

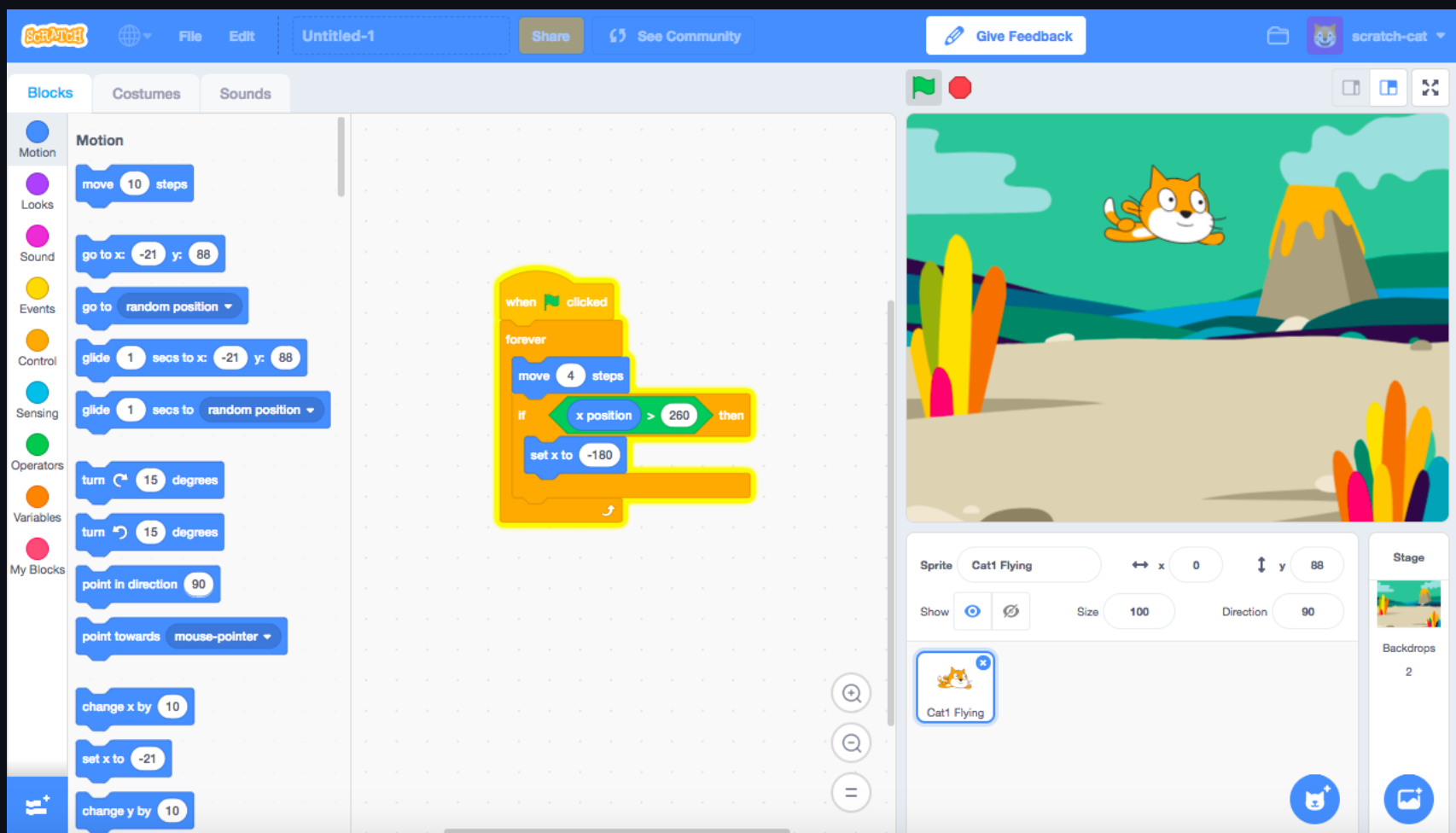
Life is Possible - 生命教育 手機程式工作坊

講課 04 - JavaScript 簡介

目錄

