Bekraf Creative Labs 2018
Sub Sektor Arsitektur

Indonesia
Creative
Digital
Architecture

// Workshop

Design Scripting Fundamentals

by Mikhael Johanes

Eks Monod Diephuis & Co.
Semarang, 24 Agustus 2018

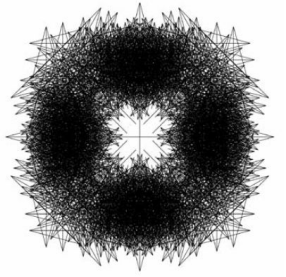
Why Scripting?

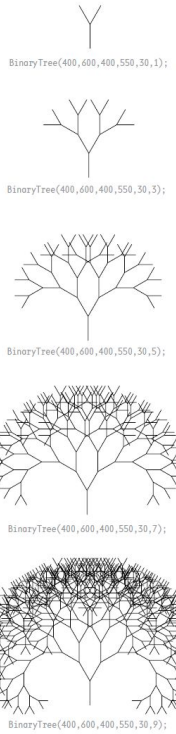
Automating routines
and repetitive activities

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.

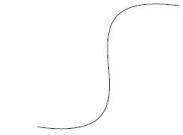
The larger design is created by repeating, rotating, inverting, connecting, and overlapping the tree forms. In code-based drawing, the designer varies the results by changing the inputs to the algorithm.



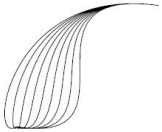


Binary Tree The drawing becomes denser with each generation. The last number in the code indicates the number of iterations. Yeohyun Ahn, MFA Studio.

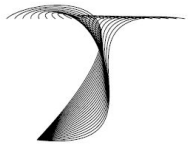
Lupton & Phillips, 2008



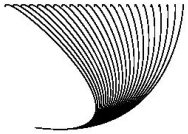
```
bezier(850,200,100,100,900,900,150,800);
```



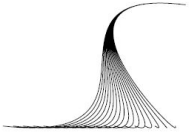
```
for(int i-0; i<900; i-i+100)
{bezier(850,200,100,100,i,900,150,800);}
```



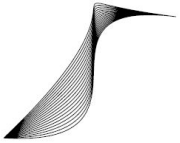
```
for(int i-0;i<900; i-i+40)
{bezier(i,200,100,100,900,i,150,800);}
```



