主　題：程式寫作編排與風格概述

講　師：紀宏宜 老師

聯成電腦

課　　程　　大　　綱

課程內容	時間	主講人
貴賓致詞	5分鐘	嘉義大學電算中心洪燕竹主任
1. 程式碼版面與樣式編排 2. 統一程式碼命名方式 3. 編寫「自文件化」程式碼 4. 編寫程式碼註解	90分鐘	聯成電腦 紀宏宜老師
數位學苑免費行動學習方案	5分鐘	聯成電腦 卓慧怡副理
Q & A 交流互動	5分鐘	學校 與 聯成電腦

紀宏宜

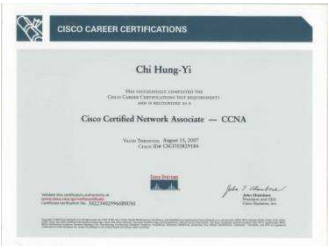
Red Hat Linux 原廠授權講師與執行考官

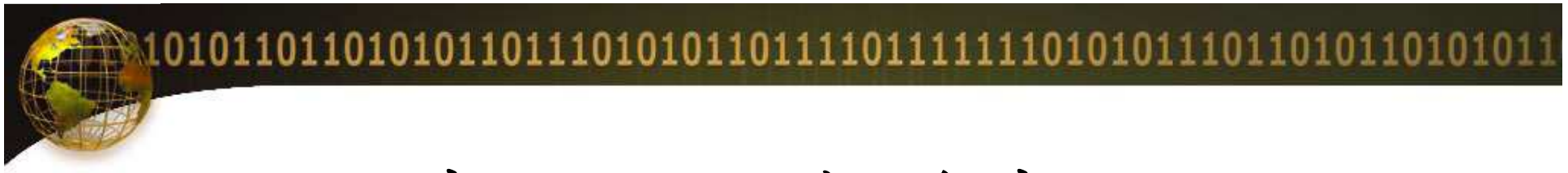
聯成電腦 Linux 訓練講師

實踐大學兼任講師、樹德科大兼任講師

國立中山大學電機系博士班

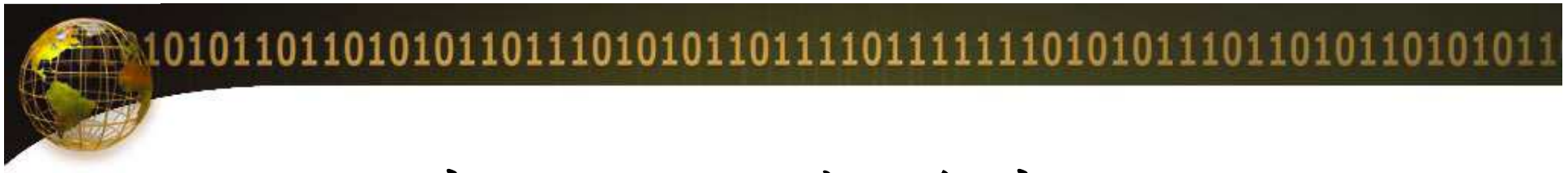
樹德科技大學電腦與通訊所





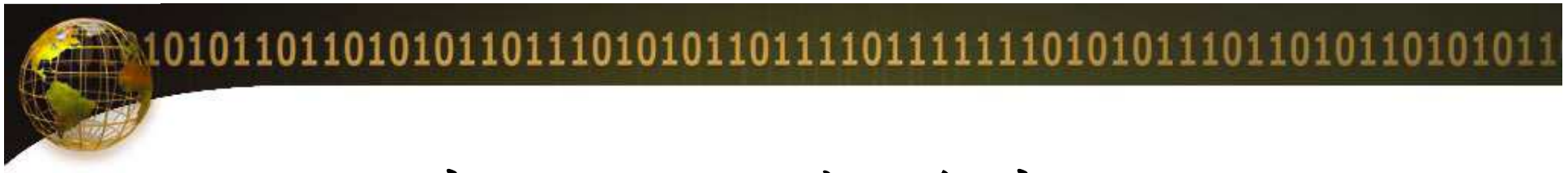
程式碼版面與樣式編排

- 程式碼到底是寫給誰看？
  - Compiler
    - 只會抱怨程式寫錯了！
  - 你自己
    - 問問你自己的良心：之前的程式是否得看得懂？
  - 你的同事
    - 放心！他們都自身難保了，不會看你的程式碼！
  - 你的老闆
    - 只要寫出會動的程式就行了！



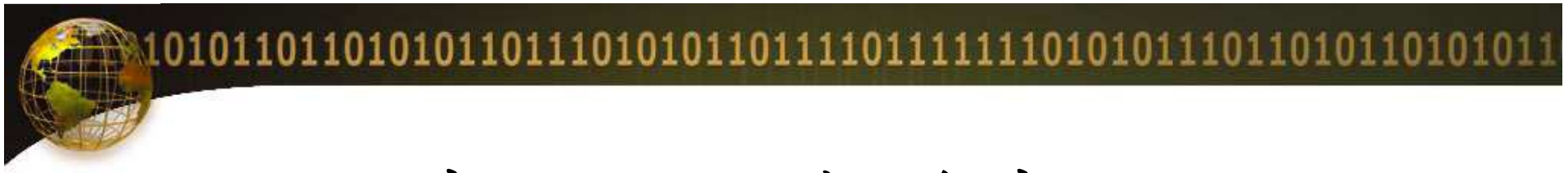
程式碼版面與樣式編排

- 版面與樣式編排有很重要嗎？
  - 樣式會「干擾」程式的可讀性！
  - 沒有人喜歡讀「看不懂」的程式碼！
  - 樣式是一種非常主觀且個性化的東西！
  - 使用熟悉的樣式會使人很舒服，但陌生的樣式則會使人感到焦慮！



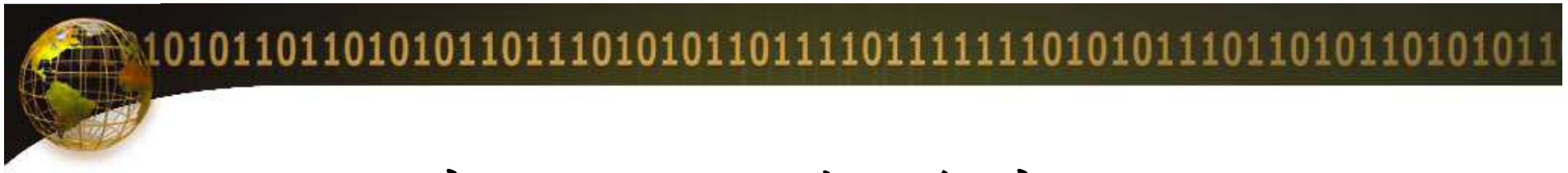
程式碼版面與樣式編排

- 什麼才是好的樣式呢？
  - 整潔的程式碼！
    - 整潔的程式碼不是指完全沒錯誤的程式碼！
  - 清晰的程式碼！
    - 強調內縮對齊！
    - 在內縮的同時，也強化了程式碼的結構與可讀性！
    - 除非必要，否則儘量不要寫太複雜的程式碼！(因為不好排版！)
    - 簡單的程式碼較為方便閱讀！
  - 程式碼版面必須是表達涵義，而不是隱藏涵義！



