Lesson 03: 控制流程

歡迎來到第三課！到目前為止，我們的程式都是從上到下一行一行執行。但真正的程式需要能夠做決定、選擇不同的路徑、重複執行某些工作。這就像是給程式一個大腦，讓它能夠根據不同情況採取不同行動。

程式的執行流程

想像你每天早上起床的流程：

1. 看看天氣如何
2. **如果下雨**，就帶雨傘；**如果沒下雨**，就不帶
3. 吃早餐
4. **重複**刷牙直到牙齒乾淨
5. 出門上班

在這個流程中，有些步驟是條件性的（看天氣決定要不要帶傘），有些步驟是重複性的（刷牙）。程式的控制流程就是在模擬這種決策和重複的邏輯。

條件判斷：if-else

基本 if 語句

最簡單的條件判斷就是「如果...就...」：

```
age = 20

if age >= 18 then
    print("你已經成年了！")
end
```

語法結構：

- `if` 後面接著條件（會是 true 或 false）
- `then` 表示「那麼就」
- `end` 表示條件判斷結束

if-else 語句

「如果...就...，否則就...」：

```
score = 85

if score >= 60 then
    print("恭喜及格！")
else
    print("需要加油喔！")
end
```

if-elseif-else 多重判斷

當有多個條件要判斷時：

```
score = 85

if score >= 90 then
    grade = "A"
elseif score >= 80 then
    grade = "B"
elseif score >= 70 then
    grade = "C"
elseif score >= 60 then
    grade = "D"
else
    grade = "F"
end

print("你的成績是：" .. grade)
```

複雜條件判斷

可以使用邏輯運算子組合多個條件：

```
age = 25
has_license = true
has_car = false

-- 使用 and（且）
if age >= 18 and has_license then
    print("可以開車")
end
```

```
-- 使用 or（或）
if has_car or has_license then
    print("有交通工具相關資格")
end

-- 使用 not（非）
if not has_car then
    print("需要購買汽車")
end

-- 複雜組合
if (age >= 18 and has_license) or age >= 21 then
    print("符合駕駛條件")
end
```

巢狀判斷

判斷裡面還可以包含判斷：

```
weather = "晴天"
temperature = 25

if weather == "晴天" then
    if temperature > 30 then
        print("天氣很熱，記得防曬！")
    elseif temperature > 20 then
        print("天氣很好，適合出遊！")
    else
        print("天氣涼爽，很舒適！")
    end
else
    print("天氣不好，待在室內比較好。")
end
```

迴圈：重複執行

當我們需要重複執行相同或類似的工作時，就要用到迴圈。

數值迴圈：for

當你知道要重複幾次時，使用 for 迴圈：

```lua
-- 基本語法：for 變數 = 開始值, 結束值 do
for i = 1, 5 do
    print("第 " .. i .. " 次")
end

-- 輸出：
-- 第 1 次
-- 第 2 次
-- 第 3 次
-- 第 4 次
-- 第 5 次
```

設定步進值

```lua
-- 每次增加 2
for i = 1, 10, 2 do
    print(i)
end
-- 輸出：1, 3, 5, 7, 9

-- 倒數
for i = 10, 1, -1 do
    print(i)
end
-- 輸出：10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

遍歷 Table 的 for 迴圈

```lua
fruits = {"蘋果", "香蕉", "橘子", "葡萄"}

-- 使用 ipairs 遍歷陣列
for index, value in ipairs(fruits) do
    print(index .. ": " .. value)
end

-- 使用 pairs 遍歷字典
student = {name = "小明", age = 20, grade = "A"}
for key, value in pairs(student) do
    print(key .. " = " .. value)
end
```

while 迴圈

當你不知道要重複幾次，只知道在什麼條件下要停止時：

```lua
-- 猜數字遊戲的輸入迴圈
answer = 42
guess = 0

while guess ~= answer do
    print("請輸入猜測的數字：")
    guess = tonumber(io.read())  -- 從鍵盤讀取輸入

    if guess < answer then
        print("太小了！")
    elseif guess > answer then
        print("太大了！")
    else
        print("猜對了！")
    end
end
```

repeat-until 迴圈

至少執行一次，然後檢查條件：

```lua
-- 密碼驗證
local password
repeat
    print("請輸入密碼：")
    password = io.read()

    if password ~= "secret123" then
        print("密碼錯誤，請重試！")
    end
until password == "secret123"

print("歡迎登入！")
```

while vs repeat-until 的差別：

- `while`：先檢查條件，可能一次都不執行
- `repeat-until`：先執行一次，再檢查條件

實用範例

範例 ：九九乘法表

