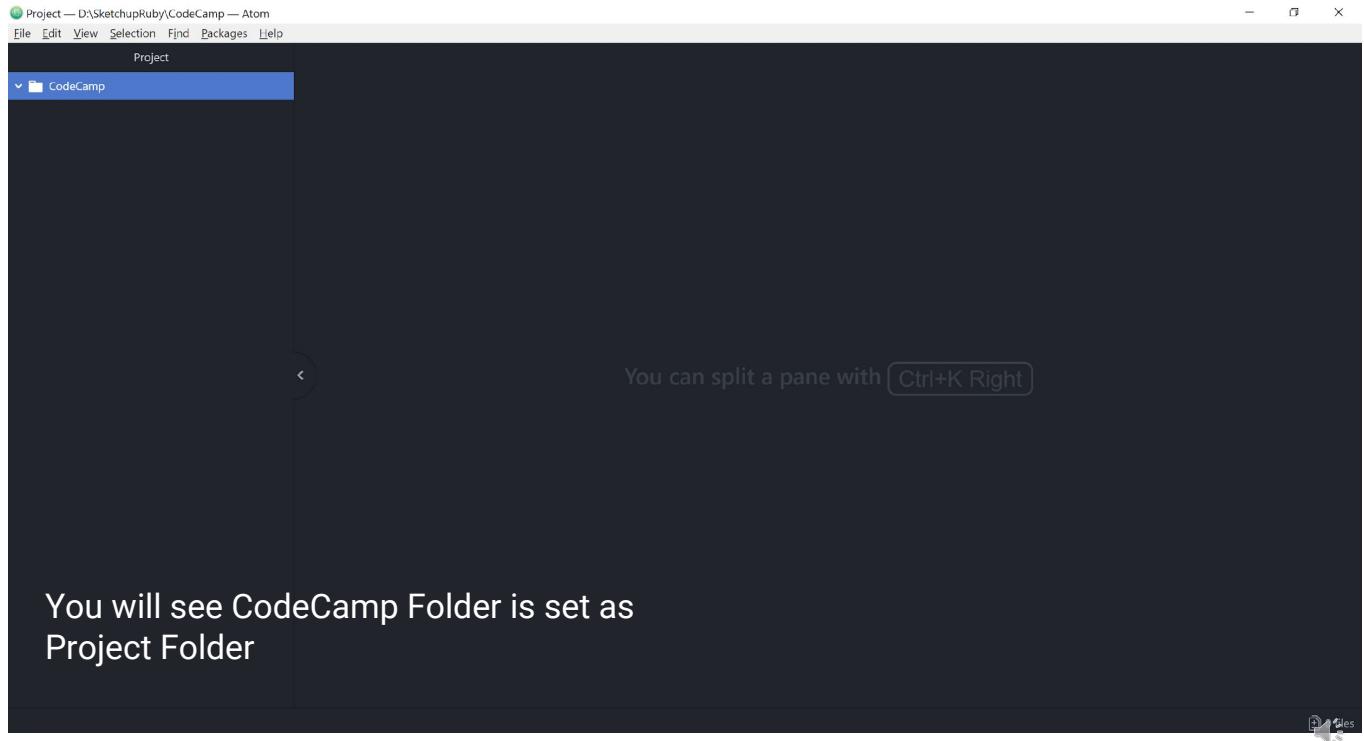




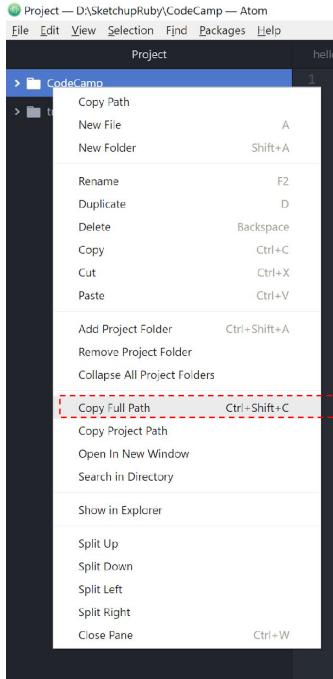
Create a “CodeCamp” folder



Set the folder as Project Folder



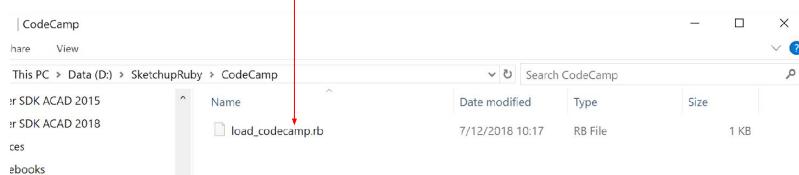
You will see CodeCamp Folder is set as
Project Folder

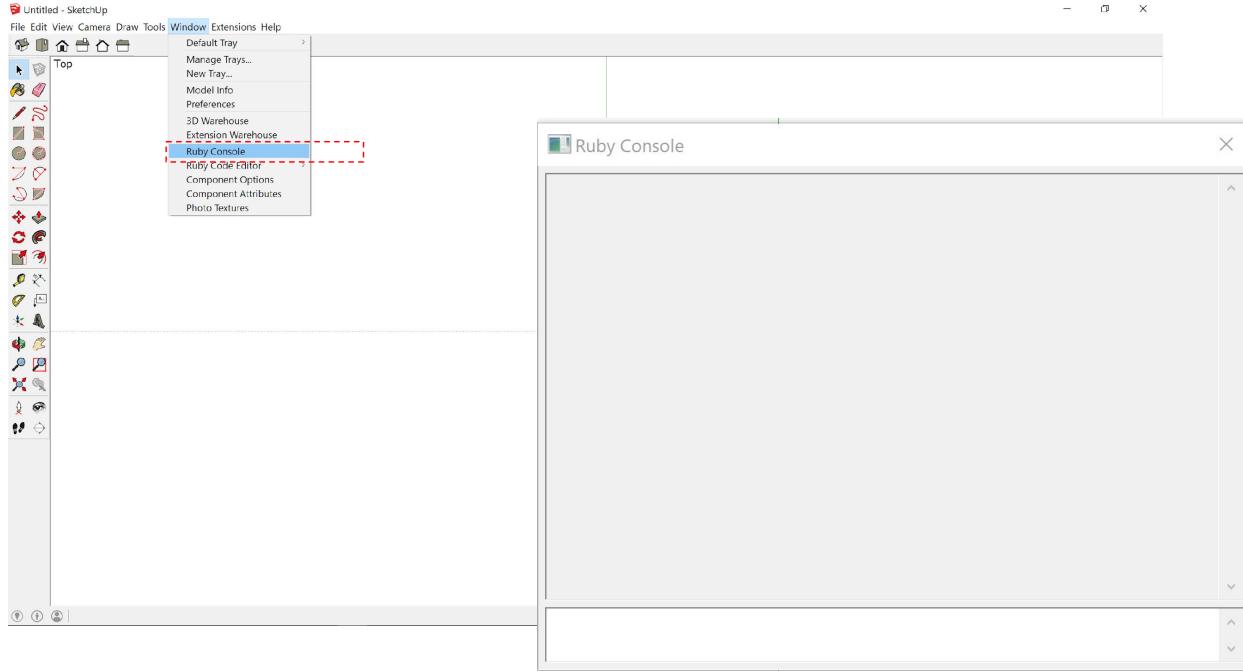


1. Create new file: File → New File
2. Right Click -> Copy Full Path on "Code Camp" folder
3. Type `$LOAD_PATH << 'paste the path here'` in the newly created file

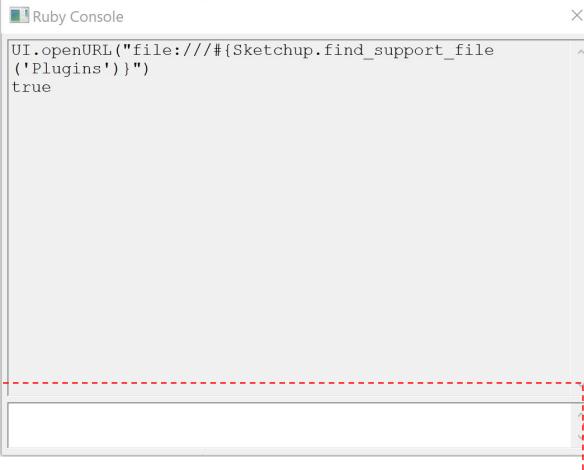
A screenshot of a code editor window titled "untitled". It contains the following code:
1 `$LOAD_PATH << 'D:\SketchupRuby\CodeCamp'`

4. Save File as "**load_codecamp.rb**"
5. You should see the "**load_codecamp.rb**" file in "CodeCamp" folder





Open the Sketchup Ruby Console



```
Ruby Console
UI.openURL("file:///#{Sketchup.find_support_file('Plugins')}")
true
```

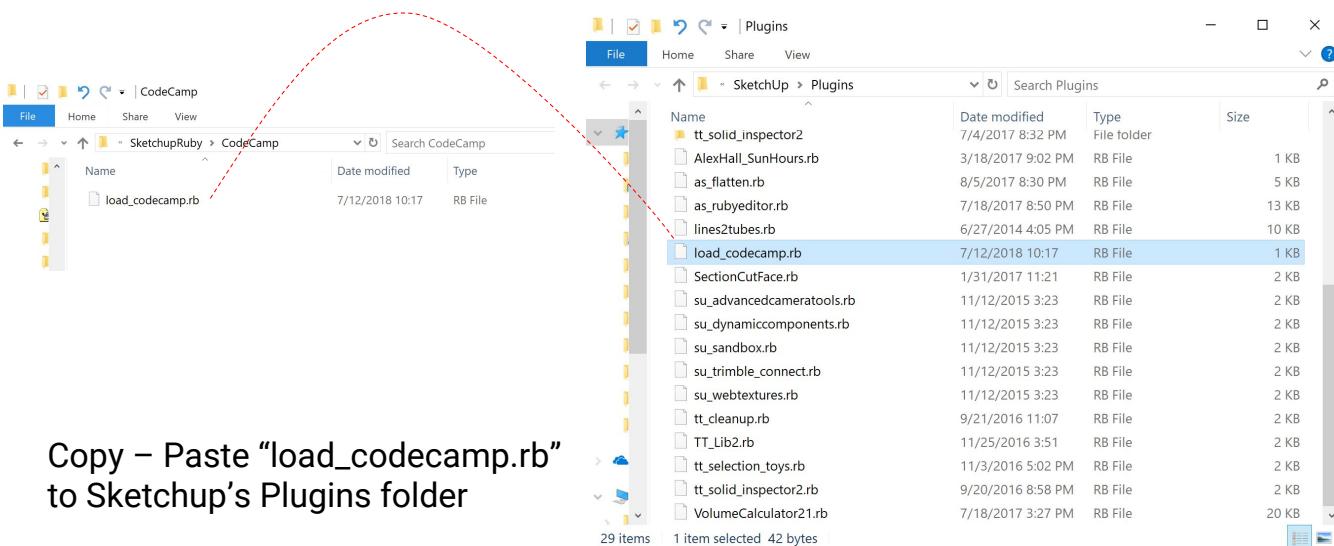
Type:

UI.openURL("file:///#{Sketchup.find_support_file('Plugins')}")

[hit Enter]