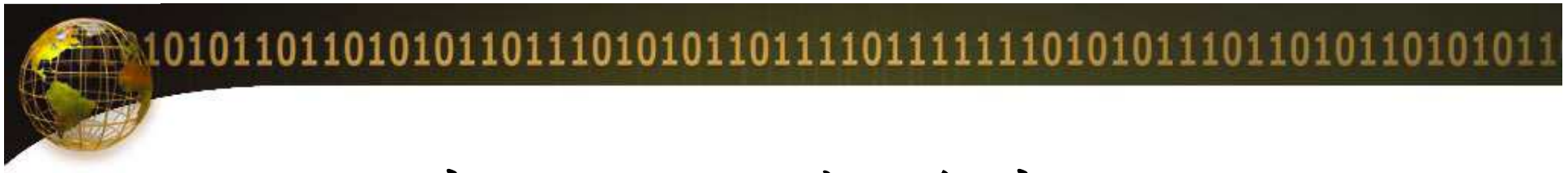
程式碼版面與樣式編排

- 衡量樣式及風格品質的標準
  - 一致性
    - 內縮的策略必須在整個專案中保持一致！
      - 如果隨意改變，看起來不專業。
      - 中途改變對齊的方式，容易誤導別人，使人以為原始檔案互不相關。
    - 如果是自訂的個性化樣式規則，應該要保持內部統一。
      - 括弧的位置也應該要遵循同一規則。
      - 內縮的空格數，也應前後一致！



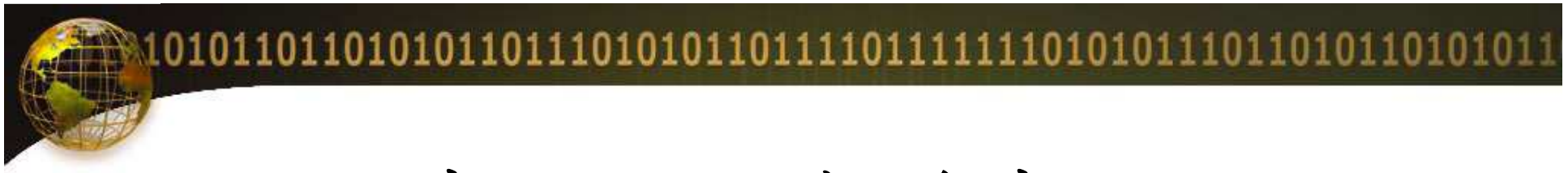
程式碼版面與樣式編排

- 傳統
  - 採用同業現行流行的樣式！
    - 確保閱讀程式碼的人，能夠快速理解程式碼！
    - 在除錯的效率上，也快上許多！
- 簡潔
  - 可以簡單的描述內縮策略！
  - 有用的風格，在擴編程式碼時，可以快速被學會，及遵守相關規則。



程式碼版面與樣式編排

- 括弧的使用方式
  - 傳統的括弧位置風格有許多種！
    - 選擇的方式
      - 取決於個人的審美觀
      - 周遭的人習慣於哪一種程式計設文化
      - 個人習慣
      - 其他
    - 在不同的環境中，應選擇不同的風格！



程式碼版面與樣式編排

- K & R 括弧風格
  - 由 C 語言之父 Kernighan & Ritchie 所確立！

```
int k_and_r() {
    int a = 0, b = 0;
    while (a != 10) {
        b++;
        a++;
    }
    return b;
}
```

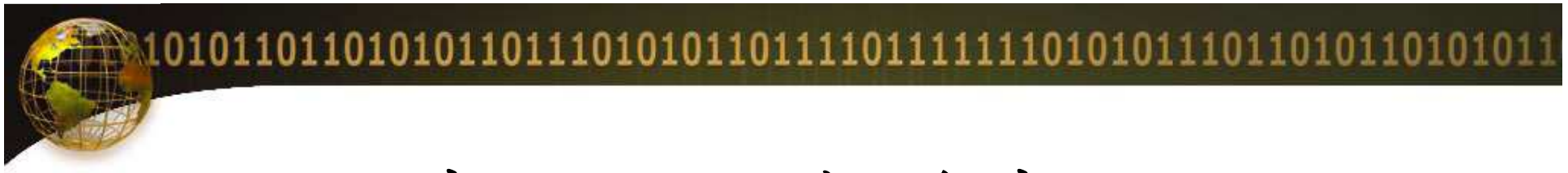
運算子前後有空一格

外圍程式與內部程式，以一個 Tab 鍵內縮，並且對齊。

後括弧與程式碼開頭對齊。

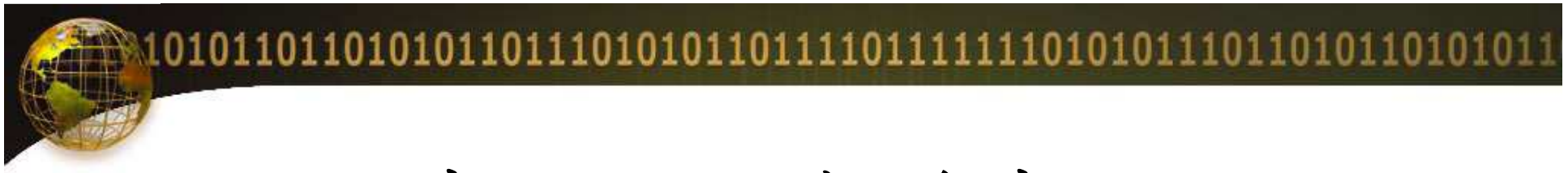
運算元靠近小括弧，甚至不空格！

前括弧在程式碼後面，並且與前方的程式碼空一格距離。



程式碼版面與樣式編排

- 優點：
  - 佔用空間小
  - 後括弧與相對應的敘述內縮在同一位置上，所以容易找到程式的終止點。
- 缺點
  - 前後括弧沒有對齊，視覺上不容易匹配。
  - 如果頁面上的右側少了括弧，很難找出來！
  - 程式碼一寫多，感覺像是一團麵球！