```
print("=== 九九乘法表 ===")

for i = 1, 9 do
    for j = 1, 9 do
        local result = i * j
        -- string.format 用來格式化輸出
        print(string.format("%d × %d = %2d", i, j, result))
    end
    print()  -- 空行分隔
end
```

控制流程的進階技巧

短路求值

```
-- and 運算：如果第一個是 false，就不會檢查第二個
age = 15
can_drive = age >= 18 and print("檢查駕照")  -- 不會印出訊息

-- or 運算：如果第一個是 true，就不會檢查第二個
name = "小明"
display_name = name or "訪客"  -- 如果 name 不存在，使用 "訪客"
```

常見錯誤和注意事項

1. 等號的使用

```
-- 錯誤：使用 = 而不是 ==
age = 18
if age = 18 then    -- 語法錯誤
    print("成年了")
end

-- 正確：使用 == 比較
if age == 18 then   -- 這才是正確的
    print("成年了")
end
```

2. 條件的真假判斷

```lua
-- 在 Lua 中，只有 false 和 nil 被視為假，其他都是真
if 0 then
    print("這會被印出來！")  -- 0 是真值
end

if '' then
    print("這也會被印出來！")  -- 空字串也是真值
end

-- 要檢查空字串，需要明確比較
local text = ""
if text == "" then
    print("字串是空的")
end
```

3. 無窮迴圈

```lua
-- 小心無窮迴圈
local i = 1
while i <= 10 do
    print(i)
    -- 忘記增加 i，會造成無窮迴圈
    -- i = i + 1  -- 記得要有這行！
end
```

4. 迴圈變數的作用域

```lua
-- for 迴圈的變數只在迴圈內有效
for i = 1, 5 do
    print(i)
end
-- print(i)  -- 錯誤！i 在這裡不存在

-- 如果需要在迴圈外使用，要在外面宣告
local i
for i = 1, 5 do
    print(i)
end
print("最後的 i：" .. i)  -- 這樣就可以
```

練習時間

基礎練習

1. **年齡分類**：
   寫一個程式判斷年齡屬於哪個階段：
   - 0-12：兒童
   - 13-19：青少年
   - 20-59：成年人
   - 60以上：長者
2. **倍數檢查**：
   使用迴圈印出 1-20 中所有 3 的倍數

grandMA2 燈光效果製作實戰應用

控制流程在燈光控制中是創造動態效果的關鍵！讓我們用條件判斷和迴圈來創造令人驚豔的燈光效果：

呼吸燈效果 - 使用 for 迴圈

```
-- grandMA2 Plugin 實戰：製作呼吸燈效果
function Start()
    gma.echo("=== 呼吸燈效果啟動 ===")

    -- 呼吸燈參數
    local fixture_range = "1 thru 10"  -- 控制燈具範圍
    local min_brightness = 5           -- 最暗亮度
    local max_brightness = 100         -- 最亮亮度
    local step = 5                     -- 每次變化的亮度
    local sleep_time = 0.1             -- 每步等待時間
    local breath_cycles = 3            -- 呼吸週期次數

    -- 執行多次呼吸週期
    for cycle = 1, breath_cycles do
        gma.echo("呼吸週期 " .. cycle .. "/" .. breath_cycles)

        -- 漸亮階段（吸氣）
        for brightness = min_brightness, max_brightness, step do
            gma.cmd("Fixture " .. fixture_range .. " at " .. brightness)
            gma.sleep(sleep_time)
        end

        -- 漸暗階段（呼氣）
```

```
        for brightness = max_brightness, min_brightness, -step do
            gma.cmd("Fixture " .. fixture_range .. " at " .. brightness)
            gma.sleep(sleep_time)
        end

        -- 週期間暫停
        gma.sleep(0.5)
    end

    gma.echo("呼吸燈效果完成")
end
```

彩虹追逐效果 - 混合使用控制流程

```
-- grandMA2 Plugin 實戰：彩虹追逐效果
function Start()
    gma.echo("=== 彩虹追逐效果啟動 ===")

    -- 彩虹色彩序列
    local colors = {"red", "orange", "yellow", "green", "cyan", "blue",
"magenta"}
    local fixtures = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  -- 燈具陣列
    local chase_speed = 0.3   -- 追逐速度
    local total_rounds = 5    -- 總追逐回合

    for round = 1, total_rounds do
        gma.echo("彩虹追逐第 " .. round .. " 回合")

        -- 每個燈具依序變色
        for i, fixture_id in ipairs(fixtures) do
            -- 計算這盞燈要顯示的顏色（循環使用顏色陣列）
            local color_index = ((i - 1 + round - 1) % #colors) + 1
            local current_color = colors[color_index]