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



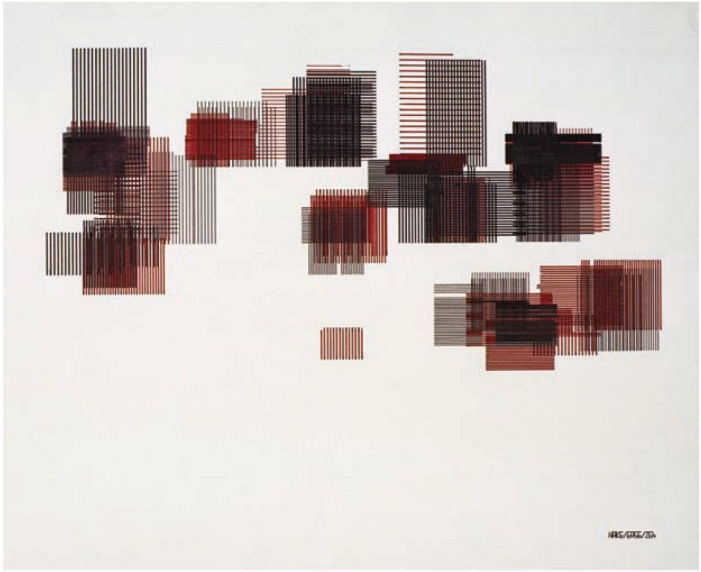
```
for(int i-0; i<900; i-i+50)
{bezier(900,200,100,100,900,900,i,800);}
```



```
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

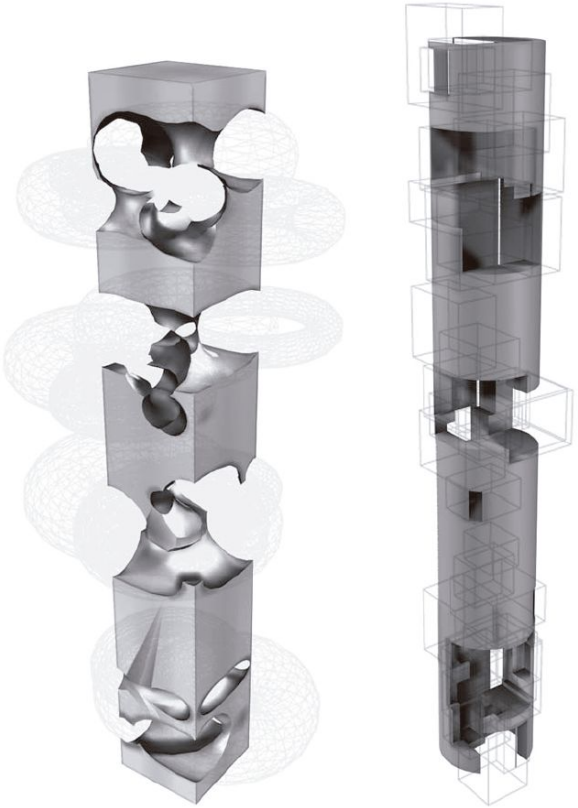


Reas, 2010

Felder von Rechteck Schraffuren Überlagert, by Frieder Nake, 1965
For this image, Nake used seven random values to control the size, location, orientation, quantity, and pen for each set of lines.

Geno Pheno Sculpture "Fractal Dice No. 1", by Keith Tyson, 2005 A statement released for the exhibition of this sculpture declared, "An algorithm written by the artist determines what will be on view in this new exhibition." The subsequent rolls of a die determined the parameters of the sculpture, including the color, 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

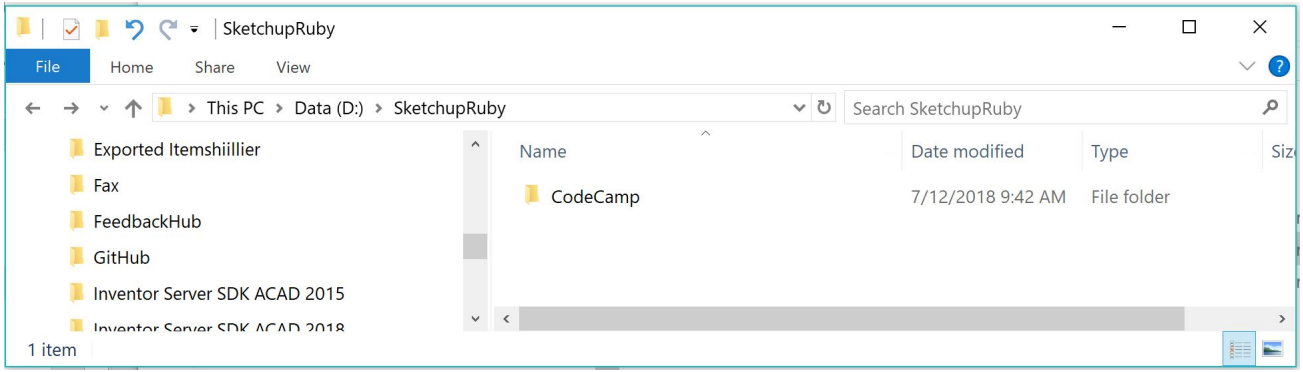
Script as Active Instructions

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

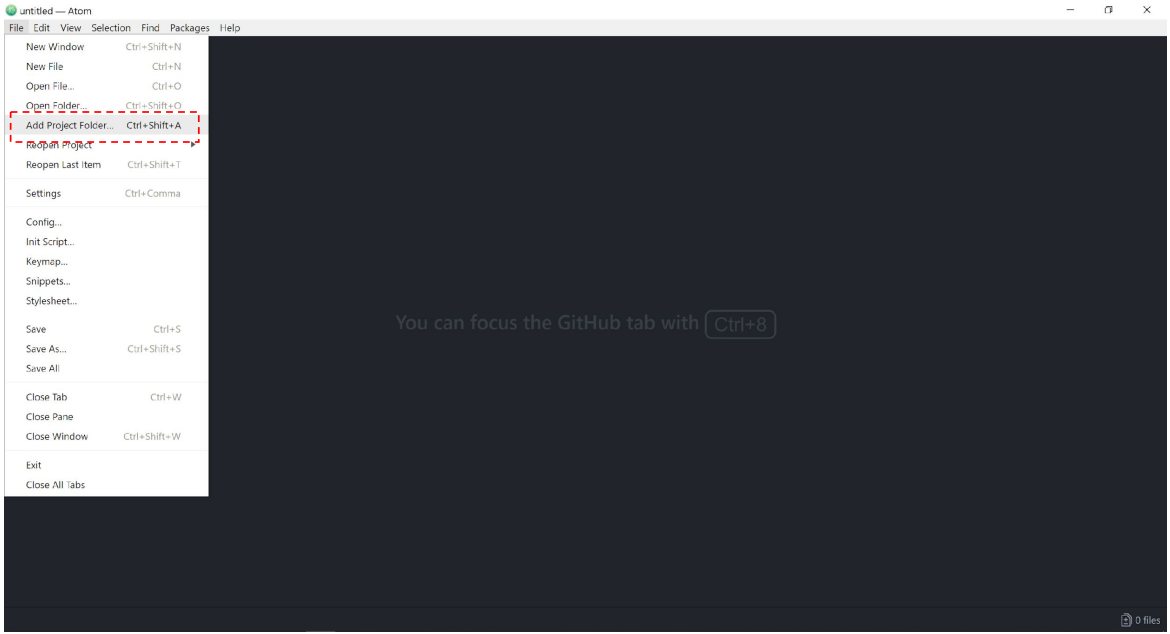
Burry, 2011 – *Scripting Cultures*

Installing
Atom Code Editor

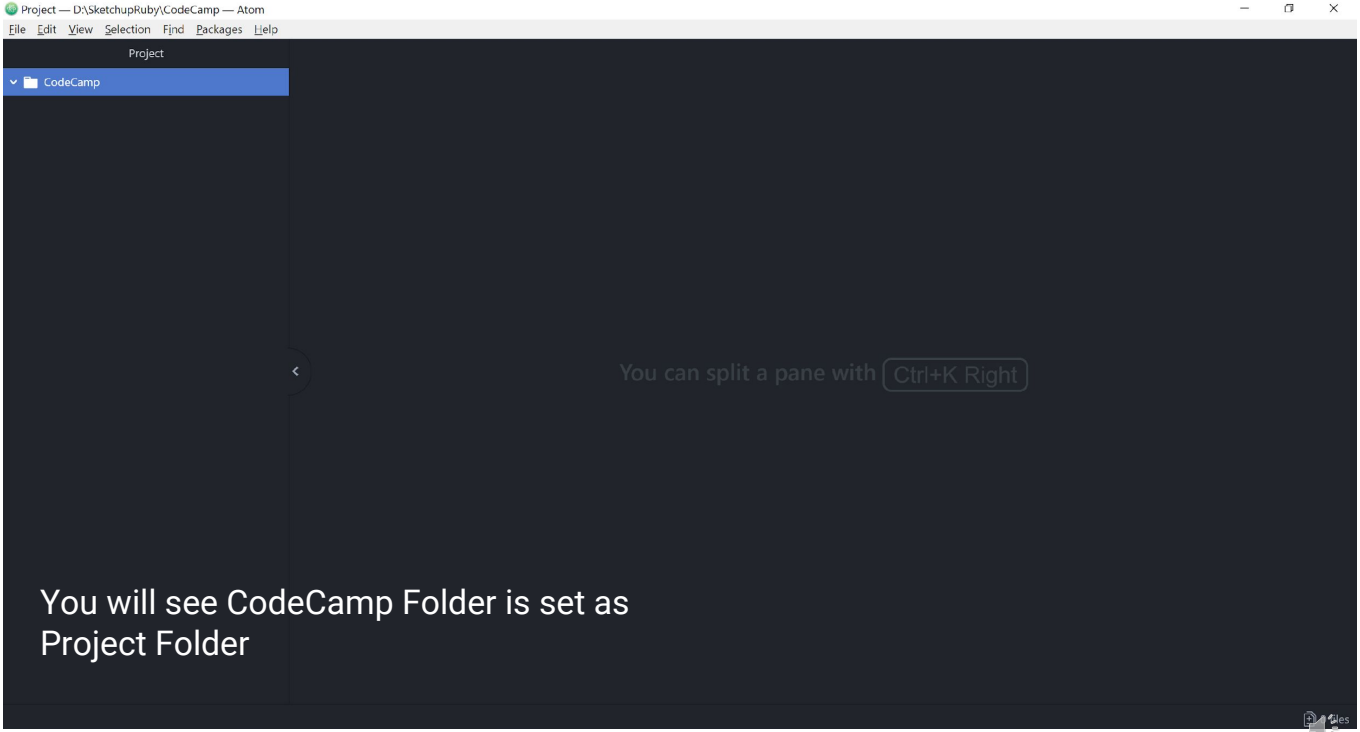
https://atom.io/

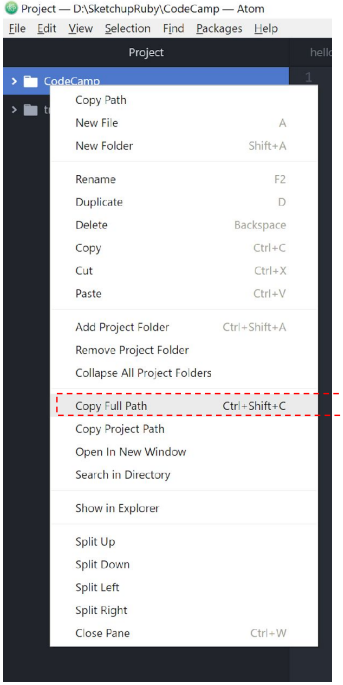


Create a “CodeCamp” folder

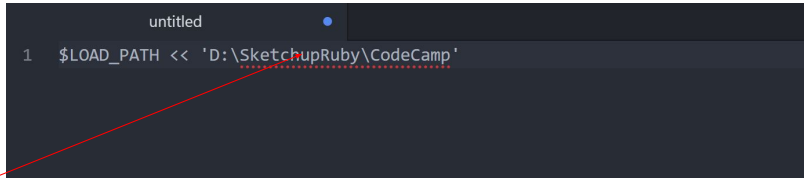


Set the folder as Project Folder

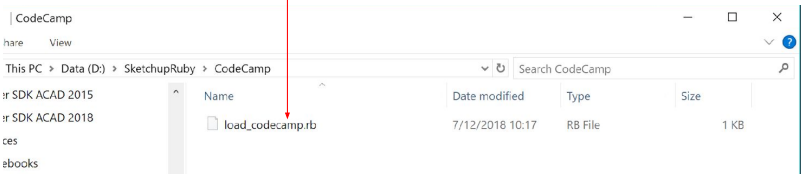

Project — D:\SketchupRuby\CodeCamp — Atom
File Edit View Selection Find Packages Help
Project
CodeCamp
You can split a pane with Ctrl+K Right
You will see CodeCamp Folder is set as Project Folder

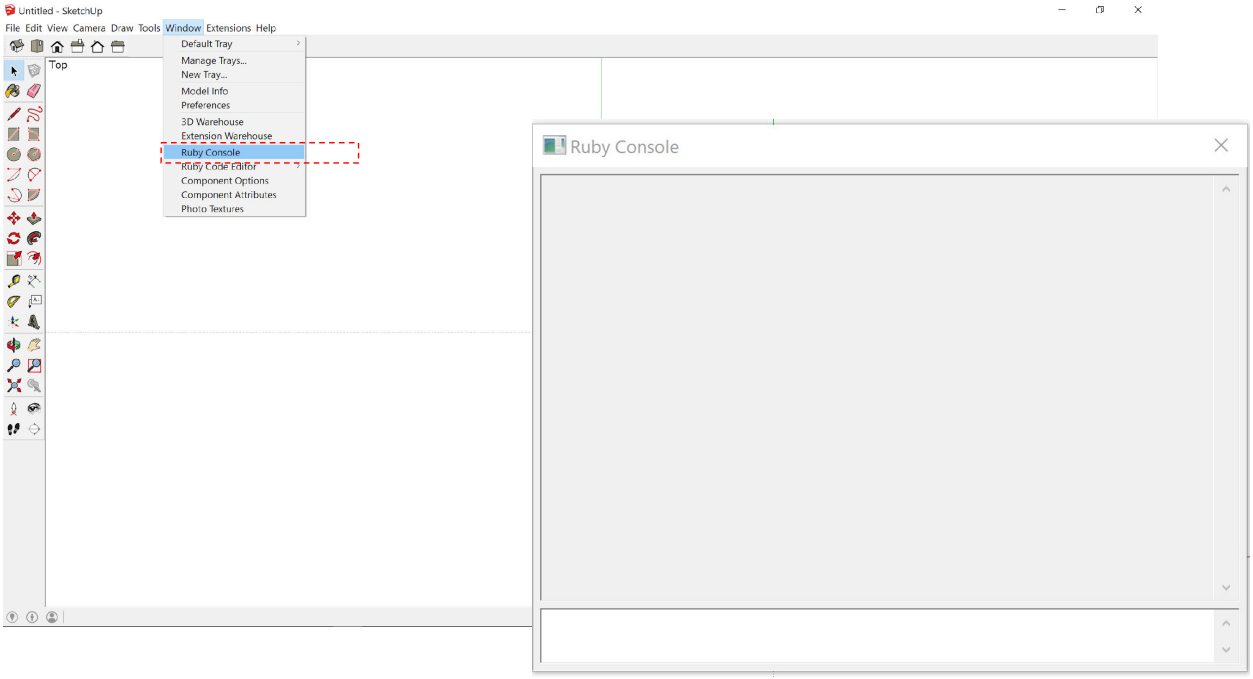


1. Create new file: File → New File
2. Right Click -> Copy Full Path on “Code Camp” folder
3. Type $LOAD_PATH << ‘[paste the path here]’ **in the newly created file**

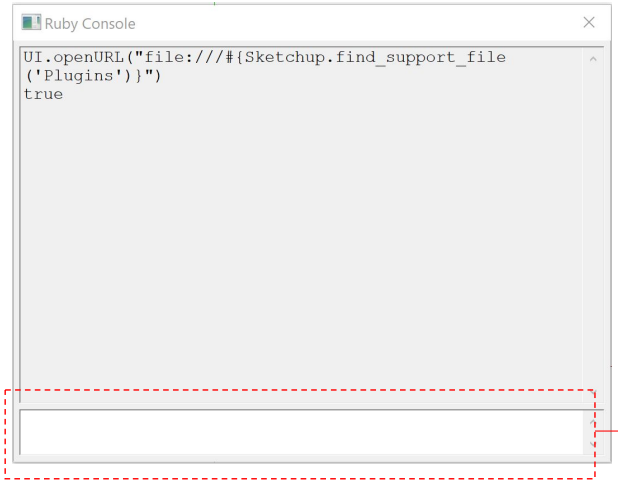


4. Save File as **“load_codecamp.rb”**
5. You should see the **“load_codecamp.rb”** file in “CodeCamp” folder





Open the Sketchup Ruby Console



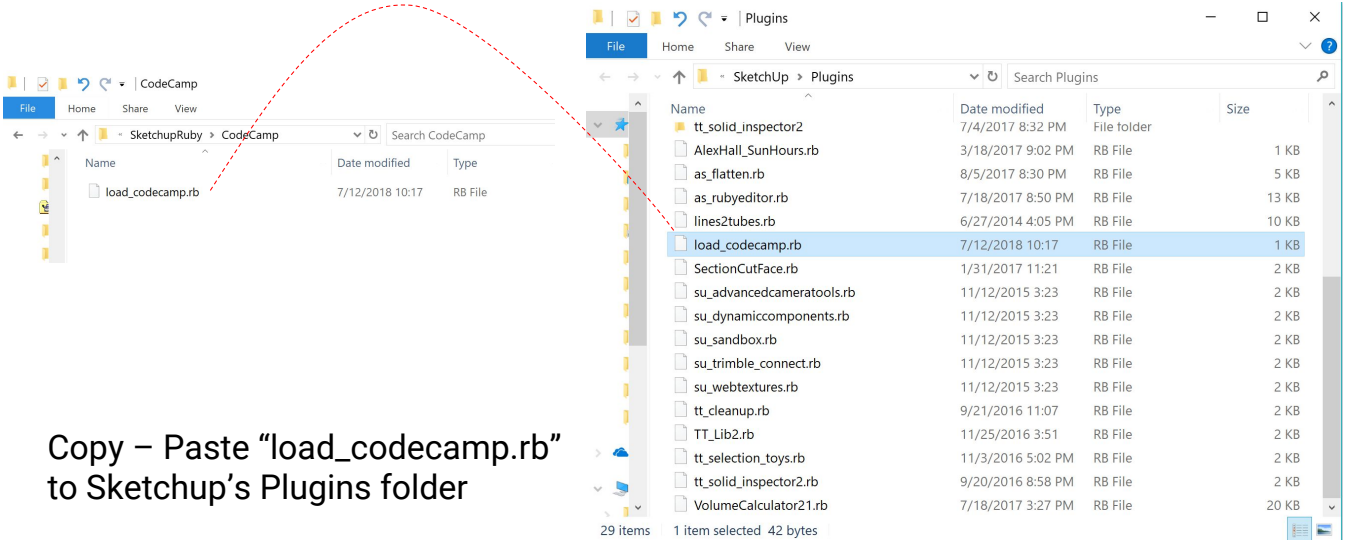
Type:

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

[hit Enter]

Sketchup Plugin folder will be shown afterwards

Identifying Sketchup’s Plugin Folder

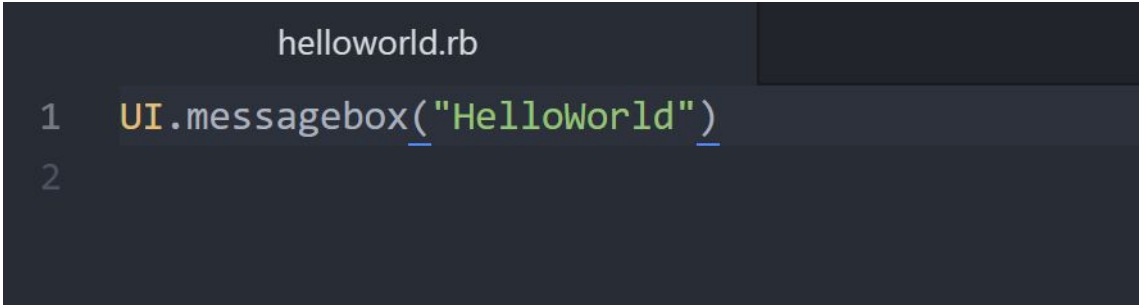


Copy – Paste “load_codecamp.rb” to Sketchup’s Plugins folder

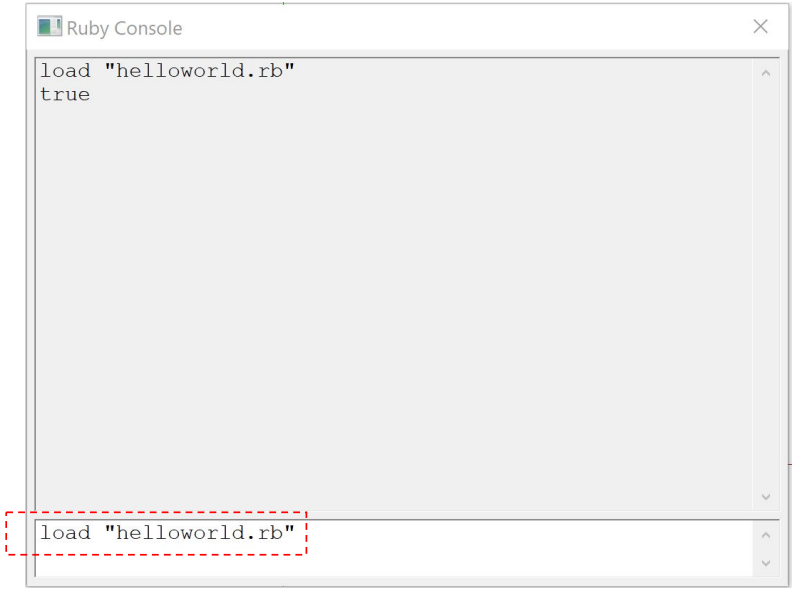
Restart your Sketchup

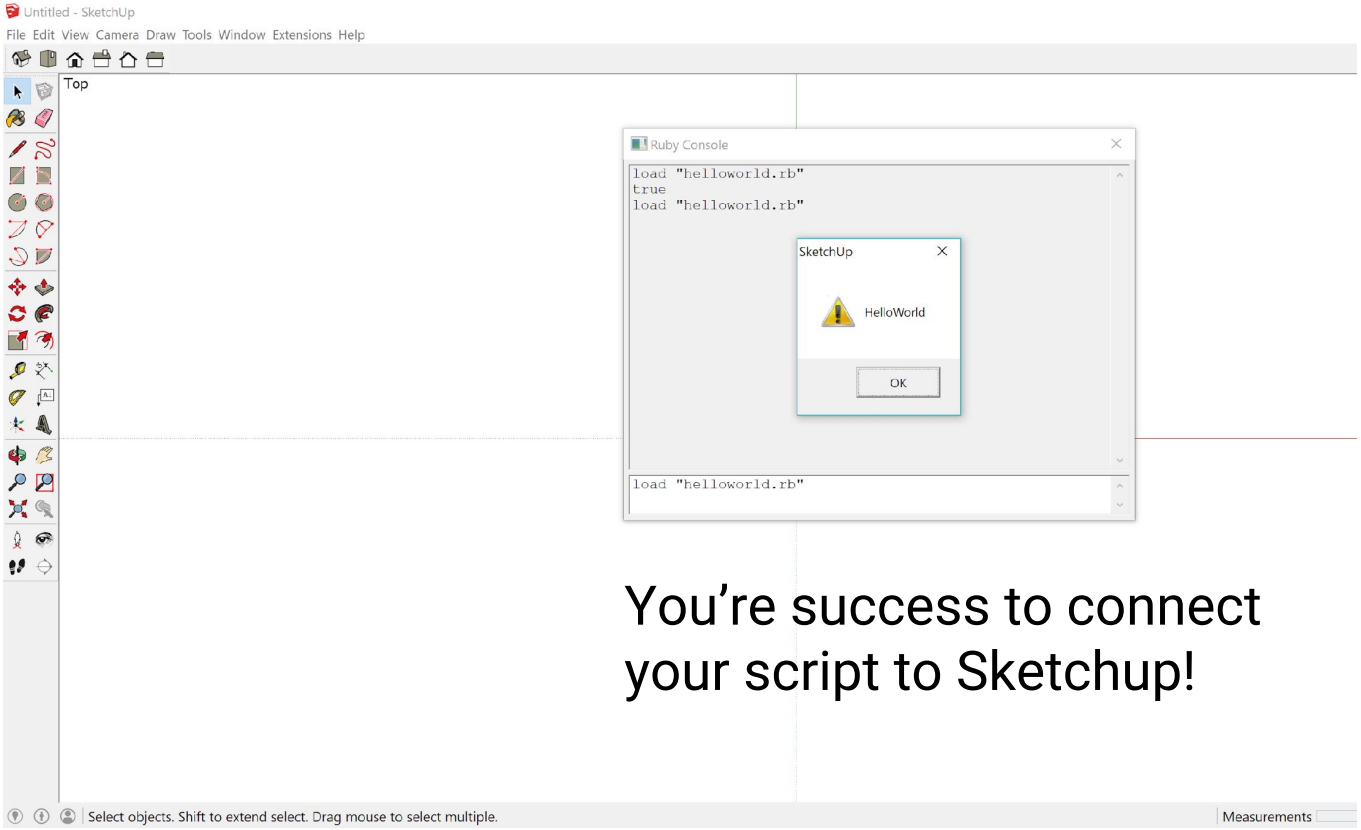
My First Script

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



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





You’re success to connect your script to Sketchup!

General ruby tutorial:

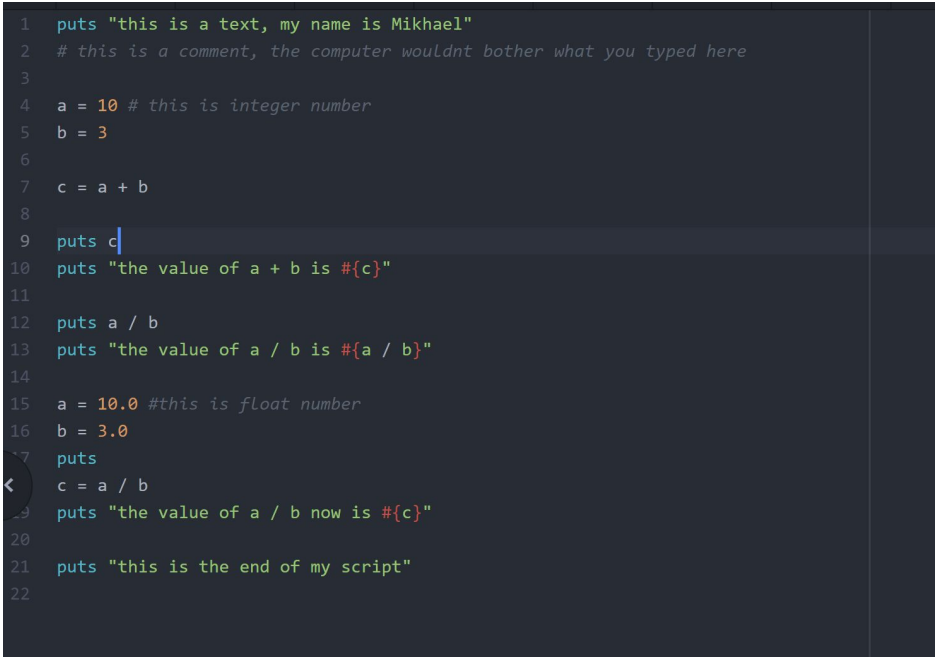
https://www.tutorialspoint.com/ruby/ruby_syntax.htm

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**”



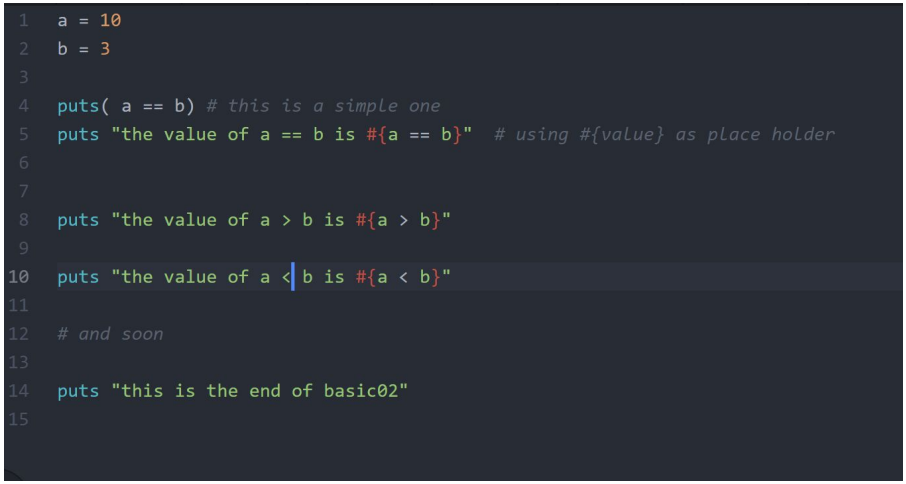
```ruby
puts "this is a text, my name is Mikhael"
# this is a comment, the computer wouldnt bother what you typed here

