BOOK 1 · SIX NIGHTS AT NEKO RAMEN

THE SUSHI CAT CODE BOOK

A paper adventure in Luau, the language of Roblox





This book belongs to chef

_____



Your game runs on this

You wrote a pitch for a game where cats cook and defend their customers from dogs. Every idea in that pitch needs code. Here is what your own sentences turn into.

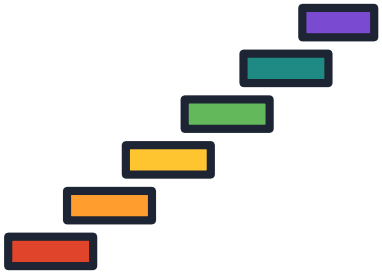
YOUR PITCH SAYS	THE CODE IDEA BEHIND IT	YOU LEARN IT ON
“You can buy passes that cost from 79 to 499 Robux.”	The game must **remember** numbers: coins, prices.	**Night 1** Variables
“You can keep upgrading things, from ramen machine to sushi.”	The game must **decide**: does the player have enough coins?	**Night 2** If
“Every day the dogs get better.”	The game must **repeat** things, day after day.	**Night 3** Loops
“…cook food and make the customers happy.”	Jobs you do many times get a **name**, so you write them once.	**Night 4** Functions
“…you can buy weapons and bodyguard cats.”	Every cat is a **thing with its own facts**: name, strength.	**Night 5** Objects
“…defend the restaurant with a frying spatula.”	The game must **react** when the player clicks or touches.	**Night 6** Events

How this book works

1. **Read the story.** Each night at Neko Ramen brings one new idea.
2. **Do the exercises with a pencil.** In this book **you are the computer**. You read the code and do exactly what it says.
3. **Check the Answer Scroll** at the back after each exercise. A wrong answer is useful: find the line where you and the computer went different ways.
4. **Collect the secret ingredient.** Every night ends with a puzzle. Its answer is a number. Write it on the Recipe Card.
5. **Use your computer tickets.** After Night 3 and after Night 6 you type real code into Roblox Studio. Your six ingredients go into it.

What you build at the end

A rainbow staircase into the sky, built by a loop. A ramen pot that pays coins when you click it. A shop that sells upgrades, checks your coins and raises its price. About 60 lines of Luau, and you will be able to read every one of them.



Luau (say “LOO-ow”) is the language every Roblox game is written in.

The crew



Chef Miso
Owner of Neko Ramen. Explains things once.



The Brain
The kitchen computer. Fast, obedient, understands nothing.



Tofu
Bodyguard cat. Strength 8. Always hungry.



The Bark Gang
They want the ramen. They get better every day.

NIGHT 1

The jars on the shelf

Variables: how a program remembers things.



It is your first night at Neko Ramen. Chef Miso points a paw at a grey box humming beside the rice cooker.

"That is the Brain. It runs the whole restaurant. It is very fast and very obedient, and it understands nothing. It does exactly what the code says, one line at a time, from the top to the bottom. Tonight you learn how it remembers."

The Brain remembers with jars. Each jar has a label on the outside and one thing inside. Programmers call a jar a **variable**.

```
local coins = 10
coins = coins + 5
print(coins)
```

`local coins = 10` Make a new jar. Write **coins** on the label. Put 10 inside.

`coins = coins + 5` Do the right side first: look in the jar (10) and add 5. That makes 15. Now put 15 into the jar. The old 10 is gone.

`print(coins)` Show what is in the jar on the screen. The screen shows 15.



A jar holds one thing. When something new goes in, the old thing is thrown away.

KITCHEN RULE

The sign `=` does not mean "is equal to". It means **"put into"**. First work out the right side. Then put the result into the jar on the left.

Three more things the Brain knows. Words go inside quotes: `local chef = "Miso"`. Numbers have no quotes. A line that starts with `--` is a note for humans, and the Brain skips it. Capital letters matter: `coins` and `Coins` are two different jars.

1.1 **Be the Brain. Fill in the jars after each line. The first row is done.**

```
1 local coins = 10
2 local bowls = 3
3 coins = coins + 5
4 bowls = bowls - 1
5 coins = coins + bowls
6 print(coins)
```

after line	coins	bowls
1	10	no jar yet
2		
3		
4		
5		

The screen shows: _____

1.2 **What does the screen show? Write both lines.**

```
1 local dish = "ramen"
2 local price = 8
3 price = price * 2
4 dish = "sushi"
5 print(dish)
6 print(price)
```

OUTPUT

The star `*` means “times”. The slash `/` means “divided by”.