            -- 設定燈光
            gma.cmd("Fixture " .. fixture_id .. " at 80 " ..
current_color)

            -- 條件判斷：如果是特定顏色就加強效果
            if current_color == "red" or current_color == "blue" then
                gma.cmd("Fixture " .. fixture_id .. " at 100")  -- 紅色和
藍色更亮
            end
```

```
            gma.sleep(chase_speed)
        end
    end

    -- 結束時所有燈光淡出
    gma.echo("效果結束，淡出...")
    gma.cmd("Fixture 1 thru 10 at 0 fade 2")
end
```

智慧反應燈光系統 - 複雜條件判斷

```
-- grandMA2 Plugin 實戰：智慧反應燈光系統
function Start()
    gma.echo("=== 智慧反應燈光系統啟動 ===")

    -- 模擬感測器數據（實際應用中會從真實感測器讀取）
    local sensors = {
        motion_detected = true,
        sound_level = 75,       -- 0-100
        time_of_day = 20,       -- 24小時制
        room_occupancy = 3      -- 人數
    }

    gma.echo("檢測環境狀況...")

    -- 多重條件判斷來決定燈光模式
    if sensors.motion_detected then
        gma.echo("偵測到動作，啟動燈光")

        -- 根據時間決定亮度
        local brightness
        if sensors.time_of_day >= 6 and sensors.time_of_day < 18 then
            brightness = 100  -- 白天全亮
            gma.echo("白天模式 - 全亮")
        elseif sensors.time_of_day >= 18 and sensors.time_of_day < 22
then
            brightness = 80   -- 傍晚適中
            gma.echo("傍晚模式 - 適中亮度")
        else
            brightness = 30   -- 夜晚昏暗
            gma.echo("夜晚模式 - 昏暗燈光")
        end

        -- 根據人數調整燈光範圍
```

```
        local fixture_count
        if sensors.room_occupancy <= 2 then
            fixture_count = "1 thru 5"
        elseif sensors.room_occupancy <= 5 then
            fixture_count = "1 thru 8"
        else
            fixture_count = "1 thru 12"
        end

        gma.cmd("Fixture " .. fixture_count .. " at " .. brightness .. "
fade 1")

        -- 根據音量決定是否啟動音樂模式
        if sensors.sound_level > 70 then
            gma.echo("高音量偵測 - 啟動音樂模式")
            MusicMode(fixture_count)
        elseif sensors.sound_level > 40 then
            gma.echo("中等音量 - 溫和反應")
            GentleResponse(fixture_count)
        end

    else
        gma.echo("無動作偵測 - 節能模式")
        gma.cmd("Fixture 1 thru 12 at 0 fade 3")
    end
end

-- 音樂模式：燈光隨節拍變化
function MusicMode(fixtures)
    local beat_colors = {"red", "blue", "green", "purple", "yellow"}

    for beat = 1, 8 do  -- 模擬8拍
        local color = beat_colors[(beat % #beat_colors) + 1]
        gma.cmd("Fixture " .. fixtures .. " at 100 " .. color)
        gma.sleep(0.5)

        -- 每兩拍閃一次
        if beat % 2 == 0 then
            gma.cmd("Fixture " .. fixtures .. " at 0")
            gma.sleep(0.1)
            gma.cmd("Fixture " .. fixtures .. " at 100 " .. color)
        end
    end
end
```

```
-- 溫和反應：輕微亮度變化
function GentleResponse(fixtures)
    for i = 1, 3 do
        gma.cmd("Fixture " .. fixtures .. " at +10 fade 0.5")  -- 增加10%
亮度
        gma.sleep(1)
        gma.cmd("Fixture " .. fixtures .. " at -10 fade 0.5")  -- 減少10%
亮度
        gma.sleep(1)
    end
end
```

條件判斷在燈光控制中的應用實例

```
-- 燈光狀態檢查與自動修正系統
function LightingHealthCheck()
    gma.echo("=== 燈光系統健康檢查 ===")

    local fixtures_to_check = {1, 2, 3, 4, 5}
    local failed_fixtures = {}

    for _, fixture_id in ipairs(fixtures_to_check) do
        local fixture_handle = gma.show.getobj.handle("Fixture " ..
fixture_id)

        -- 條件判斷：檢查燈具是否存在
        if fixture_handle then
            local dmx_value = gma.show.getdmx(fixture_id)