a = 10 # this is integer number
b = 3

c = a + b

puts c
puts "the value of a + b is #{c}"

puts a / b
puts "the value of a / b is #{a / b}"

a = 10.0 #this is float number
b = 3.0
puts
c = a / b
puts "the value of a / b now is #{c}"

puts "this is the end of my script"

```

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

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

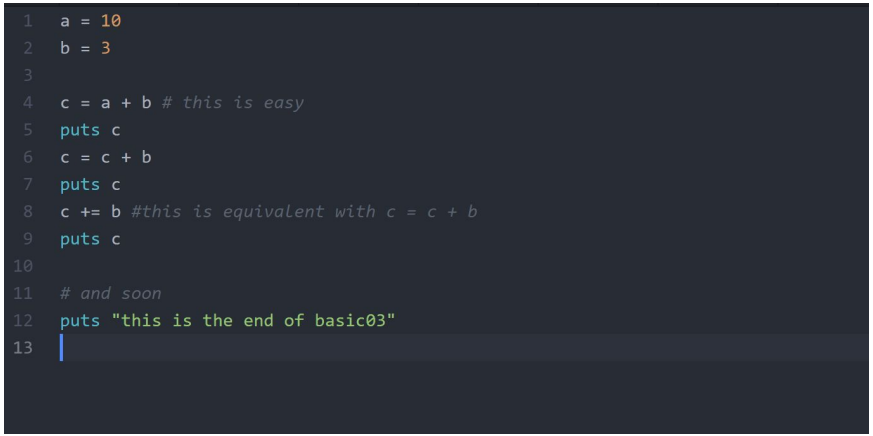


```ruby
a = 10
b = 3

puts( a == b) # this is a simple one
puts "the value of a == b is #{a == b}"  # using #{value} as place holder


puts "the value of a > b is #{a > b}"

puts "the value of a < b is #{a < b}"

# and soon

puts "this is the end of basic02"

```

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**”



```ruby
a = 10
b = 3

c = a + b # this is easy
puts c
c = c + b
puts c
c += b #this is equivalent with c = c + b
puts c

# and soon
puts "this is the end of basic03"
```

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**”

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod


width = 100.mm
length = 200.mm
height = 300.mm

pt1 = Geom::Point3d.new(0, 0, 0)
pt2 = Geom::Point3d.new(width, 0, 0)
pt3 = Geom::Point3d.new(width, length, 0)
pt4 = Geom::Point3d.new(0, length, 0)

face = ent.add_face(pt1, pt2, pt3, pt4)

if face.normal.z < 0
  face.reverse!
end

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**”

```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model

width = 100.mm # initiating width variable
length = 100.mm # length variable
height = 150.mm # height variable

pt1 = Geom::Point3d.new(0, 0, 0) # create points for each points
pt2 = Geom::Point3d.new(width, 0, 0)
pt3 = Geom::Point3d.new(width, length, 0)
pt4 = Geom::Point3d.new(0, width, 0)

group = ent.add_group # create Sketchup Group in "ent"
group_ent = group.entities # accessing "Entities" in "group"
face = group_ent.add_face(pt1, pt2, pt3, pt4) # create a Face in "group_ent" then assigned to "face" variable
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
face.pushpull(height) # push the face to create volume

```

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

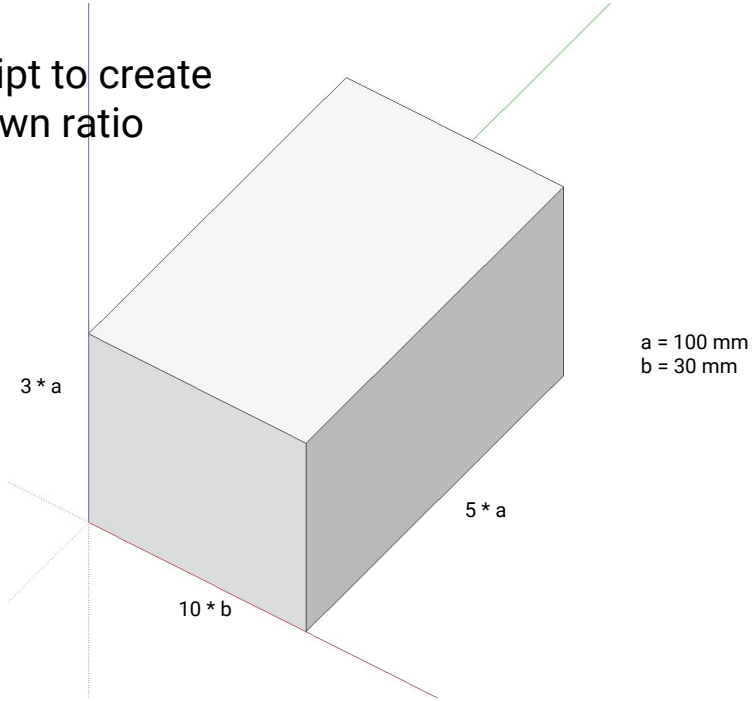
Save as “**mygeom_01_box_group.rb** ” to “**mygeom_02_box.rb**”