1.3 **A trap. Most people get this wrong the first time. Go slowly, line by line.**

```
1 local a = 4
2 local b = 9
3 a = b
4 b = a
5 print(a)
6 print(b)
```

after line	a	b
1		
2		
3		
4		

1.4 **Bug hunt. Each line has one mistake. Circle it and write the line correctly.**

`locl tips = 5` _____

`local chef = Miso` _____

`local tips = 5` then `print(Tips)` _____

1.5 **Your turn to write. Make a jar called `tips` with 0 in it. Add 12 to it. Show it on the screen.**

1.6 **The swap, fixed. In 1.3 the jars did not swap, because the 4 was thrown away too early. A third jar keeps it safe. Fill in the table.**

```
1 local a = 4
2 local b = 9
3 local spare = a
4 a = b
5 b = spare
```

after line	a	b	spare
1			
2			
3			
4			
5			

Secret ingredient 1: the price

Be the Brain. What does the screen show?

```
1 local price = 5
2 local coins = 40
3 price = price * 4
4 coins = coins - price
5 price = price + coins / 4
6 print(price)
```



NIGHT 2

Who gets ramen?

If, else: how a program makes a decision.



On the second night there is a queue down the street. Chef Miso looks through the window.

"Not everyone gets ramen. Some have no coins. And that one in the long coat has a tail that wags. The Brain must learn to choose."

To choose, the Brain asks a question that has only two answers: **true** or **false**. Then it runs one piece of code or the other. It never runs both.

```
local coins = 30
local price = 12
if coins ⩾ price then
    coins = coins - price
    print("Enjoy your ramen!")
else
    print("Sorry, not enough coins.")
end
print(coins)
```

`if coins ⩾ price then` Ask: is coins bigger than or the same as price? Here `30` is bigger than `12`, so the answer is **true**.

`else` If the answer was true, run the lines above `else` and skip the lines below it. If it was false, do the opposite.

`end` The choice is over. Every `if` needs an `end`. The Brain carries on with the next line, whatever the answer was.

QUESTION	IT ASKS
`a == b`	Are they the same?
`a ~= b`	Are they different?
`a > b` `a < b`	Bigger? Smaller?
`a ⩾ b`	Bigger or the same?
`a ⩽ b`	Smaller or the same?

KITCHEN RULE

One `=` puts something into a jar. Two `==` ask a question. Mixing them up is the most common mistake in all of programming.

You can join two questions. With `and`, both must be true. With `or`, one true answer is enough.

2.1 **True or false? The jars are:** `coins = 12`, `price = 8`, `name = "Tofu"`.

`coins > price`	_____	`name == "tofu"`	_____
`coins == 12`	_____	`coins > 10 and price > 10`	_____
`price ~= 8`	_____	`coins > 10 or price > 10`	_____
`coins >= 20`	_____		

2.2 **Look at the code at the top of this night. A new customer comes in with `coins = 9`. What does the screen show now? Write both lines.**

OUTPUT

2.3 **The dogs get better every day. With `elseif` the Brain can ask a second question when the first answer was false. It stops at the first true answer.**

```lua
local weapon = "none"
if day >= 5 then
    weapon = "soy-sauce pistol"
elseif day >= 3 then
    weapon = "stick"
else
    weapon = "bark"
end
```

day	weapon
1	
3	
4	
5	
7	

Extra spicy: Chef Miso swaps the two questions, so `day >= 3` is asked first. What weapon do the dogs have on day 5 now? _____ Why? _____

2.4 **The lines fell on the floor. Number them 1 to 6 in the right order. The code must close the door on dogs and welcome everyone else.**

_____	`end`	_____	`else`
_____	`    print("Welcome!")`	_____	`    print("Door closed!")`
_____	`if animal == "dog" then`	_____	`local animal = "dog"`

2.5 **Your turn to write. If `coins` is 100 or more, show “Hire Tofu!”. Otherwise show “Keep saving”.**

Secret ingredient 2: the red

Careful. There are two choices, one after the other.

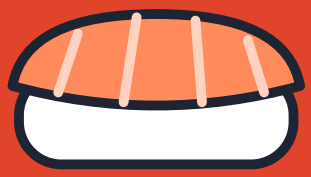
```lua
local red = 200
local day = 6
if day > 6 then
    red = red + 100
elseif day == 6 then
    red = red + 55
else
    red = 0
end
if red > 255 then
    red = 255
end
print(red)
```



NIGHT 3

Twenty hungry customers

Loops: how a program does something many times.



A bus stops outside. Twenty customers walk in at once. Chef Miso starts typing `print("Ramen!")` again and again. After the sixth line his paw hurts.

You show him a better way. A **loop** tells the Brain to run the same lines many times.

```
1  local bowls = 0
2  for i = 1, 4 do
3      bowls = bowls + 2
4  end
5  print(bowls)
```

`for i = 1, 4 do` Make a counting jar called **i**. Start it at 1.

`end` Add 1 to **i**. If **i** is still 4 or less, jump back up and run the inside lines again. If not, carry on below.

The lines between `do` and `end` run four times: once with `i = 1`, then 2, then 3, then 4. You may use **i** inside the loop like any other jar. You may also give it a better name, such as `day` or `seat`.

Glue. Two dots `..` glue text and numbers together. If `seat` holds 3, then `"Ramen for seat " .. seat` makes the text **Ramen for seat 3**.

