Javascript 基礎

一、Javascript 介紹

Javascript是什麼？

1. Javascript是一種運行在客戶端(瀏覽器)的編程語言
2. 作用：
    a. 網頁特效
    b. 表單驗證(針對表單數據的合法性進行判斷)
    c. 數據交互(獲取後台的數據，渲染到前端)
    d. 服務端編程(node.js)
3. 組成：
    a. ECMAscript
    b. Dom
    c. Bom

書寫位置

1. 內部 Javascript：寫在 HTML 裡，用 <script>標籤 包住
    a. <script>標籤寫在 </body> 上面
    b. <script>標籤放在HTML文件的底部，原因是瀏覽器會按照代碼在文件中的順序加載HTML
2. 外部 Javascript
    a. 代碼寫在以.js結尾的文件裡
    b. 語法：通過<script>標籤，引用到HTML頁面中
3. 行內 Javascript
    a. 代碼寫在標籤內部
    b. 不常用

註釋

1. 單行註釋
    a. 符號：//
    b. 作用：// 右邊的代碼會被忽略
    c. 快捷鍵：crl + /
2. 塊註釋
    a. 符號：/* */
    b. 作用：/* 和 */ 之間的內容會被忽略
    c. 快捷鍵：shift + alt +a

結束符

1. 作用：使用英文的『;』代表語句結束

2. 實際情況：可寫也可不寫，瀏覽器可以自動推斷語句的結束位置
3. 約定：為了風格統一，按照團隊要求

輸入和輸出語法

1. 輸出語法：
    a. document.write("內容")
        i. 作用：向 <body> 內輸出內容
        ii. 如果輸出內容寫的是標籤，也會被解析成網頁元素
    b. alert("警告對話筐")
        - 作用：頁面彈出警告對話筐
    c. console.log("控制台打印")
        - 作用：控制台輸出語法，測試使用
2. 輸入語法：
    - prompt("請輸入你的姓名：")
        - 作用：顯示一個對話筐，用來提示用戶輸入文字

字面量

- 在計算機科學中，字面量是在計算機中描述事 / 物
    a. 字符串 字面量：小宅
    b. 數字 字面量：20
    c. 數組 字面量：[ ]
    d. 對象 字面量：{ }

二、變量

變量是什麼？

1. 變量是計算機中用來儲存數據的『容器』，它可以讓計算機變得有記憶
2. 注意：變量不是數據本身，而是用於儲存數據的容器

變量的基本使用

1. 變量的聲明
    a. 聲明變量：要想使用變量，首先需要創建變量(也稱作定義變量)
    b. 語法：`let` 變量名
        i. 聲明變量有兩部分組成：聲明關鍵字、變量名(標誌)
        ii. `let`(允許、許可、讓)即關鍵字，所謂關鍵字是系統提供用來聲明(定義)變量的詞語
2. 變量的賦值
    a. 定義一個變量後，就能夠初始化它(賦值)。在變量名之後跟上一個賦值號 『=』，然後是數值
    b. 『=』：賦值運算符
    c. 簡單作法：聲明變量時，直接完成賦值操作(變量初始化)
3. 更新變量
    a. 變量賦值後，可以簡單地給它一個不同的值來更新它
    b. 注意：`let` 不允許多次聲明同一個變量
4. 聲明多個變量
    a. 語法：多個變量中間用逗號隔開(不推薦)
    b. `let age=18 , name="小宅"`

變量的本質

1. 內存：計算機中儲存數據的地方，相當於一個空間
2. 變量本質：是程序在內存中申請的一塊用來存放數據的小空間

變量命名規定與規範

1. 規則：
    a. 不使用關鍵字：`let`、`if`、`for`…
    b. 只能用底線、字母、數字、$組成，且數字不能開頭
    c. 字母嚴格區分大小寫
2. 規範：
    a. 命名要有意義
    b. 遵守小駝峰命名法
        - 第一個單詞首字母小寫，後面每個單詞首字母大寫

變量拓展 - 數組

1. 數組(`Array`)，一種 將一組數據儲存在單個變量名下 的優雅方式

a. `let arr = [ 10, 20, 30, 40 ]`

b. 數組是按順序保存，每個數據都有自己的編號

c. 計算機中的編號從 0 開始

d. 在數組中，數據的編號也叫做 索引 或 下標

e. 數組可以儲存任意類型的數據

2. 數組術語：

a. 元素：數組中保存的每個數據都叫做數組元素

b. 下標：數組中數據的編號

c. 長度：數組中數據的個數，通過數組的 length 屬性 獲得

i. arr.length

ii. 數組長度 = 索引號 + 1

三、常量

常量是什麼？

1. 常量：使用 `const` 聲明的變量
2. 使用場景：當某個變量永遠不會改變時，就可以使用 `const` 來聲明，而不是 `let`
3. 命名規範：和變量一致
4. 注意：常量不允許重新賦值、聲明的時候必須賦值(初始化)
5. 小技巧：不需要重新賦值的數據使用 `const`