- 深入探索程式設計
- 介紹使用 JavaScript 實現真正的程式設計

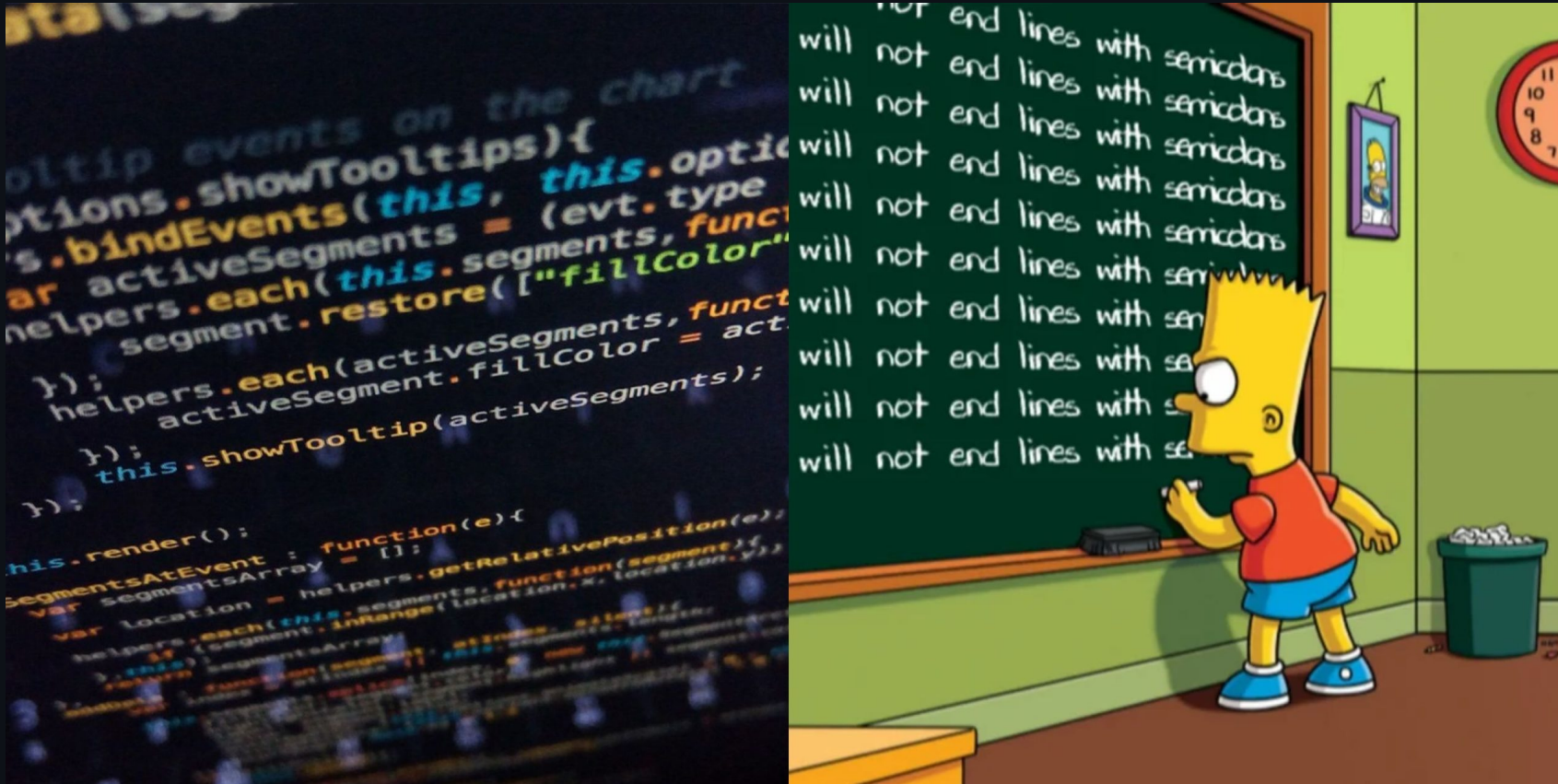
使用邏輯積木編程嗎？