3.1 **Be the Brain for the loop above. Fill in the jar after each round.**

round (i)	bowls after
1	*2*
2	
3	
4	

The screen shows: _____

3.2 **This loop uses the counting jar inside. What does the screen show?**

```
local total = 0
for i = 1, 5 do
    total = total + i
end
print(total)
```

i	total after
1	
2	
3	
4	
5	

3.3 **Write exactly what the screen shows.**

```
for seat = 1, 3 do
    print("Ramen for seat " .. seat)
end
```

OUTPUT

3.4 **A loop with a choice inside. How many dogs are outside after seven days?**

```
local dogs = 0
for day = 1, 7 do
    if day ⩾ 5 then
        dogs = dogs + 3
    else
        dogs = dogs + 1
    end
end
print(dogs)
```

day	dogs after
1	
2	
3	
4	
5	
6	
7	

3.5 **A `while` loop has no counting jar. It repeats as long as its question is true. How many clicks until you have 50 coins or more?**

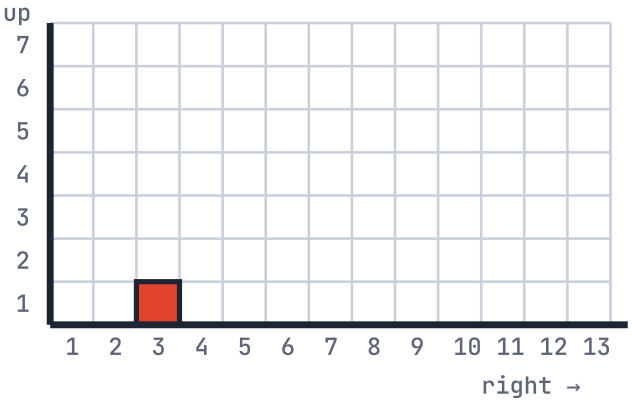
```
local coins = 0
local clicks = 0
while coins < 50 do
    coins = coins + 8
    clicks = clicks + 1
end
print(clicks)
```

Coins after each round:

The screen shows: _____

3.6 **Build with a loop. For each round, colour one step on the grid: go `i * 2` squares to the right and `i` squares up, then colour that square. The first step is done.**

```
for i = 1, 6 do
    -- colour the square at
    -- right: i * 2
    -- up:    i
end
```



This is how your staircase in Roblox is made. One loop, and each step uses **i** to find its place.

Secret ingredient 3: the steps

```
local steps = 2
for i = 1, 3 do
    steps = steps * 2
end
steps = steps + 4
print(steps)
```

WRITE IT HERE

1

You have earned Computer Ticket 1

Go to **Computer Mission 1** near the end of the book. You will build the rainbow stairs with a loop. Then come back for Night 4.

NIGHT 4

Write it down once

Functions: how to give a job a name.



Chef Miso has worked out the bill three hundred times tonight. Bowls times eight. Bowls times eight. His whiskers hang down.

"Enough. I will write the recipe on a card, once. After that I only shout its name."

A recipe card with a name is called a **function**. You write it once and then use it as often as you like.

```
1 local function bill(bowls)
2     local total = bowls * 8
3     return total
4 end
5
6 print(bill(3))
```

`local function bill(bowls)` Make a recipe card called **bill**. It needs one thing to work with. Inside the card that thing is called **bowls**.

`return total` Hand the result back to whoever asked.

`bill(3)` Use the card now, with 3 for **bowls**. This is named **calling** the function. The result is 24, so the screen shows 24.

KITCHEN RULE

Writing the card does not cook anything. Lines 1 to 4 only **save** the recipe. Nothing happens until a line calls it by name with brackets: `bill(3)`.

4.1 **Use the `bill` card. What does each call hand back?**

`bill(1)` _____ `bill(5)` _____ `bill(0)` _____

4.2 **A card can need two things. They are filled in the same order as you give them.**

```
1 local function change(paid, cost)
2     return paid - cost
3 end
```

`change(50, 24)` _____

Extra spicy: `change(bill(2), 10)`

_____

Work out the inside call first.

4.3 **A card can hand back true or false. This one is the heart of every shop.**

```lua
local function canBuy(coins, price)
    if coins ⩾ price then
        return true
    else
        return false
    end
end
```

`canBuy(30, 25)` _____

`canBuy(10, 25)` _____

`canBuy(25, 25)` _____

4.4 **In which order does the Brain show things? Write the three lines of the screen.**

```lua
local function greet(name)
    print("Welcome, " .. name)
end
print("Doors open")
greet("Tofu")
greet("Miso")
```

OUTPUT

4.5 **Your turn to write. Make a function called `tip`. It needs one thing, `total`. It hands back `total` divided by 10.**

```lua

```

Test your card: `tip(80)` hands back _____

Secret ingredient 4: the green

The same card is used three times. Each time, the old result goes back in.

```lua
local function upgrade(speed)
    return speed * 2 + 10
end
local green = upgrade(5)
green = upgrade(green)
green = upgrade(green) + 10
print(green)
```

WRITE IT HERE

NIGHT 5

The bodyguards

Objects: things with their own facts.