```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model


a = 100.mm
width = a
length = a * 2
height = a * 3

pt1 = Geom::Point3d.new(0, 0, 0)
pt2 = Geom::Point3d.new(width, 0, 0)
pt3 = Geom::Point3d.new(width, length, 0)
pt4 = Geom::Point3d.new(0, length, 0)

group = ent.add_group
group_ent = group.entities
face = group_ent.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
face.pushpull(height)

```

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**”

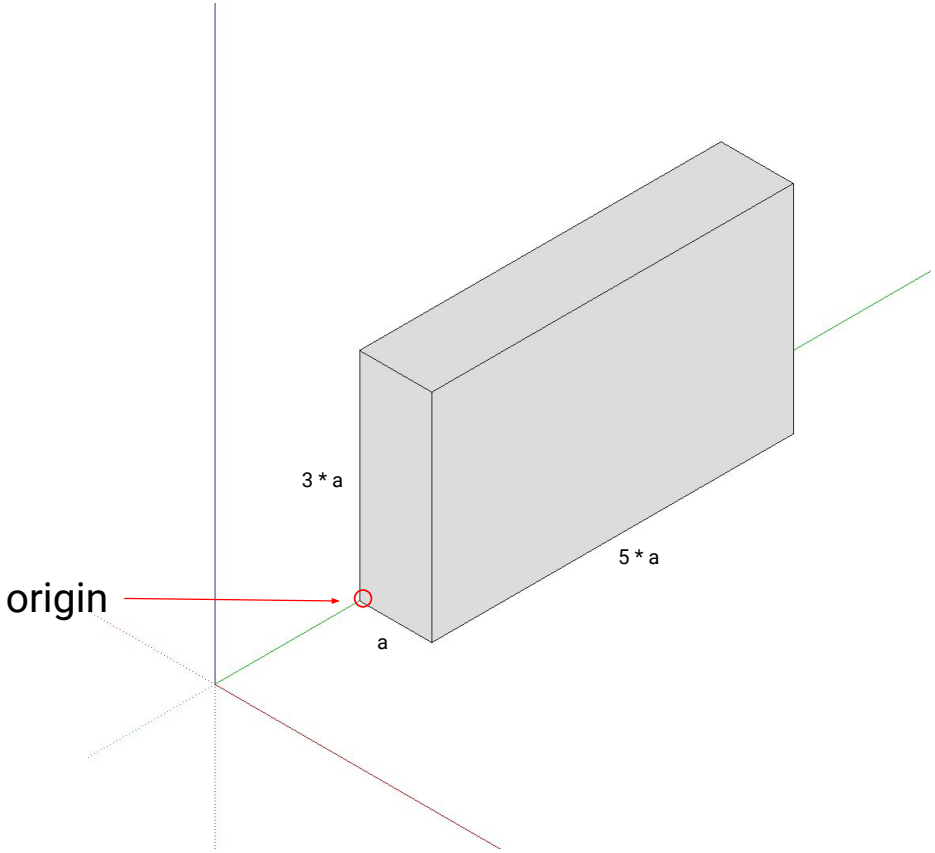
```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model

origin = Geom::Point3d.new(100.mm, 100.mm, 100.mm)
a = 100.mm
width = a
length = a * 5
height = a * 3

pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

group = ent.add_group
group_ent = group.entities
face = group_ent.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
face.pushpull(height)
```

Set a Point3d as origin of the box

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

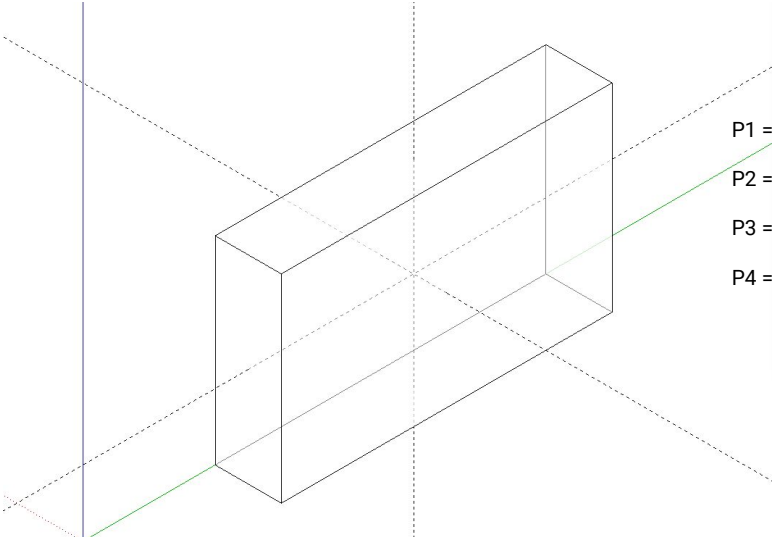
The origin point components (x ,y z) can be retrieved one by one by using:

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


3 * a
5 * a
origin
a

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

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



P1 = (origin.x - width/2), (origin.y - length/2), (origin.z - height/2)

P2 = ...

P3 = ...

P4 = ...

Repetitions, Loops, and Iterations

```
1
2  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

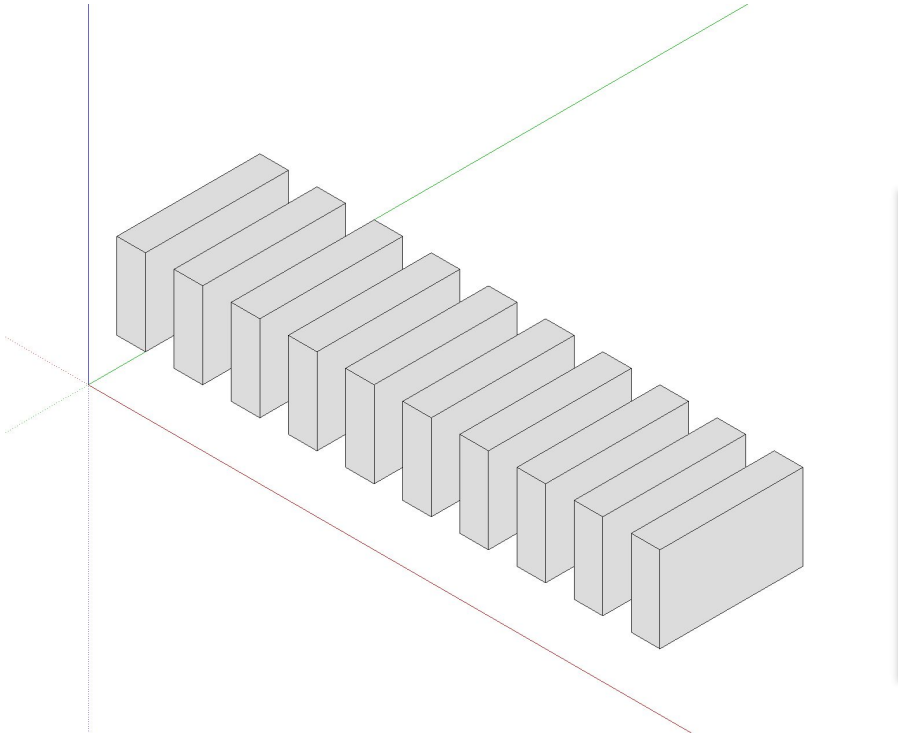
Creating a series of boxes using loop

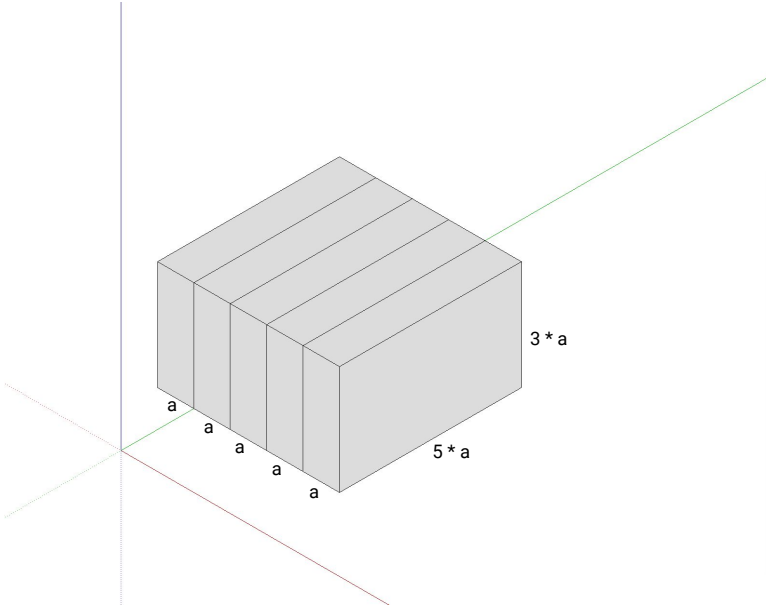
```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model


for i in 0..9
  origin = Geom::Point3d.new(i * 200.mm, 100.mm, 100.mm)
  a = 100.mm
  width = a
  length = a * 5
  height = a * 3

  pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)
  if face.normal.z < 0  # this code is used to flip the face if not upward
    face.reverse!
  end
  face.pushpull(height)
end

```

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**”

```ruby
puts "this is 10 random decimal number"
for i in 0..9
  random = rand
  puts(random)
end

puts "this is 10 random integer number from 1..15"
for i in 0..9
  random = rand(1..15)
  puts(random)
end

puts "this is random decimal number from 1.5..2.0"
for i in 0..9
  random = rand(1.5..2.0)
  puts(random)
end
```

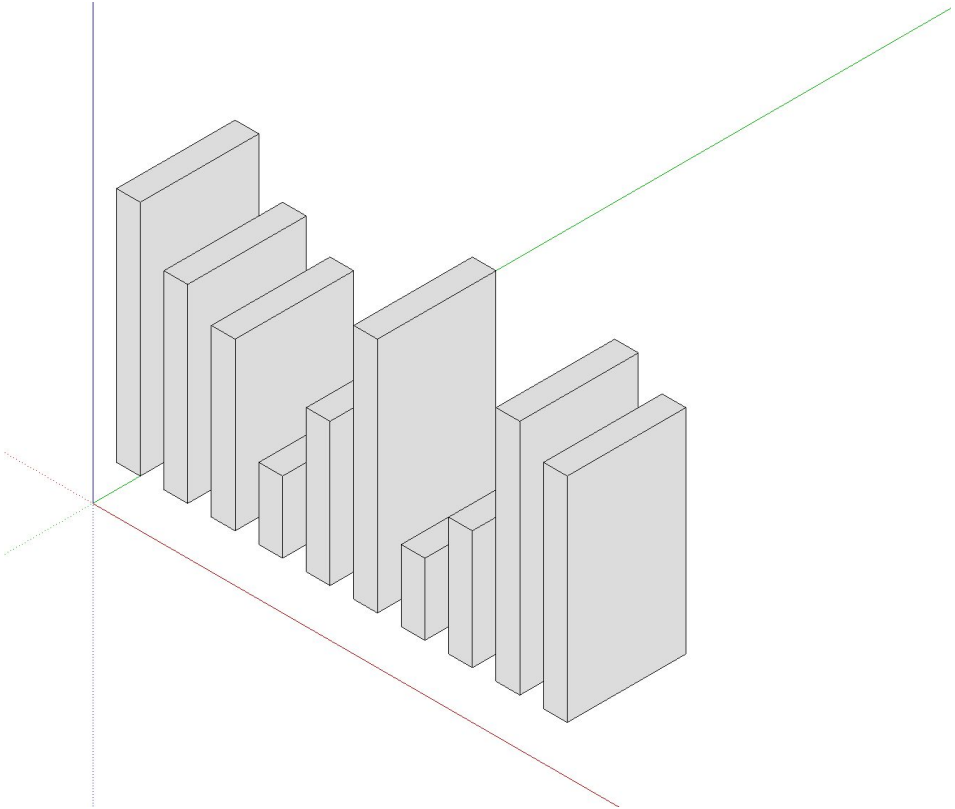
Boxes with random value

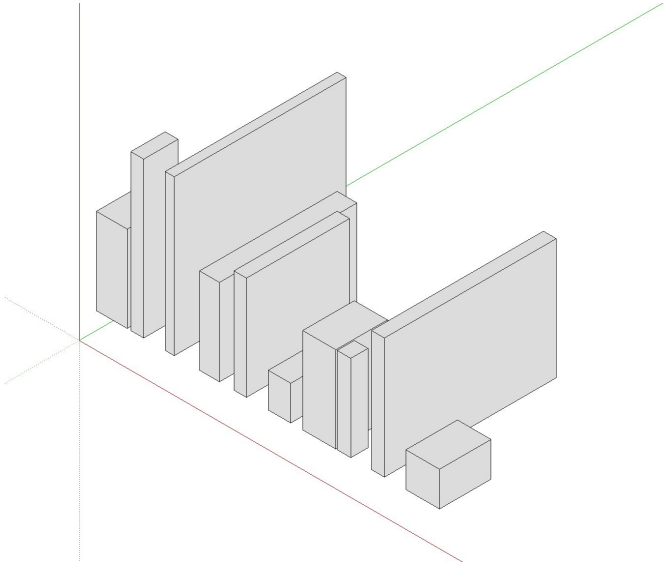
```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model


for i in 0..9
  origin = Geom::Point3d.new(i * 200.mm, 100.mm, 100.mm)
  a = 100.mm
  width = a
  length = a * 5
  random_int = rand(1..10) # create random int number from 1..10
  height = a * random_int # put random number as height multiplier

  pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)
  if face.normal.z < 0  # this code is used to flip the face if not upward
    face.reverse!
  end
  face.pushpull(height)
end