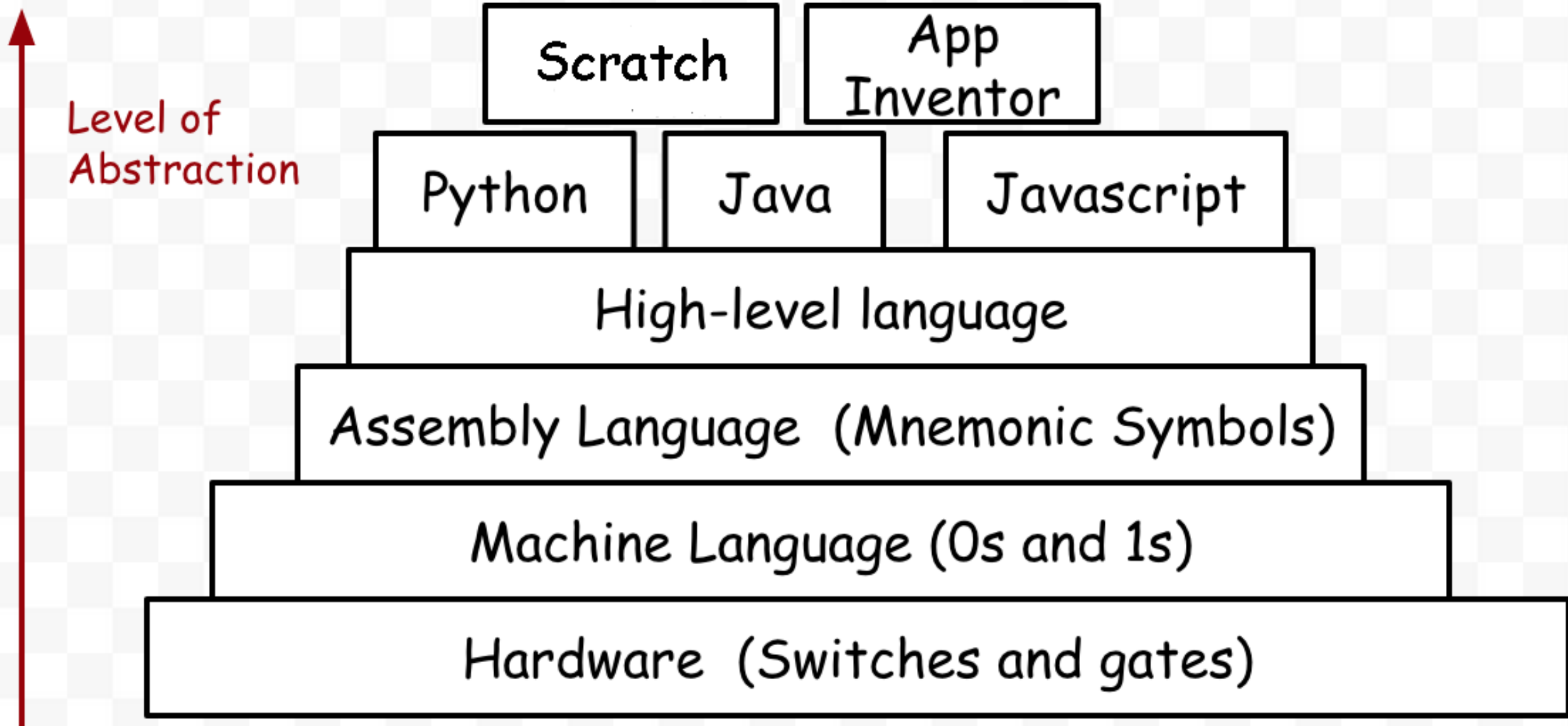
真正的程式設計師不這麼做...



真正的程式設計師僅用文字/單字編寫程式碼



程式語言









為什麼我們學習 JavaScript?