Night five. The lights flicker. Outside, the Bark Gang is back, and tonight they carry soy-sauce pistols. Chef Miso picks up the phone.

“Send me bodyguards. Big ones.”

A bodyguard cat is more than one number. He has a name, a strength and a count of wins. You could use three jars, but with ten cats that is thirty loose jars. So you put the jars for one cat together in one box. That box is called an **object**. The jars inside it are called its **properties**.

```lua
local tofu = { name = "Tofu", strength = 8, wins = 0 }

print(tofu.name)
tofu.strength = tofu.strength + 2
```



tofu

name	"Tofu"
strength	8
wins	0

The curly brackets `{ }` make the box. A dot means **“look inside”**. So `tofu.strength` means: open the box called tofu and find the jar called strength.

A blueprint makes many cats

Typing every cat by hand is slow. A function can be a **blueprint**: give it a name and a strength, and it hands back a finished cat.

```lua
local function newGuard(name, strength)
    local guard = { name = name, strength = strength, wins = 0 }
    return guard
end

local tofu = newGuard("Tofu", 8)
local wasabi = newGuard("Wasabi", 5)
```

KITCHEN RULE

Programmers call this **object-oriented programming**, or OOP. It means: your program is made of things. Each thing has its own properties. Things of the same kind come from one blueprint. Every bodyguard, every dog and every sushi machine in your game will be an object.

5.1 **Tofu and Wasabi are fresh from the blueprint above. Then these three lines run. Fill in both cards.**

```
wasabi.strength = wasabi.strength + tofu.strength
tofu.name = "Big Tofu"
tofu.wins = tofu.wins + 1
```



tofu

name	_____
strength	_____
wins	_____



wasabi

name	_____
strength	_____
wins	_____

5.2 **Start again with fresh cats: Tofu has strength 8, Wasabi has strength 5, both have 0 wins. What does each fight hand back?**

```
local function fight(guard, dogPower)
    if guard.strength > dogPower then
        guard.wins = guard.wins + 1
        return "dog runs away"
    else
        return "guard needs help"
    end
end
```

`fight(tofu, 6)`

_____

`fight(wasabi, 5)`

_____

`fight(tofu, 7)`

_____

After all three, `tofu.wins` is

_____

In Roblox, everything is an object

Every block, every player and every model you bring in from Blender is an object. Its properties are the list you see in the Properties window in Roblox Studio. Code can change them with a dot, the same way you changed Tofu.

```
local plate = Instance.new("Part")
plate.Size = Vector3.new(6, 1, 6)
plate.Position = Vector3.new(0, 10, 0)
plate.Color = Color3.fromRGB(255, 0, 0)
plate.Anchored = true
plate.Parent = workspace
```

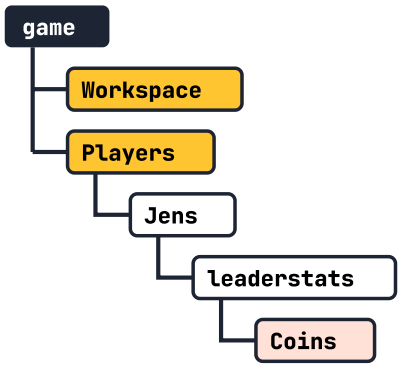
`Instance.new("Part")`	Roblox's own blueprint. It hands back a new block.
`Vector3.new(6, 1, 6)`	Three numbers that belong together. For Size: wide, tall, deep. For Position: sideways, up, forward.

`Color3.fromRGB(255, 0, 0)`	A colour mixed from red, green and blue light. Each number goes from 0 (none) to 255 (full).
`plate.Anchored = true`	The block stays where it is and does not fall.
`plate.Parent = workspace`	Put the block into the world. Before this line it exists, but nobody can see it.

5.3 **Mix the colours. Draw a line from each code to its colour.**

`Color3.fromRGB(255, 0, 0)`	white
`Color3.fromRGB(0, 0, 0)`	yellow
`Color3.fromRGB(255, 255, 255)`	blue
`Color3.fromRGB(255, 255, 0)`	red
`Color3.fromRGB(0, 0, 255)`	black

5.4 **Objects live inside other objects. Use dots to write the path. The first one is done.**



From game to Jens:
`game.Players.Jens`

From game to Jens' Coins:

```
----------
```

The number inside Coins is a property called `Value`. Write the whole path to it:

```
----------
```

Secret ingredient 5: the blue

```lua
local pot = { name = "RamenPot", heat = 10, blue = 0 }
for i = 1, 3 do
    pot.heat = pot.heat + 10
end
if pot.heat ⩾ 40 then
    pot.blue = pot.heat
end
print(pot.blue)
```

WRITE IT HERE

NIGHT 6

Ring the bell

Events: how a program reacts when something happens.



The bodyguards won. The dogs ran. But the Brain is hot and tired. All night it has asked the same question a million times a second: has someone clicked the pot? Has someone clicked the pot?

