RAZOR IDE · DEVELOPER

RAZOR IDE — Board Developer Guide

คู่มือพัฒนา Board สำหรับ Developer · ฉบับเพิ่มตัวอย่างบล็อกใช้งานจริง

สร้าง Board | เพิ่ม Block | เขียน Generator | อัปเดตผ่าน GitHub | ★ ภาพบล็อกต่อใช้จริง

v1.1 — 2026 · RazorRobotics · ทุกหัวข้อมีภาพตัวอย่างการต่อบล็อกจริงประกอบ

เอกสารฉบับนี้คงเนื้อหาเดิมทั้งหมด และ **เพิ่มภาพบล็อกที่ต่อใช้งานจริง** (สไตล์ Blockly จริงของ IDE) ในทุกหัวข้อที่เกี่ยวข้อง พร้อมโค้ด C++ ที่ระบบ Generate ออกมา เพื่อให้เห็นภาพว่า "บล็อกหน้าตาแบบนี้ → ได้โค้ดแบบนี้" ทันที

1 โครงสร้างโฟลเดอร์ Board

Board ทุกตัวอยู่ใน `boards/` ของโปรเจกต์ แต่ละ Board มีโฟลเดอร์ของตัวเอง ไม่แชร์ข้ามกัน

```
boards/
  {board_id}/            <- ID ของ Board
    config.json          <- ข้อมูล Board (metadata)
    toolbox.xml          <- เมนู Block ของ Blockly
    common/
      blocks.js          <- นิยาม Block (รูปร่าง/สี/field)
      generator.js       <- แปลง Block เป็น Code
    runtime/
      includes.txt       <- #include ต้นไฟล์
      globals.txt        <- ตัวแปร global
      setup.txt          <- code ใน setup()
      functions.txt      <- ฟังก์ชัน helper
    libraries/           <- ไฟล์ .h เฉพาะ Board
    static/cover.png     <- รูปใน Board Manager
```

ไฟล์	หน้าที่	ขาดได้?
`config.json`	ชื่อ, FQBN, version, รายชื่อ Block	ไม่ได้
`toolbox.xml`	โครงสร้างเมนู Block ทางซ้ายของ Blockly	ไม่ได้
`common/blocks.js`	นิยาม Block: รูปร่าง, field, สี	ไม่ได้

`common/generator.js`	แปลง Block เป็น C++/Python code	ไม่ได้
`runtime/includes.txt`	#include ที่ใส่ต้นไฟล์ .ino	ไม่ได้
`runtime/globals.txt`	ตัวแปร global เช่น `int _off = 0;`	ไม่ได้
`runtime/setup.txt`	code ใน setup() เช่น `Serial.begin(115200);`	ไม่ได้
`runtime/functions.txt`	ฟังก์ชัน helper เช่น `s2Sonar()`	ไม่ได้
`libraries/`	ไฟล์ .h library เฉพาะ Board	ได้
`static/cover.png`	รูปแสดงใน Board Manager	ได้

2 config.json — ข้อมูล Board

ไฟล์หลักที่ระบุ metadata ของ Board ต้องครบทุก field ที่จำเป็น

```
{
  "name": "RAZOR-S2",
  "fqbn": "esp32:esp32:esp32",   // สำหรับ arduino-cli compile
  "icon": "",
  "color": "#dc2626",
  "version": "1.0.0",           // *** สำคัญที่สุด ***
  "toolchain": "arduino",       // "arduino" หรือ "micropython"
  "upload": { "method": "arduino-cli" },
  "sdk": {
    "includes": "runtime/includes.txt",
    "globals": "runtime/globals.txt",
    "setup": "runtime/setup.txt",
    "functions": "runtime/functions.txt"
  },
  "blockTypes": [ "my_block_1", "my_block_2" ],
  "colours": {
    "MyBoard": { "bg": "linear-gradient(135deg,#ef4444,#991b1b)", "glow": "#fca5a5" }
  }
}
```

Note: `version` ใน config.json ต้องตรงกับ manifest.json บน GitHub — ถ้า manifest มีเลขสูงกว่า user จะเห็นว่ามีอัปเดต