Sketchup Plugin folder will be shown afterwards

Identifying Sketchup's Plugin Folder



Copy – Paste “load_codecamp.rb”
to Sketchup’s Plugins folder

Restart your Sketchup

My First Script

1. Create a new file and save as “**helloworld.rb**”

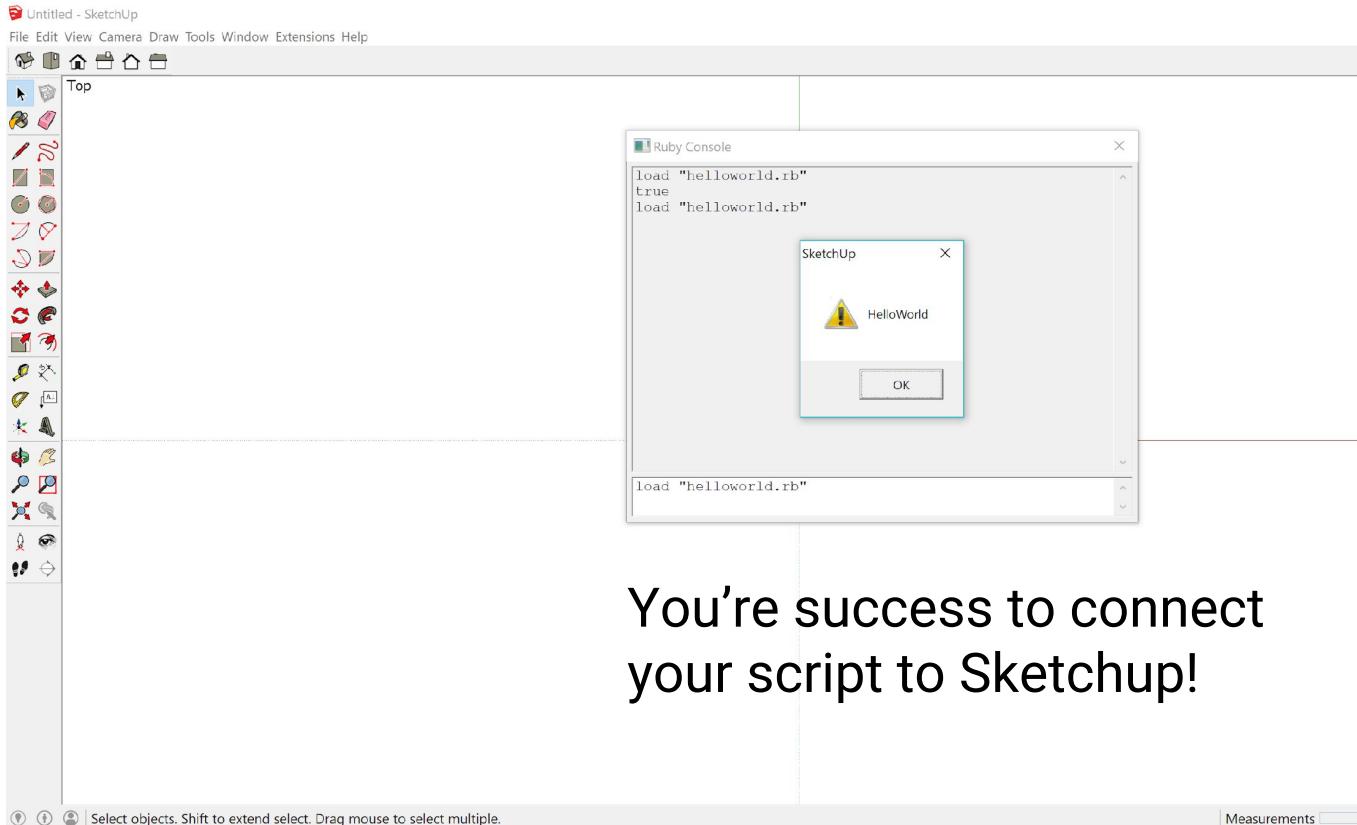
```
helloworld.rb
1 UI.messagebox("HelloWorld")
2
```

1. Open Ruby Console
2. Type load "helloworld.rb" in console



```
Ruby Console
load "helloworld.rb"
true
```

The screenshot shows a standard Windows-style window titled "Ruby Console". Inside the window, there is a text area containing the command "load \"helloworld.rb\"". Below this command, the word "true" is displayed, indicating the result of the command execution. At the bottom of the window, there is another line of text that appears to be a continuation of the command, specifically "load \"helloworld.rb\"", which is highlighted with a red dashed rectangular box. The window has standard window controls like minimize, maximize, and close buttons.



You're success to connect
your script to Sketchup!

General ruby tutorial:

https://www.tutorialspoint.com/ruby/ruby_syntax.htm

Ruby Arithmetic Operators

Operator	Description	Example
+	Addition – Adds values on either side of the operator.	$a + b$ will give 30
-	Subtraction – Subtracts right hand operand from left hand operand.	$a - b$ will give -10
*	Multiplication – Multiplies values on either side of the operator.	$a * b$ will give 200
/	Division – Divides left hand operand by right hand operand.	b / a will give 2
%	Modulus – Divides left hand operand by right hand operand and returns remainder.	$b \% a$ will give 0
**	Exponent – Performs exponential (power) calculation on operators.	$a^{**}b$ will give 10 to the power 20

https://www.tutorialspoint.com/ruby/ruby_operators.htm

Create a new file and save as “**basic_01_arithmetic.rb**”

```
1 puts "this is a text, my name is Mikhael"
2 # this is a comment, the computer wouldnt bother what you typed here
3
4 a = 10 # this is integer number
5 b = 3
6
7 c = a + b
8
9 puts c
10 puts "the value of a + b is #{c}"
11
12 puts a / b
13 puts "the value of a / b is #[a / b]"
14
15 a = 10.0 #this is float number
16 b = 3.0
17 puts
18 c = a / b
19 puts "the value of a / b now is #{c}"
20
21 puts "this is the end of my script"
22
```

Run the script in console by typing: load “basic_01_arithmetic.rb” [hit enter]
Pay attention to the results inside the console

Ruby Comparison Operators

Operator	Description	Example
<code>==</code>	Checks if the value of two operands are equal or not, if yes then condition becomes true.	<code>(a == b)</code> is not true.
<code>!=</code>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	<code>(a != b)</code> is true.
<code>></code>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	<code>(a > b)</code> is not true.
<code><</code>	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	<code>(a < b)</code> is true.
<code>>=</code>	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	<code>(a >= b)</code> is not true.
<code><=</code>	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	<code>(a <= b)</code> is true.
<code><=></code>	Combined comparison operator. Returns 0 if first operand equals second, 1 if first operand is greater than the second and -1 if first operand is less than the second.	<code>(a <=> b)</code> returns -1.
<code>==></code>	Used to test equality within a when clause of a case statement.	<code>(1...10) ==> 5</code> returns true.
<code>.eq?</code>	True if the receiver and argument have both the same type and equal values.	<code>1 ==> 1.0</code> returns true, but <code>1.eql?(1.0)</code> is false.
<code>equal?</code>	True if the receiver and argument have the same object id.	if <code>aObj</code> is duplicate of <code>bObj</code> then <code>aObj == bObj</code> is true, <code>a.equal?bObj</code> is false but <code>a.equal?aObj</code> is true.

Create a new file and save as “**basic_02_comparison.rb**”

```
1  a = 10
2  b = 3
3
4  puts( a == b) # this is a simple one
5  puts "the value of a == b is #{a == b}"  # using #{value} as place holder
6
7
8  puts "the value of a > b is #{a > b}"
9
10 puts "the value of a < b is #{a < b}"
11
12 # and soon
13
14 puts "this is the end of basic02"
15
```

Ruby Assignment Operators

Operator	Description	Example
=	Simple assignment operator, assigns values from right side operands to left side operand.	c = a + b will assign the value of a + b into c
+=	Add AND assignment operator, adds right operand to the left operand and assign the result to left operand.	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator, subtracts right operand from the left operand and assign the result to left operand.	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator, multiplies right operand with the left operand and assign the result to left operand.	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator, divides left operand with the right operand and assign the result to left operand.	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator, takes modulus using two operands and assign the result to left operand.	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator, performs exponential (power) calculation on operators and assign value to the left operand.	c **= a is equivalent to c = c ** a

Create a new file and save as “**basic_03_assignment.rb**”

```
1  a = 10
2  b = 3
3
4  c = a + b # this is easy
5  puts c
6  c = c + b
7  puts c
8  c += b #this is equivalent with c = c + b
9  puts c
10
11 # and soon
12 puts "this is the end of basic03"
13 |
```

Things to remember on defining a local variable

Always define a variable name in lowercase and use underscore (_) instead of [space]

This is allowed:

```
a = 100  
variable_a = 50  
var01 = 100  
var_one = 10.0
```

This is not allowed

```
01var = 100 (do not start your variable with number)  
Var = 50 (do not start your variable with capital letter)  
variableOne = 10.0 (this will be processed by Ruby, but camelCase style is not the convention in Ruby)
```


Creating geometries in Sketchup

Create a new file “**mygeom_00_box.rb**”

```
1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4
5 width = 100.mm
6 length = 200.mm
7 height = 300.mm
8
9 pt1 = Geom::Point3d.new(0, 0, 0)
10 pt2 = Geom::Point3d.new(width, 0, 0)
11 pt3 = Geom::Point3d.new(width, length, 0)
12 pt4 = Geom::Point3d.new(0, length, 0)
13
14 face = ent.add_face(pt1, pt2, pt3, pt4)
15
16 if face.normal.z < 0
17   face.reverse!
18 end
19
20 face.pushpull(height)
```

Save as “**mygeom_00_box.rb**” to “**exercise_00_box.rb**”

Create two boxes with:

Box 1: origin point at (0, 0, 0),

w 100 mm

l 200 mm

h 300 mm

Box 2: origin point at (100 mm, 0, 100 mm)

w 300 mm

l 400 mm

h 100 mm

Creating geometries in Sketchup using group

Save as “**mygeom_00_box.rb**” to “**mygeom _01_box_group.rb**”

```
1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4 width = 100.mm # initiating width variable
5 length = 100.mm # length variable
6 height = 150.mm # height variable
7
8 pt1 = Geom::Point3d.new(0, 0, 0) # create points for each points
9 pt2 = Geom::Point3d.new(width, 0, 0)
10 pt3 = Geom::Point3d.new(width, length, 0)
11 pt4 = Geom::Point3d.new(0, width, 0)
12
13 group = ent.add_group # create Sketchup Group in "ent"
14 group_ent = group.entities # accessing "Entities" in "group"
15 face = group_ent.add_face(pt1, pt2, pt3, pt4) # create a Face in "group_ent" then assigned to "face" variable
16 if face.normal.z < 0 # this code is used to flip the face if not upward
17   face.reverse!
18 end
19 face.pushpull(height) # push the face to create volume
20 |
```

Save as “**exercise_00_box.rb**” into “**exercise_01_box_group.rb**”

Put the two boxes into each group

Hint:

don't use single variable group for the boxes

Use variable group01 and group02 instead

Creating a box using single variable

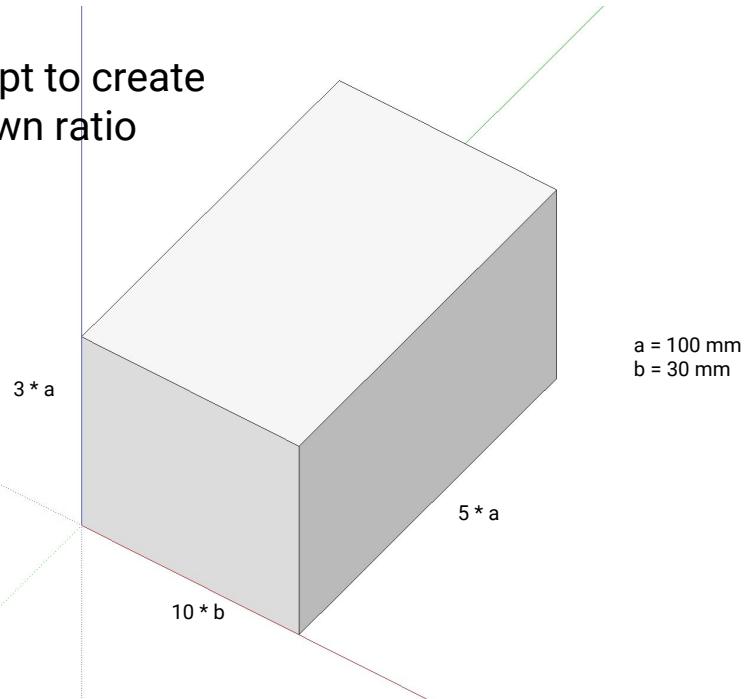
Save as “**mygeom_01_box_group.rb**” to “**mygeom_02_box.rb**”

```
1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4
5 a = 100.mm
6 width = a
7 length = a * 2
8 height = a * 3
9
10 pt1 = Geom::Point3d.new(0, 0, 0)
11 pt2 = Geom::Point3d.new(width, 0, 0)
12 pt3 = Geom::Point3d.new(width, length, 0)
13 pt4 = Geom::Point3d.new(0, length, 0)
14
15 group = ent.add_group
16 group_ent = group.entities
17 face = group_ent.add_face(pt1, pt2, pt3, pt4)
18 if face.normal.z < 0 # this code is used to flip the face if not upward
19   face.reverse!
20 end
21 face.pushpull(height)
22
```