            -- 檢查 DMX 數值是否正常
            if dmx_value == 0 then
                gma.echo("警告：Fixture " .. fixture_id .. " 可能故障 (DMX
= 0) ")
                table.insert(failed_fixtures, fixture_id)
            elseif dmx_value > 0 then
                gma.echo("正常：Fixture " .. fixture_id .. " DMX = " ..
dmx_value)
            end
        else
            gma.echo("錯誤：Fixture " .. fixture_id .. " 不存在")
            table.insert(failed_fixtures, fixture_id)
        end
    end
```

```
    -- 根據檢查結果採取行動
    if #failed_fixtures == 0 then
        gma.echo("✓ 所有燈具運作正常")
        gma.cmd("Fixture " .. table.concat(fixtures_to_check, ",") .. "
at 100 fade 2")
    else
        gma.echo("⚠ 發現 " .. #failed_fixtures .. " 個問題燈具")

        -- 只控制正常的燈具
        local working_fixtures = {}
        for _, fixture_id in ipairs(fixtures_to_check) do
            local is_failed = false
            for _, failed_id in ipairs(failed_fixtures) do
                if fixture_id == failed_id then
                    is_failed = true
                    break
                end
            end
            if not is_failed then
                table.insert(working_fixtures, fixture_id)
            end
        end

        if #working_fixtures > 0 then
            gma.cmd("Fixture " .. table.concat(working_fixtures, ",") ..
" at 80")
        end
    end
end
```

while 迴圈應用：互動式燈光控制

```
-- 互動式燈光調節器
function InteractiveLightController()
    gma.echo("=== 互動式燈光調節器 ===")
    gma.echo("輸入指令：up (增亮), down (調暗), color (變色), quit (結束)")

    local current_brightness = 50
    local current_color = "white"
    local fixture_range = "1 thru 8"

    -- 設定初始狀態
    gma.cmd("Fixture " .. fixture_range .. " at " .. current_brightness
.. " " .. current_color)
```

```
    -- 持續等待使用者輸入直到選擇退出
    while true do
        local user_input = gma.textinput("燈光控制", "請輸入指令
(up/down/color/quit)")

        -- 判斷使用者指令
        if user_input == "quit" or user_input == "exit" then
            gma.echo("結束燈光控制器")
            break

        elseif user_input == "up" then
            current_brightness = current_brightness + 10

            -- 防止超過最大值
            if current_brightness > 100 then
                current_brightness = 100
                gma.echo("已達最大亮度")
            end

            gma.cmd("Fixture " .. fixture_range .. " at " ..
current_brightness)
            gma.echo("亮度增加到：" .. current_brightness .. "%")

        elseif user_input == "down" then
            current_brightness = current_brightness - 10

            -- 防止低於最小值
            if current_brightness < 0 then
                current_brightness = 0
                gma.echo("已達最小亮度")
            end

            gma.cmd("Fixture " .. fixture_range .. " at " ..
current_brightness)
            gma.echo("亮度降低到：" .. current_brightness .. "%")

        elseif user_input == "color" then
            local colors = {"red", "blue", "green", "yellow", "purple",
"white"}

            -- 循環切換顏色
            local current_index = 1
            for i, color in ipairs(colors) do
                if color == current_color then
```

```
                    current_index = i
                    break
                end
            end

            current_index = (current_index % #colors) + 1
            current_color = colors[current_index]

            gma.cmd("Fixture " .. fixture_range .. " at " ..
current_brightness .. " " .. current_color)
            gma.echo("顏色變更為：" .. current_color)

        else
            gma.echo("無效指令，請重新輸入")
        end

        gma.sleep(0.1)  -- 短暫暫停避免過快操作
    end
end

function Start()
    InteractiveLightController()
end
```

為什麼控制流程在燈光控制中如此重要？

控制結構	燈光控制應用	實際效果
if-else	場景判斷、狀態檢查	根據時間/事件切換不同燈光模式
for 迴圈	漸變效果、序列控制	呼吸燈、彩虹效果、追逐燈
while 迴圈	持續監控、互動控制	感應器監控、使用者互動
巢狀結構	複雜效果組合	多層次的動態燈光表演

小結

控制流程是程式邏輯的核心。有了條件判斷，程式可以做決定；有了迴圈，程式可以處理重複性工作。在燈光控制中，這些工具讓我們能夠：

1. **創造動態效果**：用迴圈產生漸變、追逐、閃爍等效果

2. **智慧判斷**：根據環境或時間自動調整燈光
3. **互動控制**：建立回應式的燈光系統
4. **系統監控**：自動檢查燈具狀態並做出反應
5. **實際應用**：把簡單的開關燈變成專業的燈光秀

這些工具讓我們的程式從簡單的計算器變成智慧的燈光控制師，能夠創造出令觀眾驚豔的視覺效果！