Field	ประเภท	ตัวอย่าง	คำอธิบาย
`name`	string	"RAZOR-S2"	ชื่อแสดงในโปรแกรม
`fqbn`	string	"esp32:esp32:esp32"	สำหรับ arduino-cli
`version`	string	"1.0.0"	เพิ่มทุกครั้งที่แก้ไข
`toolchain`	string	"arduino"	arduino หรือ micropython
`blockTypes`	array	["my_block"]	รายชื่อ Block ID ทั้งหมด

`sdk`	object	{includes,...}	path ของ runtime files
`colours`	object	{Cat:{bg,glow}}	สีของหมวด Block

3 toolbox.xml — เมนู Block

กำหนดโครงสร้างเมนูซ้ายของ Blockly แบ่งเป็น `category` และ `block`

```
<xml xmlns="https://developers.google.com/blockly/xml">
  <category name="MyBoard" colour="#ef4444">
    <label text="── LED ──"></label>
    <block type="my_led_on"></block>
    <block type="my_led_off"></block>
    <block type="my_motor">
      <value name="SPEED">
        <block type="math_number"><field name="NUM">50</field></block>
      </value>
    </block>
  </category>
</xml>
```

โค้ด XML ด้านบนนี้ เมื่อเปิดใน Blockly จะกลายเป็นบล็อกในเมนูที่ลากใช้ได้ เช่นบล็อก `my_motor` ที่มีค่า default 50 เสียบมาในช่อง SPEED ดังภาพ:

Tag	คำอธิบาย
`<category>`	หมวด Block — name=ชื่อหมวด, colour=สี hex
`<label>`	หัวข้อย่อยใน category — text=ข้อความ
`<block>`	Block ที่แสดง — type=Block ID
`<value>`	ค่า default ของ input — name=ชื่อ input

4 common/blocks.js — นิยาม Block

กำหนดรูปร่าง สี และ input/output ของแต่ละ Block · บล็อกมี 3 ทรงหลัก:

▸ **Statement** — มีรอยต่อบน/ล่าง ต่อกันเป็นลำดับได้ (ไม่มีค่าคืน)

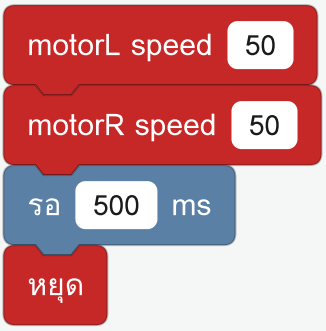
▸ **Expression/Value** — มีเดือยซ้าย เสียบเข้าช่องค่าของบล็อกอื่น (มีค่าคืน)

▸ **C-block** — ครอบบล็อกอื่นไว้ข้างใน เช่น if / loop

4.1 Statement Block (ไม่มีค่าคืน — ต่อกันได้)

```
Blockly.Blocks['my_motor_l'] = {
  init: function() {
    this.appendValueInput('SPEED')        // ช่องเสียบค่า
        .setCheck('Number')             // รับเฉพาะ Number
        .appendField('Motor ');         // label หน้า input
    this.setInputsInline(true);
    this.setPreviousStatement(true, null);    // ต่อบล็อกด้านบนได้
    this.setNextStatement(true, null);        // ต่อบล็อกด้านล่างได้
    this.setColour('#b91c1c');
    this.setTooltip('ความเร็ว -100 ถึง 100');
  }
};
```

เมื่อ `setPreviousStatement/setNextStatement` เป็น true บล็อกจะมีรอยต่อ ให้นำมาเรียงต่อกันเป็นลำดับได้ เช่นนี้:

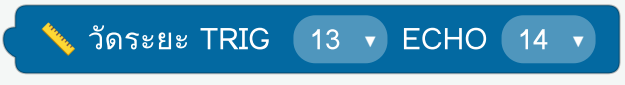


รอยหยักบน-ล่างคือจุดที่ทำให้บล็อก "ต่อกัน" เป็นลำดับการทำงานจากบนลงล่าง

4.2 Expression Block (มีค่าคืน)

```
Blockly.Blocks['my_sonar'] = {
  init: function() {
    this.appendDummyInput()
        .appendField('Sonar TRIG')
        .appendField(new Blockly.FieldDropdown([
          ['A8 (13)', '13'], ['A9 (14)', '14'],
        ]), 'TRIG')
        .appendField('ECHO')
        .appendField(new Blockly.FieldDropdown([
          ['A8 (13)', '13'], ['A9 (14)', '14'],
        ]), 'ECHO');
    this.setOutput(true, 'Number');     // คืนค่า Number
    this.setColour('#6d28d9');
  }
};
```