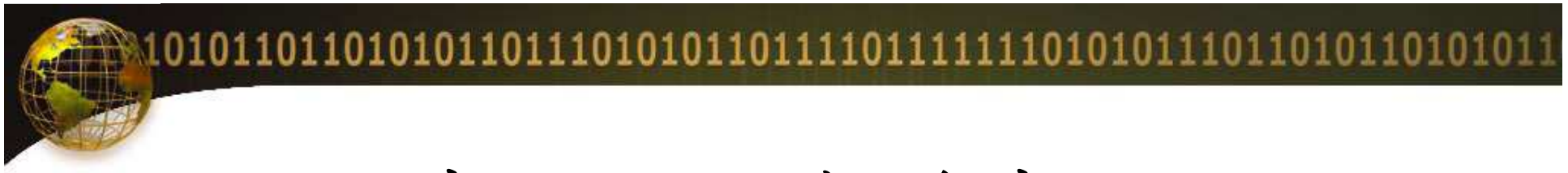
程式碼版面與樣式編排

- 懸掛式的括弧風格
  - 空間上比 K & R 更為開闊的做法

前括弧與程式開頭在不同行但同一列位置上。

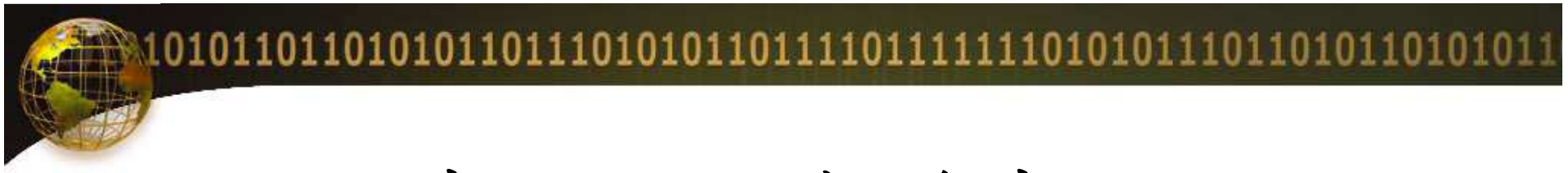
後括弧與前括弧對齊。

```
int exdented()
{
    int a = 0, b = 0;
    while (a != b)
    {
        b++;
        a++;
    }
    return b;
}
```



程式碼版面與樣式編排

- 懸掛式風格的優缺點
  - 優點：
    - 格式清晰、整潔！
    - 前後括弧放在很明顯的位置，使得程式碼區塊更容易區分。
  - 缺點：
    - 佔用太多的垂直空間
    - 如果有太多程式區塊內只包含一行程式碼，則太佔用空間。
    - 對於某些會使用 Pascal 語言的人而言，容易造成混淆。

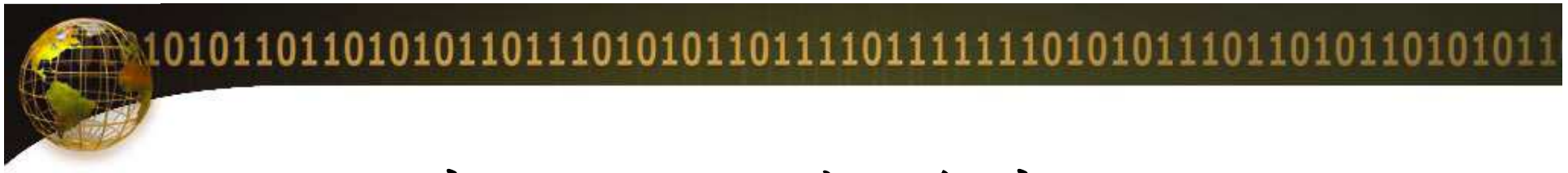


程式碼版面與樣式編排

- 內縮的括弧風格 (Whitesmith)
  - 內縮括弧風格不常見，但仍是有人在用！
  - 早期 Whitesmith 的 C 編譯器範例程式碼就是如此寫法！

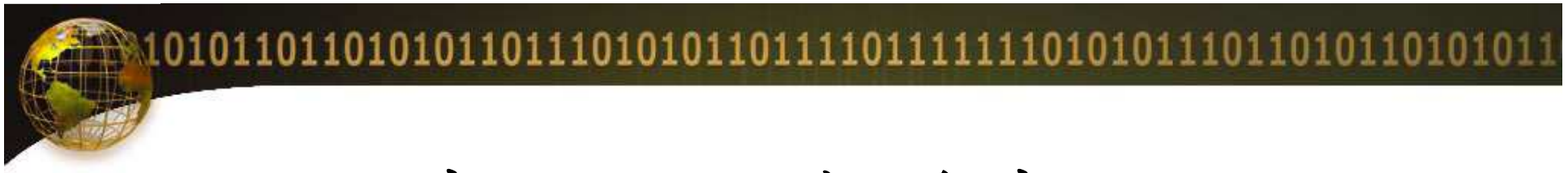
```
int exdented()
    {
    int a = 0, b = 0;
    while (a != b)
        {
        b++;
        a++;
        }
    return b;
    }
```

括弧內縮到與內部程式碼開頭在同一列位置上。



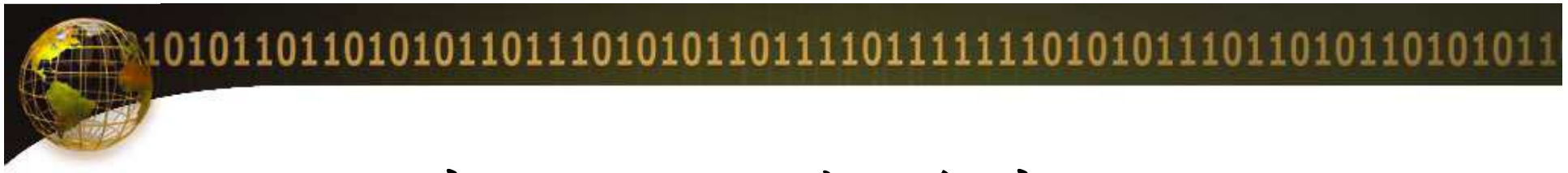
程式碼版面與樣式編排

- 內縮程式碼的優缺點
  - 優點：
    - 程式碼區塊以及包括程式碼區塊的括弧連接在一起。
  - 缺點：
    - 許多人不習慣這種使用的方式！



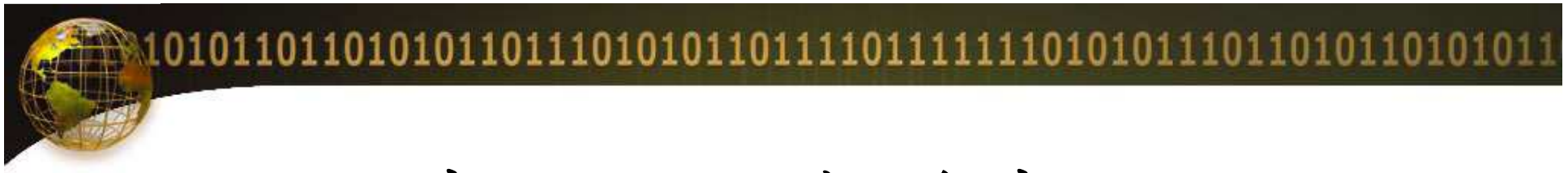
程式碼版面與樣式編排

- 其他的括弧風格
  - GNU 風格
    - 介於懸掛式和內縮式風格之間
  - Linux 核心風格
    - 介於 K & R 與懸掛式之間
  - 其他的自訂風格