```

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

```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model


for i in 0..9
  a = 100.mm
  width = a
  length = a
  height = a

  # create random (x, y) position
  random_x = (rand(1..10) * 10 * a).mm
  random_y = (rand(1..10) * 10 * a).mm
  origin = Geom::Point3d.new(random_x, random_y, 0)

  pt1 = Geom::Point3d.new(origin.x - width/2 , origin.y - length/2, origin.z - height/2)
  pt2 = Geom::Point3d.new(origin.x + width/2, origin.y - length/2, origin.z - height/2)
  pt3 = Geom::Point3d.new(origin.x + width/2, origin.y + length/2, origin.z - height/2)
  pt4 = Geom::Point3d.new(origin.x - width/2, origin.y + length/2, origin.z - height/2)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)

  if face.normal.z < 0  # this code is used to flip the face if not upward
    face.reverse!
  end

  face.pushpull(height)
end

```

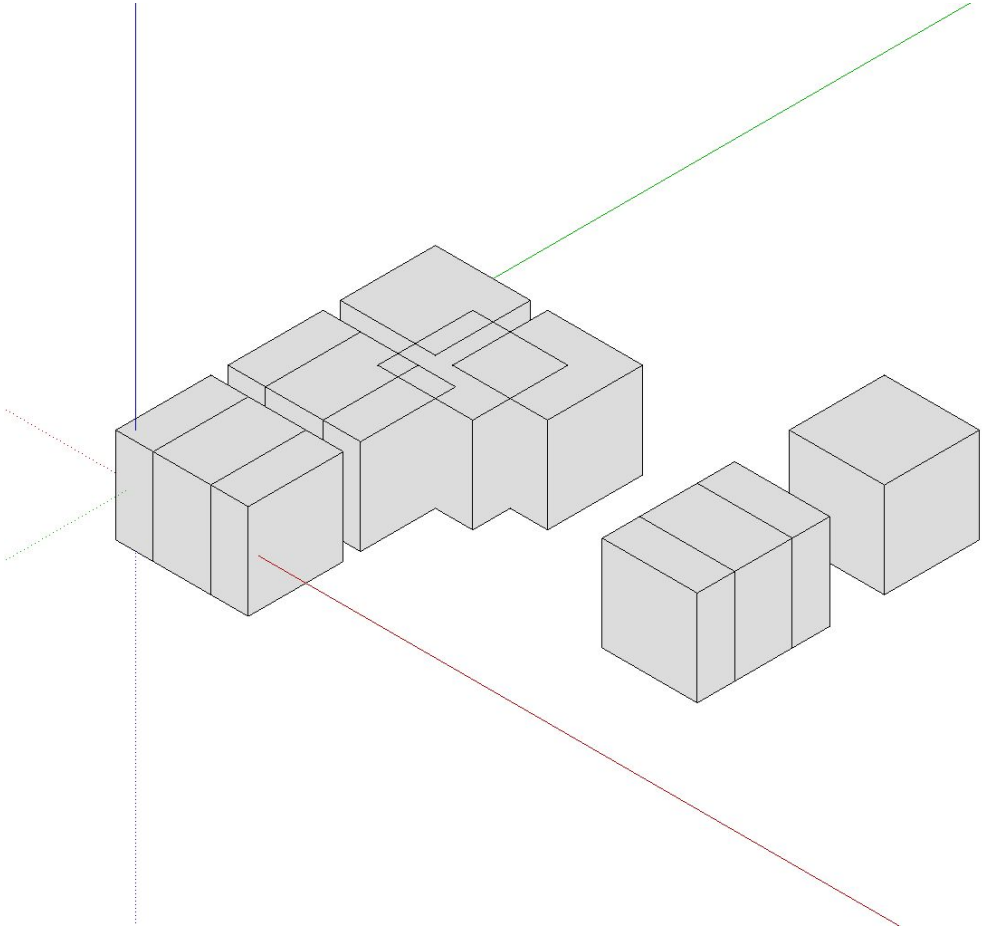
Save as “**02_box_from_center.rb**” (box with center origin point) to “**exercise_05_rand_pos.rb**”

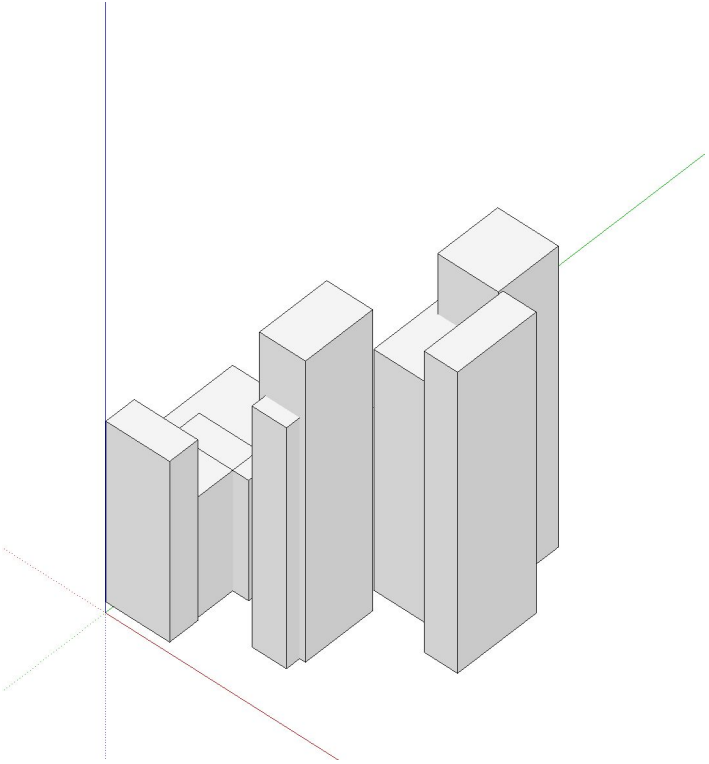
Set the box ratio to 1:1:1 with a = 100.mm

Modify it into loop of 10 boxes

Put random x and y position to origin point

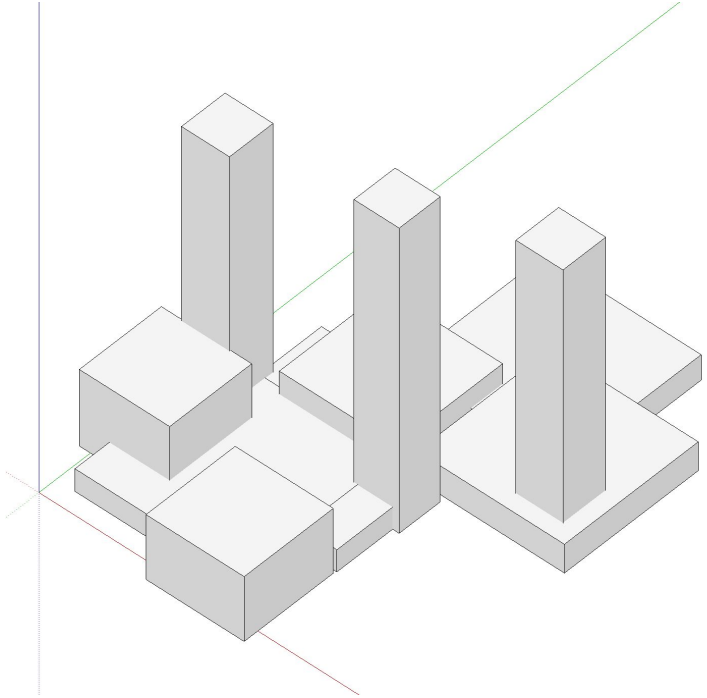
random_x = (random(1..10) * 10 * a).mm
random_y = (random(1..10) * 10 * a).mm





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

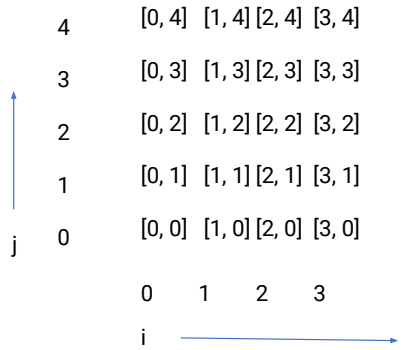
Printscreen your results and save as

exercise_05_your_idea.jpg

2 dimensional grid: loop in loop

```ruby
for i in 0..3
  puts "-----------------------------------------------------"
  puts "the value of i is #{i}"
  for j in 0..4
    puts "the value of j is #{j}"
    puts "the value of [i,j] is [#{i},#{j}]"
  end
end
```

Save as
“basic_06_loop_in_loop.rb”



Grid of boxes

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

for i in 0..9
  for j in 0..5 # create inner loop with j counter
    a = 100.mm
    width = a #set the ratio to 1:1:1
    length = a
    height = a

    #set the j to correspond with y coordinate
    origin = Geom::Point3d.new(i * width, j * length , 0)

    pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

    group = ent.add_group
    group_ent = group.entities
    face = group_ent.add_face(pt1, pt2, pt3, pt4)

    if face.normal.z < 0
      face.reverse!
    end

    face.pushpull(height)
  end
end
```

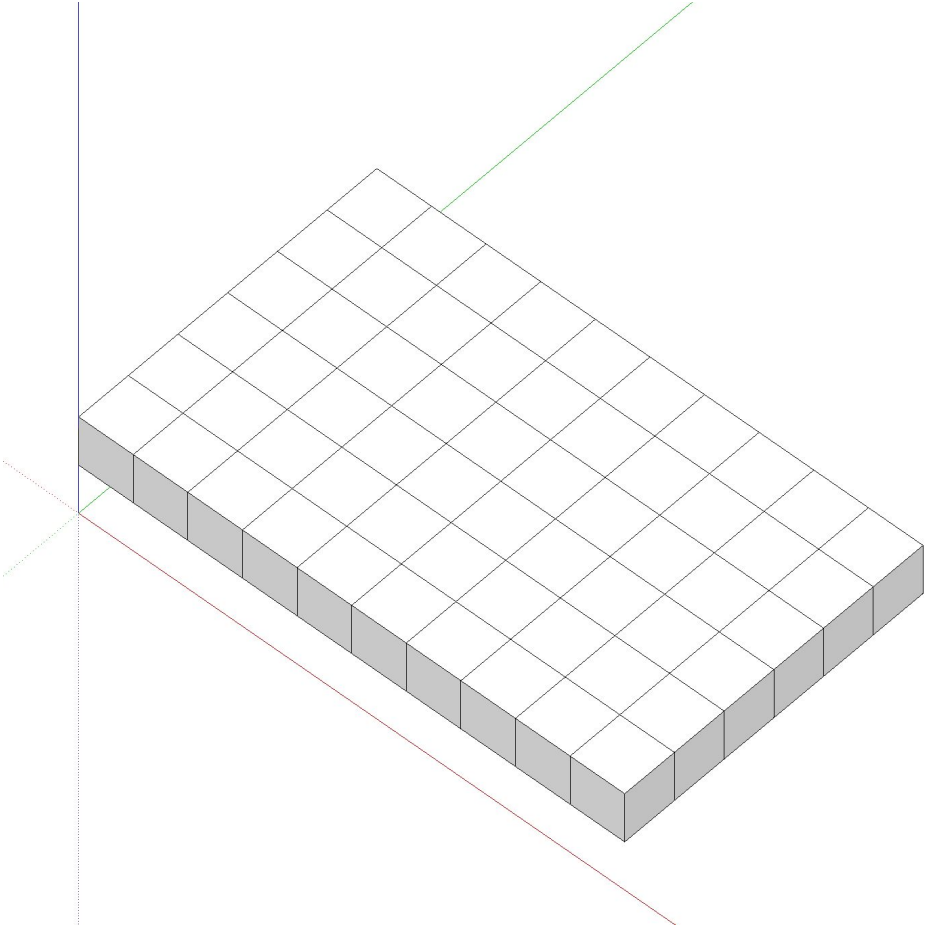
Start from "**mygeom_04_box_series.rb**" (series of boxes),

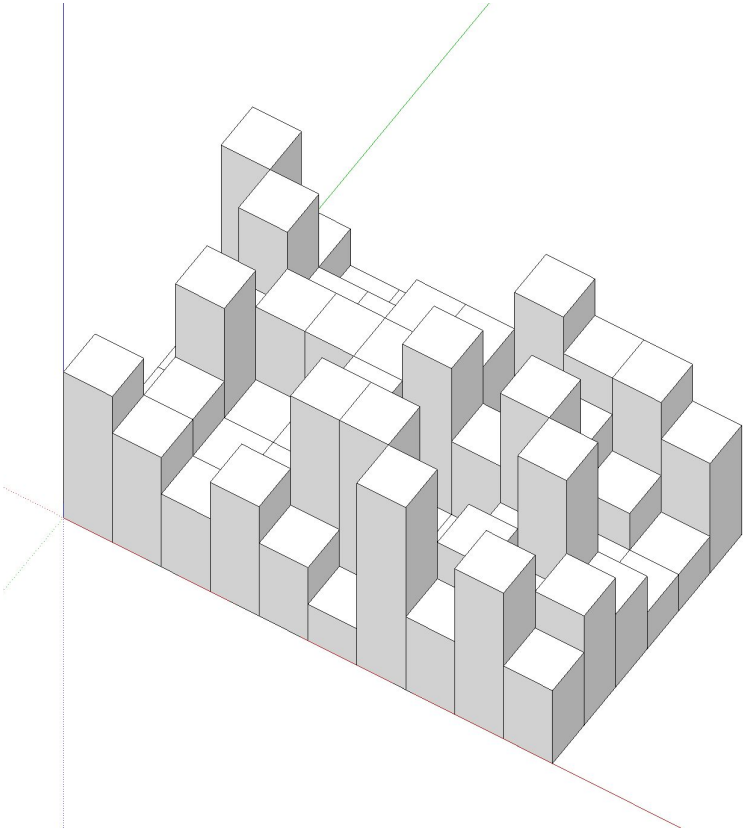
save as "**mygeom_06_grid.rb**"

Set the box ratio to 1:1:1

Create a loop within existing loop

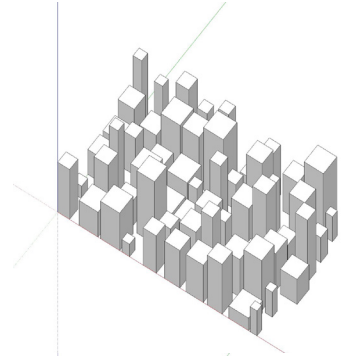
Set the origin point of the boxes to correspond with i and j value

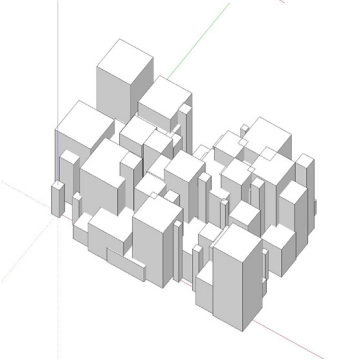




Put random value on its height

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

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod
for k in 0..6
  for j in 0..5
    for i in 0..4
      a = 25.mm
      width = a * rand(1..4)
      length = a * rand(1..4)
      height = a * rand(1..4)

      x = a * rand(0..4) + j * 1000.mm # add 1000.m grid
      y = a * rand(0..4) + k * 1000.mm # add 1000.m grid
      z = a * rand(0..4)

      origin = Geom::Point3d.new(x, y, z)

      pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
      pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
      pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
      pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

      group = ent.add_group
      group_ent = group.entities
      face = group_ent.add_face(pt1, pt2, pt3, pt4)

      if face.normal.z < 0
        face.reverse!
      end
      face.pushpull(height)
    end
  end
end
```