Expression block มี **เดือยทางซ้าย** สำหรับเสียบเข้า "ช่องค่า" ของบล็อกอื่น (ไม่มีรอยต่อบน-ล่าง):



และนำไปเสียบในช่องค่าของบล็อก statement ได้ เช่นเอาค่าที่อ่านได้ไปเป็นเวลา:

Field ที่ใช้บ่อย

Field	ตัวอย่างการใช้	ผลลัพธ์
`FieldDropdown`	`new Blockly.FieldDropdown([['A0','0']])`	dropdown เลือกค่า
`FieldTextInput`	`new Blockly.FieldTextInput('Hello')`	กล่องพิมพ์ข้อความ
`FieldNumber`	`new Blockly.FieldNumber(0,-100,100)`	กล่องตัวเลข
`FieldCheckbox`	`new Blockly.FieldCheckbox('TRUE')`	checkbox
`appendField(str)`	`this.appendField('ชื่อ')`	ข้อความธรรมดา

หน้าตาของ field แต่ละแบบเมื่ออยู่บนบล็อกจริง — dropdown (เม็ดยาเข้ม มีลูกศร ▾), กล่องตัวเลข และกล่องข้อความ (พื้นขาว):

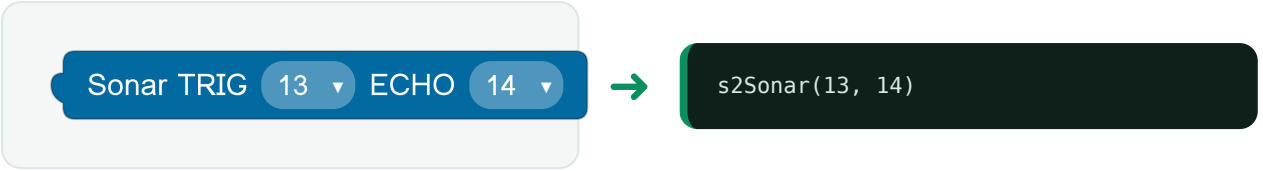
5 common/generator.js — แปลง Block เป็น Code

ทุก Block ต้องมี generator — รับ block object แล้วคืน string เป็น C++ code

5.1 Generator ของ Statement Block

```
Blockly.Arduino.forBlock['my_motor_l'] = function(block) {
  const spd = Blockly.Arduino.valueToCode(
    block, 'SPEED', Blockly.Arduino.ORDER_ATOMIC
  ) || '0';                        // fallback ถ้าช่องว่าง
  return `motorL(${spd});\n`;        // statement ลงท้าย \n เสมอ
};
```

5.2 Generator ของ Expression Block




```
Blockly.Arduino.forBlock['my_sonar'] = function(block) {
  const trig = block.getFieldValue('TRIG');   // อ่าน dropdown
  const echo = block.getFieldValue('ECHO');
  return [`s2Sonar(${trig}, ${echo})`, Blockly.Arduino.ORDER_ATOMIC];
};
```

Method	ใช้เมื่อ	ตัวอย่างผลลัพธ์
`block.getFieldValue('NAME')`	Field: dropdown, text, number	'13' หรือ 'HIGH'
`valueToCode(block,'NAME',ORDER)`	Value input รับ block อื่น	'50' หรือ 'x + 1'
`statementToCode(block,'DO')`	Statement input (body ของ loop)	'motorL(50);\n'

> **Note:** Statement block คืน string ลงท้าย `\n` เสมอ · Expression block คืน array 2 ตัว `[code, ORDER]` เสมอ

6 Runtime Files — Code ที่ฝังในทุก Sketch

ไฟล์ทั้ง 4 ถูก inject เข้า .ino โดยอัตโนมัติ ไม่ต้องให้ user เพิ่มเอง

includes.txt

```
#include <Arduino.h>
#include <WiFi.h>
#include "S2_motor.h"
#include "S2_oled.h"
```

globals.txt

```
int _s2MotorOffL = 0;
int _s2MotorOffR = 0;
```

setup.txt

```
Serial.begin(115200);
pinMode(2, OUTPUT);          // Motor pin
motorL(0); motorR(0);
razors2_oled();              // init OLED
```

