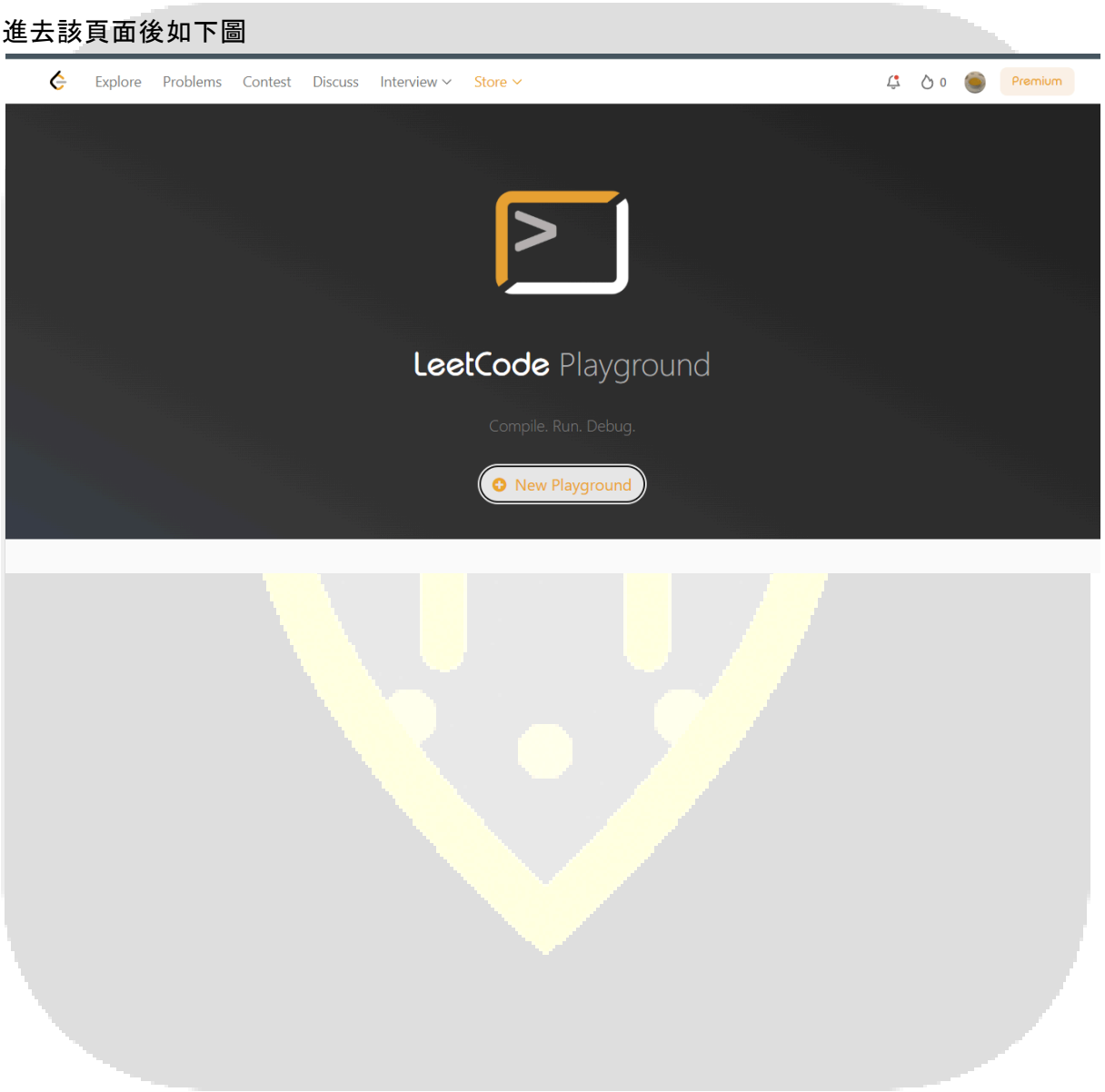
基礎程式講義

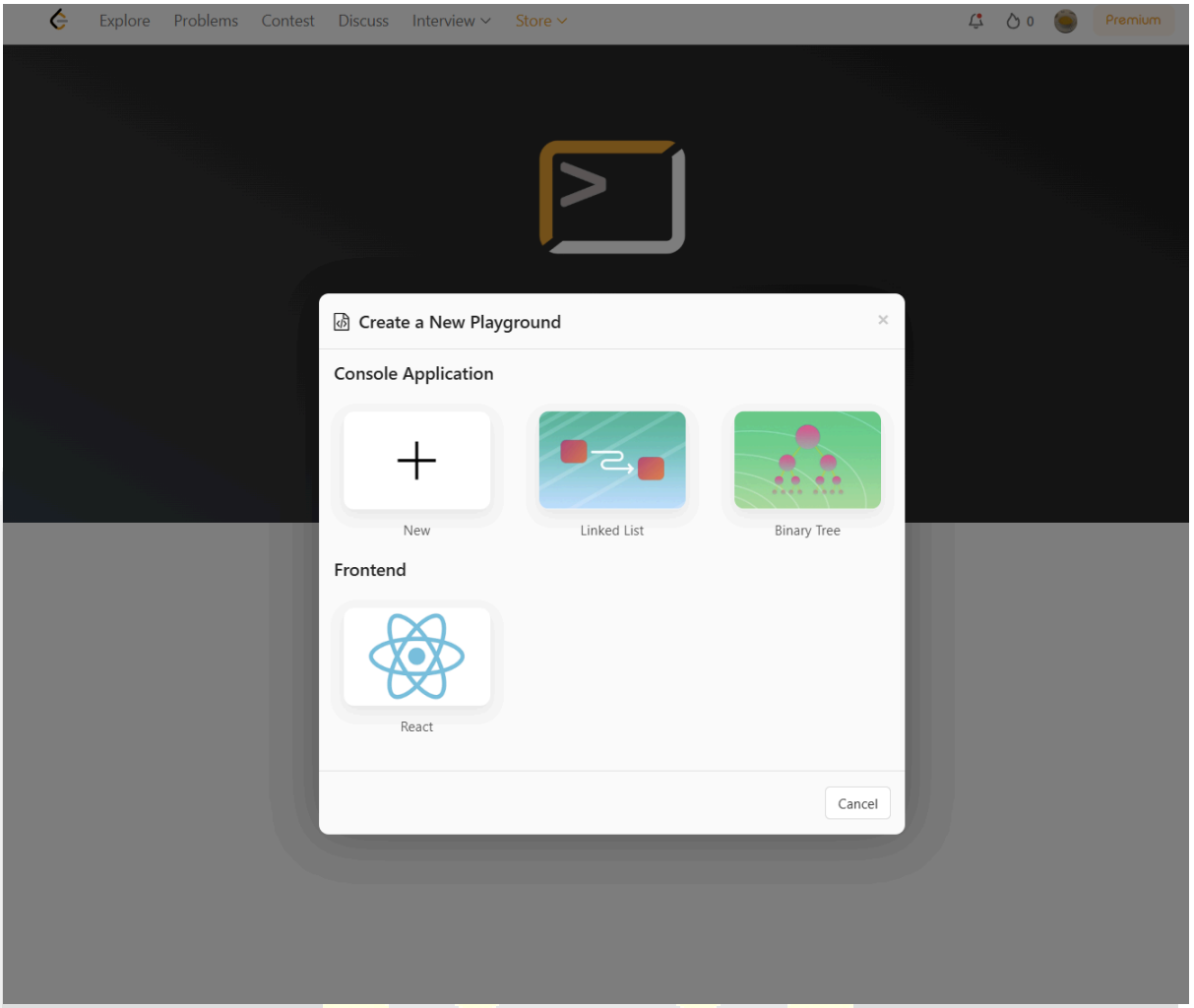
第零章 課前準備

學習程式語言不一定需要先安裝軟體。在初期練習中，我們推薦使用 LeetCode 提供的線上編輯器，操作簡單、不需安裝，能專注在練習語法與邏輯。

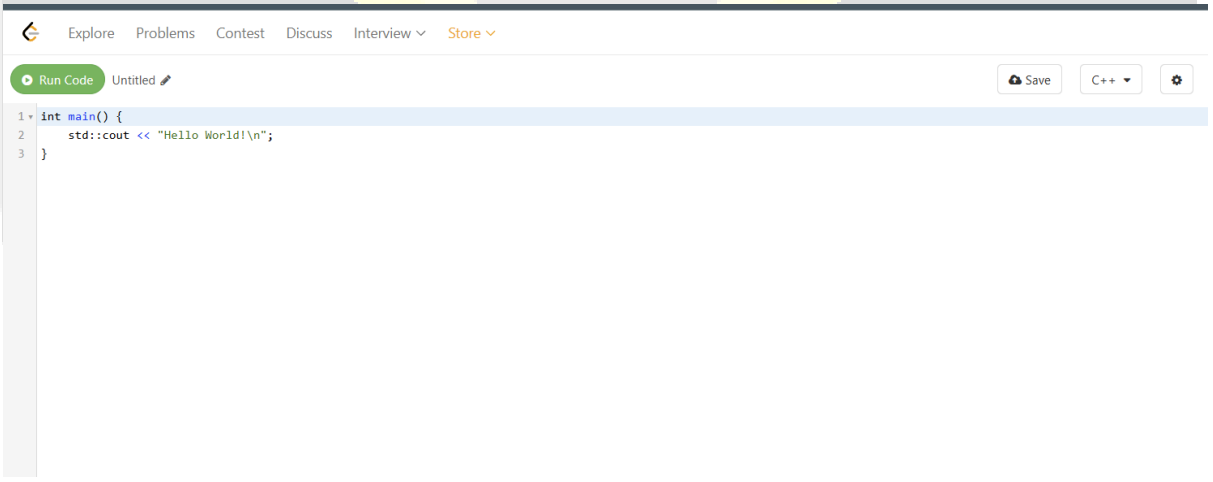
進去該頁面後如下圖



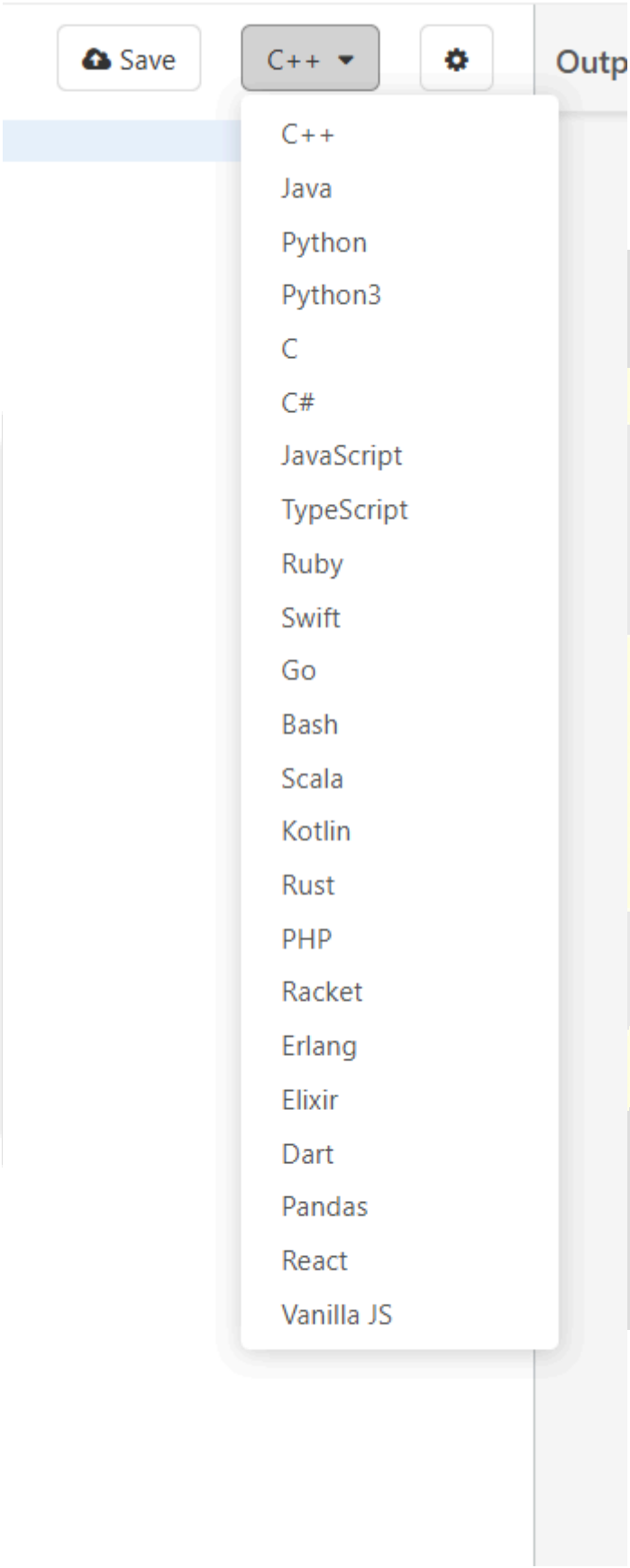
點擊中間 new playground會顯示下圖



選取new之後會如下圖

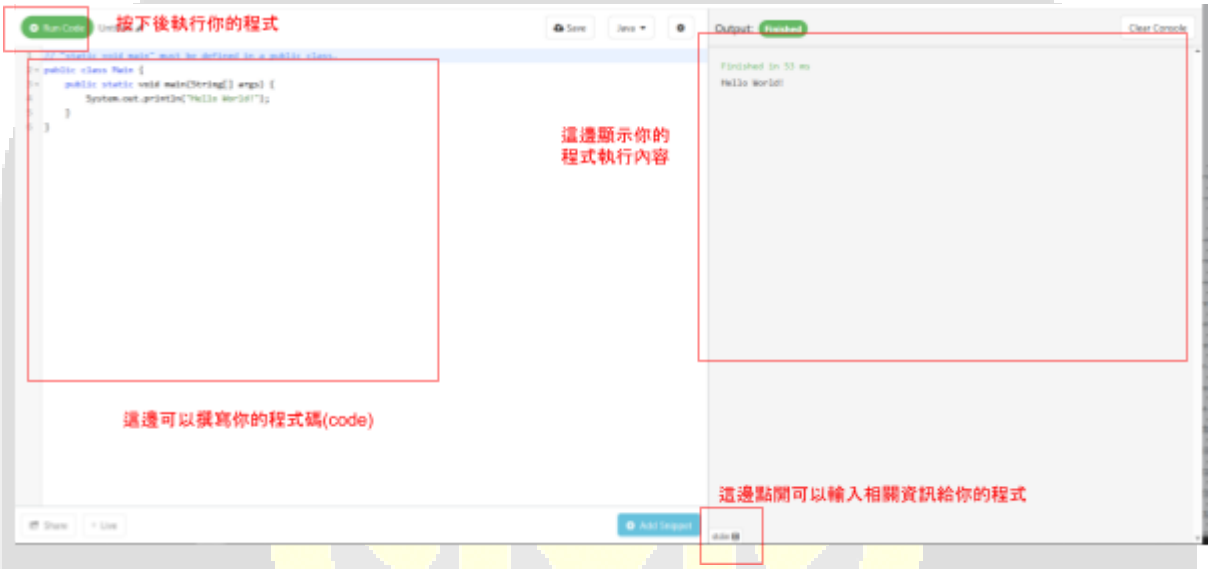


而於右上角C++區域可以選擇自己適合的語言。



由於後續課程我們將使用Java來練習，因此如沒有特殊需求可以選擇Java來做為你的第一個程式語言。

(本課程使用 Java 做為主要語言，但大多數語法概念在其他語言中也很通用。)
(以下相關概念為為了快速銜接後續實戰營而寫，有些例外狀況並不會特別提到，也鼓勵同學針對這幾個章節，有大略概念後可以再自己深入針對這些關鍵字去查詢相關資料，並嘗試寫程式及觀察)