Generating choices

Start from “**08_random_box_composition.rb**”

Save as **“mygeom_07_box_compositions.rb”**

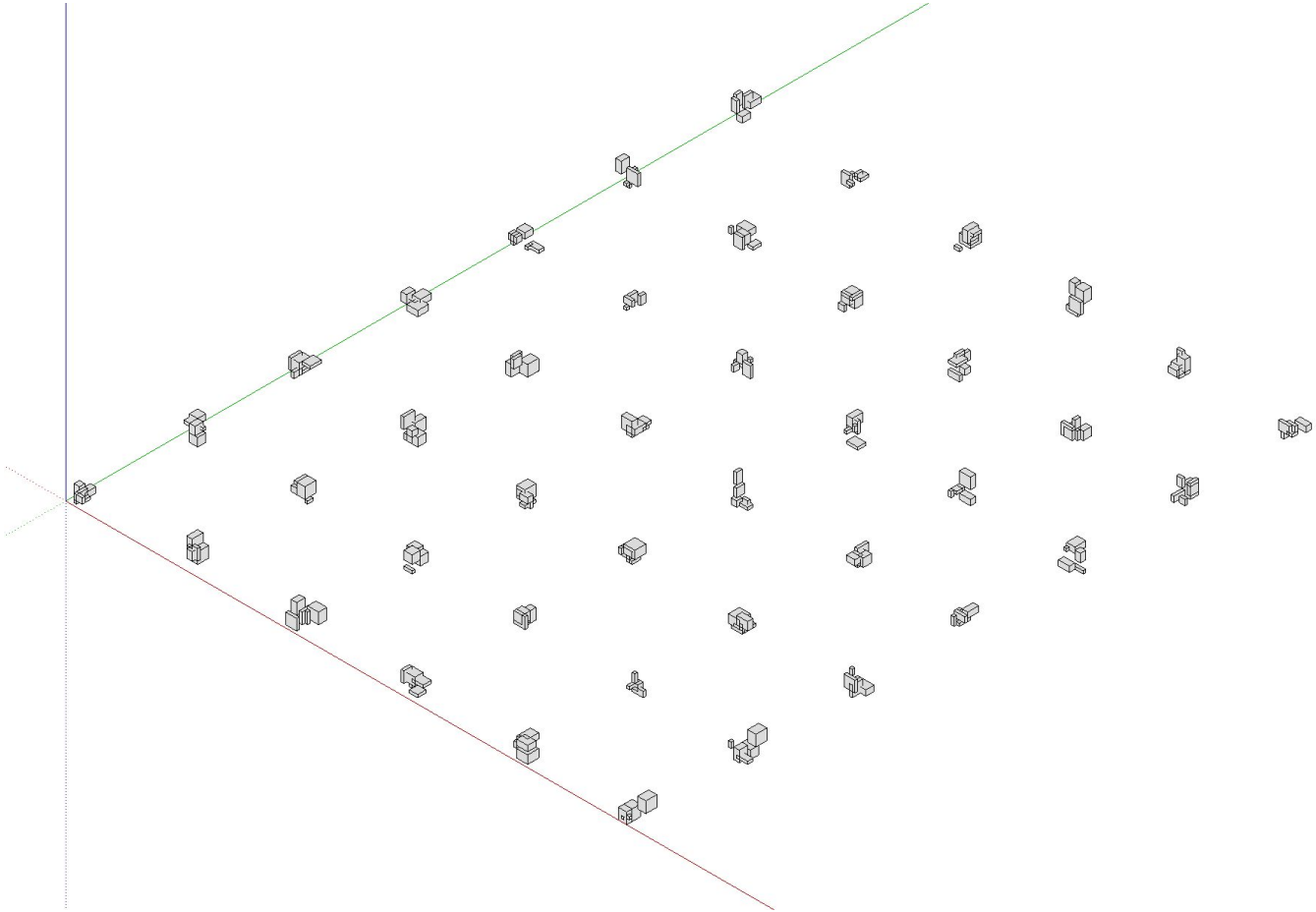
Create two loop j and k outside the existing loop i

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)
```

```ruby
mod = Sketchup.active_model # Open model
ent = mod.entities # All entities in model

origin = Geom::Point3d.new(0.mm, 0.mm, 0.mm)
a = 100.mm
width = a
length = a * 5
height = a * 3

pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

group = ent.add_group
group_ent = group.entities
face = group_ent.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
face.pushpull(height)

group_copy = group.copy                    #copy the group to group_copy
vector_tr = Geom::Vector3d.new(2 * a, 0, 0)  #set the vector according to a in x axis
tr = Geom::Transformation.translation(vector_tr) # create a translation base on vector
group_copy.transform!(tr)                   # apply the translation to the object
```

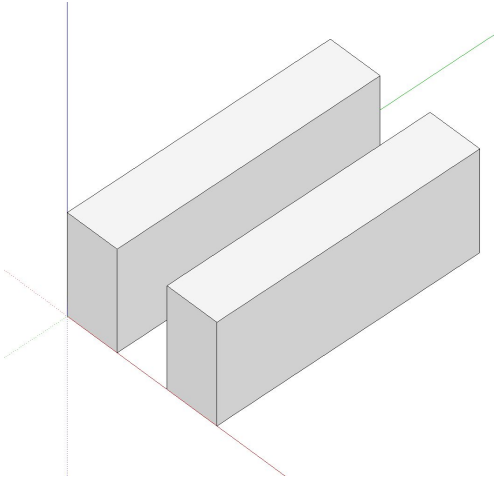
Translation

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

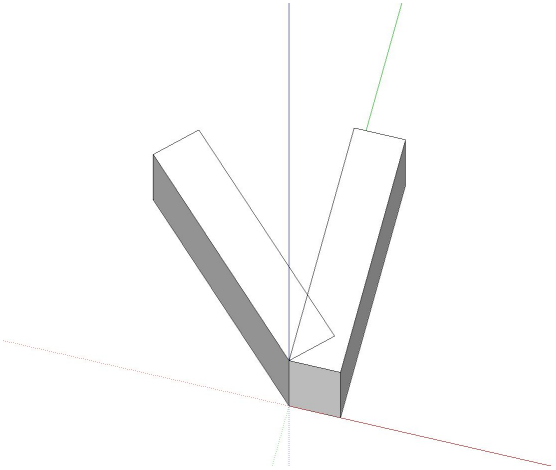


Rotation

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

Remove the translation portion from the script and add rotation transformation

```
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
```

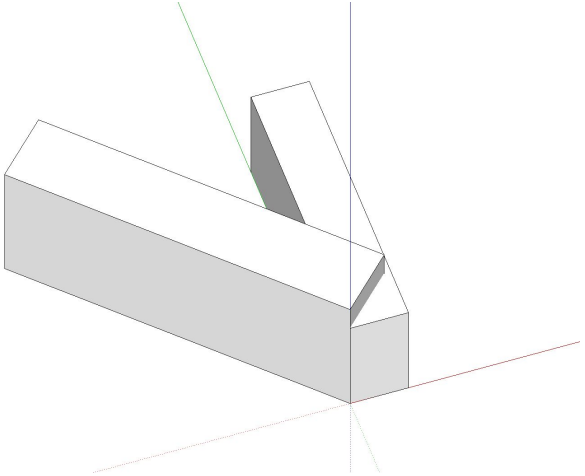


```ruby
group_copy = group.copy #copy the group to group_copy

axis = Geom::Vector3d.new(0, 0, 1) # set z axis for rotation axis
angle = 0.25 * Math::PI # set the angle as 1/4 PI rad or 45 degree
# note that ruby work in radians instead of degree
rotation_point = origin # set origin above as rotation point
tr = Geom::Transformation.rotation(rotation_point, axis, angle)
group_copy.transform!(tr) # apply the translation to the object
scaling_point = origin
tr_scale = Geom::Transformation.scaling(scaling_point, 1.25) # set scale Transformation
group_copy.transform!(tr_scale) # apply the scaling to the object

```

Scaling

Start from
“**mygeom_09_trans.rb**”
Save as
“**mygeom_10_trans.rb**”

Add scale transformation
to the rotated group



```ruby

for i in 0..9
  a = 100.mm
  width = a
  length = a * 5
  height = a * 3
  origin = Geom::Point3d.new(0, 0, i * height)  # the origin z is set according
                                                # height and counter i

  pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)
  if face.normal.z < 0  # this code is used to flip the face if not upward
    face.reverse!
  end
  face.pushpull(height)
  #create rotation Transformation
  axis = Geom::Vector3d.new(0, 0, 1) #z axis
  point = origin # sam as origin point
  angle = Math::PI * 0.25 * (i / 9.0) #incrementally according to i
  tr = Geom::Transformation.rotation(point, axis, angle) # create rotation
  group.transform!(tr) #apply rotation
end

```

Start from
“**mygeom_04_box_series.rb**”
Save as
“**mygeom_11_series_rotate.rb**”

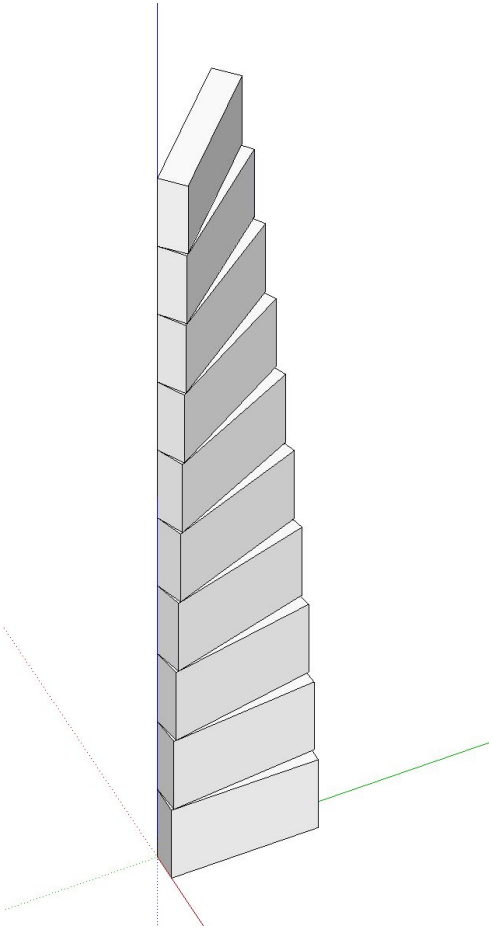
Create a series of box in z axis

Set the ratio accordingly

Apply rotation incrementally
from the bottom:

Remember:

Full rotation = 2 * Math::Plaa