functions.txt

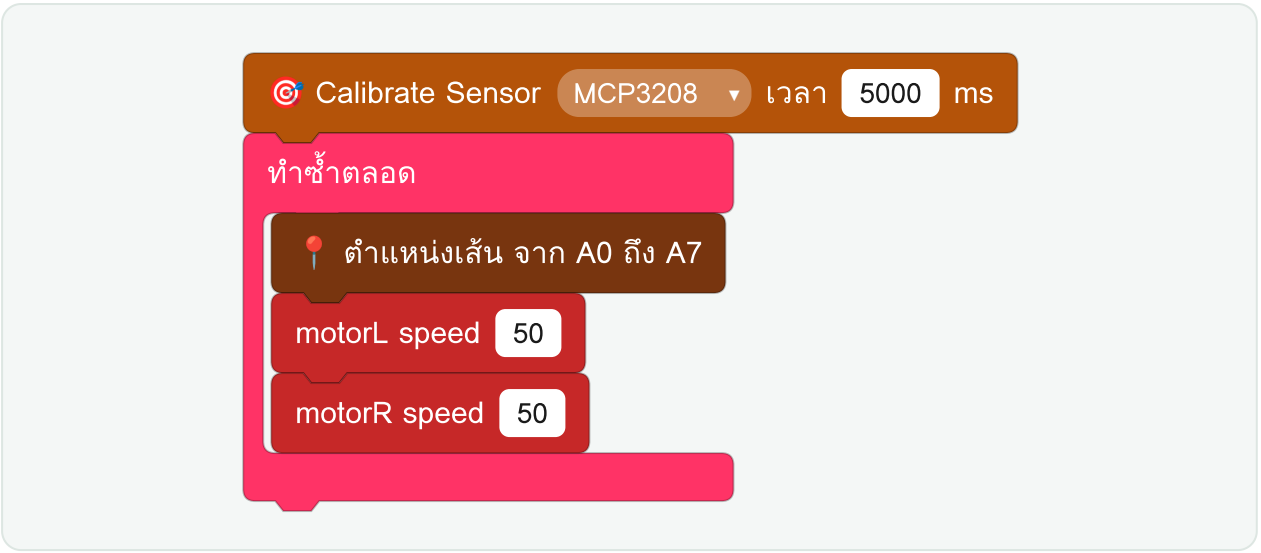
```
long s2Sonar(int trig, int echo) {
  pinMode(trig, OUTPUT); pinMode(echo, INPUT);
  digitalWrite(trig, LOW);  delayMicroseconds(2);
  digitalWrite(trig, HIGH); delayMicroseconds(10);
  digitalWrite(trig, LOW);
  long dur = pulseIn(echo, HIGH, 23200UL);
  return dur > 0L ? dur / 58L : -1L;
}
```

> **Note:** library .h ที่วางใน `libraries/` จะถูก add ด้วย `--library` flag ของ arduino-cli ใช้ `#include "MyLib.h"` ใน includes.txt ได้ทันที

7 ขั้นตอนสร้าง Board ใหม่

1. **Copy template** — คัดลอก `boards/_template/` → `boards/{board_id}/`
2. **แก้ config.json** — ตั้ง name, fqbn, version, blockTypes, colours
3. **เพิ่ม Block ใน blocks.js** — `Blockly.Blocks['my_block_1'] = { init: function(){...} };`
4. **เพิ่ม Generator ใน generator.js** — `Blockly.Arduino.forBlock['my_block_1'] = function(block){ return '...;\n'; };`
5. **แก้ toolbox.xml** — เพิ่ม `<block type="my_block_1">` ในหมวด
6. **ทดสอบ** — รัน IDE → เลือก Board → ลากบล็อก → Generate → Compile → ดู error log

เมื่อทำครบ บล็อกที่สร้างจะนำมาต่อเป็นโปรแกรมจริงได้ทันที ตัวอย่างโปรแกรมที่ประกอบจากบล็อกของ Board (calibrate เซนเซอร์ครั้งเดียว แล้ววนเดินตามเส้น):