"Stop asking," says Chef Miso. He puts a small brass bell on the counter. "When the bell rings, cook. Until then, rest."

In Roblox the bell is called an **event**. Roblox rings it when something happens: a click, a touch, a player joining. You tell the Brain which function to run when it rings.

```
1  local function cook(player)
2      print(player.Name .. " cooked ramen")
3  end
4
5  pot.ClickDetector.MouseClick:Connect(cook)
6  print("Kitchen is open")
```

`pot.ClickDetector`	An object inside the pot. It notices clicks.
`.MouseClick`	The bell. It rings each time a player clicks the pot.
`:Connect(cook)`	A promise: "each time this bell rings, run **cook**." Notice there are no brackets after cook. You hand over the card. You do not call it.

KITCHEN RULE

`Connect` does not run the function. It only makes the promise, and the Brain moves straight on to the next line. Later, each ring runs the function once. Roblox also tells the function **who** clicked: that is the `player`.

6.1 **The code above runs. Later Jens clicks the pot two times. Write all three lines of the screen, in order.**

OUTPUT

6.2 **The big one. This is a whole shop. Be the Brain through one evening. After each thing that happens, fill in the jars.**

```lua
local coins = 0
local coinsPerClick = 1
local price = 5

local function cook()
    coins = coins + coinsPerClick
end

local function buy()
    if coins ≥ price then
        coins = coins - price
        coinsPerClick = coinsPerClick + 1
        price = price * 2
        print("Upgrade!")
    else
        print("Not enough coins")
    end
end

pot.ClickDetector.MouseClick:Connect(cook)
shop.ClickDetector.MouseClick:Connect(buy)
```

what happens	coins	coinsPerClick	price	screen shows
start	*0*	*1*	*5*	*nothing*
pot clicked 6 times				
shop clicked				
pot clicked 3 times				
shop clicked				
pot clicked 2 times				
shop clicked				

You have just run the upgrade system of a tycoon game in your head. “From ramen machine to sushi” works exactly like this.

Secret ingredient 6: the growth

That night the bell rings five times.

```lua
local growth = 0

local function ring()
    if growth < 2 then
        growth = growth + 1
    end
end

bell.MouseClick:Connect(ring)
```



What is in the jar `growth` at the end of the night?

The Recipe Card

Copy your six secret ingredients here. Then add them all together.

night	ingredient	your number	what it does in your game
1	price		*what the first upgrade costs*
2	red		*red light in the colour of your pot*
3	steps		*how many stairs the loop builds*
4	green		*green light in the colour of your pot*
5	blue		*blue light in the colour of your pot*
6	growth		*how fast the price grows*
all six added together			*the kitchen lock*

KITCHEN RULE

The kitchen lock opens at 462. If your sum is 462, all six ingredients are right. If not, one night needs another look. With the right numbers your pot comes out in ramen orange.

Computer tickets

A grown-up signs each ticket when you use it. Do the paper nights first. They make the computer time short and fun.

1

After Night 3 · Computer Mission 1 · 25 minutes

Build the rainbow stairs. Signed: _____

2

After Night 6 · Computer Mission 2 · 30 minutes

Build the ramen pot and earn coins. Signed: _____

3

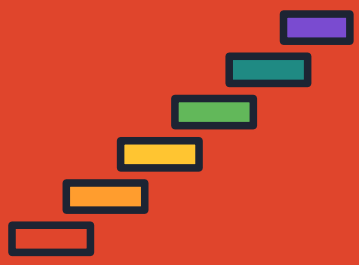
After Mission 2 · Computer Mission 3 · 30 minutes

Build the shop. Then make it yours. Signed: _____

COMPUTER MISSION 1

The rainbow stairs

Ticket 1. You need: secret ingredient 3.



Get ready

1. Open **Roblox Studio**. Choose the **Baseplate** template.
2. Find the **Explorer** window. It is the list on the right side.
3. In the Explorer, move the mouse over **ServerScriptService**. Click the **+** that appears. Choose **Script**.
4. A script opens with one line in it. Delete that line.
5. Click the **Output** button in the toolbar at the top. A window opens. It is your screen for `print` and for error messages. They never show inside the game.

Predict first

Read the code below before you type it. Answer on paper.

How many steps will the loop build? _____ How high up is step 1? _____ How high up is the last step? _____

Type it

Type this exactly. Put your own number in the box. Press the Tab key to move a line to the right. Four lines use dots and capital letters: they are Night 5 ideas, and for now you type them as they are.

```
-- NEKO RAMEN by Jens

local steps = 

for i = 1, steps do
    local step = Instance.new("Part")
    step.Size = Vector3.new(8, 1, 4)
    step.Position = Vector3.new(0, i * 2, -10 - i * 4)
    step.Color = Color3.fromHSV(i / steps, 1, 1)
    step.Anchored = true
    step.Parent = workspace
end
```

Run it

1. Press the **Play** button at the top (or the F5 key).
2. Find your stairs. Walk up them to the top.
3. Press **Stop** (the red square) before you change the code.