第一章 程式的基本概念

程式就是一組讓電腦按順序執行的指令。大多數程式都是從上往下一行一行地執行。在 Java 中，每個應用程式的進入點是：

```
public static void main(String[] args) {
    // 程式從這裡開始
}
```

這行是 Java 執行的起點，不寫這行的話，Java 不知道該從哪裡開始。

最基本的程式為一行一行執行的，而作為程式練習，首先第一步就是找到程式的起始點，在Java的基礎程式，可以看到 `public static void main(String[] args)` 這一行，而程式就會這一行， 並由這一行向下執行， 直到 `public static void main(String[] args)`， 的最後一行。

```
    public static void main(String[] args) {
        System.out.println("Hello");
        System.out.println("World!");
    }
```

以上述程式為例，當程式開始時，會先執行第一行 `System.out.println("Hello");`接著會執行第二行 `System.out.println("World!");` 接著來講解這兩行分別代表的意思。

第二章 變數及資料型別

變數是用來「儲存資料」的。程式需要記憶某些資料（像是年齡、名字、成績），你需要一個地方來放。

```
int age = 18;
String name = "Alice";
```

若沒有變數則無法儲存和操作資料。譬如說你要將上述年齡+1，就無法做到。

Java 是強型別語言，也就是你必須詳細告訴程式這個變數要儲存的資料類型，如 `int`, `String`, `boolean`。

以下為常見的幾個類型

資料型別	說明	範例
`int`	整數	5, 100, -3
`double`	小數	3.14, 2.718
`String`	字串	"Hello", "World"
`boolean`	布林值（真或假）	true, false

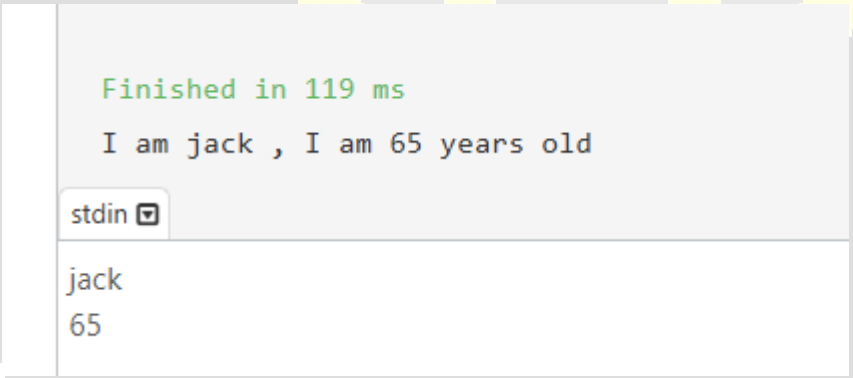
第三章 程式標準輸入輸出(standard input/ standard output)

接著，我們希望能藉由電腦顯示某些資訊給我們，譬如一些文字，而搭配前面的變數章節，也可以將變數的資料顯示出來，可以嘗試跑下列程式

```
public static void main(String[] args) {
        int age = 18;
        String name = "Alice";
        System.out.println("I am " + name + " , I am " + age + "
years old");
    }
```

而若只能輸入固定的東西，那程式會顯得有些無聊，若能配合使用者的輸入去變換輸出的東西
程式可以更活用，所以就有了以下的程式，

```
    public static void main(String[] args) {
        int age;
        String name;
        Scanner scanner = new Scanner(System.in);
        name = scanner.next();
        age = scanner.nextInt();
        System.out.println("I am " + name + " , I am " + age + "
years old");
    }
```



