Variable (const vs let)

let 以及const則是為了改善var可能造成混淆的問題而產生的新一代variable種類。

let基本上使用方式與var相同，可進行declaration以及re-assign，惟不允許相同變數名稱被重複declaration，故可減少混淆風險。

```
let x;         // declaration without assignment, it's ok
let y = 5;
y = 6;         // re-assign, it's ok
let x = 6;     // re-declaration, it's invalid
```

const則用以表示常數，其於初次declaration就必須對其assign，不可僅進行decalration。且不可re-assign。因此若是不會再需要改動的值，開發者應以const為首要考量，避免後續開發造成混淆。

```
const x;      // declaration without assignment, it's valid
const y = 5;
y = 6;        // re-assign, it's valid
const x = 6; // redeclaration, it's valid
```

值得一提的是const 只是給予immutable reference to a value.不代表 the value it holds is immutable，舉例來說假如const設定為一個object：

```
const student = {name: 'Tom'};
student.age = 18;
//set the new property inside student object, it's ok
student.name = {'Ben'};
//change the property value inside student object, it's ok
student = {name: 'Alex'};
//re-assign, it's valid
```

reference:

https://javascript.plainenglish.io/understand-variable-declaration-initialisation-and-assignment-in-javascript-4f98e54ecda5

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

Assignment 1 - revised

將變數max預設為numbers[0], 後續比大小for loop從index = 1開始。

```
function max(numbers) {
  // your code here, for-loop method preferred
  let max = numbers[0];

  for (let i = 1; i < numbers.length; i++) {
    if (numbers[i] > max) {
      max = numbers[i];
     }
  }

  return max;
}
```

Assignment 4 - revised

(app.js)

```
//Request 1
const welcomeText = document.querySelector("#welcome-message");
const welcomeArea =
document.querySelector("#welcome-message-area");

welcomeArea.addEventListener("click", () => {
  welcomeText.textContent = "Have a GoodTime!";
});

//Request 2
const btnShow = document.querySelector(".show-btn");
const hiddenArea = document.querySelector("#hidden");

btnShow.addEventListener("click", () => {
  hiddenArea.removeAttribute("id");
});
```

Assignment 6 - revised

此處應先下載jQuery再下載app.js。故此處應使用defer，因為defer會按照條列順序依序執行script。若使用async則無視script條列順序，先下載好的先執行，可能會造成jQuery尚未下載完成，而app.js先下載好先執行卻無法使用jQuery資料庫的bug。

reference: https://gcdeng.com/blog/script-tag-async-defer-attributes

```
<head>
    ...
    <script
      defer
      src="https://code.jquery.com/jquery-3.7.1.min.js"
integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
      crossorigin="anonymous"
    ></script>
    <script defer src="./app.js"></script>
  </head>
```

Big(O) notation

Big(O) natation用來表示時間複雜度的一種analysis tool。

簡單的定義為當$f(n)$成長得比$g(n)$慢時，

即可稱為：

$f(n) = O(g(n))$

有關於Big(O) natation 正式的定義為：

假設f(n)與g(n)皆為正函數，

只要存在常數c 以及某數 n_0，能滿足：

當$n > n_0$時，$f(n) \leq c \cdot g(n)$

就可得出結論：

$f(n) = O(g(n))$

這代表即使$f(n)$沒有limit（不收斂）也沒關係，只要有boundary即可。

適用於$|sin(n)| = O(1)$的證明。

reference: (我在林軒田老師youtube公開的DSA課程學的)

https://www.csie.ntu.edu.tw/~htlin/course/dsa20spring/

https://www.youtube.com/watch?v=d-PDYIqg_Kc&list=PLXVfgk9fNX2Kda9rttSvGROCtRQ3Sb8bA&index=6

findPosition 和 binarySearchPosition 的 big O 是什麼呢？

findPosition:

```
function findPosition(numbers, target) {
  // your code here, for-loop method preferred
  let index = -1;

  //reverse the search order so that the later index number will be smaller
  for (let i = numbers.length - 1; i >= 0; i--) {
    if (target === numbers[i]) {
      index = i;
    }
  }
  return index;
}
```

best case（target就出現在index = 0的位置）: $time\Theta(1)$ //Θ為big theta

worst case（target就出現在index 最後的位置或是不存在）: $time\Theta(n)$

Big(O) natation 應以**upper bound**也就是**worst case**的結果為準,

故**findPosition** 為**O(n)-algorithm(linear complexity)**

binarySearchPosition

```
function binarySearchPosition(numbers, target) {
  // your code here
  let begin = 0;
  let end = numbers.length - 1;

  while (begin <= end) {
    let mid = Math.round((begin + end) / 2);
    if (numbers[mid] > target) {
      end = mid - 1;
    } else if (numbers[mid] < target) {
      begin = mid + 1;
    } else {
```

```
      return mid;
    }
  }
  return -1;
}
```

best case（target第一次就出現在index = mid的位置）: $time\Theta(1)$　　//Θ為big theta

worst case（target不存在）: $time\Theta(log(n))$

Big(O) natation 應以**upper bound**也就是**worst case**的結果為準,

故**binarySearchPosition** 為**O(log(n))-algorithm(logarithmic complexity)**