四、數據類型

弱數據類型

- Javascript只有賦值之後，才能確認數據類型

基本數據類型

1. 字串型(string)
    a. 通過 單引號、雙引號、反引號 包裹的數據
    b. “ ” 空字串
    c. 字符串拼接：『+』(數字相加、字符相連)
    d. 模板字符串
        i. 外面使用` `反引號，裡面使用 ${變量名}
        ii. let age = 27
            document.write(` 我今年 ${age} 歲了 `)
2. 數字型(number)
    a. 正數、負數、小數
    b. 算數運算符：+(加)、-(減)、*(乘)、/(除)、%(取餘數)
    c. NaN(Not a Number)：代表計算不正確的錯誤
3. 布林值型(boolean)
    - 肯定(true)、否定(false)
4. 未定義型(undefined)
    a. 只聲明變量，不賦值的情況下，變量的默認值為undefined
    b. 檢測變量是undefined，說明沒有值傳遞過來
5. 空類型(null)
    a. 代表無、空、值未知
    b. 官方解釋：把null作為尚未創建的對象
    c. 將來有個變量裡面存放的是一個對象，但對象還沒創建好，可以先給個null
6. undefined和null的差別：
    a. undefined表示沒有賦值
    b. null代表賦值了，但內容為空

引用數據類型

1. 對象(Object)

檢測數據類型

1. 控制台
2. typeof x 、typeof(x)

五、類型轉換

為什麼要類型轉換？

- 坑：使用表單、prompt( )獲取而來的數據默認是字符串類型，不能直接使用數字運算

隱式轉換

1. 『+』兩邊只要有一個是字符串，都會把另一個 轉換成 字符串型
2. 除了『+』以外，『-』『*』『/』會把數據 轉換成 數字型
3. 『+』作為 正號 解析可以轉換成 數字型
   - console.log(+"100") // 100 （數字型）
4. 任何數據和字符串相加，結果都是字符串

顯式轉換

1. 自己寫代碼告訴系統該轉成什麼類型
2. 轉換為 數字型
   a. Number(數據)
      i.
```
let str = "100"
console.log(Number(str))     // 123 （數字型）
```
      ii. 如果字串裡有非數字，轉換失敗，結果為 NaN （不是一個數字）
   b. parseInt(數據)
      i. 只保留整數
      ii.
```
console.log(parseInt("12px"))    // 12
```
      iii.
```
console.log(parseInt("12.77px"))     // 12
```
      iv.
```
console.log(parseInt("abc12px"))     // NaN
```

   c. parseFloat(數據)
      i. 可以保留小數點
      ii.
```
console.log(parseFloat("12.55px"))  // 12.55
```
      iii.
```
console.log(parseFloat("abc12.55px"))    // NaN
```

六、運算符

賦值運算符

1. 『=』：賦值，對變量進行賦值的運算符
2. 等號右邊的值賦予給左邊，要求左邊必須是一個容器
3. 『+=』、『-=』、『*=』、『/=』、『%=』

一元運算符

1. 只需要一個操作數就可以運算的運算符叫一元運算符，例子：正負號等
2. 二元運算符：`let num = 10 + 20`
3. 自增（減）運算符的用法：
    a. 前置自增
    ```
    let num = 1
    ++num
    console.log(num)    // 2
    ```
    b. 後置自增
    ```
    let num = 1
    num++
    console.log(num)          // 2
    ```
4. 前置自增 和 後置自增 的差別：
    a. 前置自增(加加在前，先自增再運算)
    ```
    let i = 1
    console.log(++i + 2)    // 4
    ```
    b. 前置自增(加加在後，先運算再自增)
    ```
    let i = 1
    console.log(i++ + 2)    // 3
    ```
5. 獨立使用時，兩者並無差別
6. 後置自增 i++，使用相對較多，且都是單獨使用

比較運算符

1. 比較結果為 **boolean** 類型，即只會得到 **true** 或 **false** (比較運算符有隱式轉換)
2. 『<』、『>』、『>=』、『<=』、『==』、『===』、『!=』、『!==』
3. 『==』：只判斷兩邊『值』是否相等
    a. `console.log("2" == 2)    // true`
    b. `console.log(undefined == null)  // true`
4. 『===』：判斷兩邊『值和數據類型』是否全等(推薦使用)
    a. `console.log("2" === 2)    // false`
    b. `console.log(undefined === null)  // false`
    c. `console.log(NaN === NaN) // false`