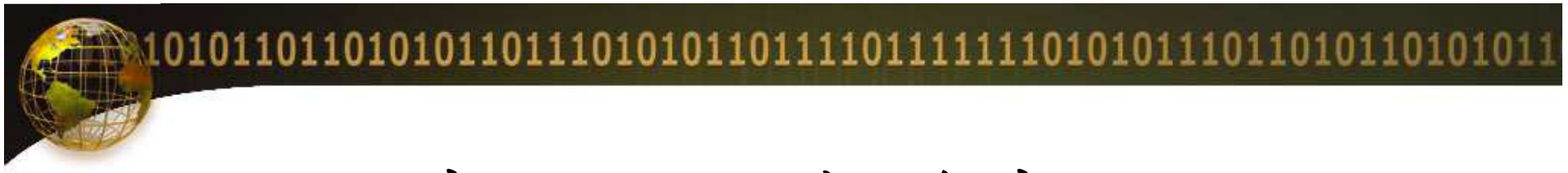
程式碼版面與樣式編排

- 良好的樣式是啥？
  - 依據閱讀程式碼的環境來決定何種樣式比較適合
    - 原始碼樣式
      - 指的是在編輯程式碼的開發環境中，閱讀程式！
      - 其重點在於樣式風格應該要協助程式設計師可以快速理解程式碼內容，並且正確地編寫程式。
    - 出版物上的程式碼樣式
      - 出版物上的程式碼，通常優先考慮的是讀物的空間。
        - » 所以在最小的空間內要置入最多行數的程式碼！
      - 文章內的程式碼，通常也只是概略的展示重點程式碼，與主題無關的內容，將被省略！



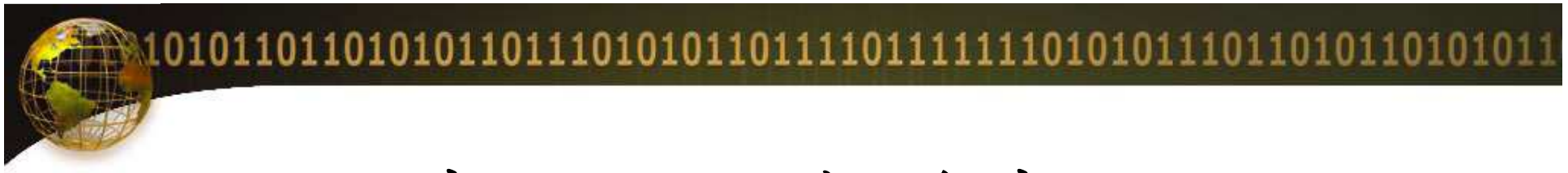
程式碼版面與樣式編排

- 列印稿上的程式碼
  - 程式碼在印列時，欄寬、行高、字體大小、顏色..等等，將會干擾閱讀程式碼的人！
  - 如果有彩色的列印，比較容易讓人分辨註解區內的程式碼與真正運作的程式碼！
  - 字體太小，會讓人閱讀時，產生誤判！
  - 欄寬太窄、太寬或是行高控制不當，亦使得程式碼閱讀起來十分困難！



程式碼版面與樣式編排

- 製訂內部風格的重要性
  - 好的風格易於讀寫，也易於維護！
  - 公司內部在訂好風格之後，可以讓所有程式碼撰寫方式一致，也容易協調與支援！
  - 具體的好處更有以下幾點：
    - 對外展現專業與嚴謹的態度
    - 公司對客戶可以保證寫出的程式符合同一個標準。可防止糟糕的程式碼產出！
    - 彌補工具的不足！因為，每一種工具都會有不同的設定方式與規則！提供一個水平線，讓所有工具做出來的風格相同！

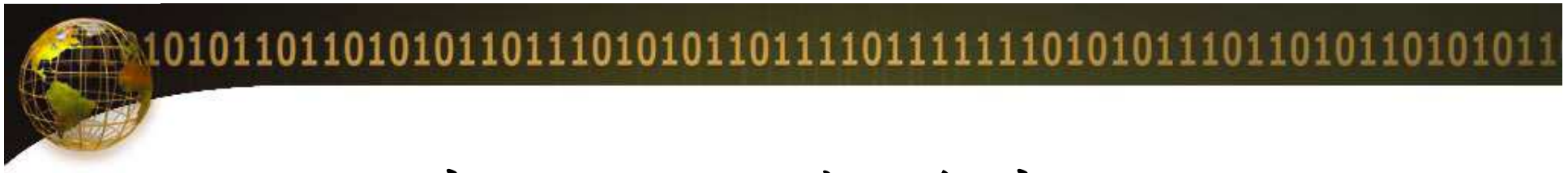


程式碼版面與樣式編排

- 程式碼容易在同事之間流通，並且同事間亦可互相支援、維護程式碼的正確性！(不但節省閱讀時間，同時也節省公司的資金)

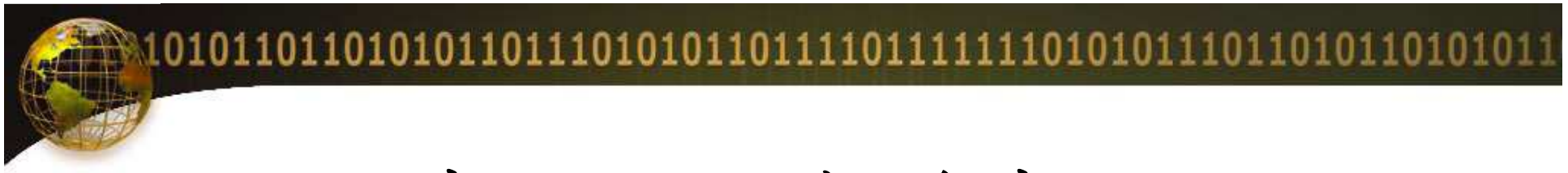
- 程式設計師在養成習實、開始使用相同風格的樣子之後，就不必常回頭修改程式碼編排樣式！如此控制版本的歷史記錄，就非常有用了！