```ruby

for j in 0..9
  for i in 0..9
    a = 100.mm
    width = a * 0.2
    length = a * 5
    height = a * 3
    origin = Geom::Point3d.new(j * a * 3, 0, i * height)  # the origin z is set according
                                                          # height and counter i
                                                          # set the origin z to corresond
                                                          # with j in x axis

    pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

    group = ent.add_group
    group_ent = group.entities
    face = group_ent.add_face(pt1, pt2, pt3, pt4)
    if face.normal.z < 0  # this code is used to flip the face if not upward
      face.reverse!
    end
    face.pushpull(height)
    #create rotation Transformation
    axis = Geom::Vector3d.new(0, 0, 1) #z axis
    point = origin # sam as origin point
    angle = Math::PI * 0.5 * ((i * j) / 81.0) #incrementally according to i * j
    tr = Geom::Transformation.rotation(point, axis, angle) # create rotation
    group.transform!(tr) #apply rotation
  end
end

```

Start from
"**mygeom_11_series_rotate.rb**"
Save as "**mygeom_12_grid_rotate.rb**"

Create a series of box in z axis

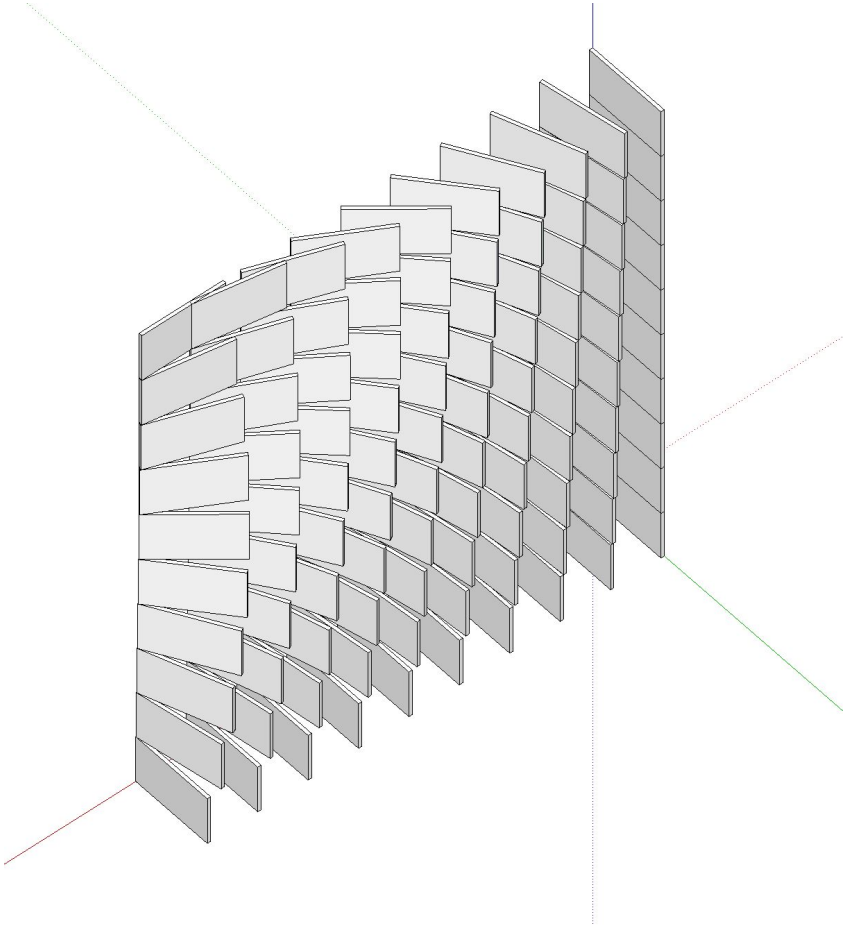
Set the ratio accordingly

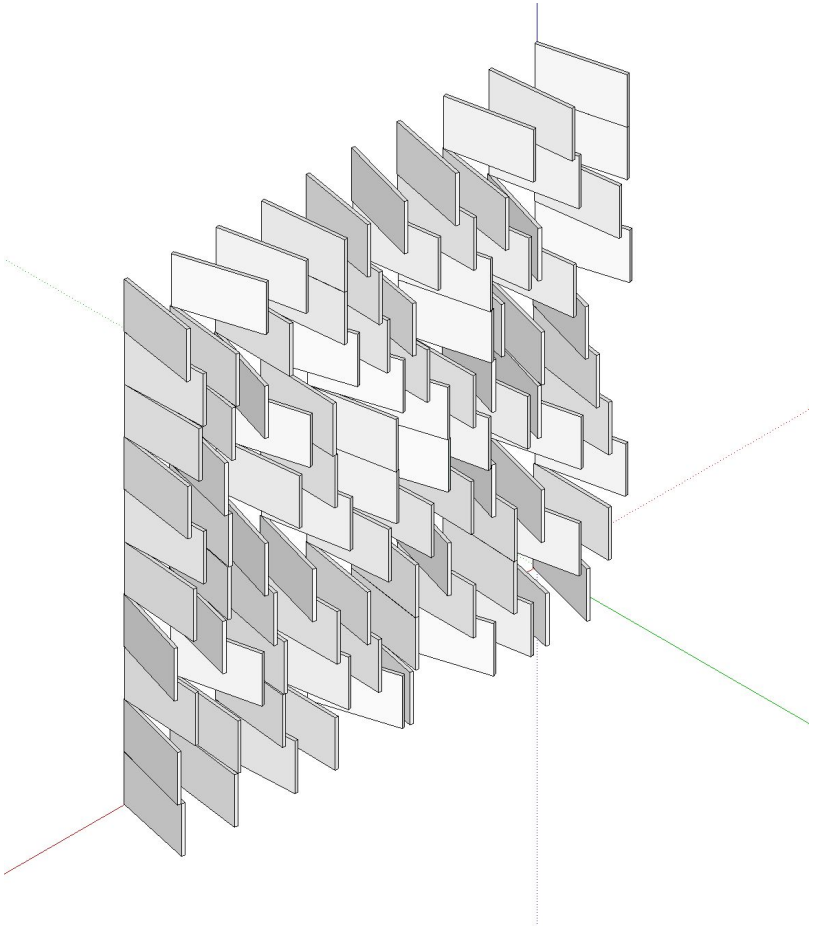
Create a new outer loop j

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

Remember:

Full rotation = 2 * Math::PI

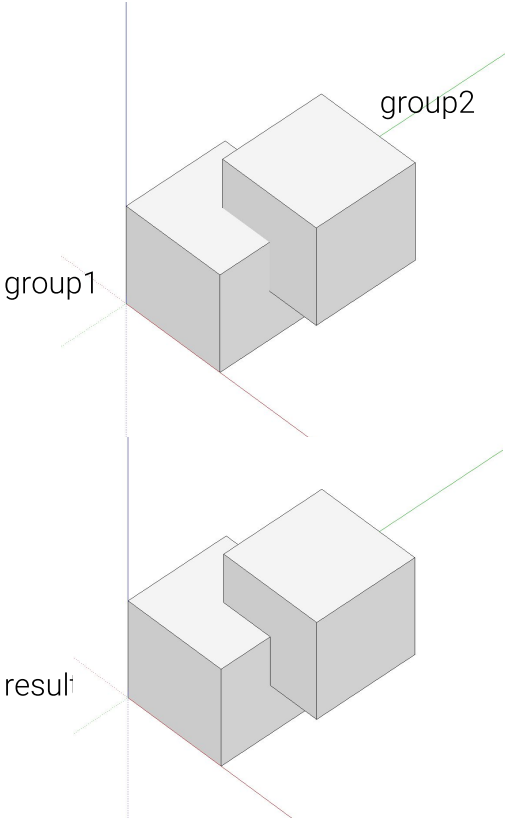




exercise_12_random_rotate.rb

Boolean Operation

result =
group2.union(group1)



```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

# this is a basic code to make a simple box

a = 100.mm      # define the module dimension
width = a       # width of the box
length = a  # Length of the box
height = 0.2 * a # height of the box

x = 0.mm        # define the x position
y = 0.mm        # define the y position
z = 0.mm        # define the z position

origin = Geom::Point3d.new(x, y, z) # create a point for origin

#define points for a rectangle in respect to origin point
pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

#create a group_ box group
box_group = ent.add_group
# create a rectanglar face
face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
# pushpull the face to create the box
face.pushpull(height)
```

```ruby
# 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)
# union box_group with box_copy
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

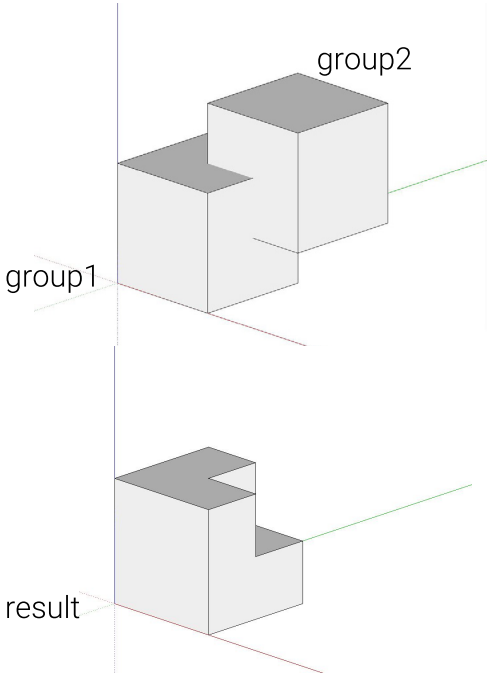
# union box_group with box_copy
union_result = box_copy.union(box_group)

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

group_union = nil

for i in 0..4
  a = 25.mm
  width = a * rand(1..4)
  length = a * rand(1..4)
  height = a * rand(1..4)

  x = a * rand(0..4)
  y = a * rand(0..4)
  z = a * rand(0..4)

  origin = Geom::Point3d.new(x, y, z)

  pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)

  if face.normal.z < 0
    face.reverse!
  end
  face.pushpull(height)
  if group_union == nil
    group_union = group
  else
    if group_union.manifold?
      group_union  = group_union.union(group)
    end
  end
end
```

Start from “**08_random_box_composition**”

Save as “**mygeom_14_series_union.rb**”

result = group2.subtract(group1)



```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

# this is a basic code to make a simple box

a = 100.mm      # define the module dimension
width = a       # width of the box
length = a  # length of the box
height = 0.2 * a # height of the box

x = 0.mm        # define the x position
y = 0.mm        # define the y position
z = 0.mm        # define the z position

origin = Geom::Point3d.new(x, y, z) # create a point for origin

#define points for a rectangle in respect to origin point
pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

#create a group_ box group
box_group = ent.add_group
# create a rectanglar face
face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
# pushpull the face to create the box
face.pushpull(height)
```