> **Nothing there? Red text in Output?** Good. Every programmer sees red text every day. Go to "When the Brain complains" after Mission 3 and find your message.

Change it

Predict on paper, then try. Change one thing at a time.

CHANGE	I THINK THIS WILL HAPPEN	WAS I RIGHT?
Set `steps` to 60.		
Change `i * 2` to `i * 1`.		
Change the Size to `(20, 1, 4)`.		

When the ticket is used up: put `steps` back to your ingredient, save the place with File → Save to File, and go back to Night 4.

COMPUTER MISSION 2

The pot that pays

Ticket 2. You need: secret ingredients 2, 4 and 5.



Find the six ideas first

This is the next part of your program. Before you type, mark it up with a pencil on this page.

Draw a **circle** around every variable when it is made. Draw a **box** around each function name where its card is written. **Underline** each place where a property of an object is set. Put a **star** beside the two lines that connect a bell.

Add these lines to your script. First, near the top, under `local steps`:

```
local red =
local green =
local blue =
local coinsPerClick = 1
```

Then at the bottom, under the `end` of your stairs loop:

```lua
local function makeBlock(name, x, color)
    local block = Instance.new("Part")
    block.Name = name
    block.Size = Vector3.new(4, 4, 4)
    block.Position = Vector3.new(x, 2, -6)
    block.Color = color
    block.Anchored = true
    block.Parent = workspace
    local click = Instance.new("ClickDetector")
    click.Parent = block
    local sign = Instance.new("BillboardGui")
    sign.Name = "Sign"
    sign.Size = UDim2.new(0, 200, 0, 50)
    sign.StudsOffset = Vector3.new(0, 4, 0)
    sign.Parent = block
    local label = Instance.new("TextLabel")
    label.Name = "Label"
    label.Size = UDim2.new(1, 0, 1, 0)
    label.Text = name
    label.TextScaled = true
    label.Parent = sign
    return block
end

local function say(block, text)
    block.Sign.Label.Text = text
end

local pot = makeBlock("RamenPot", -8, Color3.fromRGB(red, green, blue))
```

Press Play now. An orange pot stands beside the stairs, with a sign above it. Press Stop, then add the coins:

```lua
local function giveCoins(player)
    local stats = Instance.new("Folder")
    stats.Name = "leaderstats"
    stats.Parent = player
    local coins = Instance.new("IntValue")
    coins.Name = "Coins"
    coins.Parent = stats
end

game.Players.PlayerAdded:Connect(giveCoins)

local function cook(player)
    local coins = player.leaderstats.Coins
    coins.Value = coins.Value + coinsPerClick
end

pot.ClickDetector.MouseClick:Connect(cook)
```

The sign. A `BillboardGui` is a sign that floats above a block. The `TextLabel` inside it holds the words. `UDim2.new(0, 200, 0, 50)` is a size on the screen: 200 wide and 50 tall. `UDim2.new(1, 0, 1, 0)` means “fill the whole sign”. The small function `say` changes the words on a sign.

Answer before you run

`makeBlock` is a blueprint. What three things does it need? _____

Which bell rings when a player joins the game? _____

Which function runs when that bell rings? _____ Which function runs on a click?

_____

Run it

1. Press **Play**. Look at the top right corner of the game. Your name is there with **Coins 0**.
2. Find the pot beside the stairs. Move the mouse over it. The pointer turns into a hand.
3. Click the pot. Watch your coins.

Why “leaderstats”? Roblox looks for a folder with exactly this name inside each player. Everything in it is shown on the board at the top right. Almost every Roblox game with coins uses it, and so will yours.

COMPUTER MISSION 3

The shop

Ticket 3. You need: secret ingredients 1 and 6.



Near the top, with your other ingredients, add:

```
local price = 
local growth = 
```

At the bottom of the script, add the shop. You have already run this code in your head on Night 6.

```
local shop = makeBlock("Shop", 8, Color3.fromRGB(60, 200, 90))
say(shop, "Upgrade: " .. price .. " coins")

local function buy(player)
    local coins = player.leaderstats.Coins
    if coins.Value ≥ price then
        coins.Value = coins.Value - price
        coinsPerClick = coinsPerClick + 1
        price = price * growth
        pot.Size = pot.Size + Vector3.new(1, 1, 1)
        say(pot, "+" .. coinsPerClick .. " per click")
        say(shop, "Upgrade: " .. price .. " coins")
        print("Upgrade! Next price: " .. price)
    else
        say(shop, "Not enough! You need " .. price)
        print("Not enough coins. You need " .. price)
    end
end

shop.ClickDetector.MouseClick:Connect(buy)
```

Predict, then play

You click the pot until you have 25 coins. You click the green shop. How many coins do you have now? _____ How many coins does one click on the pot give now? _____ What does the next upgrade cost? _____

Press Play and check. Click the shop too early and read its sign. Buy an upgrade: the pot grows and both signs change. The messages from `print` are in the Output window. If a message ends with `(x11)`, the same message came 11 times.