– 如果接的案子多數來自公司外部，則選擇與多數業界相同主流的風格就非常重要了！



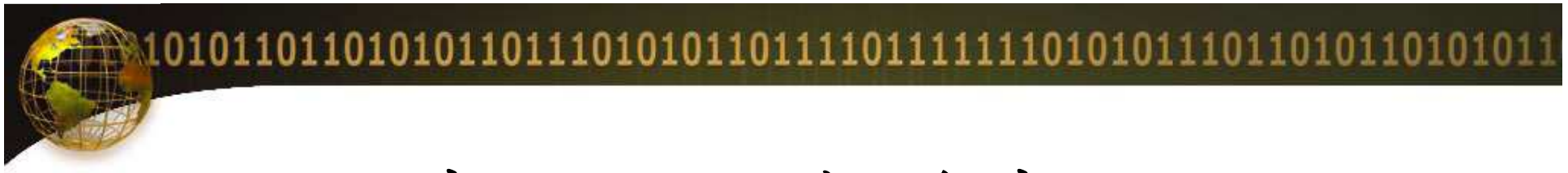
程式碼版面與樣式編排

- 設立標準
  - 建立一套程式設計標準是一項需要慎重考慮的任務，應該適當且堅定地進行！
  - 建議作法：
    - 標準為誰而立？
      - 良好的個人風格，不見得對團隊內的所有程式設計師都好！
      - 建立標準時，不應只適合個人的審美觀！
      - 標準最好是簡單、易懂
    - 凝聚共識
      - 讓需要使用該標準的人，一起參與標準的制訂！
      - 推舉一人為主席，裁決最後的版本！



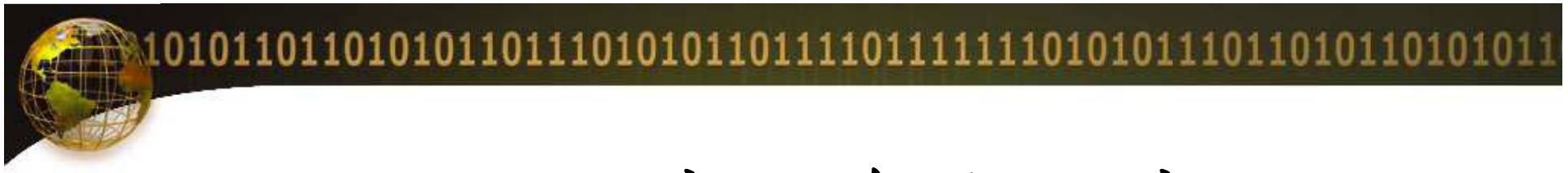
程式碼版面與樣式編排

- 做出成果→將標準寫成文件存放
  - 最終的成果，應該不止是一個模糊的檔案。
  - 應該是一個可理解的內容。
  - 可以拿出來訓練新手的檔案。
  - 將來可提供查閱與參考。
  - 對於爭議條款的解釋。
- 突出重點
  - 只為團隊所重視的程式語言訂出標準！
- 避免成為熱點
  - 把罕見且麻煩的情況交給個人決定
  - 不要過份死板，規則可以是被打破的！



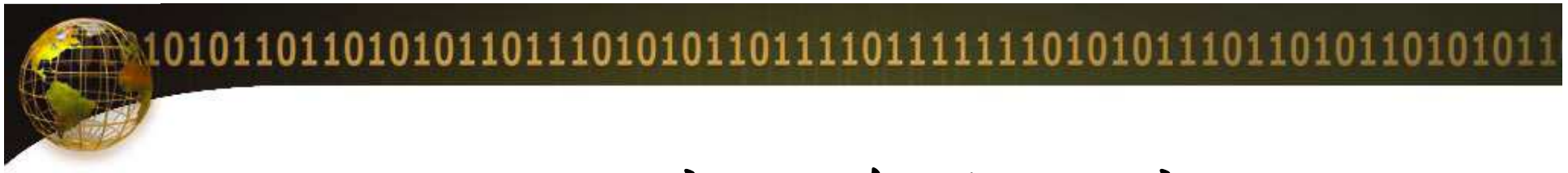
程式碼版面與樣式編排

- 逐步完善標準內容
  - 一次只開發一小部份內部風格
  - 慢慢經由撰寫大型的程式來累積標準
- 為推廣標準做準備
  - 為推廣標準做一些說明與相關的咨詢！
  - 盡量以表揚的方式來促進所有人遵守標準
  - 儘量取得管理階層的認同與授權！



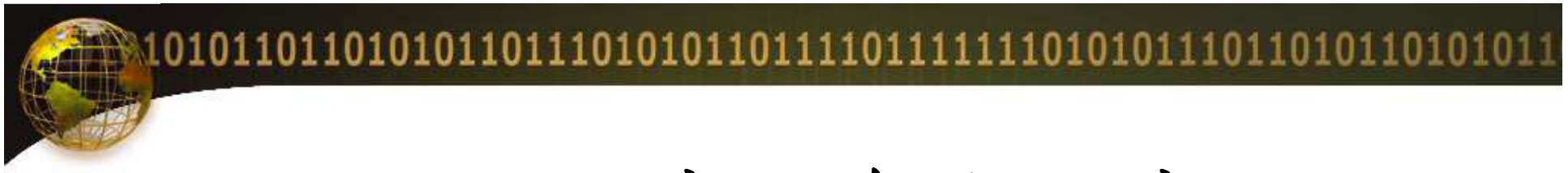
統一程式碼命名方式

- 名稱有何重要性？
  - 身份的描述
    - 名稱是身份概念的基礎！例：冠夫姓的婦女
  - 行為的描述
    - 名稱不僅說明了身份，而且也暗示著行為。
    - 名稱不見得直接描述物件的行為，但會影響你怎樣跟這個物件互動！
    - 例：你與你的女兒
  - 識別個體
    - 名稱將一個事物標明為一個性質獨特的實體。
    - 例：電



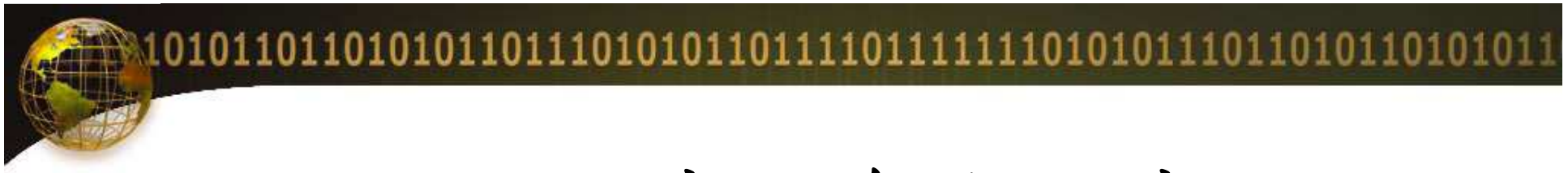
統一程式碼命名方式

- 適當的命名
  - 瞭解名稱就可以瞭解對象
    - 加速程式設計師之間的交流
  - 清晰的命名是優秀程式碼的特點之一
- 需要合適命名的對象
  - 變數、函數、型別、C++命名空間和 Java Package、巨集、原始檔案
  - 還有其他的東西，不過，一切仍回歸於上述幾項的用法！