- 對於初學者來說相對容易學習
- 只需學習一次，可在任何地方使用 (Web / 桌面 / Linux)
- 在全球具有高度的普及度

<https://www.stackscale.com/blog/most-popular-programming-languages/>

- 有龐大的社群支援以及擁有豐富的庫支援 `npm`

Javascript 可以運行於

-  網頁瀏覽器 (使用 V8 引擎)
-  Android (使用 V8 引擎和瀏覽器)
-  iOS (使用 V8 引擎和瀏覽器)
-  Windows (使用 Nodejs)
-  MacOS (使用 Nodejs)
-  Linux (使用 Nodejs)

幾乎你日常生活中所看到的任何東西都可以運行 JavaScript。

撰寫歡迎程式

app.js

```
let str = "reemo"  
console.log("Hello all");  
console.log(str);  
console.log("This is " + str);
```

然後在終端機中輸入 `node app.js` 以執行腳本。

注意：如果您沒有安裝 nodejs，請立即從 <https://nodejs.org/en/> 安裝它。

console.log()

這是 JavaScript 中用來列印訊息的基本函數

用法

```
console.log("Hello world"); // "Hello world"  
console.log("Hello " + "Peter"); // "Hello Peter"  
console.log("Hello", "Peter"); // "Hello Peter"
```

```
console.log(100); // 100  
console.log(100 + 45); // 145
```

今天的 JavaScript 課程

- 資料類型 Data Types
- 算術運算子 Arithmetic Operators
- 賦值運算子 Assignment Operators
- 定義變數 Define variables
- 條件語句 Conditional Statements
- 比較語句 Compare Statements
- 函式 Functions

JavaScript Comments 註解

用於解釋代碼，並使其更可讀
註解中的代碼將不會運行/執行/編譯

單行註解以 `//` 開頭。

```
// I will not run  
// console.log("Hello")  
// Hello mate  
  
console.log("yo hi")
```

JavaScript Comments 註解

多行註解以 `/*` 開頭, 以 `*/` 結尾.

```
/*  
Hello  
This is multi line comments  
I can contains a lot of stuff  
in the same block  
console.log("Yooooo")  
*/  
  
console.log("mate")
```

常見的資料型別

- `string` 字串型別, 例如 `"Hello"`, `"Good day"`, `"I go to school by bus"`
- `number` 數值型別, 例如 `12`, `-3`, `32.476`, `0x012`, `11010010`
- `boolean` 布林型別, 例如 `true`, `false`

string

顯示常見的字串 / 字元

- `"Hello world"`, `"a"`, `"😊"` (Using `"`) (使用雙引號)
- `'Hello mate'`, `'b'`, `'😍'` (Using `'`) (使用單引號)
- ``Hello mom``, ``c``, ``😎`` (Using ```) (使用反引號)

number

代表數學運算中的數值資料型別 (`+` , `-` , `*` , `/` , `%`)

- Integer 整數: `1` , `384`
- Signed Integer 有正負號的整數: `1` , `384` , `-43`
- Float / Double 浮點數 / 雙精度浮點數: `1.23` , `-34.3423` , `0.001`
(Float 浮點數 = 單精度, Double 雙精度浮點數 = 雙精度)
- Infinity 無限大: `infinity` , `-infinity`

number 基本算術運算子

- `+` : Addition 加法 (e.g. `3 + 5 = 8`)
- `-` : Subtraction 減法 (e.g. `8 - 2 = 6`)
- `*` : Multiplication 乘法 (e.g. `3 * 4 = 12`)
- `/` : Division 除法 (e.g. `10 / 5 = 2`, `14 / 5 = 2.8`)

更多 `number` 算術運算符

通常在一般情況下使用的

- `**` : 指數運算 Exponentiation (`2^3` => `2**3`)
- `%` : 取模運算 / 求餘數 Modulus / Remainder (`5 % 2 == 1` , `10 % 2 == 0`)

通常在循環中使用的

(將在下一課中介紹更多)

- `++` : 自增 (`i++`)
- `--` : 自減 (`i--`)

`boolean` 布爾值

它是一種用於確定條件是否發生的數據類型。在此數據類型中，僅出現 `true` 和 `false`。

- `true` : 表示條件將發生 / 匹配。
- `false` : 表示條件將不會發生 / 不匹配。

boolean 示例

```
13 == 100 // Is 13 equal to 100 ? false
100 > 50  // Is 100 bigger than 50 ? true
0 < 2     // Is 0 smaller than 2 ? true

"tom"    == "hello tom" // Is "tom" equal to "hello tom" ? false
"apple"  == "apple"     // Is "apple" equal to "apple" ? true
```

boolean and &&, or ||, not !