這一章再未來實戰營用到機會比較少，大家可以稍微看過就可以了

第四章 註解

回頭去看第一章的程式，我們發現有一個有趣的東西

```
public static void main(String[] args) {
```

```
    // 程式從這裡開始
}
```

“//程式從這裡開始”，該行的作用主要是補充說明讓之後的開發人員可以更好理解這隻程式要做的事情，也就是我們俗稱的註解，註解是給人看的文字，電腦在執行時會忽略它。通常用來說明程式邏輯、標記待辦事項或暫時停用某段程式碼。

單行註解：

在Java中單行註解通常用”//”表示，表示這行到最後面全部皆為註解

```
// 這是一段註解
System.out.println("Hello"); // 在這行末尾註解
```

多行註解:

在Java中單行註解通常用”/*及”*/”表示，夾在這兩個字串中間的即為註解

```
/*
  這是多行註解
  可以寫很多說明文字
*/
System.out.println("World");
```

為什麼需要註解？

- 幫助自己或他人快速理解程式碼
- 留下解釋、備註或提醒
- 暫時停用某段邏輯測試其他部分

後續章節將運用註解提示學員應該在哪裡撰寫程式。

第五章 函式(function)

在程式設計中，「函式」是非常重要的一個概念。它就像是把一段可以重複使用的邏輯包裝起來，讓你可以隨時呼叫、使用，減少重複程式碼，並讓程式碼更清楚、更有結構。

一個完整的函式通常可以分為三個主要部分：

回傳值型態（**Return Type**）

這是函式執行完後會「回傳」給呼叫者的資料型態。可以是整數（`int`）、小數（`double`）、布林值（`boolean`）、字串（`String`）、陣列、甚至是 `void`（表示沒有回傳值）。

為什麼需要回傳值？

因為很多時候我們會希望函式計算出一個結果後傳回來，方便我們在主程式中使用。

```
int add(int a, int b) {
    return a + b; // 回傳加總結果
}
```

`add` 函式的回傳型態是 `int`，表示這個函式會傳回一個整數。

傳入參數（**Parameters**）

這是函式在被呼叫時，需要輸入進來的資料。函式可以根據不同的輸入參數，做出不同的行為。

為什麼需要參數？

函式是一段可重複使用的程式邏輯，但如果每次執行都只能針對同一組資料，就失去意義了。藉由傳入參數，我們可以讓函式變得「有彈性」。

```
void greet(String name) {
    System.out.println("Hello, " + name + "!");
}
```

傳入一個 `String` 型別的參數 `name`，就可以對不同人打招呼。

函式內容（**Function Body**）

這是函式最重要的部分，也就是函式真正在做什麼事的地方。會根據傳入參數進行運算、邏輯判斷、迴圈處理，最後產生回傳值或執行某些動作。

```
int square(int x) {
```

```
    return x * x;
}
```

這段程式中，函式的本體是計算平方後回傳。

後續的章節，學員主要需要修改函式的內容，來完成我們的練習題

第六章 數字運算

在程式語言中，我們可以對數字做各種運算，這些運算大多與數學類似，但在語法與使用上有一些需要注意的地方。

基本運算子

運算子	名稱	說明	範例
+	加法	將兩個數字相加	3 + 2 = 5
-	減法	將左邊的數字減去右邊的數字	5 - 2 = 3
*	乘法	將兩個數字相乘	4 * 3 = 12
/	除法	將左邊的數字除以右邊的數字（整數除整數會省略小數）	7 / 2 = 3（不是3.5）
%	餘數（mod）	取除法的「餘數」	7 % 2 = 1

注意：大部分程式語言中若兩個整數相除，結果也是整數，小數部分會被無條件捨棄。

---

x = x + 1 是什麼意思？

你可能會看到這樣的程式碼：

```
x = x + 1;
```

這在數學上看起來像是不合理的等式，因為左邊和右邊的值不一樣。但在程式中，這不是「等式」，而是「指令」，意思是：

把目前的 **x** 加 **1**，然後存回 **x** 裡面」

所以這行的執行流程是：

1. 取出 x 現在的值
2. 加 1
3. 把結果重新存到 x 裡

範例：

```
int x = 3;
x = x + 1; // 現在 x 變成 4
```

自增與自減運算子

為了簡化 `x = x + 1` 這種常見寫法，我們可以使用 自增運算子 **`++`**，或 自減運算子 **`--`**。

`x++`：等同於 `x = x + 1`

`x--`：等同於 `x = x - 1`

這些語法讓程式更簡潔、好讀。

```
int x = 5;
x++; // x 現在是 6
x--; // x 現在又變回 5
```

為什麼需要這些運算？

在許多實際應用中，我們會經常需要對某個變數進行「加一」、「減一」的操作，例如：

- 計數器（像是計算有幾個人通過門口）
- 迴圈(第八章)中的次數控制
- 對陣列(第九章)進行遍歷（例如 `i++`）

練習

輸入為華氏溫度轉換成攝氏溫度

華氏溫度 = 攝氏溫度 * 9 / 5 + 32

請將下列程式碼貼入code部分並完成指定函式

```
public class Main {
    public static int celsiusConvertToFahrenheit(int celsius) {// celsius 為攝氏溫度
        // 請由此撰寫你的程式
    }
    public static void main(String[] args) {
        int celsius;
        Scanner scanner = new Scanner(System.in);
        celsius = scanner.nextInt();
        System.out.println(celsiusConvertToFahrenheit(celsius));
    }
}
```