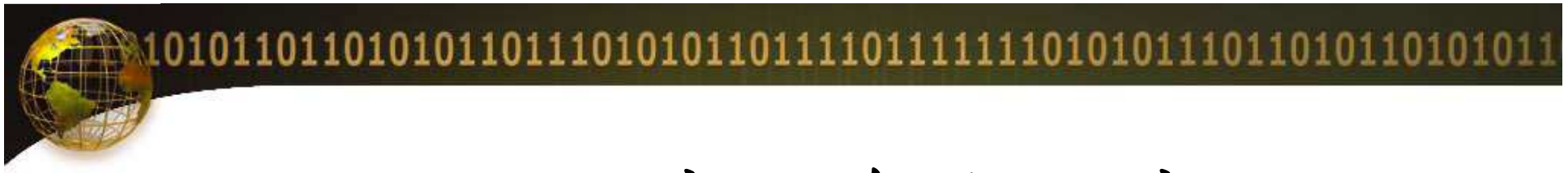
統一程式碼命名方式

- 好的名稱所應具有的特質
  - 描述性
    - 透過謹慎地命名來表達正確的第一印象
    - 從外行讀者的角度來選擇名稱，而不是從自己內行的角度來選擇
  - 技術上正確的名稱
    - 大部份的現代程式語言，都有一些命名的準則。
    - C++語言的命名，不應使用 str開頭後面跟一個小寫字母或是底線開頭的全域識別字！或是不要使用std 的命名空間識別字。



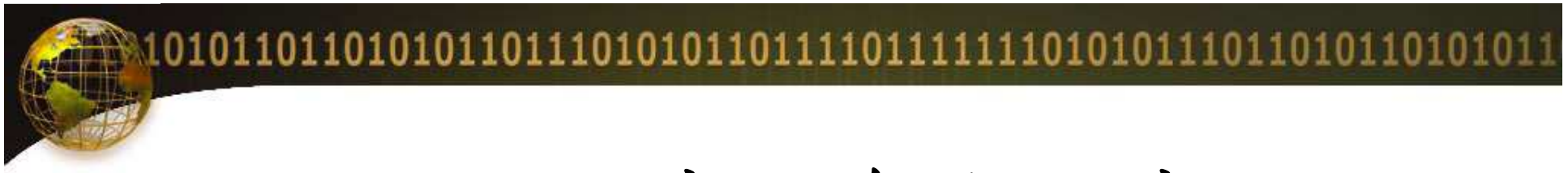
統一程式碼命名方式

- 符合語言習慣
  - 清晰明瞭的名稱，應遵從讀者所期望的慣例。
    - 程式語言的流暢程度，端看有沒有遵循符合程式語言習實的用法。
- 恰當的名稱
  - 長度
    - 如果一個名稱的涵義很明確，則其名稱的長度並不重要。
      - » 例： apple_count
    - 如果只是用於迴圈運算的變數，使用太長的名稱是無意義的。
      - » 例： for (int i = 0 ; i < apple_count ; i++)
  - 格調
    - 欠缺考慮的名稱會破壞程式碼的專業程度。例：boring_key …
    - 避免使用類似 foo 和 bar 等古怪的名稱！



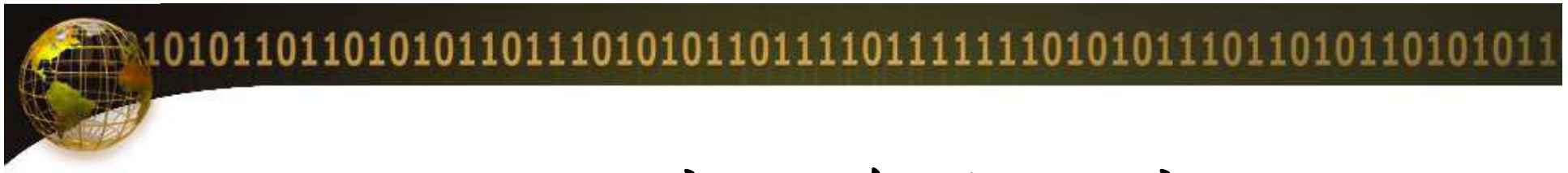
統一程式碼命名方式

- 命名方式概要
  - 命名變數
    - 通常都是名詞，可以直接反應出變數所代表的意義，例： elapsed_time 或是 exchange_rate
    - 如果不用名詞，通常也會是名詞化的動詞！例：widget_length 等等。
    - 另外，針對物件導向語言的慣例，會利用名稱來說明它們是屬於那一個物件的成員。例：BallSize、Cars_prize 等等。



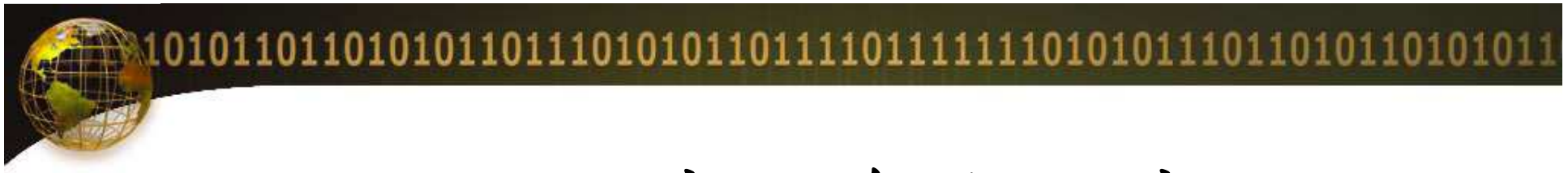
統一程式碼命名方式

- 命名函數
  - 通常使用動詞來表達函數的動作。例如：countApples() 、 metal.getName() 等。
  - 儘量隱藏函數內部的具體實作過程，始終從使用者的角度來為函數命名。
- 名稱空間的命名
  - C++ 的名稱空間和 Java 語言的package 就像袋子一樣，主要用於對構造物進行分組。
  - 選擇描述其內容的關係的詞語最好。例：controls 、 filesystem 等等



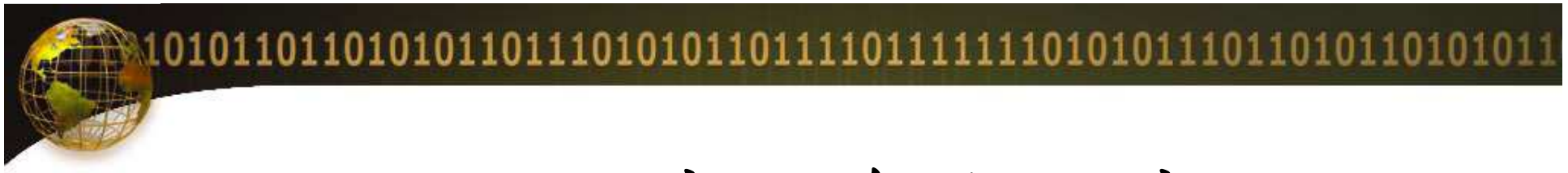
統一程式碼命名方式

- 命名巨集
  - 巨集有個很傳統的作法：名稱全使用大寫字母。
  - 應該被賦予在程式碼中其他地方都不會出現的獨特名稱。
  - 使用專案名稱來命名更好！例：PROJECT_MY_MACRO
- 命名檔案
  - 儘量將程式碼分到多個檔案中，使得每個檔案的功能單純化。
  - 檔案的名稱，則以該檔案有何功能來命名。