When the Brain complains

Red text in Output tells you the line number. Look at that line **and the line above it**.

THE MESSAGE SAYS	IT USUALLY MEANS
`Expected 'end'`	An `if`, a `for` or a `function` is missing its `end`.
`attempt to index nil`	The Brain looked for a jar and found nothing. A name is spelled wrong, or a capital letter is wrong.
`is not a valid member of`	The property name is wrong. It is `Color` and `Anchored`, with these exact letters.
`Expected ')'` or `Expected '='`	A bracket, a comma or a quote mark is missing on that line.
No message, but nothing happens	Did you press Play? Is the script inside ServerScriptService? Is the last line with `Connect` there?
A wavy line under a word	Studio does not know this name. Check the spelling against the book.

Make it yours

The program works. Now it is your turn to change it. Each mission uses ideas you already have. For each one, write your plan on paper before you open Studio.

MISSION	WHAT TO DO	IDEAS
1. Your colours	Give the pot and the shop new colours. Mix them yourself with three numbers.	Properties
2. The fair price	The price doubles each time. Change `growth` so that it grows more slowly. Then make each upgrade add 2 to `coinsPerClick`. Find numbers that make the game fun.	Variables
3. The golden plate	Build one big golden block at the top of the stairs. Copy the six lines that make a step, outside the loop. Work out the Position from `steps`.	Objects, maths
4. The sushi machine	Add a second shop block. It costs 100 coins and adds 5 to `coinsPerClick`. You need a new block, a new price jar and a new function.	Functions, events
5. The days	Make days pass. Put this at the very bottom of the script. Nothing below a `while true` loop ever runs, because the loop never ends.	Loops
6. The dogs arrive	Inside that loop: if `day` is 5 or more, build a brown block near the pot and call it “Dog”. One new dog each day.	Loops, if, objects
7. Your Blender model	Bring a model you made in Blender into Studio and put it where the pot is. Put a ClickDetector inside it. Change the code so the click on your model pays the coins.	Objects, events

```lua
local day = 1
while true do
    task.wait(10)
    day = day + 1
    print("Day " .. day)
end
```

From this to Sushi Cat

Your real game is the same ideas, used many more times.

IN YOUR GAME	HOW YOU WOULD BUILD IT
Ramen machine, then sushi machine, then more	A list of upgrades. Each one is an object with a name, a price and a power. One `buy` function works for all of them.
Dogs that get better every day	A `day` variable and a loop. The dog blueprint uses `day` to set strength and weapon, like exercise 2.3.
Bodyguard cats	A blueprint function like `newGuard`, and a `fight` function like exercise 5.2.
Passes for Robux	An event from Roblox that rings when a player has bought a pass. You connect a function to it.
Cat waiters that walk to tables	A function for each job (walk, pick up, serve) and events that say when the cat has arrived.

Luau on one page

Keep this page beside the computer.

CODE	WHAT IT MEANS
`local coins = 10`	Make a variable (a jar) and put 10 in it.
`coins = coins + 1`	Work out the right side, then put the result in the jar.
`"ramen"`	Text. It is called a string. Always in quotes.
`"Day " .. day`	Glue text and numbers together.
`-- a note`	A comment. The computer skips it.
`print(coins)`	Show something in the Output window.
`==` `~=` `>` `<` `>=` `<=`	Questions: same, different, bigger, smaller, bigger or same, smaller or same.
`if … then … else … end`	Choose. Run the first part if the answer is true, the second part if it is false.
`for i = 1, 10 do … end`	Repeat 10 times. `i` counts 1, 2, 3 and so on.
`while coins < 50 do … end`	Repeat as long as the answer is true.
`local function bill(bowls) … end`	Save a recipe with a name. `bowls` is what it needs.
`return total`	Hand the result back.
`bill(3)`	Call the function: run the recipe now.
`{ name = "Tofu", strength = 8 }`	An object: several jars in one box.
`tofu.strength`	The dot means “look inside”.
`Instance.new("Part")`	Make a new Roblox object.
`part.Parent = workspace`	Put it into the world.
`Vector3.new(x, y, z)`	Three numbers: sideways, up, forward.
`Color3.fromRGB(r, g, b)`	A colour from red, green and blue, each 0 to 255.
`bell:Connect(cook)`	When this event happens, run the function `cook`.

KITCHEN RULE

The three habits. Read code from the top, one line at a time. Change one thing, then run. When it breaks, read the red message and look at that line and the line above.

CHECK AFTER EACH EXERCISE

The Answer Scroll

Night 1

1.1 After line 2: coins 10, bowls 3. Line 3: 15, 3. Line 4: 15, 2. Line 5: 17, 2. The screen shows 17.

1.2 sushi, then 16. The jar `dish` first held "ramen", but line 4 replaced it.

1.3 9 and 9. Line 3 puts 9 into `a`, and the 4 is gone. Line 4 then copies that 9 back into `b`. The jars do not swap. To swap you need a third jar to hold one number for a moment.