第七章 if else條件

在程式設計中，「條件判斷」可以讓我們根據不同的情況，做出不同的處理方式。舉個簡單的例子：如果考試分數大於 60，就顯示「及格」，否則顯示「不及格」。這樣的邏輯在程式中，就是透過 `if` 與 `else` 來實作的。

`if` 的基本語法

```
if (條件) {
    // 條件為 true 時會執行的程式碼
}
```

範例：

```
int score = 85;
if (score >= 60) {
    System.out.println("及格！");
}
```

```
```

如果 `score >= 60` 為真，就會執行 `{}` 內的程式。

二、`if-else` 結構

當我們想根據「條件成立與否」來做兩種不同處理時，可以使用 `if-else`：

```
if (條件) {
    // 條件為 true 時執行
} else {
    // 條件為 false 時執行
}
```

範例：

```
int score = 45;
if (score >= 60) {
    System.out.println("及格！");
} else {
    System.out.println("不及格！");
}
```

多重條件：`else if`

當條件不只有兩種情況時，我們可以使用 `else if` 來擴充判斷邏輯。

範例：成績分級

```
int score = 87;
if (score >= 90) {
    System.out.println("優等");
} else if (score >= 80) {
    System.out.println("甲等");
```

```
} else if (score >= 70) {
    System.out.println("乙等");
} else if (score >= 60) {
    System.out.println("丙等");
} else {
    System.out.println("不及格");
}
```

條件運算子

在 `if` 條件中，常用的運算子如下：

運算子	說明	範例（結果）
`==`	等於	`a == 3`（true）
`!=`	不等於	`a != 0`
`>`	大於	`a > 10`
`<`	小於	`b < 5`
`>=`	大於或等於	`a >= 60`
`<=`	小於或等於	`b <= 100`
`&&`	並且（AND）	`a > 0 && b > 0`
`||`	或（or）	`a > 0 && b > 0`

注意 `&&` 與 `||` 若只打單個，會是完全不一樣的意思，`&` 與 `|` 主要是針對數字，將數字轉換成二進位後做位元運算，例如 5 `&` 12，我們將5跟12轉換成二進位，$(0101)_2$跟$(1100)_2$，他們 `&` 出來的結果就會是$(0100)_2$也就是4。

範例：同時判斷年齡與是否學生

```
int age = 18;
boolean isStudent = true;

if (age <= 18 && isStudent) {
    System.out.println("可以申請學生優惠票");
}
```

六、常見錯誤與注意事項

1. 忘記用雙等號（==）判斷

```
if (x = 5)  // 錯誤！這是指定不是判斷
```

應該改成：

```
if (x == 5)
```

2. 忘記加大括號 {}（尤其在多行時）

```
if (x > 0)
    System.out.println("正數");
    System.out.println("這行會不管條件照常執行！"); // 錯誤行為
```

建議習慣即使只有一行，也加上 {}，避免日後維護出錯：

```
if (x > 0) {
    System.out.println("正數");
}
```

3. 多重條件順序錯誤

```
if (score >= 60) {
    System.out.println("及格");
} else if (score >= 90) {
    System.out.println("優等");
}
// 錯誤：永遠不會進入 >=90 的分支，因為已經先判斷 >=60
```

程式練習

練習題：成績等級判斷

請撰寫一個程式，讓使用者輸入一個整數（0～100之間），代表學生成績，然後根據下列條件回傳對應的等級：

分數範圍	顯示等級
90 分以上	A
80～89 分	B
70～79 分	C
60～69 分	D
60 分以下	F

請將下列程式碼貼入code部分並完成指定函式

```
public class Main {
    public static String getGrade(int score) {
        // 請由此撰寫你的程式 可先嘗試執行以下程式觀察結果
        return "A";
    }
    public static void main(String[] args) {
        int score;
        Scanner scanner = new Scanner(System.in);
        score = scanner.nextInt();
        System.out.println(getGrade(score));
    }
}
```

第八章 迴圈(loop)

在現實生活中，我們常常會做「重複性的動作」，例如：

- 每天刷牙 2 次，連續 7 天

- 每個人發一封信
- 從 1 數到 100

這些事情用程式來描述，就會用到「迴圈（Loop）」的概念。

基本語法

```
for (初始化區; 條件區; 更新區) {
    // 每次重複執行的程式區塊
}
```

若不需要初始化區及更新區也可以使用while

```
while (條件區) {
    // 每次重複執行的程式區塊
}
```

範例一:

```
for (int i = 1; i <= 5; i++) {
    System.out.println("第 " + i + " 次 Hello");
}
```