Set a = 100.mm as a value to define the dimension of the box

Save as “**mygeom_02_box.rb** ” to “**exercise_02_box.rb**”

Modify the script to create
a box with shown ratio



Creating box using origin point

Save as “**mygeom_02_box.rb**” to “**mygeom_03_box.rb**”

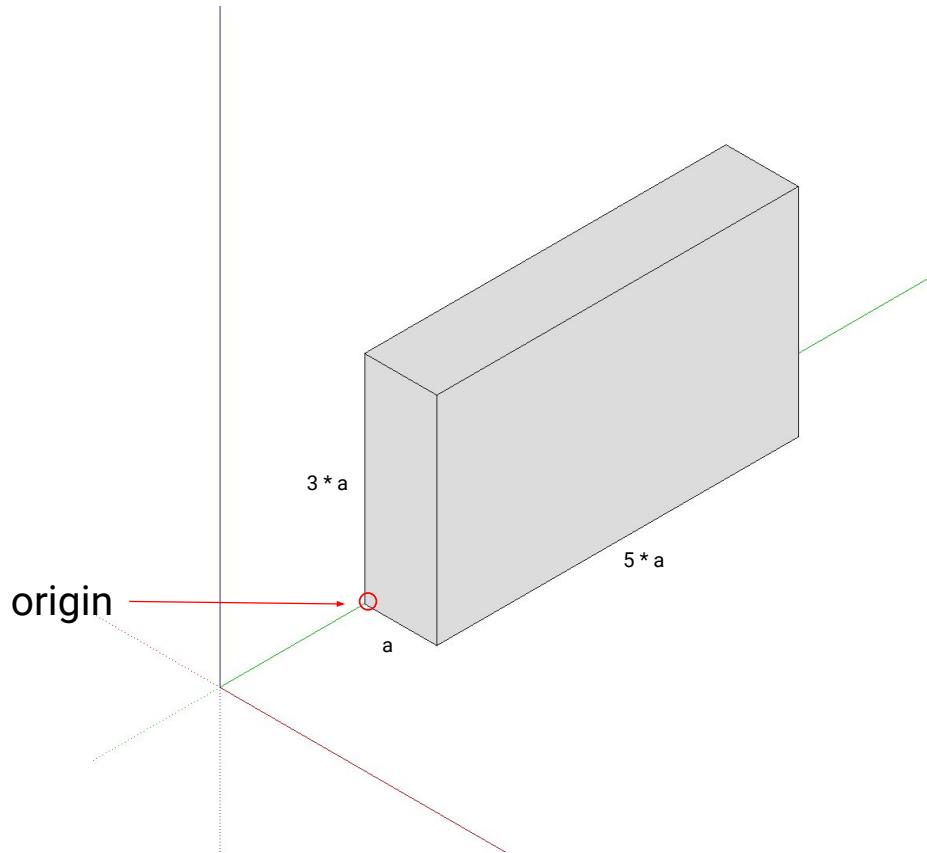
```
1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4 origin = Geom::Point3d.new(100.mm, 100.mm, 100.mm)
5 a = 100.mm
6 width = a
7 length = a * 5
8 height = a * 3
9
10 pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
11 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
12 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
13 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
14
15 group = ent.add_group
16 group_ent = group.entities
17 face = group_ent.add_face(pt1, pt2, pt3, pt4)
18 if face.normal.z < 0 # this code is used to flip the face if not upward
19   face.reverse!
20 end
21 face.pushpull(height)
22
```

Set a Point3d as origin of the box

Geom::Point3d(x, y, z) is used to create a Point3d object, and assigned to origin

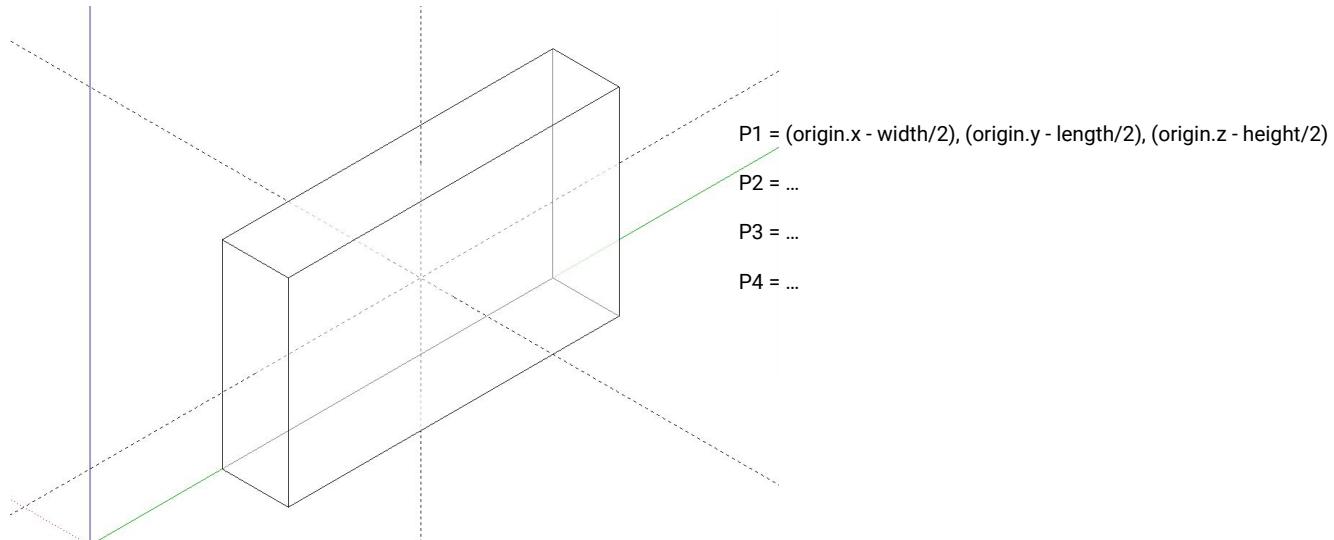
The origin point components (x, y, z) can be retrieved one by one by using:

x = Point3d.x
y = Point3d.y
z = Point3d.z



Save as “**mygeom_03_box.rb**” to “**exercise_03_box.rb**”

Create a box with origin point at the center of the box



Repetitions, Loops, and Iterations

```
1
2 v for i in (0..9)
3     puts i
4 end
5 |
```

Save as “**basic_04_for_loop.rb**”

https://www.tutorialspoint.com/ruby/ruby_loops.htm

Save as “**exercise_basic_loop.rb**”

Create a loop that shows this in your Sketchup Ruby Console:

```
50  
60  
70  
80  
90  
. . .  
150
```

Hint:

create a loop from 5..15
multiply it by 10

Creating a series of boxes using loop

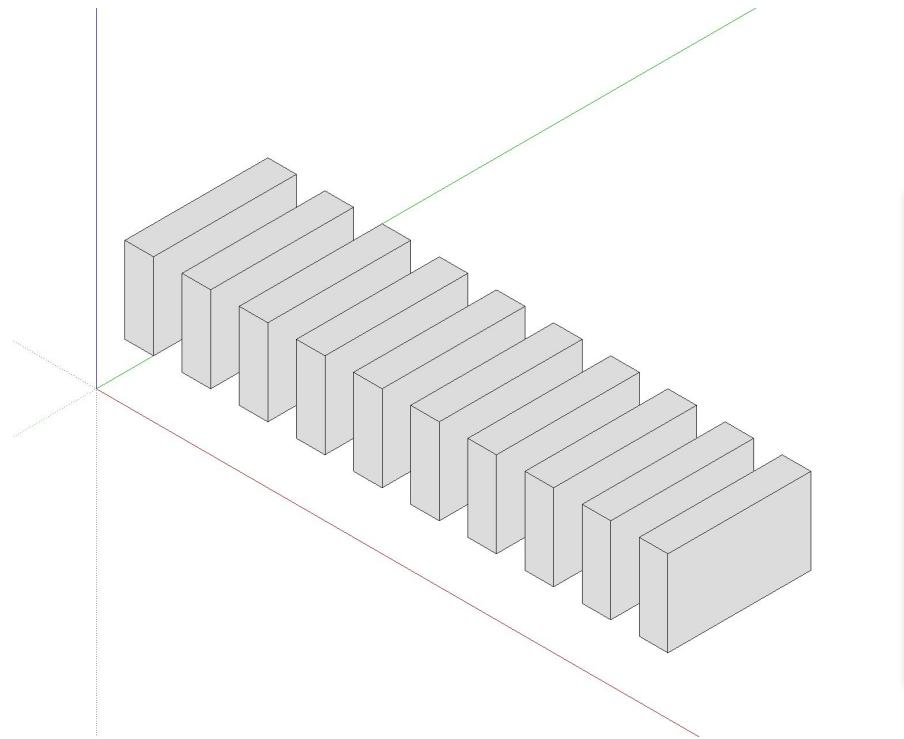
```
1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4
5 for i in 0..9
6   origin = Geom::Point3d.new(i * 200.mm, 100.mm, 100.mm)
7   a = 100.mm
8   width = a
9   length = a * 5
10  height = a * 3
11
12  pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
13  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
14  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
15  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
16
17  group = ent.add_group
18  group_ent = group.entities
19  face = group_ent.add_face(pt1, pt2, pt3, pt4)
20  if face.normal.z < 0 # this code is used to flip the face if not upward
21    face.reverse!
22  end
23  face.pushpull(height)
24 end
25
```

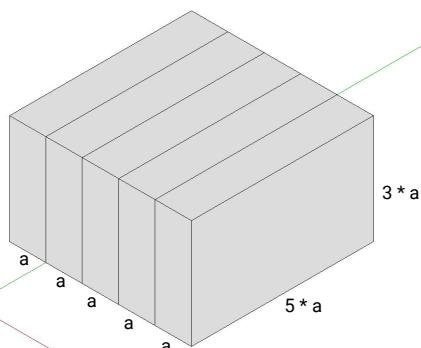
Save as

"mygeom_03_box.rb" to
"mygeom_04_box_series.rb"

Put the whole script within the loop
i from 0..9

Set the x component of origin point
to :i * 200.mm





Save as
"mygeom_04_box_series.rb"
to
"exercise_04_box_series.rb"

Try to create :
5 boxes with distance that
correspond to the width a
of the box

Randomise

rand → random float between 0 and 1

rand(9..20) → random integer in 9..20

rand(1.4..2.5) → random float between 1.4 and 2.5

Create a new file as “**basic_05_random.rb**”

```
1 puts "this is 10 random decimal number"
2 for i in 0..9
3     random = rand
4     puts(random)
5 end
6
7 puts "this is 10 random integer number from 1..15"
8 for i in 0..9
9     random = rand(1..15)
10    puts(random)
11 end
12
13 puts "this is random decimal number from 1.5..2.0"
14 for i in 0..9
15     random = rand(1.5..2.0)
16     puts(random)
17 end|
18
```

Boxes with random value

```
1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4
5 for i in 0..9
6   origin = Geom::Point3d.new(i * 200.mm, 100.mm, 100.mm)
7   a = 100.mm
8   width = a
9   length = a * 5
10  random_int = rand(1..10) # create random int number from 1..10
11  height = a * random_int # put random number as height multiplier|
12
13  pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
14  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
15  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
16  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
17
18  group = ent.add_group
19  group_ent = group.entities
20  face = group_ent.add_face(pt1, pt2, pt3, pt4)
21  if face.normal.z < 0 # this code is used to flip the face if not upward
22    face.reverse!
23  end
24  face.pushpull(height)
25
26
```