- NaN 不等於任何人，包括 NaN 它本身

5．特殊說明：

a．如果是數字和 〝其他值〞 的比較，則其他值會自動轉換成數字去比較

b．如果是字串符和字串符的比較，則會比較每一個字符的ASCII碼，按位進行比較

c．如果是布林值參與比較，布林值會轉換成數字 0 和 1

d．數字只有 0 是 false，其餘為 true

e．字符只有 〝 〞 是 false，其餘為 true

邏輯運算符

1．比較結果為 boolean 類型，即只會得到 true 或 false

2．&&：並且，符號兩邊為 true，結果才為 true，一假則假

3．||：或者，符號兩邊有一個 true，結果就為 true，一真則真

4．！ ：取反，true 變 false、false 變 true

運算符優先級

優先級	運算符	順序
1	小括號	（ ）
2	一元運算符	++ - - ！
3	算數運算符	先 * % 後 + -
4	比較運算符	>、>=、<、<=
5	相等運算符	==、!=、===、!===
6	邏輯運算符	順序：！、&&、 ||
7	賦值運算符	=
8	逗號運算符	,

七、語句

表達式和語句

1. 表達式：可以被求值的代碼，Javascript 解釋器會將其計算出一個結果
    a. x = 7
    b. 3 + 4
    c. num ++
2. 語句：一段可以執行 Javascript 某個命令的代碼，比如：prompt( )
    - 順序語句、if 條件語句、for 循環語句

- 表達式 和 語句 的差別：
    a. 表達式：可以計算出一個值，所以可以寫在賦值語句的右側
        - num = 3 + 4
    b. 語句：不一定有值，用來自行以使某件事發生
        - alert()、console.log()

八、分支語句

- 分支語句：可以有 選擇性 的執行想要的代碼

if 語句

1. 單分支 語法：
   - ```
     if (條件) {
         滿足條件要執行的代碼
     }
     ```
   a. 括號內的條件為true時，進入大括號裡執行代碼
   b. 小括號內的結果若不是布林值，會發生 隱式轉換 變成 布林值
   c. 除了 0 以外，所有的 數字 都為true
   d. 除了 ” ” 以外，所有的 字符串 都為true
2. 雙分支 語法
   - ```
     if (條件) {
         滿足條件要執行的代碼
     }
     else {
         不滿足條件要執行的代碼
     }
     ```
3. 多分支 語法
   - ```
     if (條件) {
         滿足條件要執行的代碼
     }
     else if (條件) {
         滿足條件要執行的代碼
     }
     else {
         以上條件都不滿足要執行的代碼
     }
     ```

三元運算符

1. if 雙分支，更簡單的寫法
2. 語法：
   - 條件 ? 滿足條件執行的代碼 : 不滿足條件執行的代碼
     ```
     console.log(3 > 5 ? true : false)    // false
     ```
3. 一般用來取值
   - ```
     let sum = 3 < 5 ? 100 : 200
     console.log(sum)     // 100
     ```

4. 案例：

- 數字補零
- ```
  let num = prompt("輸入一個數字：")
  num = num < 10 ? 0 + num : num
  console.log(num)
  ```

switch 語句

1. 語法：
   - ```
     switch (數據) {
         case 值1:
             代碼1
             break
         case 值2:
             代碼2
             break
         default:
             代碼n
     }
     ```
2. 找到跟小括號全等『===』的 case 值，並執行對應的代碼，反之，執行 default 的代碼
3. switch case 語句一般用於等值判斷，不適合區間判斷
4. 搭配 break 用，否則會造成 case 穿透

- if 語句 和 switch 語句的差別：
  a. if 語句：通常用於範圍判斷，分支少時，執行效率高
  b. switch 語句：通常處理 case 為比較確定值的情況，分支多時，執行效率高，結構更清晰

九、循環語句

- 斷點調試：幫助理解代碼運行

while 循環

1. while 循環：在滿足條件期間，重複執行某些代碼
2. 語法：
   - while（循環條件）{
     要重複執行的代碼（循環體）
     }
3. 滿足小括號裡的條件為 true 才會進入 循環體 執行代碼

4. while 大括號裡代碼執行完畢後不會跳出，而是繼續回到小括號裡判斷條件是否滿足，
   若滿足執行大括號裡的代碼，然後再回到小括號判斷條件，直到小括號內條件不滿足，即跳出

5. while 循環三要素 （循環是以某個變量為起始值，然後不斷產生變化量，慢慢靠近終止條件的過程）
   a. 變量起始值
   b. 終止條件（沒有終止條件，循環不停止，造成死循環）
   c. 變量變化量（自增、自減）

```
- let i = 1
  while (i <= 3) {
      console.log("循環3次")
      i++
  }
```

6. 退出循環
   a. continue：結束本次循環，繼續下次循環（一般用於 排除 或者 跳過某一選項 的時候）