執行流程:

1. `int i = 1`：設定初始值（只做一次）
2. `i <= 5`：判斷是否繼續執行
3. 印出 `Hello`
4. `i++`：更新變數
5. 回到第 2 步，直到條件不成立為止

範例二:

```
int i = 5

while(i!=0){
  System.out.println(“倒數第 " + i + " 次 Hello");
  i--;
```

```
}
```

迴圈常見應用

從 1 加到 100

```
int sum = 0;
for (int i = 1; i <= 100; i++) {
    sum += i;
}
System.out.println("總和為：" + sum);
```

列出所有偶數

```
for (int i = 2; i <= 20; i += 2) {
    System.out.print(i + " ");
}
```

倒數

```
for (int i = 10; i >= 1; i--) {
    System.out.println(i);
}
```

常見錯誤與陷阱

錯誤例子	說明
`for (int i = 0; i < 10; )`	忘記遞增變數，會造成無限迴圈
`for (int i = 10; i < 0; i++)`	條件永遠為 `false`，完全不會執行

程式練習

輸入一個數字n，印出一個三角形總共有n行，第一行有一個星號”*”,第二行有兩個星號”**”，第三行有三個星號”***”。

範例輸入

```
5
```

範例輸出

```
*
**
***
****
*****
```

```
public class Main {
    //印出一個換行
    public static void printNextLine() {
        System.out.println("");
    }
    //印出一個星號
    public static void printStar() {
        System.out.print("*");
    }
    //印出一個空格
    public static void printSpace() {
        System.out.print(" ");
    }
    public static void printTriangle(int n) {
        // 請由此撰寫你的程式 可先嘗試執行以下程式觀察他的結果
        printSpace();
        printStar();
        printNextLine();
    }
    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        printTriangle(n);
    }
}
```

程式練習2

輸入一個數字n，印出一個金字塔總共有n行，第一行有一個星號”*”,第二行有三個星號”***”，以此類推，並讓所有輸出的文字置中。

範例輸入

```
5
```

範例輸出

```
    *
   ***
  *****
 *******
*********
```

```
public class Main {
    //印出一個換行
    public static void printNextLine() {
        System.out.println("");
    }
    //印出一個星號
    public static void printStar() {
        System.out.print("*");
    }
    //印出一個空格
    public static void printSpace() {
        System.out.print(" ");
    }
    public static void printPyramid(int n) {
        // 請由此撰寫你的程式 可先嘗試執行以下程式觀察他的結果
        printSpace();
        printStar();
        printNextLine();
    }
    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        printPyramid(n);
    }
}
```

程式練習3
輸入一個數字，判斷他有幾個因數，也就是有幾個>0的數字可以將他整除

範例輸入

```
6
```

範例輸出

```
4
```

範例解釋：6總共有 1 2 3 6這四個因數

```
public class Main {

    public static int numberOfFactor(int n) {
        // 請由此撰寫你的程式
    }
    public static void main(String[] args) {
        int n;
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        System.out.println(numberOfFactor(n));
    }
}
```

第九章 陣列(array)

在寫程式時，我們經常會遇到需要儲存「一組相似資料」的情況，例如：

- 存放 5 位同學的成績
- 紀錄一週的氣溫
- 儲存多個輸入數字後進行加總、平均、排序

這時候就不能用一個變數變數 score1、score2、score3 來處理，我們需要一個「可以一次儲存多個數值的容器」，那就是—陣列（**Array**）。

什麼是陣列？

陣列就是「一組相同型別的變數集合」，你可以用一個名字來代表整組資料，透過索引（index）來讀取或修改裡面的每一個元素。

陣列的宣告與初始化

方法 1：先宣告、再給值

```
int[] scores = new int[5]; // 宣告一個整數陣列，有 5 個元素（index 0 ~ 4）

scores[0] = 80;
scores[1] = 90;
scores[2] = 85;
scores[3] = 70;
scores[4] = 60;
```

方法 2：宣告時直接給值（常用）

```
int[] scores = {80, 90, 85, 70, 60};
```

陣列索引（index）

- 陣列的索引是從 0 開始編號
- 若有 5 個元素，索引會是：0, 1, 2, 3, 4
- 使用語法 array[索引] 來存取

```
System.out.println(scores[0]); // 印出第 1 個數：80
System.out.println(scores[4]); // 印出第 5 個數：60
```

搭配 for 迴圈使用

陣列最常與迴圈一起使用，因為可以逐一處理每個元素。

```
for (int i = 0; i < scores.length; i++) {
    System.out.println("第 " + (i+1) + " 位同學分數：" + scores[i]);
}
```

常見錯誤與陷阱

錯誤類型	說明

`ArrayIndexOutOfBoundsException`	超出索引範圍，例如：`scores[5]`（應該只有 `0~4`）
索引從 `1` 開始	錯誤觀念：陣列索引從 `0` 開始！