8 ขั้นตอนอัปเดต Board

ทุกครั้งที่แก้ไขไฟล์ใน Board ต้องทำตามลำดับนี้ให้ครบ

ขั้น	ทำอะไร	รายละเอียด
1	แก้ไขไฟล์ Board	blocks.js / generator.js / toolbox.xml / runtime files
2	เพิ่ม version ใน config.json	เช่น 1.0.0 → 1.0.1 (เพิ่มทุกครั้งไม่ว่าแก้อะไร)
3	รัน pack_boards.bat	ดับเบิลคลิก `C:\razor_ide1.39\pack_boards.bat`
4	คัดลอก zip ขึ้น GitHub	วาง .zip ใน `razor-ide-updates/boards/` → Commit
5	แก้ manifest.json บน GitHub	เปลี่ยน version ของ board ให้ตรงกับ config.json

```
// ขั้น 2 — boards/RAZOR-S2/config.json
{ "version": "1.0.1" }          // จาก "1.0.0"

// ขั้น 5 — manifest.json บน GitHub
{
  "id": "RAZOR-S2",
  "version": "1.0.1",
  "download_url": "https://raw.githubusercontent.com/9richer/razor-ide-updates/main/boards/R
}
```

> **Note:** ต้องทำทั้ง config.json (ในเครื่อง) และ manifest.json (บน GitHub) พร้อมกัน ถ้าเปลี่ยนแค่อย่างเดียว user จะเห็น update แต่ไม่ได้ของใหม่

9 Checklist ก่อน Release Board

#	รายการตรวจสอบ	ผ่าน?
1	config.json มี version เพิ่มขึ้นจากเดิม	☐

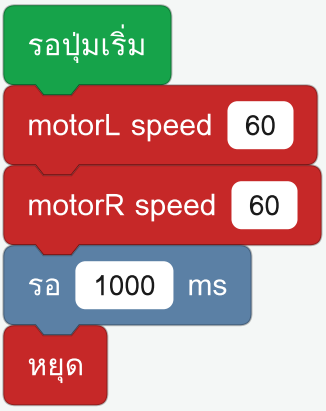
2	blockTypes array มีครบทุก Block ID	☐
3	sdk field ชี้ถูก path ทั้ง 4 ไฟล์	☐
4	blocks.js มี Blockly.Blocks[...] ครบทุก Block	☐
5	generator.js มี forBlock[...] ครบทุก Block	☐
6	toolbox.xml มี <block type=...> ครบทุก Block	☐
7	runtime/includes.txt มี #include ครบ	☐
8	ทดสอบ Compile ผ่านอย่างน้อย 1 โปรแกรม	☐
9	รัน pack_boards.bat ได้ zip ใหม่	☐
10	อัปโหลด zip ขึ้น GitHub (boards/ folder)	☐
11	แก้ manifest.json: version ตรงกับ config.json	☐
12	Commit + Push GitHub repo	☐

10 ★ ตัวอย่างการต่อบล็อกใช้งานจริง

หัวข้อนี้รวมโปรแกรมสำเร็จรูปที่ประกอบจากบล็อกจริงของ Board (เช่น RAZOR-S4) แต่ละตัวอย่างแสดง **ภาพบล็อกที่ต่อกัน** คู่กับ **โค้ด C++ ที่ระบบ Generate ออกมา** ให้เห็นความสัมพันธ์ชัดเจน

A · ขับเคลื่อนพื้นฐาน (เรียง Statement)

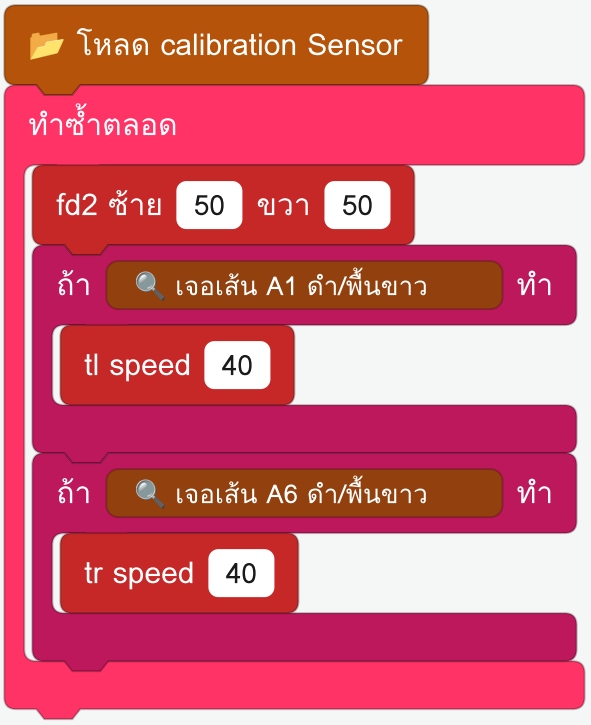
รอกดปุ่มเริ่ม → เดินหน้า 1 วินาที → หยุด · ตัวอย่างการต่อบล็อก statement เป็นลำดับ




```
sw_go();
motorL(60);  motorR(60);
delay(1000);
ao();
```