```ruby

 # 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)
 # subtract box_group with box_copy
 substract_result = box_copy.union(box_group)
```

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

Changes the union method to subtract

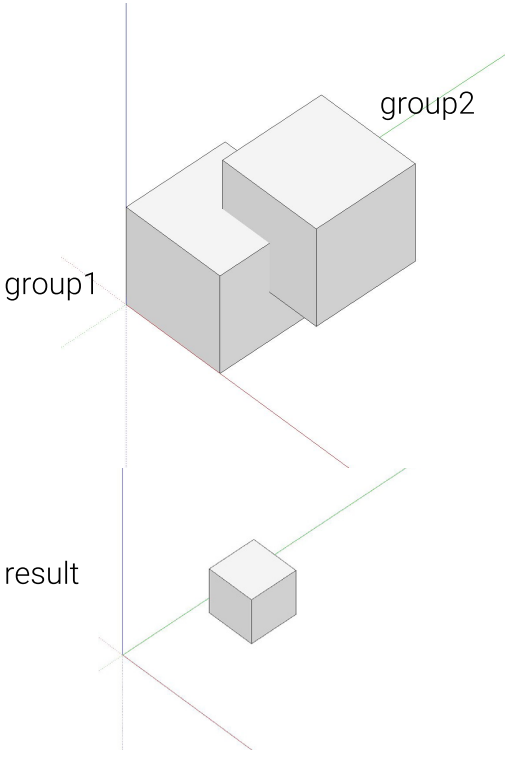
subtract box_group with box_copy
~~union_result = box_copy.union(box_group)~~
subtract_result = box_copy.union(box_group)

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

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

    # define the module dimension
width = 100.mm       # width of the box
length = 100.mm  # length of the box
height = 200.mm  # height of the box

x = 0.mm        # define the x position
y = 0.mm        # define the y position
z = 0.mm        # define the z position

origin = Geom::Point3d.new(x, y, z) # create a point for origin

#define points for a rectangle in respect to origin point
pt1 = Geom::Point3d.new(origin.x, origin.y, origin.z)
pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

#create a group_ box group
box_group = ent.add_group
# create a rectanglar face
face = box_group.entities.add_face(pt1, pt2, pt3, pt4)
if face.normal.z < 0  # this code is used to flip the face if not upward
  face.reverse!
end
# pushpull the face to create the box
face.pushpull(height)
```

```ruby
for i in 0..15
  a = 25.mm
  width = a * rand(1..4)
  length = a * rand(1..4)
  height = a * rand(1..4)

  x = a * rand(-4..4)
  y = a * rand(-4..4)
  z = a * rand(0..4)

  origin = Geom::Point3d.new(x, y, z)

  pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
  pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
  pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
  pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

  group = ent.add_group
  group_ent = group.entities
  face = group_ent.add_face(pt1, pt2, pt3, pt4)

  if face.normal.z < 0
    face.reverse!
  end
  face.pushpull(height)
  box_group = group.subtract(box_group)
  if !box_group.manifold?
    break
  end
end
```

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)

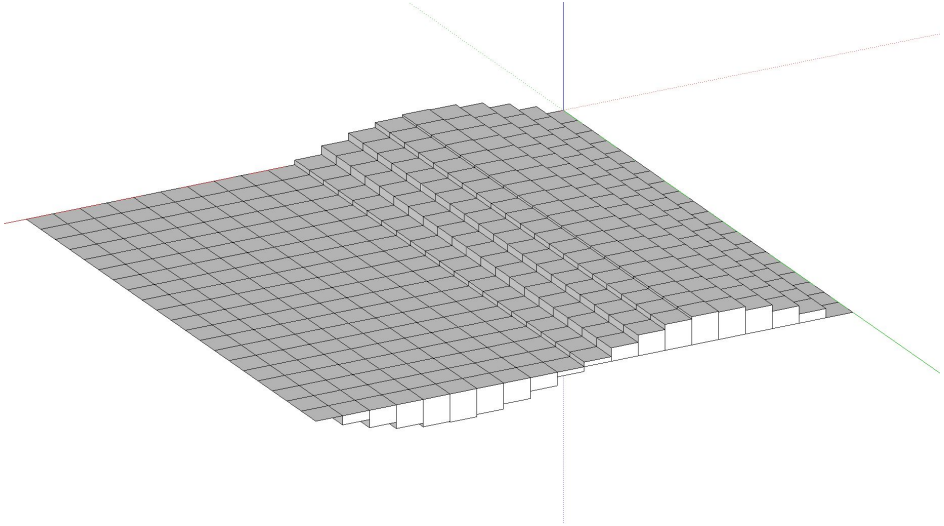
sin = Math.sin(degree)

tan = Math.sin(degree)

```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

for i in 0..19
  for j in 0..19 # create inner loop with j counter
    a = 100.mm
    width = a #set the ratio to 1:1:1
    length = a
    # add sin function to box height
    height = a * Math.sin(2 * i / 19.0 * Math::PI)

    #set the j to correspond with y coordinate
    origin = Geom::Point3d.new(i * width, j * length , 0)

    pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

    group = ent.add_group
    group_ent = group.entities
    face = group_ent.add_face(pt1, pt2, pt3, pt4)

    if face.normal.z < 0
      face.reverse!
    end

    face.pushpull(height)
  end
end

```

Start from “**mygeom_06_grid.rb**”
Save as “**mygeom_17_math_sin_i.rb**”

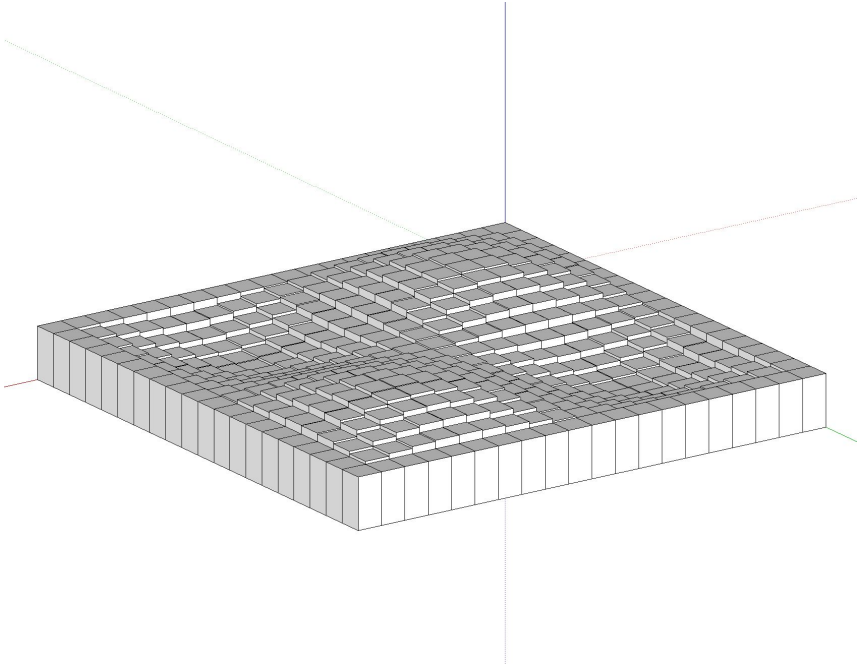
# add sin function to box height
height = a * Math.sin(2 * i / 19.0 * Math::PI)



```ruby
mod = Sketchup.active_model # open sketchup active_model
ent = mod.entities # accessing the entities in mod

for i in 0..19
  for j in 0..19 # create inner loop with j counter
    a = 100.mm
    width = a #set the ratio to 1:1:1
    length = a
    # add sin function to box height
    height = a * Math.sin(2 * i / 19.0 * Math::PI) * Math.sin(2 * j / 19.0 * Math::PI) + 200.mm

    #set the j to correspond with y coordinate
    origin = Geom::Point3d.new(i * width, j * length , 0)

    pt1 = Geom::Point3d.new(origin.x , origin.y, origin.z)
    pt2 = Geom::Point3d.new(origin.x + width, origin.y, origin.z)
    pt3 = Geom::Point3d.new(origin.x + width, origin.y + length, origin.z)
    pt4 = Geom::Point3d.new(origin.x, origin.y + length, origin.z)

    group = ent.add_group
    group_ent = group.entities
    face = group_ent.add_face(pt1, pt2, pt3, pt4)

    if face.normal.z < 0
      face.reverse!
    end

    face.pushpull(height)
  end
end
```

Start from
“**mygeom_17_math_sin_i**”
Save as
“**mygeom_18_math_sin_ij.rb**”

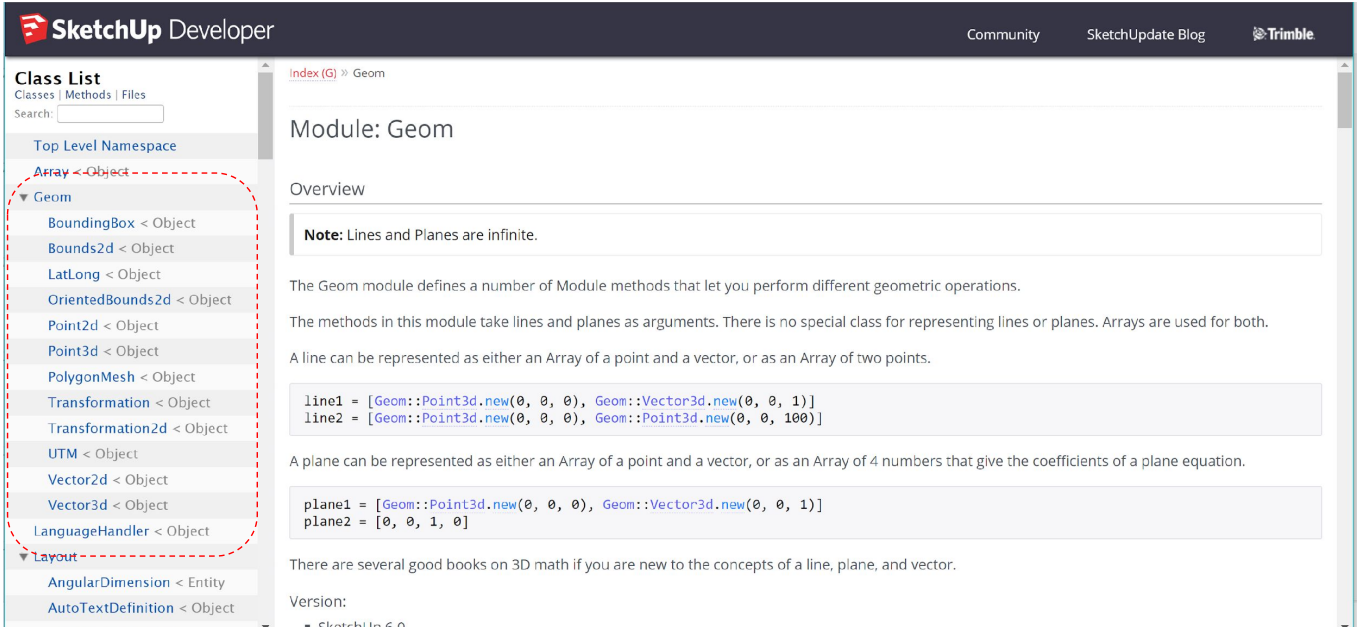
add sin function to box height
according to i and j values



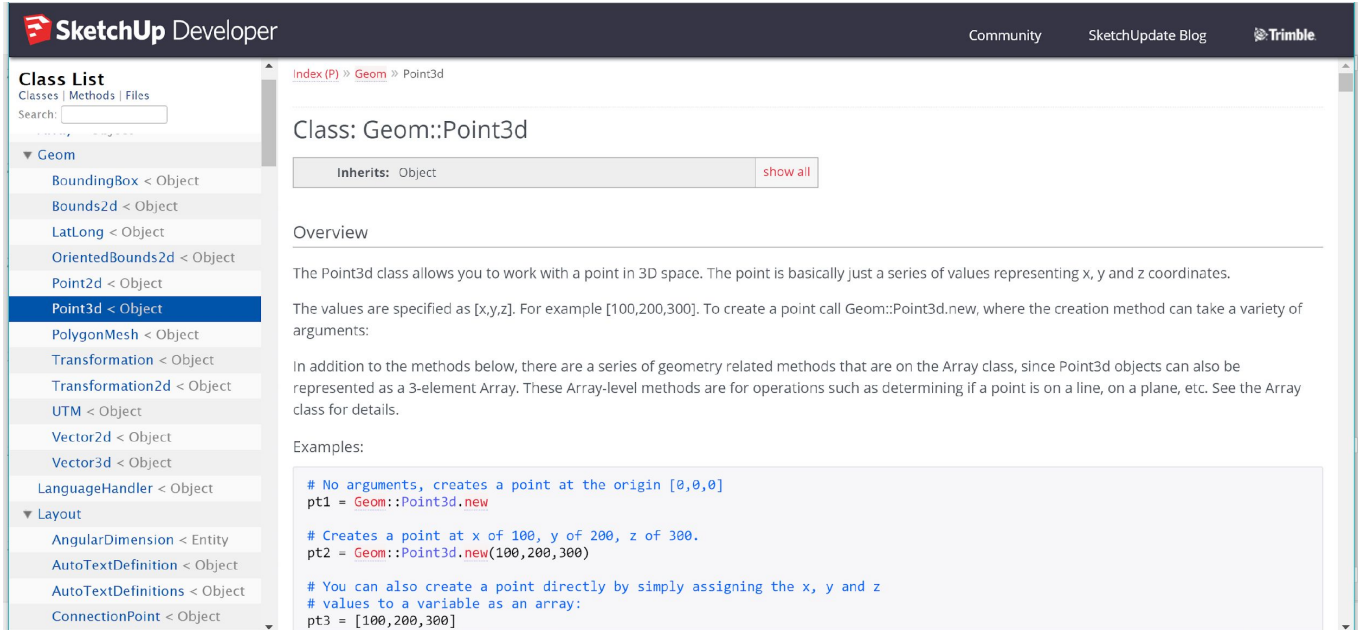
Understanding Sketchup Ruby API

https://ruby.sketchup.com

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



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



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