Save as

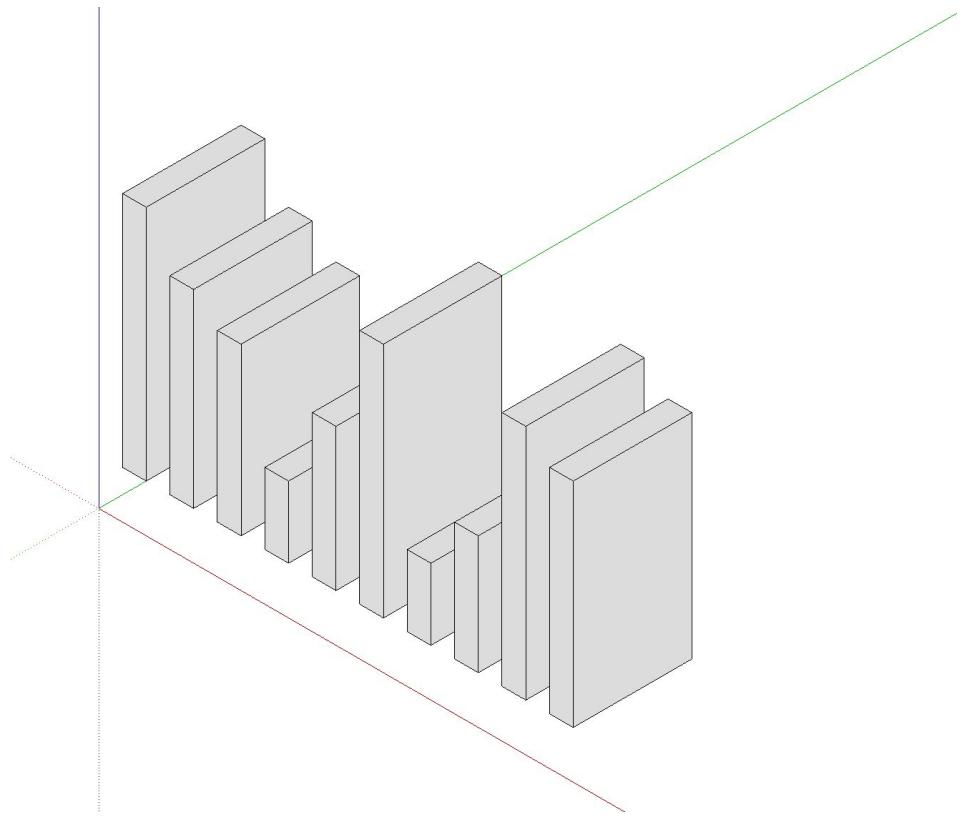
"mygeom_04_box_series.rb"

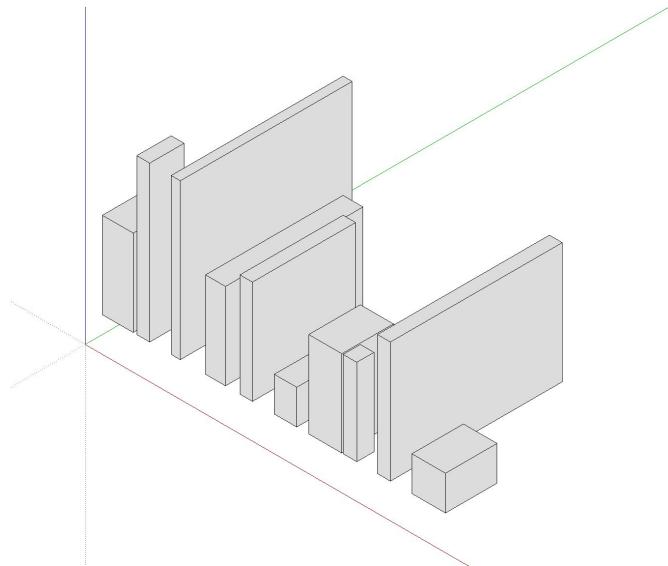
to

"mygeom_05_box_random.rb"

Assign random_int with
random integer from 1..10

Put height variable to
correspond with random
integer random_int





Save as

"mygeom_05_box_random.rb"
to **"exercise_05_rand_size.rb"**

Random width from
0.5 to 2.0

Random height from
1..10

Random length from
1..10

```

1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4
5 for i in 0..9
6   a = 100.mm
7   width = a
8   length = a
9   height = a
10
11  # create random (x, y) position
12  random_x = (rand(1..10) * 10 * a).mm
13  random_y = (rand(1..10) * 10 * a).mm
14  origin = Geom::Point3d.new(random_x, random_y, 0)
15
16  pt1 = Geom::Point3d.new(origin.x - width/2 , origin.y - length/2, origin.z - height/2)
17  pt2 = Geom::Point3d.new(origin.x + width/2, origin.y - length/2, origin.z - height/2)
18  pt3 = Geom::Point3d.new(origin.x + width/2, origin.y + length/2, origin.z - height/2)
19  pt4 = Geom::Point3d.new(origin.x - width/2, origin.y + length/2, origin.z - height/2)
20
21  group = ent.add_group
22  group_ent = group.entities
23  face = group_ent.add_face(pt1, pt2, pt3, pt4)
24
25  if face.normal.z < 0 # this code is used to flip the face if not upward
26    face.reverse!
27  end
28
29  face.pushpull(height)
30 end
31

```

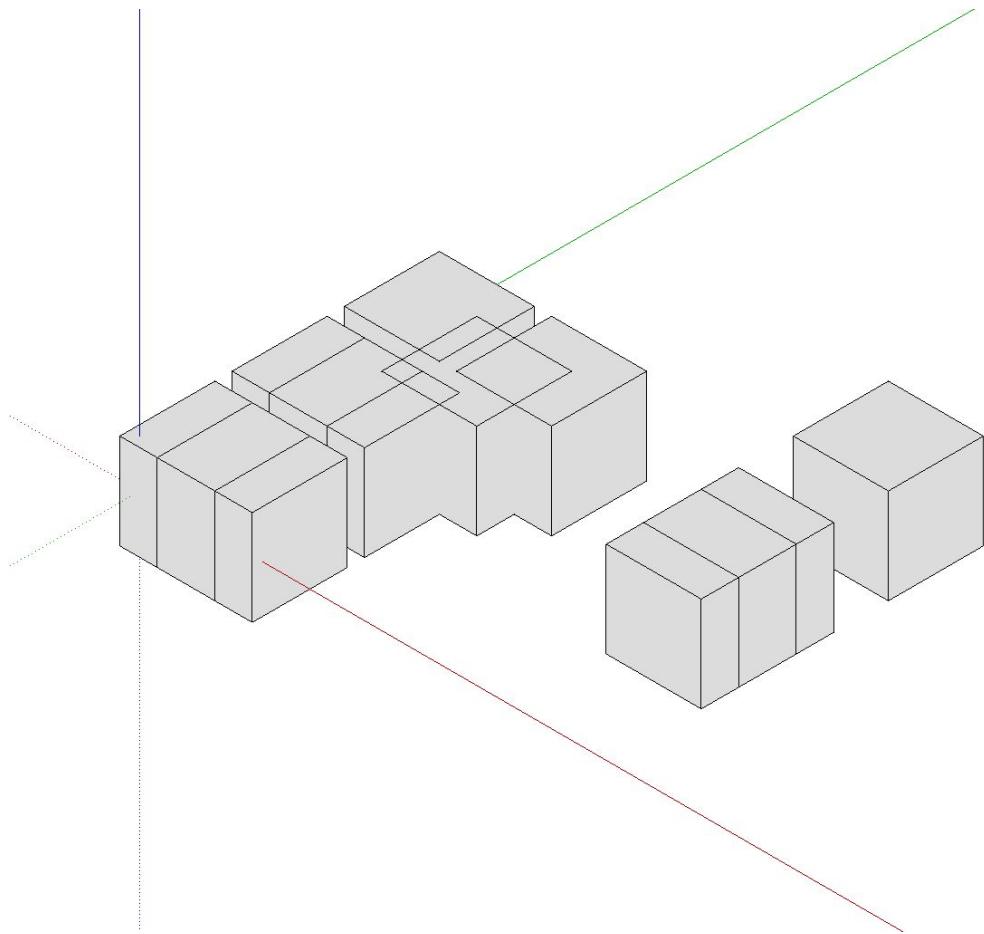
Save as “**02_box_from_center.rb**”
 (box with center origin point)
 to “**exercise_05_rand_pos.rb**”

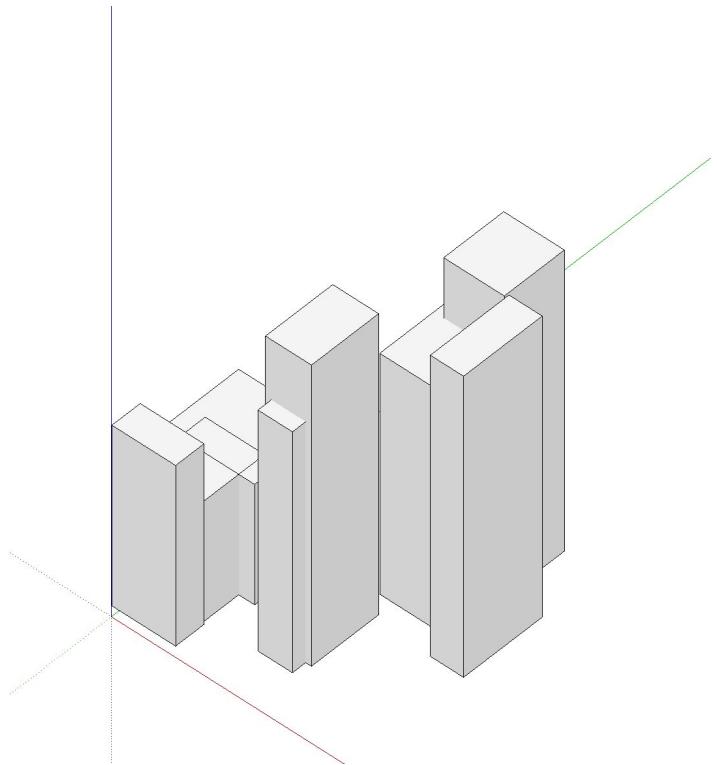
Set the box ratio to 1:1:1 with
 $a = 100.mm$

Modify it into loop of 10 boxes

Put random x and y position to
 origin point

$\text{random_x} = (\text{random}(1..10) * 10 * a).\text{mm}$
 $\text{random_y} = (\text{random}(1..10) * 10 * a).\text{mm}$



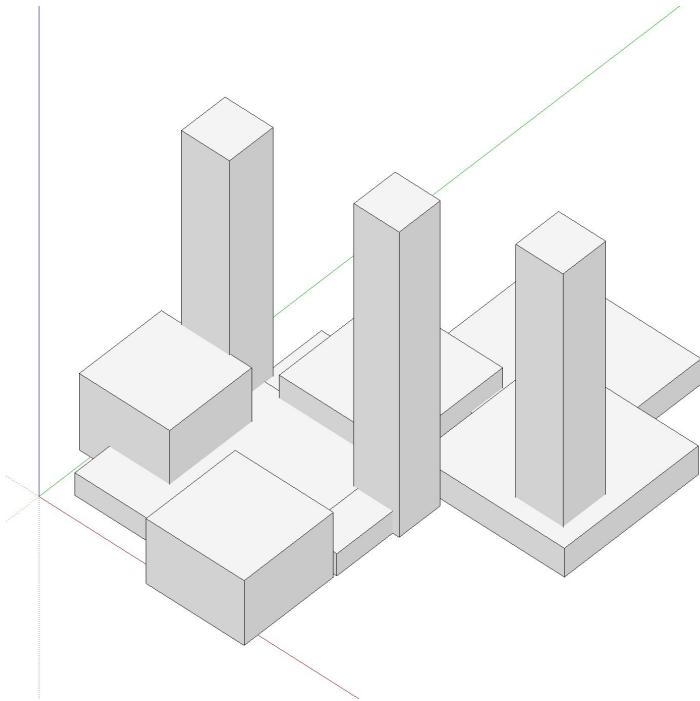


Save as "**02_box_from_center.rb**"
(box with center origin point)
to "**exercise_05_rand_pos_size.rb**"

```
# set the origin point to center bottom of the box  
pt1 = Geom::Point3d.new(..., ..., origin.z)  
pt2 = ...  
pt3 = ...  
pt4 = ...
```

set random width, length and height

```
width = a * rand(0.5..1.5)  
length = a * rand(0.5..1.5)  
height = a * rand(1..5)
```