在所有的程式語言中，幾個符號代表邏輯運算。

- **and** : 使用 **&&**
- **or** : 使用 **||**
- **not** : 使用 **!**

Boolean Operators

AND

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

OR

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

NOT

A	NOT A
True	False
False	True

boolean 與 and &&, or ||, not !

- `and` : 表示兩個條件都成立時, 會回傳 `true`。
- `or` : 表示其中一個條件成立時, 會回傳 `true`。
- `not` : 反轉條件 (`!true => false`, `!false => true`)

為什麼要使用 `and` `&&`, `or` `||`, `not` `!`

其實這是一個現實生活中的簡單問題或邏輯句子。

如果我們要用表示一個條件：

如果(If)今天是星期一，且(and)天氣沒有下雨，那麼我會去彼得家。

在程式語言中，我們會這樣寫：

```
if(today.day == "monday" && today.weather != "raining"){  
    goToPeterHome();  
}
```

為什麼要使用 `and`, `or`, `not`

更複雜的條件可能會像這樣：

這是湯姆。嘿，彼得，我想跟你玩一個遊戲。
從1到6猜一個數字，我會開始擲骰子。

如果數字是偶數且是1或6，你將贏得遊戲。
如果數字是偶數但是是4，你將輸。
但如果數字是奇數，你會輸。
但如果數字是奇數且是3，你將贏得遊戲。

如果你是湯姆，想要跟彼得玩一個遊戲，你會在程式語言中如何寫呢？

samples.js

```
// let assume this function will return a random number of 1 to 6
let diceNumber = randomDiceNumber();

if(diceNumber % 2 == 0 && diceNumber == 4){
    lose()
}
else if(diceNumber % 2 == 0 && (diceNumber == 1 || diceNumber == 6)){
    win()
}
else if(diceNumber % 2 == 1 && diceNumber == 3){
    win()
}
else if(diceNumber % 2 == 1){
    lose()
}
```

在 JS 中的進階資料型別

(本課程中不會涵蓋，但在未來很重要)

常用的

- 物件 (Object) and 陣列 (Array)
- Null and 未定義 (Undefined)

進階的

- 錯誤 (Error)
- 日期 (Date)
- 映射 (Map) and 集合 (Set)
- 類別 (Class)

休息一下

Assignment Operators 指派運算子

Operator Short hand 縮寫運算子	Example 範例	Same As 相同於
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$
**=	$x ** = y$	$x = x ** y$

Define a variables 定義變量

```
var names = "reemo";  
let age = 123;  
const isMale = true;
```

一般來說，我們可以使用 `var`、`let` 或 `const` 來定義變量。

在大多數情況下（99.99%），由於歷史原因，我們不建議使用 `var`。

[了解更多](#)

Define a variables 定義變量

```
// <declare_words> <declare_variables_name> = <data_values>  
let names = "reemo";  
const age = 10;
```

- `let` 是一個定義詞，用於告訴計算機定義一個可更改的變量。
- `const` 也是一樣的，但我們假設使用 `const` 定義的變量是在定義之後不能被更改的變量。

let

1. 使用 `let` 定義的變量可以被重新定義
2. 具有區塊作用域 (block-scope)

```
let names = "reemo";  
console.log(names) // reemo  
  
names = "tom"  
console.log(names) // tom
```

const

1. 通常情況下，使用 `const` 定義的變量不能被重新定義。
2. 具有區塊作用域 (block-scope)

```
const names = "reemo";  
console.log(names) // reemo  
  
names = "tom" // Cannot assign to 'names' because it is a constant.  
console.log(names) // error: Uncaught TypeError: Assignment to constant variable.
```

在像 `array` 和 `object` 這樣的數據類型中使用 `const` 有例外情況，但我們首先假設所有 `const` 變量都不能被重新分配。

條件語句

- Compare 比較運算子: `==`, `===`, `!=`, `!==`
- For Maths 數學運算子: `>`, `<`, `>=`, `<=`
- Logics 邏輯運算子: `if`, `else if`, `else`

比較運算子 `==`, `===`, `!=`, `!==`

- `==`: 弱等於
- `===`: 強等於
- `!=`: 弱不等於
- `!==`: 強不等於

Strong compare 強等 和 Weak compare 弱等 ?