```
- let i = 1
  while (i <= 5) {
      if (i === 3) {
          i++
          continue    //  退出本次循環，countine 以下的語句不再執行
      }
      console.log(i)  //  跳過第3次循環，1245”
  }
```

   b. break：退出循環（一般用於結果已經得到，後續的循環不需要的時候）

```
- let i = 1
  while (i <= 5) {
      if (i === 3) {
          break   //  結束 for 循環
      }
```

```
            console.log(i)   //   只循環2次，12
            i++
        }
    p33
```

for 循環

1. for 循環：在滿足條件期間，重複執行某些代碼，比 **while** 循環 一目瞭然
2. 語法：
   - for (變量起始值; 終止條件; 變量變化量(自增、自減)) {
     循環體
     }
   - ```
     for (let i = 1; i <= 3; i++) {
         console.log("循環3次")
     }
     ```
3. for 循環 最大價值：遍歷數組
   - ```
     let arr = ["春", "夏", "秋", "冬"]
     for (let i = 0; i <= arr.length - 1; i++) {
         console.log(arr[i])
     }
     ```
   - ```
     let arr = ["春", "夏", "秋", "冬"]
     for (let i = 0; i < arr.length; i++) {
         console.log(arr[i])
     }
     ```
4. 退出循環
   a. countine 退出本循環，一般用於 排除 或 跳過某一個選項 的時候
   - ```
     for (let i = 1; i <= 5; i++) {
         if (i === 3) {
             continue    //  退出本次循環，countine 以下的語句不再執行
         }
         console.log(i)     //  跳過第3次循環，1245
     }
     ```

   b. break 退出整個for循環，一般用於 結果已經得到，後續的循環不需要的時候
   - ```
     for (let i = 1; i <= 5; i++) {
         if (i === 3) {
             break          //  結束 for 循環
         }
         console.log(i)     //  只循環2次，12
     }
     ```
5. 無限循環
     ```

- `while (ture) { }`、`for (; ;) { }`，用來構造無限循環，都需要使用 `break` 退出循環

- `while` 循環 和 `for` 循環 的差別：
  a. 當 明確 循環的次數，推薦使用 `for` 循環
  b. 當 不明確 循環的次數，推薦使用 `while` 循環

6. 雙層 `for` 循環 崁套
   - 一個循環裡再套一個循環，一般用在 `for` 循環

```
for (let i = 1; i <= 3; i++) {
        console.log(`第 ${i} 天`)
      for (let n = 1; n <= 5; n++) {
           console.log(`記住了第 ${n} 個單字`)
        }
}
```

十、數組

數組是什麼？

1. 數組 (Array)：按順序保存數據的 數據類型
2. 將多個數據用 數組 保存起來，然後放到一個變量中，方便管理

數組的基本使用

1. 語法：
    a. 字面量 聲明數組
        - let 數組名 = [數據1, 數據2, … 數據n]
    b. 使用 new Array 構造函數 聲明
        - let arr = new Array (數據1, 數據2, … 數據n)
2. 取值 語法：
    - 數組名[下標]

3. 數組術語：
    a. 元素：數組中保存的每個數據都叫做 數組元素
    b. 下標：數組中數據的 編號
    c. 長度：數組中數據的 個數，通過數組的 length 屬性獲得
        - 數組長度 = 索引號 + 1
4. 遍歷數組：
    a. 用循環把數組中每個元素都訪問到，一般用 for 循環遍歷
    b. for (let i = 0; i < 數組名.length; i++) {
        console.lo(數組名[i])
    }

操作數組

1. 查：查詢數組數據
    - 數組名[下標]
2. 改：重新賦值
    - 數組名[下標] = 新值
3. 增：添加新的數據
    a. 數組.push(新增的內容)：將一個或多個元素添加到數組的 末尾，並返回數組的新長度
        - 語法：
            - arr.push(元素1, 元素2, … 元素n)
            - let arr = ["春", "夏", "秋"]
              arr.push("冬")
              console.log(arr) // ["春", "夏", "秋","冬"]
              console.log(arr.push("冬")) // 4

b. 數組.unshift(新增的內容)：將一個或多個元素添加到數組的 開頭，並返回數組的新長度

4. 刪：刪除數組中的數據（元素）

a. 數組.pop( )：從數組中刪除最後一個元素，並返回該元素的值

- 語法：

```
let arr = [ "春", "夏", "秋" ]
arr.pop( )
console.log( arr )      //[ "春", "夏" ]
console.log( arr.pop( ) )      //  "秋"
```

b. 數組.shift( )：從數組中刪除第一個元素，並返回該元素的值

c. 數組.splice(start , deletCount)：從數組中刪除指定元素

i. start：指定開始的位置(從0計數)

ii. deleteCount：表示要移除的元素的個數，若省略則刪除到最後

- 語法：

```
let arr = [ "春", "夏", "秋", "冬" ]
arr.splice( 1, 2 )     //  從索引號1的位置開始刪，刪掉2個
console.log( arr )     //  [ "春", "冬" ]
```