B · เดินตามเส้น (Loop + If ซ้อน)

โหลด calibration แล้ววนตลอด: เดินหน้า ถ้าเซนเซอร์ขอบซ้าย/ขวาเจอเส้น ให้เลี้ยวแก้




```
loadCalibration();
while (true) {
  fd2(50, 50);
  if (sensorOnLine(1, 0)) { tl(40); }
  if (sensorOnLine(6, 0)) { tr(40); }
}
```

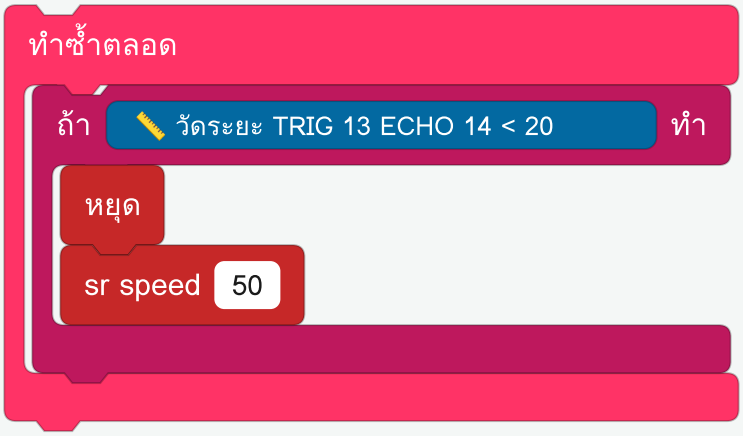
C · หุ่นยนต์กู้ภัย — อ่านสีพื้นจุดปล่อย (TCS34725)

เริ่มเซนเซอร์สี → สอนสีหน้างานทีละสี → วนเช็ค ถ้าพื้นเป็นสีแดงให้หยุด แสดงผล และส่งเสียง


```
tcs_begin(0xEB, 1);
tcs_teach(0, "WHITE");
tcs_teach(1, "RED");
tcs_teach(2, "GREEN");
while (true) {
  if (tcs_readColor() == 1) {   // 1 = RED
    ao();
    oled(0, "RED!");
    Beep();
  }
}
```

D · หลบกำแพงด้วย Ultrasonic (Expression ในเงื่อนไข)

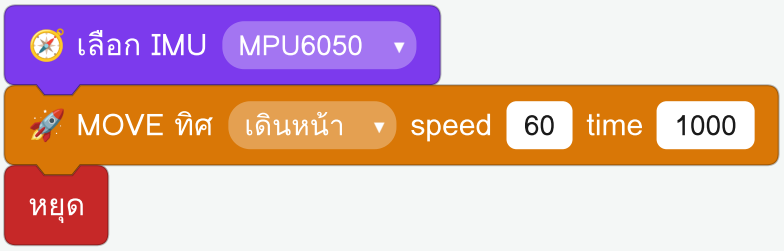
วนตลอด: ถ้าระยะที่วัดได้ < 20 ซม. ให้หยุดแล้วหมุนหลบ · บล็อกค่า "วัดระยะ" เสียบในเงื่อนไข if




```
while (true) {
  if (ultrasonic_read(13, 14) < 20) {
    ao();
    sr(50);
  }
}
```

E · เดินตรงด้วย IMU

เลือกชนิด IMU → สั่งเดินหน้าตรงด้วย PID ของ IMU 1 วินาที → หยุด




```
imuSelect(IMU_MPU6050);
imuMove(/* เดินหน้า */ 'F', 30, 60, 1000, 1, 0, 0.1, 0, 0);
ao();
```

เคล็ดลับการสอน: ให้ผู้เรียนสังเกตว่า "รอยหยักบน-ล่าง" = ลำดับการทำงาน, "เดือยซ้าย" = ค่าที่ส่งเข้าบล็อกอื่น, และ "บล็อกครอบ (C-block)" = if/loop ที่มีบล็อกข้างใน จับคู่ภาพบล็อกกับโค้ดที่ได้ จะเข้าใจ generator ได้เร็วขึ้นมาก