Save as “**02_box_from_center.rb**”
(box with center origin point)
to “**exercise_05_rand_vol_constant.rb**”

#set random rectangular base

...

random_int = rand(0.5..2.0)

width = a * random_int

length = a * random_int

#set the volume of the box constant

height = (a * a * a) / (width * height)

Exercise yourself with your own idea!

exercise_05_your_idea.rb

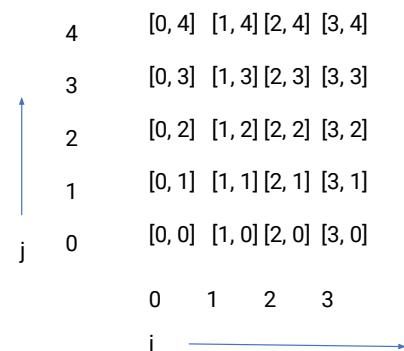
Printscreen your results and save as

exercise_05_your_idea.jpg

2 dimensional grid: loop in loop

```
1 for i in 0..3
2   puts "-----"
3   puts "the value of i is #{i}"
4   for j in 0..4
5     puts "the value of j is #{j}"
6     puts "the value of [i,j] is [#{i},#{j}]"
7   end
8 end
9
```

Save as
"basic_06_loop_in_loop.rb"



Grid of boxes

```
1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 for i in 0..9
5   for j in 0..5 # create inner loop with j counter
6     a = 100.mm
7     width = a #set the ratio to 1:1:1
8     length = a
9     height = a
10
11   #set the j to correspond with y coordinate
12   origin = Geom::Point3d.new(i * width, j * length , 0)
13
14   pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
15   pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
16   pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
17   pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
18
19   group = ent.add_group
20   group_ent = group.entities
21   face = group_ent.add_face(pt1, pt2, pt3, pt4)
22
23   if face.normal.z < 0
24     face.reverse!
25   end
26
27   face.pushpull(height)
28 end
29 end
30
```

Start from

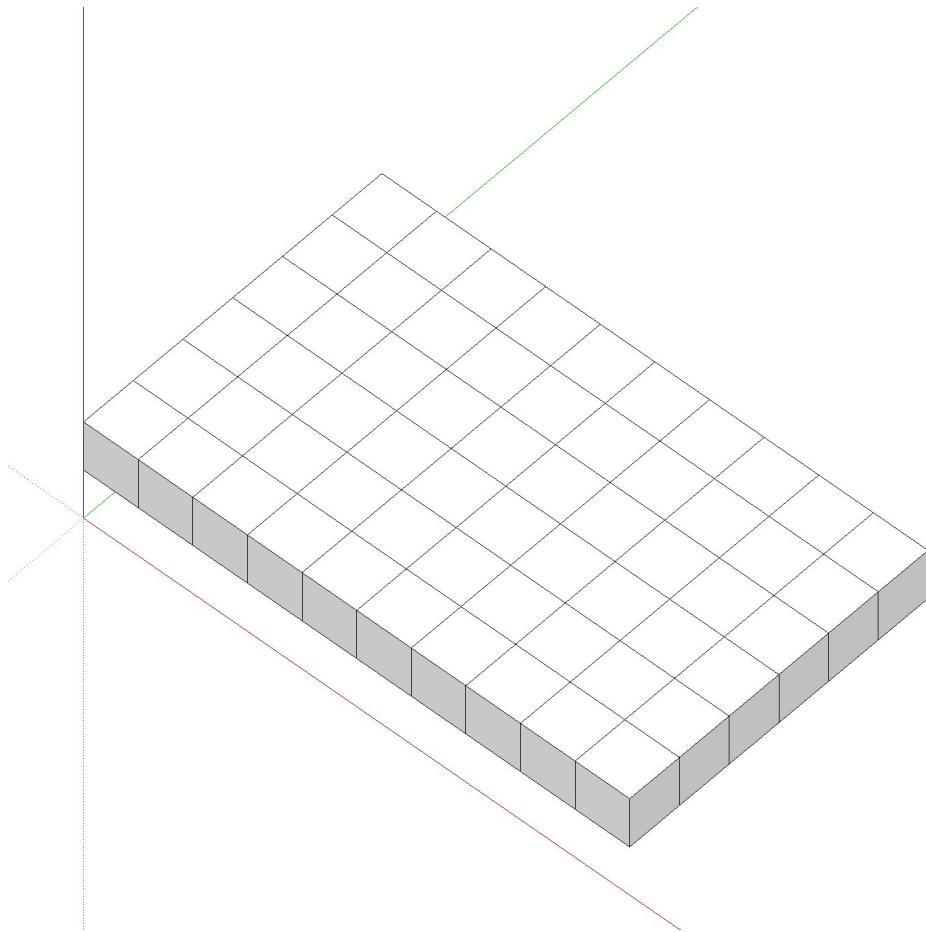
“**mygeom_04_box_series.rb**” (series of boxes),

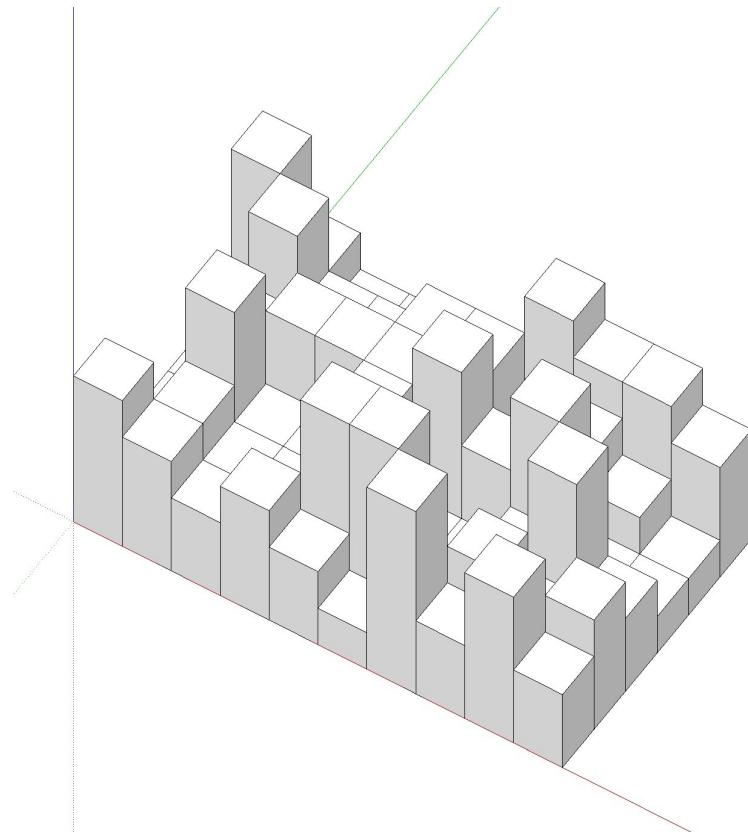
save as “**mygeom_06_grid.rb**”

Set the box ratio to 1:1:1

Create a loop within existing loop

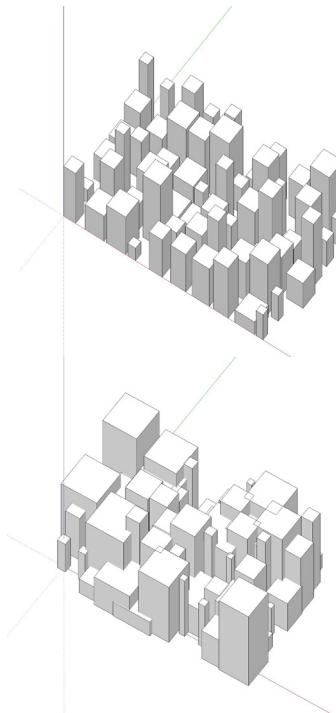
Set the origin point of the boxes to correspond with i and j value





Put random value on its height

save as "**exercise_06_grid_rand_height.rb**"



Exercise yourself with
your own idea!

exercise_06_your_idea.rb

Printscreen your results and save as

exercise_06_your_idea.jpg


```

1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3 for k in 0..6
4   for j in 0..5
5     for i in 0..4
6       a = 25.mm
7       width = a * rand(1..4)
8       length = a * rand(1..4)
9       height = a * rand(1..4)
10
11      x = a * rand(0..4) + j * 1000.mm # add 1000.m grid
12      y = a * rand(0..4) + k * 1000.mm # add 1000.m grid
13      z = a * rand(0..4)
14
15      origin = Geom::Point3d.new(x, y, z)
16
17      pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
18      pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
19      pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
20      pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
21
22      group = ent.add_group
23      group_ent = group.entities
24      face = group_ent.add_face(pt1, pt2, pt3, pt4)
25
26      if face.normal.z < 0
27        face.reverse!
28      end
29      face.pushpull(height)
30    end
31  end
32 end

```

Generating choices

Start from "**08_random_box_composition.rb**"

Save as "**mygeom_07_box_compositions.rb**"

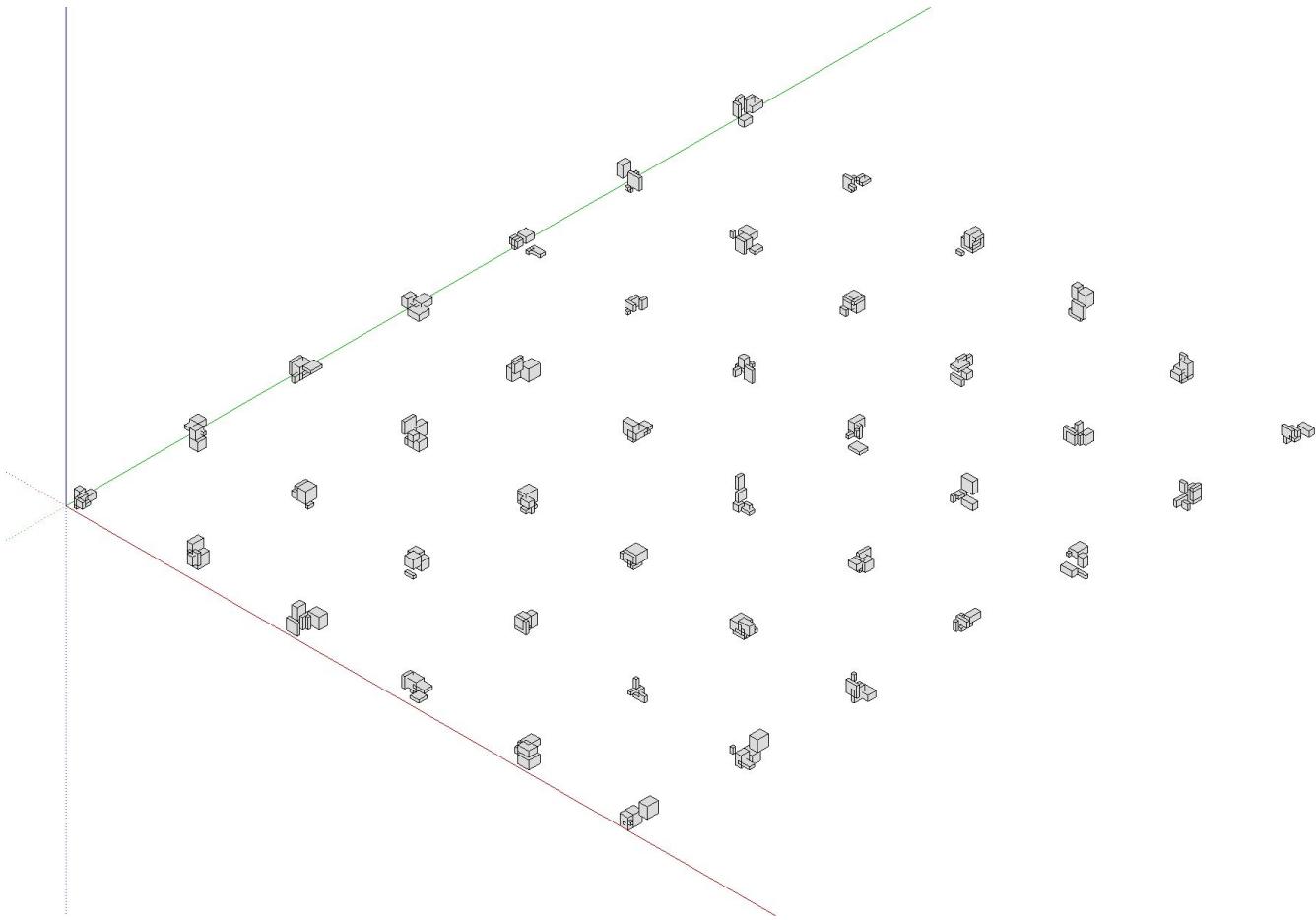
Create two loop j and k outside the existing loop i

Set the j and k to correspond with

x and y components of boxes' origin point

By adding i * 1000.mm and j * 1000.mm

to the origin point definition



Exercise yourself with
your own idea!