- Strong 意味著不僅值相同，而且數據類型也需要匹配。
- Weak 意味著如果值相同（無論是字符串還是數字），且內容相同，則返回 true。

強等與弱等的例子

```
let num = 100; // type: number
let stringNum = "100" // type: string

console.log( num == stringNum ) // Weak compare: true
console.log( num === stringNum ) // Strong compare: false
```

注意事項：在這個例子中，`num` 是一個 `number` 的資料型態，但是 `stringNum` 是一個 `string` 的資料型態，儘管兩者都代表了 `100`。

如果不確定要使用哪種比較類型，總是使用強比較 `===` `!==` 來確保安全。

數學比較運算子：>, <, >=, <=

- >: 大於
- >=: 大於或等於s
- <: 小於
- <=: 小於或等於

```
10 > 6    // true
5 < 89    // true
10 > 10   // false, why? Since 10 not not bigger than 10, is equal to 10
10 >= 10  // true
```

邏輯比較運算子: `if` , `else if` , `else`

內部程式碼將在條件滿足時運行

- `if(){}` 如果條件成立，運行這個區塊的程式碼。
- `else if(){}` 如果前面的條件不成立，則檢查這個條件，如果成立，運行這個區塊的程式碼。
- `else{}` 如果前面的所有條件都不成立，則運行這個區塊的程式碼。

if

demo.js

```
// Only the conditions between the () is true, the under code will run
```

```
if(true){  
    console.log("I will run yeah 😊")  
}  
  
if(false){  
    console.log("I will NOT run oh no 😞")  
}
```

ifDemo.js

```
let nums = 100;

if(nums > 10){
    nums += 23 // this code will run since 100 > 10 is true
}

console.log(nums) // 123
```

ifDemoTwo.js

```
let nums = 5;

if(nums > 10){
    nums += 23 // this code will NOT run since 5 > 10 is false
}

console.log(nums) // 5
```

if 與 else

demo.js

```
// If the conditions in `if` is false, it will run the else code sections
if(false){
  console.log("I will NOT run oh no 😞")
}
else{
  console.log("I will run yeah 😊")
}
```

if 與 else

ifElseDemo.js

```
let nums = 20;

if(nums > 10){
    nums += 23 // if the nums > 10 is true, this line will run
}
else{
    nums -= 10 // if the nums > 10 is false, this line will run
}

console.log(nums) // 43
```

if 與 else

ifElseTwoDemo.js

```
let nums = 5;

if(nums > 10){
    nums += 23 // if the nums > 10 is true, this line will run
}
else{
    nums -= 10 // if the nums > 10 is false, this line will run
}

console.log(nums) // -5
```

進階 `if` 與 `else` 使用

以下代碼邏輯上是不一樣

```
if(true){  
    console.log("I will run yeah 😊")  
}  
  
console.log("I will run no matter if() is true or false")
```

```
if(false){  
    console.log("I will run yeah 😊")  
}  
else{  
    console.log("I will run But only if() is false")  
}
```

if 與 else 與 else if

ifElseDemo.js

```
let nums = 50;

if(nums == 10){
    console.log("I am a 10")
}
else if(nums == 20){
    console.log("I am a 20")
}
else{
    console.log("nope") // This line will be printed
}

// "nope"
```

if 與 else 與 else if

ifElseDemo.js

```
let nums = 10;

if(nums == 10){
    console.log("I am a 10") // This line will be printed
}
else if(nums == 20){
    console.log("I am a 20")
}
else{
    console.log("nope")
}

// "I am a 10"
```


if 與 else 與 else if

ifElseDemo.js

```
let nums = 20;

if(nums == 10){
    console.log("I am a 10")
}
else if(nums == 20){
    console.log("I am a 20") // This line will printed
}
else{
    console.log("nope")
}

// "I am a 20"
```