數組案例

p46 47 48

十一、函數

為什麼要學函數？

1. 實現代碼重複使用、封裝
2. 函數 function ：被設計為執行特定任務的代碼塊

函數使用

1. 聲明語法：
   - function 函數名() {
     函數體
     }
2. 函數命名規範
   a. 和變量名基本一致
   b. 小駝峰命名法
   c. 前綴應該為動詞
   d. 命名建議：常用動詞約定，can、has、is、get、set、load、add
3. 調用語法：
   - 函數名()

函數傳參

1. 聲明語法（形參）：
   - function 函數名(參數列表) {
     函數體
     }
   - 形參：聲明函數時，小括號裡的 形式上的參數，本質上就是在函數內部 聲明變量
   - 作用：接受傳遞過來的 實參，如果沒有實參傳遞過來，默認是 undefined
2. 調用語法（實參）：
   - 函數名(參數列表)
   - 實參：調用函數時，小括號裡 實際上的參數，可以是變量
3. 參數中間用 逗號 隔開
4. 參數默認值
   a. 形參可看作 變量，但是如果一個變量不給值，默認值為何？ Ans：undefined
   b. 用戶不輸入實參，可以給 形參默認值，可以默認為 0，這樣程序更嚴謹

```
function getSum(x = 0, y = 0) {
    console.log(x + y)
}
getSum()    //  0
getSum(1, 3)    //  4
```

   c. 默認值只會在 缺少實參 參數傳遞時執行，其餘會優先執行傳遞過來的實參

函數返回值

1. 函數 是被設計為 執行特定任務 的代碼塊
2. 提問：執行特定任務後，然後呢？

   Ans：把任務結果給我們

   - 缺點：把計算後的結果處理方式寫死了，內部處理了
   - 解決：把處理結果返回給調用者
3. 提問：為什麼要讓函數有返回值？

   Ans：函數執行後得到結果，結果是調用者想要拿到的

   a. 函數內部不需要輸出結果，而是返回結果

   b. 對執行結果的拓展性更高，可以讓其他的程序使用這個結果
4. 例如：

   a. `let num = prompt("請輸入一個數字")`

   b. `result = parseInt("100")`
5. 有些函數，沒有返回值：`alert("我是彈跳匡，不需要返回值")`
6. 當函數需要返回值時，用 `return` 關鍵字
7. 語法：

   - `return` 數據

   a.
```
function fn() {
      return 20  //  相當於執行 fn( ) = 20
}
let result = fn()
      console.log(result)   //   20
```

   b.
```
function prompt(str) {
      return str
}
let result = prompt("請輸入：")
      console.log(result)
```
8. 結論：

   - `return` 的值給了 函数名（ =調用者 ），再把函数名裡面的值給變量，最後再輸出變量
9. 細節：

   a. 在函數體中使用 `return` 關鍵字能將內部的執行結果交給函數外部使用

   b. `return` 後面代碼不會再被執行，會立即結束當前函數，所以 `return` 後面的數據不要換行寫

   c. `return` 後面不寫數據 或是 函數內不寫 `return`，函數的默認返回值為 `undefined`

   d. 兩個相同的函數，後面的會覆蓋前面的函數

   e. Javascript 中，實參和形參的個數可以不一定

      i. 如果形參過多，自動填上 `undefined`

      ii. 如果實參過多，過多的實參會被忽略

   f. 函數一旦碰到 `return` 就不會再往下執行，函數的結束用 `return`
10. 提問：如何返回多個數據？ Ans：使用數組

    - 求數組中最大值和最小值：

```
<script>
    function getArrMax(arr = []) {
        let max = arr[0]
        let min = arr[0]
        for (let i = 1; i < arr.length; i++) {
            if (max < arr[i]) {
                max = arr[i]
            }
        }
        for (let i = 1; i < arr.length; i++) {
            if (min > arr[i]) {
                min = arr[i]
            }
        }
        return [max, min]
    }
    let newArr = getArrMax([10, 730, 300, 60, 40])
    console.log(`數組的最大值：${newArr[0]}`)
    console.log(`數組的最小值：${newArr[1]}`)
</script>
```