陣列常見應用範例

加總所有數字

```
int sum = 0;
for (int i = 0; i < scores.length; i++) {
    sum += scores[i];
}
System.out.println("總分為：" + sum);
```

找出最大值

```
int max = scores[0];
for (int i = 1; i < scores.length; i++) {
    if (scores[i] > max) {
        max = scores[i];
    }
}
System.out.println("最高分為：" + max);
```

二維陣列（2D Array）

什麼是二維陣列？

如果說一維陣列就像是一排座位，二維陣列就像是「教室裡的座位表」—一個表格（矩陣）狀的結構。

簡單來說，二維陣列就是：

> 陣列中的每個元素又是一個陣列。

宣告與初始化

宣告格式

```
資料型態[][] 陣列名稱 = new 資料型態[列數][欄數];
```

範例：3列4欄的整數表格

```
int[][] table = new int[3][4];
```

這代表有 3 列、4 欄，總共 3*4 = 12 個元素。

宣告時直接給值

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

存取與指定元素

```
matrix[0][0] = 10; // 第一列第一欄
System.out.println(matrix[1][2]); // 第二列第三欄：6
```

巢狀 `for` 迴圈搭配使用

二維陣列幾乎一定會搭配「巢狀 `for` 迴圈」來處理每個元素。

範例:印出二維陣列內容

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
```

第 十 章：結構體（struct / class）與資料封裝

為什麼需要 struct / class？

在程式中，我們常常會遇到一種狀況:一個實體需要多個資料欄位來描述。
例如要表示一個學生，可能包含:

- 姓名(字串)

- 年齡(整數)
- 成績(浮點數)

雖然可以用三個變數去儲存，但這樣資料零散且不易管理，這時我們就會用結構化資料來幫助我們把資料「包」在一起。

Java 中的用法：class

Java 可以用 **class + 成員變數** 來達成資料封裝的目的。

宣告範例：

```
public class Student {
    String name;
    int age;
    float score;
    public Student(String name, int age, float score) {
        this.name = name;
        this.age = age;
        this.score = score;
    }

    void printInfo() {
        System.out.println(name + " 年齡：" + age + " 成績：" +
score);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student("Alice", 20, 88.5f);
        s.printInfo();
```

```
    }
}
```

第 十一 章：泛型（Generics）－讓你的程式更通用

為什麼需要「泛型」？

在寫程式的時候，我們常常會遇到這種情況：

- 想做一個函式，能處理 任意型別的資料。
- 又希望 型別是安全的，不要錯誤傳入不對的資料。

例如一個「取得陣列第一個元素」的函式，不管傳入的是整數陣列、字串陣列還是學生物件陣列，邏輯是一樣的，不該重複寫多份。

泛型的目標：一次寫好邏輯、套用不同型別。

會特別介紹這一章主要是，未來在使用相關資料結構時，將會很常看到這類用法，譬如說 `Map<string, int>`。

Java 中的泛型（Generics）

泛型類別範例：

```
public class Box<T> {
    private T value;

    public void set(T value) {
        this.value = value;
    }

    public T get() {
        return value;
    }
}
```

使用泛型：

```
public class Main {
    public static void main(String[] args) {
        Box<Integer> intBox = new Box<>();
        intBox.set(123);
        System.out.println(intBox.get());

        Box<String> strBox = new Box<>();
        strBox.set("Hello");
        System.out.println(strBox.get());
    }
}
```

說明:

- `<T>` 是一個「型別參數」，可替代為任何型別，也可以使用多組例如`<T , P>`。

附錄 debug

我們程式寫出來可能有些錯誤，這時候需要去找我們程式有哪裡寫錯，這個動作就稱為debug，而debug有很多種方式，以下介紹一個最樸素的方式。

使用 **stanrdard out / log debug**

System.out.println("目前 x 值: " + x);

- 確認變數是你預期的值
- 確認程式是否進入 if / else
- 確認 for 迴圈是否有照預期執行

首次 debug 看不懂錯在哪，選用將變數當下的直印出來參照是最簡單最直接的方式，
這種方式也就是常聽到的寫log。(未來會有其他更方便的工具可以協助debug)

相關練習題
https://leetcode.com/problems/harshad-number/description/ (迴圈 if else條件判斷)
https://leetcode.com/problems/running-sum-of-1d-array/description/ (迴圈 陣列)
https://leetcode.com/problems/ant-on-the-boundary/description/ (迴圈 陣列 數字運算)
https://leetcode.com/problems/find-lucky-integer-in-an-array/description/ (迴圈 陣列 數字運算 if else條件判斷)
https://leetcode.com/problems/shuffle-the-array/description/ (迴圈 陣列)
https://leetcode.com/problems/number-of-students-doing-homework-at-a-given-time/description/ (迴圈 if else條件判斷 數字運算)
https://leetcode.com/problems/type-of-triangle/description/ (if else條件 數字運算)

https://leetcode.com/problems/find-numbers-with-even-number-of-digits/description/ (迴圈 if else條件 數字運算)