統一程式碼命名方式

- 命名檔案名稱時，更為細節的注意事項
  - 注意大小寫。有些系統是不分大小寫的。
  - 大小寫不同的系統中，儘量不要同時有大小寫不同的檔案名稱，例： foo.h 或是 Foo.h
  - 如果專案中使用了不同語言，則儘量不要使用相同的檔名而不同的副檔名。例： foo.c foo.java
  - 儘量確保所有的檔案名稱都不相同，即使是在不同的目錄內。

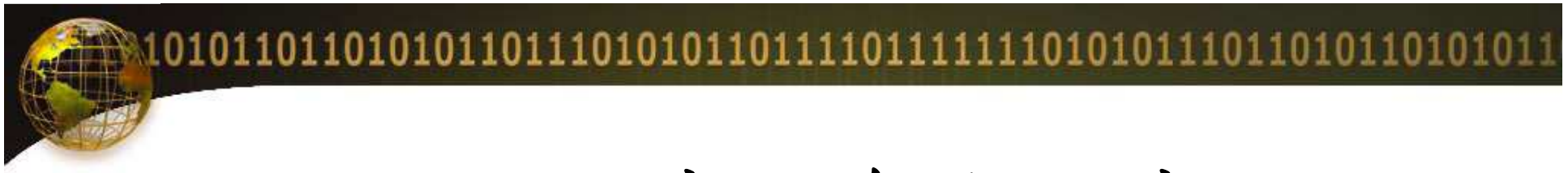


統一程式碼命名方式

- 爆爛的命名方式
  - A,B,C,D…..
    - 天曉得你在寫啥？(大概只有老天爺知道你！)

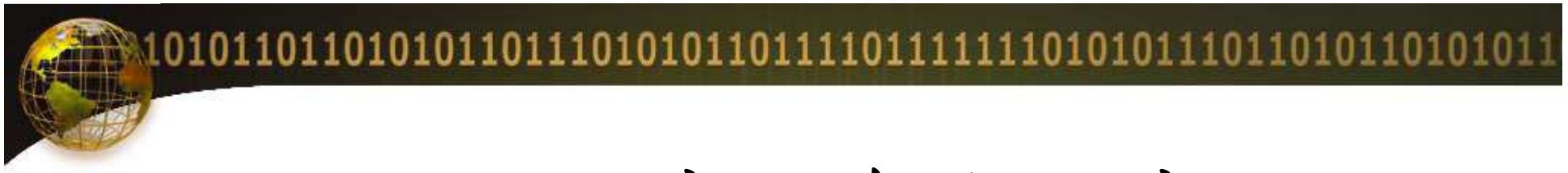
```
int exdented()
    {
    int a = 0, b = 0;
    while (a != b)
        {
        b++;
        a++;
        }
    return b;
    }
```

大概只有你肚子裡的迴蟲才知道你在寫啥吧？



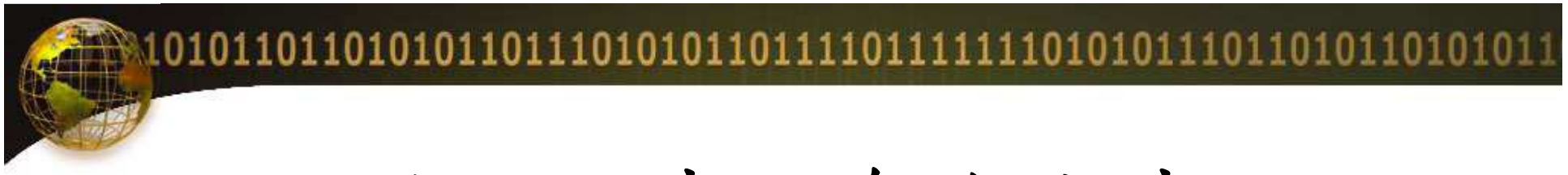
統一程式碼命名方式

- 全世界有名的命名方式
  - 匈牙利命名法
    - 此命名法又可細分為：系統匈牙利命名法和匈牙利應用命名法。
    - 一個變數名由一個或多個小寫字母開始，這些字母有助於記憶變數的類型和用途，緊跟着的就是程式設計師選擇的任何名稱。
    - 後半部分的首字母可以大寫，以區別前面的類型指示字母



統一程式碼命名方式

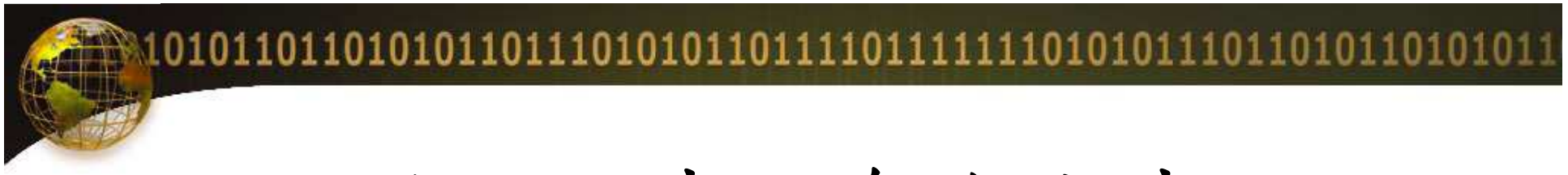
- 範例：
  - fBusy： boolean 型別
  - nSize：整數型別
  - fpPrice：浮點數
  - pszOwner：指向零結束字符串的指標
  - g_nWheels：全域名稱空間的成員
  - m_nWheels：structure／class 成員



統一程式碼命名方式

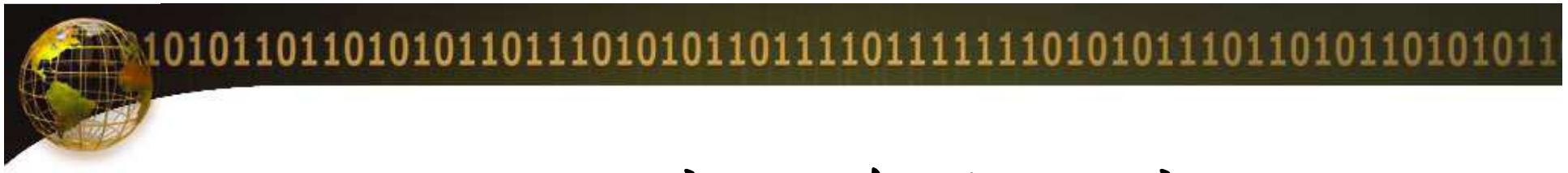
– 駝峰式命名法

- 當變數名和函式名是由二個或多個單字連結在一起，而構成的唯一識別字時，利用「駝峰式大小寫」來表示，可以增加變數和函式的可讀性。
- 命名規則可視為一種慣例，並無絕對與強制，為的是增加識別和可讀性。
- 一旦選用或設定好命名規則，在程式編寫時應保持一致格式。