作用域

1. 全局作用域：作用於函數外部，所有代碼執行的環境（整個<script>標籤內部）或獨立的.js文件
    - 全局變量：全局變量在任何區域都可以訪問和修改
2. 局部作用域：作用於函數內部，因為跟函數有關係，也稱為函數作用域
    - 局部變量：只能在當前函數內部訪問和修改
3. 變量有一個，特殊情況：
    - 如果函數內部，變量沒有 `let` 聲明，直接賦值，也當全局變量使用，但強烈不推薦使用
    - 函數內部的形參可以當作是 局部變量 來看
4. 變量的訪問原則：
    a. 只要是代碼，至少有一個作用域
    b. 如果函數中又有函數，在這個作用域中又可以誕生一個作用域
    c. 就近原則：在能夠訪問到的情況下，先局部，局部沒有再往上查找變量最終的值

匿名函數

1. 匿名函數：`function () { }`
    a. 函數表達式：將匿名函數賦值給一個變量，並通過變量名進行調用
        - 語法：
            - ```
              let fn = function () {
                  consloe.log("函數表達式")
              }
              ```
              ```

2. 具名函數：function fn() { }，調用 fn()
3. 匿名函數 和 具名函數 的不同
    a. 具名函數的調用可以寫在任何位置
    b. 函數表達式，必須先聲明函數表達式，後調用（順序）
4. 匿名函數 之 立即執行：
    a. 無需調用，立即執行，其實本質上已經調用了
    b. 避免全局變量之間的污染
    c. 多個立即執行函數要用『 ; 』隔開，否則報錯

```
<script>
    // 寫法 1
    (function (x, y) {
        console.log(x + y)
    })(10, 5);

    // 寫法 2
    (function (x, y) {
        console.log(x + y)
    }(20, 30));
</script>
```

案例

- 用戶輸入秒數，自動轉換時、分、秒
    1. 用戶輸入總秒數（注意默認值）
    2. 計算時、分、秒（封裝函數）裡面包含數字補 0
    3. 打印輸出
- 公式：
    a. 小時：h = parseInt(總秒數 / 60 / 60)
    b. 分鐘：m = parseInt(總秒數 / 60 % 60)
    c. 秒數：s = parseInt(總秒數 % 60)

```
<script>
    // 1. 用戶輸入
    let second = +prompt("輸入秒數：")
    // 2. 封裝函數
    function getTime(t) {
        // 3. 轉換
        let h = parseInt(t / 60 / 60)
        let m = parseInt(t / 60 % 60)
        let s = parseInt(t % 60)
        h = h < 10 ? "0" + h : h
        m = m < 10 ? "0" + m : m
        s = s < 10 ? "0" + s : s
        return `${h}時 ${m}分 ${s}秒`
```

```
    }
    let time = getTime(second)
    console.log(time)
</script>
```

邏輯中斷

1. 類似參數的默認值寫法
2. 邏輯運算符裡的短路
   - 短路：只存在於 && 和 || 中，當滿足一定條件會讓右邊代碼不執行

符號	短路條件
&&	左邊為 false 就短路
||	左邊為 ture 就短路

3. 原因：通過左邊能得到整個式子的結果，因此沒必要再判斷右邊
4. 運算結果：無論 && 還是 ||，運算結果都是最後被執行的表達式值，一般用在變量賦值

轉換為 Boolean 型

1. 顯示轉換：
   - Boolean(內容)
     - ” ”、0、false、undefined、null、NaN 轉換為布林型後都是 false，其餘為 true
2. 隱式轉換：
   a. 有字串符的加法 “ ” + 1，結果是 “1”
   b. 減法『 - 』(像大多數數學運算一樣)只用於數字，它會使空字串 ” ” 轉換為 0
   c. null 經過數字轉換之後會變為 0
   d. undefined 經過數字轉換之後會變為 NaN

十二、對象

對象是什麼？

1. 對象(object)：Javascript 中的一種數據類型
2. 對象可以理解為是一種 無序的 數據集合，注意：數組是 有序的 數據集合

對象的基本使用

1. 聲明語法
   - let 對象名 = {}
   - let 對象名 = new Object{ }
2. 對象 由 屬性 和 方法 組成
   a. 屬性：信息或特徵（名詞）
   b. 方法：功能或行為（動詞）

```
let 對象名 = {
    屬性名:屬性值,
    方法名:函數
}
```

3. 屬性：
   a. 數據描述性的信息稱為屬性

```
let obj = {
    person: "李晨瑋",
    age: 27,
    gender: "男"
}
```

   b. 屬性都是成對出現，包括 屬性名 和 值，用『 : 』隔開
   c. 多個 屬性 之間，使用『 , 』隔開
   d. 屬性 就是依附在對象上的 變量（外面是變量，對象內是屬性）
   e. 屬性名 可以使用 " " 或 ' '，一般情況下省略，除非名稱遇到特殊符號，如：空格、中橫線等
4. 方法：
   a. 數據行為性的信息稱作方法，其本質是函數

```
let obj = {
    person: "李晨瑋",
    age: 27,
    sayHi: function () {
        document.write("hello world")
    }
}
```

   b. 方法是由 方法名 和 函數 組成，用『 : 』隔開
   c. 多個 屬性 之間，使用『 , 』隔開

d. 方法 就是依附在對象中的 函數

e. 方法名 可以使用 " " 或 ` `，一般情況下省略，除非名稱遇到特殊符號，如：空格、中橫線等

對象的操作

```
let obj = {
    person: "李晨瑋",
    age: 27,
    sayHi: function () {
        document.write("hello world")
    },
    num: function (x, y) {
        document.write(x + y)
    }
}
```