1.4 `local tips = 5` (local was spelled wrong). `local chef = "Miso"` (text needs quotes). `print(tips)` (small t: `Tips` is a different, empty jar).

1.5

```
local tips = 0
tips = tips + 12
print(tips)
```

1.6 After line 3: a 4, b 9, spare 4. Line 4: a 9, b 9, spare 4. Line 5: a 9, b 4, spare 4. Now they have swapped.

Ingredient 1 25. price becomes 20. coins becomes 40 − 20 = 20. Then price = 20 + 20 / 4 = 20 + 5.

Night 2

2.1 Left column: true, true, false, false. Right column: false (capital T matters), false (price is not above 10), true (one true answer is enough for `or`).

2.2 "Sorry, not enough coins." and then 9. The coins did not change, because the lines above `else` were skipped.

2.3 Day 1 bark. Day 3 stick. Day 4 stick. Day 5 soy-sauce pistol. Day 7 soy-sauce pistol. Extra spicy: stick. On day 5 the question `day ⩾ 3` is already true, so the Brain stops there and never asks about 5. Ask the hardest question first.

2.4

```
local animal = "dog"
if animal == "dog" then
    print("Door closed!")
else
    print("Welcome!")
end
```

2.5

```
if coins ⩾ 100 then
    print("Hire Tofu!")
else
    print("Keep saving")
end
```

Ingredient 2 255. Day is 6, so `day > 6` is false and `day == 6` is true: red becomes 255. The second choice asks if 255 is bigger than 255. It is not, so nothing changes.

Night 3

3.1 2, 4, 6, 8. The screen shows 8.

3.2 total after each round: 1, 3, 6, 10, 15. The screen shows 15.

3.3 Ramen for seat 1 / Ramen for seat 2 / Ramen for seat 3.

3.4 1, 2, 3, 4, 7, 10, 13. The screen shows 13.

3.5 Coins: 8, 16, 24, 32, 40, 48, 56. At 48 the question "is coins smaller than 50" is still true, so there is one more round. The screen shows 7.

3.6 Six squares in a staircase: (2, 1), (4, 2), (6, 3), (8, 4), (10, 5), (12, 6).

Ingredient 3 20. steps goes 2 → 4 → 8 → 16 in the loop. Then 16 + 4.

Night 4

4.1 8, 40, 0.

4.2 26. Extra spicy: `bill(2)` is 16, so it becomes `change(16, 10)`, which is 6.

4.3 true, false, true. The last one is true because `⩾` also accepts "the same".

4.4 Doors open / Welcome, Tofu / Welcome, Miso. Lines 1 to 3 only save the card.

4.5 `tip(80)` hands back 8.

```
local function tip(total)
    return total / 10
end
```

Ingredient 4 120. upgrade(5) = 20. upgrade(20) = 50. upgrade(50) = 110, plus 10.

Night 5

5.1 tofu: name “Big Tofu”, strength 8, wins 1. wasabi: name “Wasabi”, strength 13, wins 0.

5.2 “dog runs away” (8 is bigger than 6). “guard needs help” (5 is not bigger than 5). “dog runs away”. `tofu.wins` is 2.

5.3 (255, 0, 0) red. (0, 0, 0) black: no light at all. (255, 255, 255) white: all light. (255, 255, 0) yellow: red and green light together. (0, 0, 255) blue.

5.4 `game.Players.Jens.leaderstats.Coins` and `game.Players.Jens.leaderstats.Coins.Value`

Ingredient 5 40. heat goes 10 → 20 → 30 → 40. Then 40 >= 40 is true, so blue becomes 40.

Night 6

6.1 Kitchen is open / Jens cooked ramen / Jens cooked ramen. `Connect` only makes the promise, so “Kitchen is open” comes first.

6.2 coins, coinsPerClick, price, screen:
pot × 6: 6, 1, 5, nothing
shop: 1, 2, 10, Upgrade!
pot × 3: 7, 2, 10, nothing
shop: 7, 2, 10, Not enough coins
pot × 2: 11, 2, 10, nothing
shop: 1, 3, 20, Upgrade!

Ingredient 6 2. The first two rings add 1 each. After that `growth < 2` is false, so the last three rings change nothing.

Computer missions

Mission 1 20 steps. Step 1 is 2 high (1 × 2). The last step is 40 high (20 × 2). With 60 steps the stairs are three times as long. With `i * 1` they are half as steep. With 20 wide they become a grand staircase.

Mission 2 makeBlock needs a name, an x position and a colour. The bell is `PlayerAdded`. It runs `giveCoins`. A click runs `cook`.

Mission 3 0 coins left. 2 coins per click. The next upgrade costs 50.

Golden plate One way to do it:

```
local gold = Instance.new("Part")
gold.Size = Vector3.new(12, 1, 12)
gold.Position = Vector3.new(0, steps * 2 +
2, -18 - steps * 4)
gold.Color = Color3.fromRGB(255, 200, 0)
gold.Anchored = true
gold.Parent = workspace
```