Functions 函数

Functions 函数

函数是一个为了完成特定任务而设计的代码块。

basic-function.js

```
function myFunctionName(){  
  // Stuff to do when this function is called.  
  console.log("Hello mate.")  
}  
  
myFunctionName() // calling the function
```

Functions with params 帶參數的函數

函數可以傳入參數，以便重複使用。

basic-function-params.js

```
function greetings(names){  
  // Stuff to do when this function is called.  
  console.log("Hello " + names)  
}
```

```
greetings("peter") // Hello peter  
greetings("tom") // Hello tom
```

Functions with more params 帶多個參數的函數

函數也可以傳入很多參數。

basic-function-params.js

```
function greetings(names, ages){  
  // Stuff to do when this function is called.  
  console.log("Hello " + names)  
  console.log("Are you the age of " + ages + " ?")  
}  
  
greetings("peter", 18) // Hello peter / Are you the age of 18 ?  
greetings("tom", 33) // Hello tom / Are you the age of 33 ?
```

Functions return 函數回傳

函數可以回傳 `return` 一個值，讓其他變數使用。

function-return.js

```
function returnSentences(names){  
  let sentences = "Hello " + names;  
  return sentences  
}  
  
let myNameSentences = returnSentences("peter"); //myNameSentences = "Hello peter"  
console.log(myNameSentences); // "Hello peter"  
  
console.log(returnSentences("tom")); // "Hello tom"
```

Advance Functions: Arrow Function (Bonus)

進階函數：箭頭函數

在 JavaScript 中，函數也可以像這樣編寫：

```
let myFunction = (a, b) => a * b;
```

休息一下

Lab 01 - 定義變數和類型

Peter 想要定義一些變數，但他不知道該怎麼做。你可以幫他嗎？

1. 定義兩個 `let` 變數 `income` 和 `outcome`，其值分別為 `10000` 和 `4000`。
2. 定義兩個 `const` 變數 `names` 和 `date`，其值分別為 `"Peter"` 和 `"2022-03-06"`。
3. 定義一個 `let` 變數 `total`，它是基於 `income - outcome` 得出的總數。

Lab 01 - 定義變數和類型

lab02-template.js

```
let someStuff = ...  
const someStuffAlso = ...
```

Lab 02 - 檢查學生階段

編寫一個程序，根據以下要求檢查學生的教育階段：

如果學生的年齡小於5歲，他/她將處於 未接受教育 狀態。

如果學生的年齡在6到12歲之間，他/她將處於 小學 狀態。

如果學生的年齡在13到18歲之間，他/她將處於 中學 狀態。

如果學生的年齡在19到22歲之間，他/她將處於 大學 狀態。

如果學生的年齡大於23歲，他/她將處於 工作 狀態。

Lab 02 - 檢查學生階段

lab02-template.js

```
function calculateAgeStatus(age){  
  console.log(age)  
  // Your code  
}  
  
// Testing  
calculateAgeStatus(4) // 未接受教育  
calculateAgeStatus(59) // 工作  
calculateAgeStatus(16) // 中學  
calculateAgeStatus(8) // 小學  
calculateAgeStatus(21) // 大學
```

總結

我們學習了以下內容：

- 執行 `app.js`
- 用 `console.log` 輸出變量
- 資料型別 `string`, `number`, `boolean`
- 算術運算符 `++`, `*`, `/` 等等
- 賦值運算符 `=`, `+=`, `*=`, `/=` 等等
- 定義變量 `let`, `const`, `var`
- 條件語句 `if`, `else`
- 比較語句 `==`, `===`, `<=` 等等
- 函數 `function add(a,b){ return a + b }`

更多練習

<https://code.tecky.io/>

<https://js.checkio.org/>

更多學習資料

在本課程中，我們沒有涉及以下內容：

- Object, Array, for loop (物件、陣列、for 迴圈)
- Array Looping (for, while) (陣列迴圈 (for, while))
- Scope (作用域)

但如果有興趣，您可以自行閱讀 `lecture-ex1` 的內容。

完結