1. 屬性：

   a. 查：聲明對象，並添加使用『 . 』獲得對象中屬性的對應的值，稱之屬性訪問

      i. 對象名.屬性名

         - console.log(obj.person) // "李晨瑋"

      ii. 對象名 [`屬性名`]

         - console.log(obj[`age`]) // 27

   b. 改：重新賦值

      - 對象名.屬性名 = 新值

   c. 增：添加新的數據

      - 對象名.屬性名 = 新值

   d. 刪：刪除數組中的數據（元素）

      - delete 對象名.屬性名

2. 方法：

   a. 調用：聲明對象，並添加方法後，可以使用『 . 』調用對象中的方法（函數），稱之方法調用

      - 對象名.方法名 ( )

         - obj.sayHi() 注意：記得要加小括號

   b. 可添加 形參 和 實參

      - obj.num(5, 10)

   c. 增加：添加新的方法

      - 對象名.方法名 = 函數

```
// 聲明一個空的對象（沒有任何屬性，也沒有任何方法）
let user = {} 或 user=null
// 動態添加屬性
user.name = "李晨瑋"
user[`age`] = 27
// 動態添加方法
```

```
        user.sayHi = function () {
            console.log("hello world")
        }
```

3. null 也是 Javascript 中數據類型的一種，通常只用來表示不存在的對象，使用typeof檢測，結果為object

遍歷對象

1. for 遍歷對象的問題
    a. 對象沒有數組一樣的 length 屬性，所以無法確定長度
    b. 對象裡是無序的鍵值對，沒有規律，不像數組裡有規律的下標
2. for in 不適合 遍歷數組
    -
```
let arr = ["李晨瑋", "Wayne", "華江高中"]
    for (let k in arr) {
        console.log(k)     // 打印 數組的下標（索引號），但是是字串符型 "0"
        console.log(arr[k])
    }
```
3. for in 遍歷對象
    a. 一般不用這方式來遍歷數組，主要是用來遍歷對象
    b. for in 語法中的 k 是一個變量，在循環的過程中依次代表對象的屬性名
    c. 由於 k 是變量，所以必須使用 [ ] 語法解析
    d. 記住：k 是獲得對象的 屬性名、對象名 [k] 是獲得 屬性值
    -
```
let obj = {
            person: "李晨瑋",
            age: 27,
            gender: "男"
        }
    for (let k in obj) {
        console.log(k)  // 打印 屬性名(字串符)，帶引號 `person` `age` `gender`
        console.log(obj[k])  // 打印 屬性值 obj[`person`]、obj[k]
    }
```

案例

1. 遍歷數組對象
    -
```
let students = [
        { name: "李晨瑋", age: 27, city: "新北" },
        { name: "黃小花", age: 30, city: "高雄" },
        { name: "吳小可", age: 20, city: "台中" }
    ]
```
    -
```
for (let i = 0; i < students.length; i++) {
        console.log(i)     // 打印 下標索引號
```

```
            console.log(students[i])     // 打印 所有對象
            console.log(students[i].name)     // 打印 所有對象 的 名字
        }
```

2．根據數據，渲染生成表格

```
<style>
    table tr {
        text-align: center;
    }

    table th {
        padding: 10px;
    }

    table td {
        padding: 10px;
    }
</style>

<body>
    <table>
        <caption>學生列表</caption>
        <tr>
            <th>序號</th>
            <th>姓名</th>
            <th>年齡</th>
            <th>戶籍</th>
        </tr>
        <script>
            // 數據準備
            let students = [
                { name: "李晨瑋", age: 27, city: "新北" },
                { name: "黃小花", age: 30, city: "高雄" },
                { name: "吳小可", age: 20, city: "台中" }
            ]
            // 渲染頁面
            for (let i = 0; i < students.length; i++) {
                document.write(`
                <tr>
                <td>${i + 1}</td>
                <td>${students[i].name}</td>
                <td>${students[i].age}</td>
                <td>${students[i].age}</td>
```

```
                </tr>
                `)
            }
        </script>
</body>
```