Save as “**exercise_07_your_idea.rb**”

Transformation

Translation (move)

```
tr = Geom::Transformation.translation(vector)  
tr = Geom::Transformation.translation(point)
```

Rotation

```
tr = Geom::Transformation.rotation(point, vector, angle)
```

Scale

```
tr = Geom::Transformation.scaling(point, scale)  
tr = Geom::Transformation.scaling(point, xscale, yscale, zscale)
```

```

1 mod = Sketchup.active_model # Open model
2 ent = mod.entities # All entities in model
3
4 origin = Geom::Point3d.new(0.mm, 0.mm, 0.mm)
5 a = 100.mm
6 width = a
7 length = a * 5
8 height = a * 3
9
10 pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
11 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
12 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
13 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
14
15 group = ent.add_group
16 group_ent = group.entities
17 face = group_ent.add_face(pt1, pt2, pt3, pt4)
18 if face.normal.z < 0 # this code is used to flip the face if not upward
19   face.reverse!
20 end
21 face.pushpull(height)
22
23 group_copy = group.copy #copy the group to group_copy
24 vector_tr = Geom::Vector3d.new(2 * a, 0, 0) #set the vector according to a in x axis
25 tr = Geom::Transformation.translation(vector_tr) # create a translation base on vector
26 group_copy.transform!(tr) # apply the translation to the object
27

```

Translation

Start from

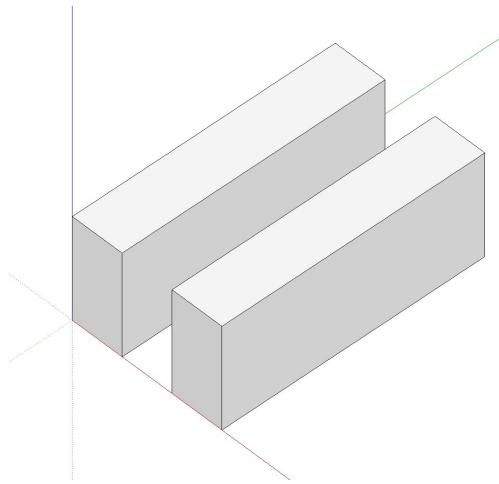
"mygeom_03_box.rb" (making a group of solid box)

Save as

"mygeom_08_trans.rb"

Set the ratio of the box accordingly

Copy the group and apply translation to the copied group

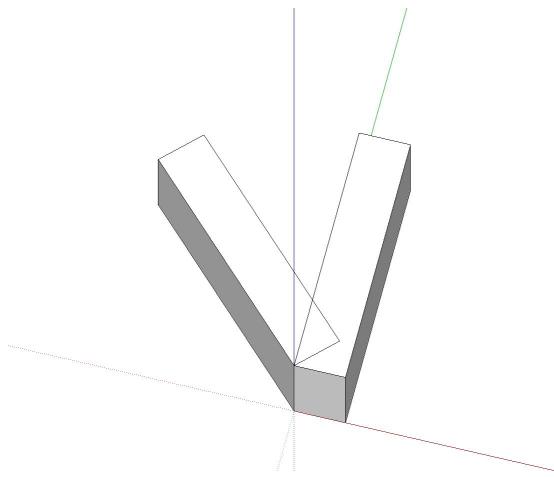


Rotation

```
23 group_copy = group.copy #copy the group to group_copy
24
25 axis = Geom::Vector3d.new(0, 0, 1) # set z axis for rotation axis
26 angle = 0.25 * Math::PI # set the angle as 1/4 PI rad or 45 degree
27 # note that ruby work in radians instead of degree
28 rotation_point = origin # set origin above as rotation point
29 tr = Geom::Transformation.rotation(rotation_point, axis, angle)
30 group_copy.transform!(tr) # apply the rotation to the object
```

Start from
“**mygeom_08_trans.rb**”
Save as
“**mygeom_09_trans.rb**”

Remove the translation
portion from the script and
add rotation
transformation

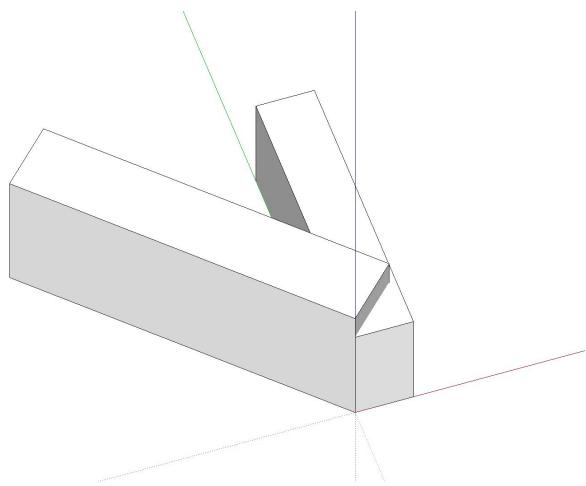


Scaling

```
23 group_copy = group.copy #copy the group to group_copy
24
25 axis = Geom::Vector3d.new(0, 0, 1) # set z axis for rotation axis
26 angle = 0.25 * Math::PI # set the angle as 1/4 PI rad or 45 degree
27 # note that ruby work in radians instead of degree
28 rotation_point = origin # set origin above as rotation point
29 tr = Geom::Transformation.rotation(rotation_point, axis, angle)
30 group_copy.transform!(tr) # apply the translation to the object
31 scaling_point = origin
32 tr_scale = Geom::Transformation.scaling(scaling_point, 1.25) # set scale Transformation
33 group_copy.transform!(tr_scale) # apply the scaling to the object|
```

Start from
"mygeom_09_trans.rb"
Save as
"mygeom_10_trans.rb"

Add scale transformation
to the rotated group



```

4
5   for i in 0..9
6     a = 100.mm
7     width = a
8     length = a * 5
9     height = a * 3
10    origin = Geom::Point3d.new(0, 0, i * height) # the origin z is set according
11                      # height and counter i
12
13    pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
14    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
15    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
16    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
17
18    group = ent.add_group
19    group_ent = group.entities
20    face = group_ent.add_face(pt1, pt2, pt3, pt4)
21    if face.normal.z < 0 # this code is used to flip the face if not upward
22      face.reverse!
23    end
24    face.pushpull(height)
25    #create rotation Transformation
26    axis = Geom::Vector3d.new(0, 0, 1) #z axis
27    point = origin # same as origin point
28    angle = Math::PI * 0.25 * (i / 9.0) #incrementally according to i
29    tr = Geom::Transformation.rotation(point, axis, angle) # create rotation
30    group.transform!(tr) #apply rotation
31  end
32

```

Start from

"mygeom_04_box_series.rb"

Save as

"mygeom_11_series_rotate.rb"

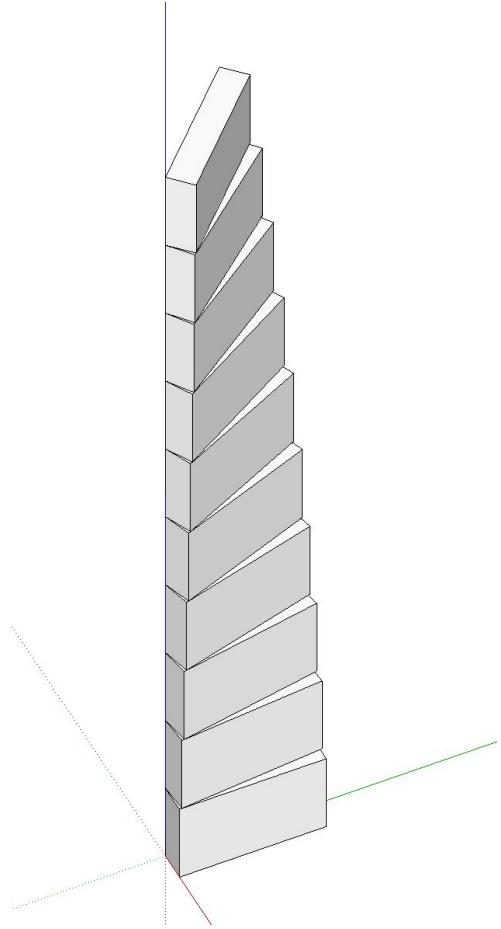
Create a series of box in z axis

Set the ratio accordingly

Apply rotation incrementally
from the bottom:

Remember:

Full rotation = $2 * \text{Math}::\text{Plaa}$



```

3
4   for j in 0..9
5     for i in 0..9
6       a = 100.mm
7       width = a * 0.2
8       length = a * 5
9       height = a * 3
10      origin = Geom::Point3d.new(j * a * 3, 0, i * height) # the origin z is set according
11          # height and counter i
12          # set the origin z to correspond
13          # with j in x axis
14
15      pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
16      pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
17      pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
18      pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
19
20      group = ent.add_group
21      group_ent = group.entities
22      face = group_ent.add_face(pt1, pt2, pt3, pt4)
23      if face.normal.z < 0 # this code is used to flip the face if not upward
24        face.reverse!
25      end
26      face.pushpull(height)
27      #create rotation Transformation
28      axis = Geom::Vector3d.new(0, 0, 1) #z axis
29      point = origin # same as origin point
30      angle = Math::PI * 0.5 * ((i * j) / 81.0) #incrementally according to i * j
31      tr = Geom::Transformation.rotation(point, axis, angle) # create rotation
32      group.transform!(tr) #apply rotation
33    end
34  end
35

```

Start from

"mygeom_11_series_rotate.rb"

Save as "**"mygeom_12_grid_rotate.rb"**"

Create a series of box in z axis

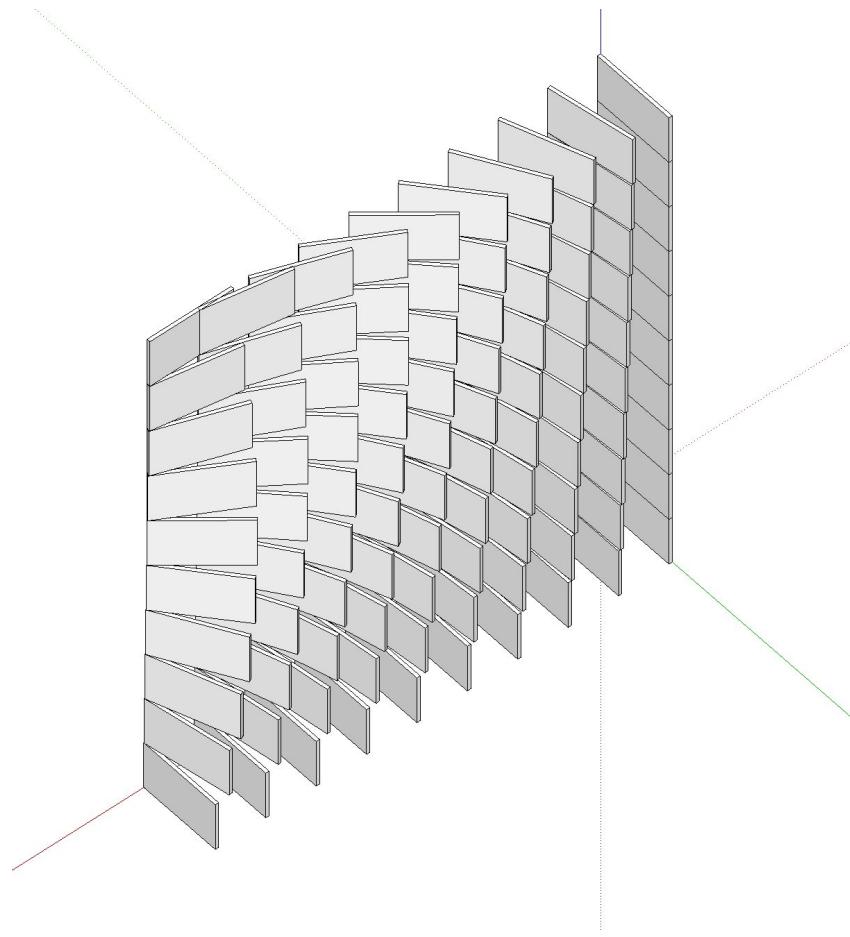
Set the ratio accordingly

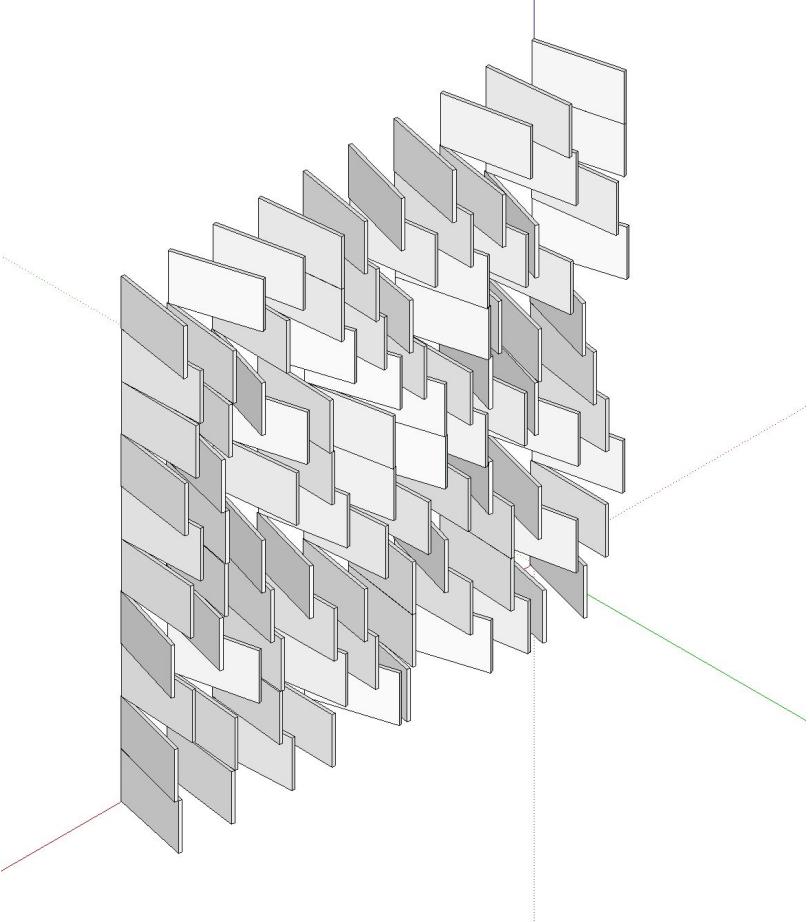
Create a new outer loop j

Apply rotation incrementally from the bottom to top and left to right

Remember:

Full rotation = $2 * \text{Math::PI}$

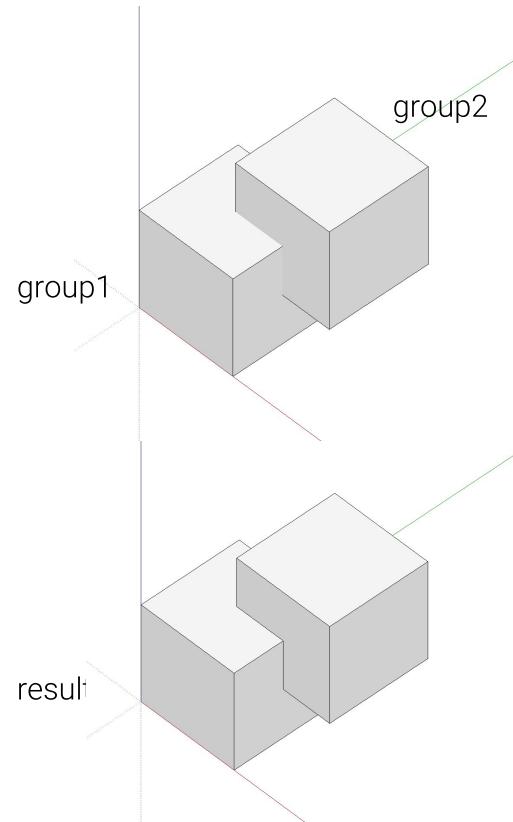




exercise_12_random_rotate.rb

Boolean Operation

```
result =  
group2.union(group1)
```



```

1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 # this is a basic code to make a simple box
5
6 a = 100.mm      # define the module dimension
7 width = a       # width of the box
8 length = a     # Length of the box
9 height = 0.2 * a # height of the box
10
11 x = 0.mm        # define the x position
12 y = 0.mm        # define the y position
13 z = 0.mm        # define the z position
14
15 origin = Geom::Point3d.new(x, y, z) # create a point for origin
16
17 #define points for a rectangle in respect to origin point
18 pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
19 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
20 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
21 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
22
23 #create a group_box group
24 box_group = ent.add_group
25 # create a rectangular face
26 face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
27 ↘ if face.normal.z < 0 # this code is used to flip the face if not upward
28   face.reverse!
29 end
30 # pushpull the face to create the box
31 face.pushpull(height)
32
33 # make a copy of box and move it
34 box_copy = box_group.copy
35 u = width * rand(-0.8..0.8)
36 v = length * rand(-0.8..0.8)
37 w = height * rand(-0.8..0.8)
38 move_vector = Geom::Vector3d.new(u, v, w)
39 tr = Geom::Transformation.translation(move_vector)
40 box_copy.transform!(tr)
41 box_copy
42 # union box_group with box_copy
43 union_result = box_copy.union(box_group)

```

Start from “**01_box.rb**”
Save as “**mygeom_13_union.rb**”

Adjust the dimension of box

a = 100.mm
width = a
length = a
height = 0.2 * a

Create two boxes by using translation transformation

```

# make a copy of box and move it
box_copy = box_group.copy
u = width * rand(-0.8..0.8)
v = length * rand(-0.8..0.8)
w = height * rand(-0.8..0.8)
move_vector = Geom::Vector3d.new(u, v, w)
tr = Geom::Transformation.translation(move_vector)
box_copy.transform!(tr)

```

Create a union from the two boxes

```

# union box_group with box_copy
union_result = box_copy.union(box_group)

```

```

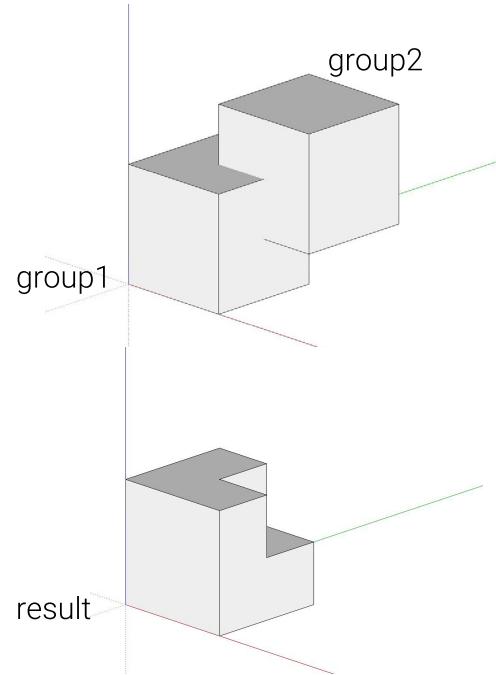
1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 group_union = nil
5
6 for i in 0..4
7   a = 25.mm
8   width = a * rand(1..4)
9   length = a * rand(1..4)
10  height = a * rand(1..4)
11
12  x = a * rand(0..4)
13  y = a * rand(0..4)
14  z = a * rand(0..4)
15
16  origin = Geom::Point3d.new(x, y, z)
17
18  pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
19  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
20  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
21  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
22
23  group = ent.add_group
24  group_ent = group.entities
25  face = group_ent.add_face(pt1, pt2, pt3, pt4)
26
27  if face.normal.z < 0
28    face.reverse!
29  end
30  face.pushpull(height)
31  if group_union == nil
32    group_union = group
33  else
34    if group_union.manifold?
35      group_union = group_union.union(group)
36    end
37  end
38 end

```

Start from “**08_random_box_composition**”

Save as “**mygeom_14_series_union.rb**”

```
result = group2.subtract(group1)
```



```

1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 # this is a basic code to make a simple box
5
6 a = 100.mm      # define the module dimension
7 width = a       # width of the box
8 length = a     # Length of the box
9 height = 0.2 * a # height of the box
10
11 x = 0.mm        # define the x position
12 y = 0.mm        # define the y position
13 z = 0.mm        # define the z position
14
15 origin = Geom::Point3d.new(x, y, z) # create a point for origin
<
16
17 #define points for a rectangle in respect to origin point
18 pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
19 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
20 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
21 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
22
23 #create a group_ box group
24 box_group = ent.add_group
25 # create a rectangular face
26 face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
27 ~if face.normal.z < 0 # this code is used to flip the face if not upward
28   face.reverse!
29 end
30 # pushpull the face to create the box
31 face.pushpull(height)

32
33 # make a copy of box and move it
34 box_copy = box_group.copy
35 u = width * rand(-0.8..0.8)
36 v = length * rand(-0.8..0.8)
37 w = height * rand(-0.8..0.8)
38 move_vector = Geom::Vector3d.new(u, v ,w)
39 tr = Geom::Transformation.translation(move_vector)
40 box_copy.transform!(tr)
41 box_copy
42 # subtract box_group with box_copy
43 subtract_result = box_copy.union(box_group)

```

Start from “**mygeom_13_union.rb**
Save as “**mygeom_15_subtract.rb**”

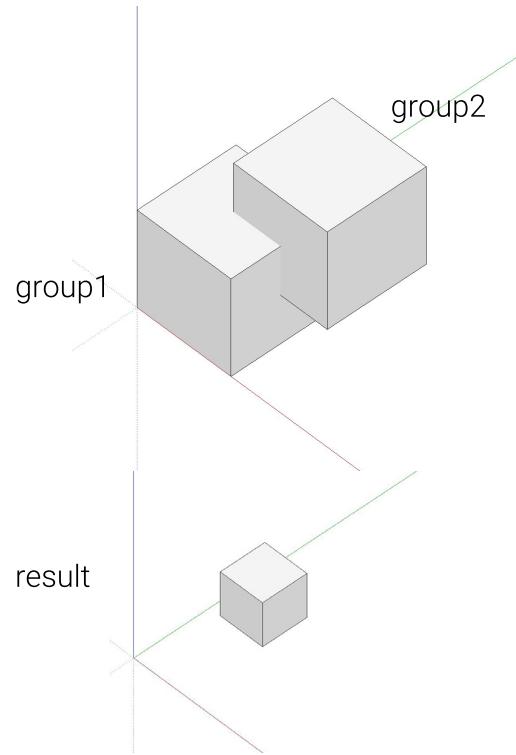
Changes the union method to subtract

```
# subtract box_group with box_copy
union_result = box_copy.union(box_group)
subtract_result = box_copy.union(box_group)
```