內置對象

內置對象是什麼？

1. Javascript 內部提供的對象，包含各種 屬性 和 方法 給開發者調用
2. console 其實就是 Javascript 中內置的對象，該對象中存在一個方法叫 log，然後調用 log 這個方法
    a. document.write( )
    b. console.log( )

內置對象 Math

1. Math 對象是 Javascript 提供的一個 “數學” 對象
2. 作用：提供了一系列做數學運算的做法，包含了屬性和方法
    a. 屬性：
        - Math.PI 圓周率
            - console.log(Math.PI) // 3.14
    b. 方法：
        i. Math.random 生成 0~1 之間的隨機數（包括0不包括1）
            - console.log(Math.random())
        ii. Math.ceil 數字向上取整數
            - console.log(Math.ceil(1.1)) // 2
              console.log(Math.ceil(1.7)) // 2
              console.log(Math.ceil(1.9)) // 2
        iii. floor 數字向下取整數
            - console.log(Math.floor(1.1)) // 1
              console.log(Math.floor(1.5)) // 1
              console.log(Math.floor(1.9)) // 1
        iv. round 四捨五入取整數
            - console.log(Math.round(1.1)) // 1
              console.log(Math.round(1.5)) // 2
              console.log(Math.round(-1.1)) // -1
              console.log(Math.round(-1.5)) // -1
              console.log(Math.round(-1.51)) // -2
        v. max 在一數組中找最大數
            - console.log(Math.max(1, 2, 3, 4, 5)) // 5
        vi. min 在一數組中找最小數

- console.log(Math.min(1, 2, 3, 4, 5)) // 1

vii. abs 絕對值

- console.log(Math.abs(-10)) // 10

viii. pow 冪方法

- console.log(Math.pow(2, 4)) // 16
  console.log(Math.pow(3, 3)) // 27

ix. sqrt 平方根

- console.log(Math.sqrt(64)) // 8

生成任意範圍隨機數

1. Math.random()
   - 隨機數函數，返回一個 0~1 之間的隨機數，並且包括 0，不包括1的隨機小數
2. 如何生成 0~10 的整數呢？
   a. Math.floor(Math.random() * (10 + 1))
   b. 隨機抽取 數組中的元素
      - let arr = ["拉麵", "燒肉", "泡菜"]
        let random = Math.floor(Math.random() * arr.length)
        console.log(random) // 隨機打印 0~2 整數
        console.log(arr[random]) // 隨機打印 "拉麵" "燒肉" "泡菜"
3. 如何生成 5~10 的隨機數呢？
   - Math.floor(Math.random() * (5 + 1)) + 5
4. 如何生成 N~M 的隨機數呢？
   - Math.floor(Math.random()*(M - N + 1)) + N

案例

1. 隨機點名案例
   - let arr = ["劉備", "關羽", "張飛", "呂布", "曹操", "趙雲", "馬超"]
     // 得到一個隨機數，作為數組的索引號，0~6
     random = Math.floor(Math.random() * arr.length)
     console.log(random)
     // 打印 數組裡面的元素
     console.log(arr[random])
2. 隨機顯示一個名字到頁面中，但是不允許重複顯示
   - let arr = ["劉備", "關羽", "張飛", "呂布", "曹操", "趙雲", "馬超"]
     let random = Math.floor(Math.random() * arr.length)
     console.log(arr[random])
     arr.splice(random, 1)
     console.log(arr)
3. 猜數字
   - // 隨機生成一個數字 1~10
     function getRandom(N, M) {

```
            return Math.floor(Math.random(M - N + 1)) + N
        }
        random = getRandom(1, 10)

        // 循環猜數字
        while (true) {
            // 用戶輸入數字
            let num = +prompt("請輸入數字：")
            if (num > random) {
                console.log("再猜小一點");
            }
            else if (num < random) {
                console.log("再猜大一點")
            }
            else {
                console.log("答對")
            break
            }
        }
```

4. 生成隨機顏色

```
<script>
    // 定義一個隨機顏色函數
    function getRandomColor(flag = true) {
        if (flag) {
            // 如果是true，返回 #ffffff
            let str = "#"
            let arr = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"a",
                       "b", "c", "d", "e", "f"]
            // 利用for循環，隨機抽6次，累加到str裡面
            for (let i = 1; i <= 6; i++) {
                // random 是數組的索引號，是隨機的
                let random = Math.floor(Math.random() * arr.length)
                str += arr[random]
            }
            return str
        }
        else {
            // 否則是 false，返回 rgb(255,255,255)
            let r = Math.floor(Math.random() * 256)
            let g = Math.floor(Math.random() * 256)
            let b = Math.floor(Math.random() * 256)
```

```
            return `rgb(${r},${g},${b})`
        }
    }
    // 2.調用函數
    console.log(getRandomColor(true))
    console.log(getRandomColor(false))
</script>
```