Start from **08_random_box_composition** Save as "**mygeom_16_series_subtract.rb**"

```
1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4     # define the module dimension
5 width = 100.mm      # width of the box
6 length = 100.mm    # Length of the box
7 height = 200.mm    # height of the box
8
9 x = 0.mm          # define the x position
10 y = 0.mm         # define the y position
11 z = 0.mm         # define the z position
12
13 origin = Geom::Point3d.new(x, y, z) # create a point for origin
14
15 #define points for a rectangle in respect to origin point
16 pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
17 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
18 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
19 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
20
21 #create a group_ box group
22 box_group = ent.add_group
23 # create a rectangular face
24 face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
25 if face.normal.z < 0 # this code is used to flip the face if not upward
26   face.reverse!
27 end
28 # pushpull the face to create the box
29 face.pushpull(height)
30
31
33 for i in 0..15
34   a = 25.mm
35   width = a * rand(1..4)
36   length = a * rand(1..4)
37   height = a * rand(1..4)
38
39   x = a * rand(-4..4)
40   y = a * rand(-4..4)
41   z = a * rand(0..4)
42
43   origin = Geom::Point3d.new(x, y, z)
44
45   pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
46   pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
47   pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
48   pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
49
50 group = ent.add_group
51 group_ent = group.entities
52 face = group_ent.add_face(pt1, pt2, pt3, pt4)
53
54 if face.normal.z < 0
55   face.reverse!
56 end
57 face.pushpull(height)
58 box_group = group.subtract(box_group)
59 if !box_group.manifold?
60   break
61 end
62 end
63
```

```
result = group2.intersect(group1)
```



Ruby Math Module

<https://ruby-doc.org/core-2.2.0/Math.html>

pi = Math::PI

Remember: 360 degree == full circle = $2 * \pi$
radians

cos = Math.cos(degree)

sin = Math.sin(degree)

tan = Math.sin(degree)

```

1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 v for i in 0..19
5 v   for j in 0..19 # create inner Loop with j counter
6 v     a = 100.mm
7 v     width = a #set the ratio to 1:1:1
8 v     length = a
9 v     # add sin function to box height
10 v    height = a * Math.sin(2 * i / 19.0 * Math::PI)
11 v
12 v    #set the j to correspond with y coordinate
13 v    origin = Geom::Point3d.new(i * width, j * length , 0)
14
15 pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
16 pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
17 pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
18 pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
19
20 group = ent.add_group
21 group_ent = group.entities
22 face = group_ent.add_face(pt1, pt2, pt3, pt4)
23
24 v if face.normal.z < 0
25   face.reverse!
26 end
27
28   face.pushpull(height)
29 end
30 end
31

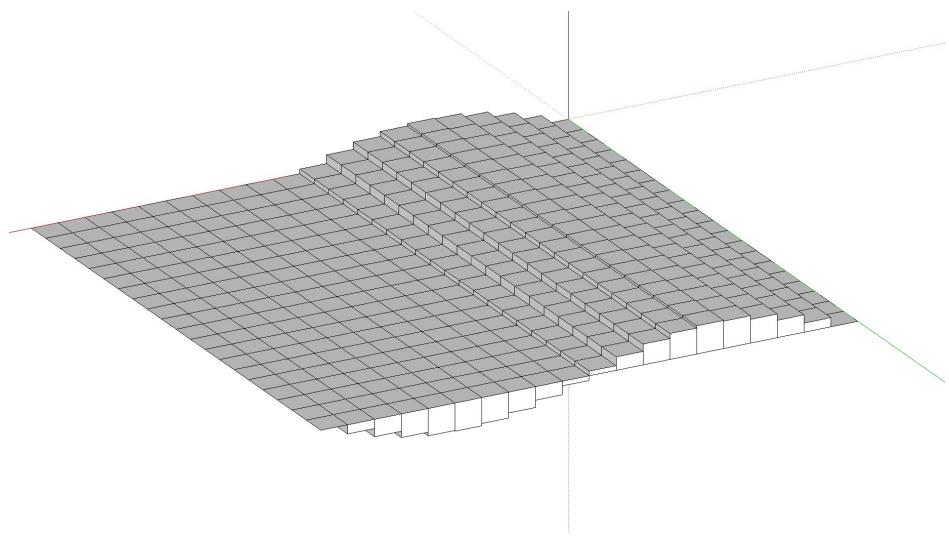
```

Start from “**mygeom_06_grid.rb**”

Save as “**mygeom_17_math_sin_i.rb**”

add sin function to box height

height = a * Math.sin(2 * i / 19.0 * Math::PI)



```

1 mod = Sketchup.active_model # open sketchup active_model
2 ent = mod.entities # accessing the entities in mod
3
4 for i in 0..19
5   for j in 0..19 # create inner loop with j counter
6     a = 100.mm
7     width = a #set the ratio to 1:1:1
8     length = a
9     # add sin function to box height
10    height = a * Math.sin(2 * i / 19.0 * Math::PI) * Math.sin(2 * j / 19.0 * Math::PI) + 200.mm
11
12    #set the j to correspond with y coordinate
13    origin = Geom::Point3d.new(i * width, j * length , 0)
14
15    pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
16    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
17    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
18    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)
19
20    group = ent.add_group
21    group_ent = group.entities
22    face = group_ent.add_face(pt1, pt2, pt3, pt4)
23
24    if face.normal.z < 0
25      face.reverse!
26    end
27
28    face.pushpull(height)
29  end
30 end

```

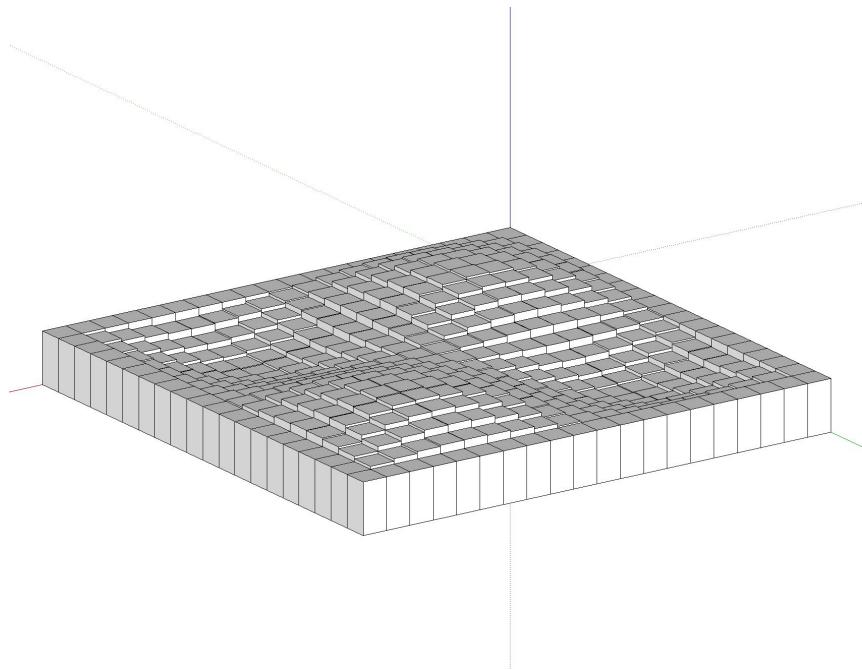
Start from

"mygeom_17_math_sin_i"

Save as

"mygeom_18_math_sin_ij.rb"

add sin function to box height
according to i and j values



Understanding Sketchup Ruby API

<https://ruby.sketchup.com>

<https://ruby.sketchup.com/Geom.html>

The screenshot shows the SketchUp Developer interface. On the left is a sidebar titled "Class List" with a search bar and a "Top Level Namespace" section. A red dashed box highlights the "Geom" class under "Top Level Namespace". Below "Geom", several other classes are listed: BoundingBox, Bounds2d, LatLong, OrientedBounds2d, Point2d, Point3d, PolygonMesh, Transformation, Transformation2d, UTM, Vector2d, Vector3d, and LanguageHandler. At the bottom of the sidebar, another red dashed box highlights the "Layout" section, which contains "AngularDimension" and "AutoTextDefinition". The main content area has a header "Index (G) » Geom" and a title "Module: Geom". It includes an "Overview" section with a note about lines and planes being infinite, followed by descriptions of how to represent lines and planes using arrays of points or vectors. It also provides code examples for creating lines and planes. Below this, there's a section about 3D math books and a "Version" section.

Index (G) » Geom

Module: Geom

Overview

Note: Lines and Planes are infinite.

The Geom module defines a number of Module methods that let you perform different geometric operations.

The methods in this module take lines and planes as arguments. There is no special class for representing lines or planes. Arrays are used for both.

A line can be represented as either an Array of a point and a vector, or as an Array of two points.

```
line1 = [Geom::Point3d.new(0, 0, 0), Geom::Vector3d.new(0, 0, 1)]
line2 = [Geom::Point3d.new(0, 0, 0), Geom::Point3d.new(0, 0, 100)]
```

A plane can be represented as either an Array of a point and a vector, or as an Array of 4 numbers that give the coefficients of a plane equation.

```
plane1 = [Geom::Point3d.new(0, 0, 0), Geom::Vector3d.new(0, 0, 1)]
plane2 = [0, 0, 1, 0]
```

There are several good books on 3D math if you are new to the concepts of a line, plane, and vector.

Version: sketchup 6.0

<https://ruby.sketchup.com/Geom/Point3d.html>

The screenshot shows the SketchUp Developer documentation for the `Geom::Point3d` class. The left sidebar contains a "Class List" with various SketchUp classes like `BoundingBox`, `Bounds2d`, `LatLong`, etc. The `Geom` category is expanded, showing `Point3d` selected. The main content area shows the `Geom::Point3d` class definition, which inherits from `Object`. It includes an "Overview" section describing the class's purpose and usage examples.

Class List
Classes | Methods | Files
Search:

Geom

- `BoundingBox < Object`
- `Bounds2d < Object`
- `LatLong < Object`
- `OrientedBounds2d < Object`
- `Point2d < Object`
- `Point3d < Object`**
- `PolygonMesh < Object`
- `Transformation < Object`
- `Transformation2d < Object`
- `UTM < Object`
- `Vector2d < Object`
- `Vector3d < Object`

LanguageHandler < Object

Layout

- `AngularDimension < Entity`
- `AutoTextDefinition < Object`
- `AutoTextDefinitions < Object`
- `ConnectionPoint < Object`

[Index \(P\)](#) » [Geom](#) » `Point3d`

Class: `Geom::Point3d`

Inherits: `Object` [show all](#)

Overview

The `Point3d` class allows you to work with a point in 3D space. The point is basically just a series of values representing x, y and z coordinates. The values are specified as [x,y,z]. For example [100,200,300]. To create a point call `Geom::Point3d.new`, where the creation method can take a variety of arguments:

In addition to the methods below, there are a series of geometry related methods that are on the `Array` class, since `Point3d` objects can also be represented as a 3-element `Array`. These `Array`-level methods are for operations such as determining if a point is on a line, on a plane, etc. See the `Array` class for details.

Examples:

```
# No arguments, creates a point at the origin [0,0,0]
pt1 = Geom::Point3d.new

# Creates a point at x of 100, y of 200, z of 300.
pt2 = Geom::Point3d.new(100,200,300)

# You can also create a point directly by simply assigning the x, y and z
# values to a variable as an array:
pt3 = [100,200,300]
```

<https://ruby.sketchup.com/Sketchup.html>

The screenshot shows the SketchUp Developer API documentation. On the left is a sidebar titled "Class List" with a search bar and a tree view of classes under "Table". A red dashed box highlights the "Sketchup" class, which is selected and has a blue background. The main content area shows the "Module: Sketchup" page with an "Overview" section and examples of Ruby code. The code demonstrates how to grab handles to entities, layers, materials, and component definitions, and then use them to display a message box.

Class List

Index (S) » Sketchup

Module: Sketchup

Overview

The Sketchup module contains a number of important utility methods for use in your Ruby scripts. Many of the classes in the API are implemented beneath this module. You can think of the Sketchup module as the “root” of the application tree. Most ruby calls start from the currently active model, and this is accessed via the Sketchup.active_model method.

Examples:

```
# Grab a handle to the currently active model (aka the one the user is
# looking at in SketchUp.)
model = Sketchup.active_model

# Grab other handles to commonly used collections inside the model.
entities = model.entities
layers = model.layers
materials = model.materials
component_definitions = model.definitions
selection = model.selection

# Now that we have our handles, we can start pulling objects and making
# method calls that are useful.
first_entity = entities[0]
UI.messagebox("First thing in your model is a " + first_entity.typename)

number_materials = materials.length
```