統一程式碼命名方式

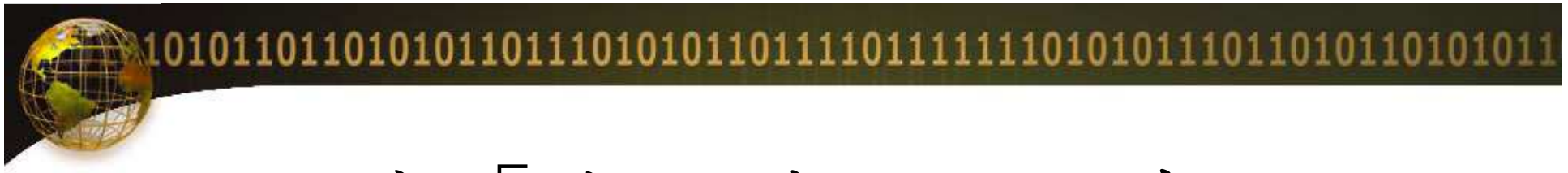
- 駝峰式命名法的分類
  - 小駝峰式命名法（lower camel case）：
    - 第一個單字以小寫字母開始；第二個單字的首字母大寫，例如：firstName、lastName。
  - 大駝峰式命名法（upper camel case）：
    - 每一個單字的首字母都採用大寫字母，例如：FirstName、LastName、CamelCase，也被稱為Pascal命名法。



統一程式碼命名方式

– 帕斯卡命名法

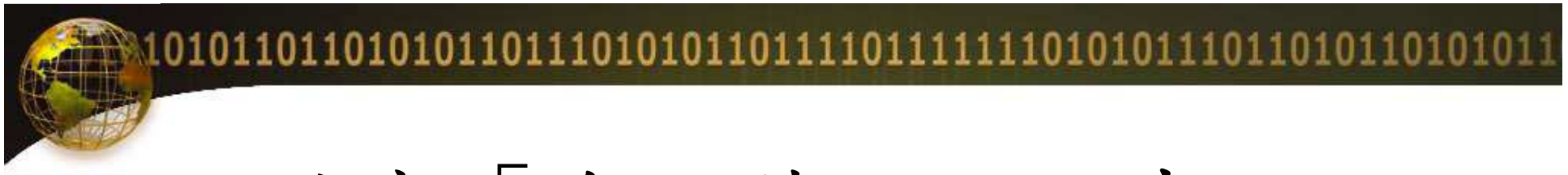
- 當變數和函式名稱是由二個或二個以上單字連結在一起，而構成的唯一識別字時，用以增加變數和函式的可讀性。
- 單字之間不以空格斷開或連接號（-）、底線（_）連結。
- 第一個單字首字母採用大寫字母；後續單字的首字母亦用大寫字母，例如：FirstName、LastName。
- 可視為一種命名慣例，並無絕對與強制，為的是增加識別和可讀性。



編寫「自文件化」程式碼

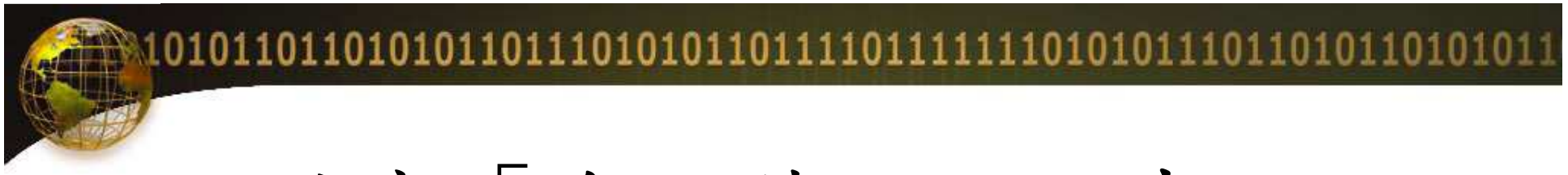
```
int fibonacci(int pisition){
    if (position < 2)    {
        return 1;
    }
    int previousButOne = 1;
    int previous = 1;
    int answer = 2;
    for (int n = 2; n < position; ++n)   {
        previousButOne = previous;
        previous = answer;
        answer = previous + previousButOne;
    }
    return answer;
}
```

有馬上看得懂，這是在寫啥程式嗎？



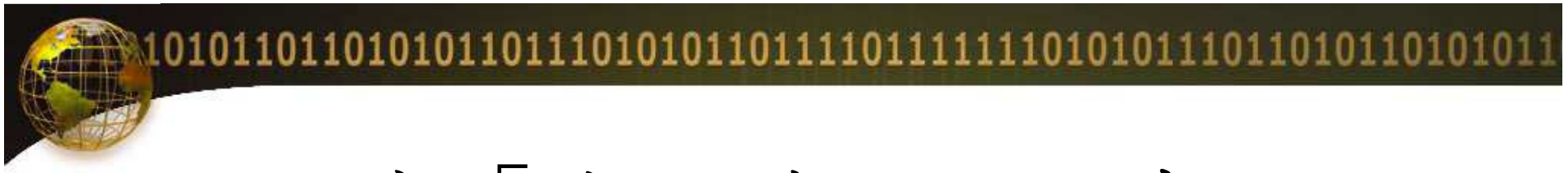
編寫「自文件化」程式碼

- 自文件化的程式碼的優點
  - 不用註解，即可明白程式碼所要表達的內容。
  - 簡潔有力的程式碼比註解來得更有閱讀上的效率。
  - 日後的維護上，不僅快速，而且也方便。



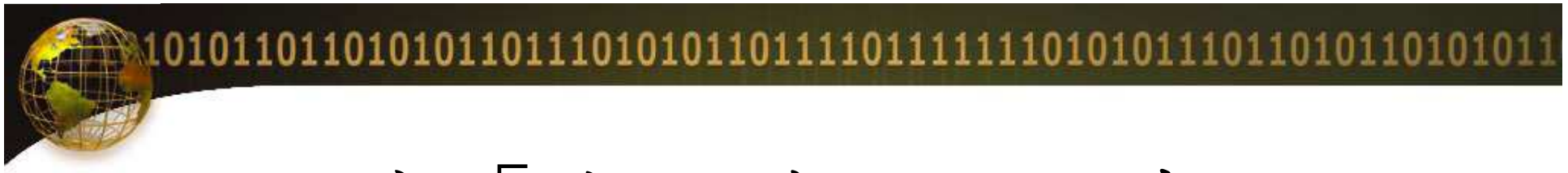
編寫「自文件化」程式碼

- 編寫自文件化程式碼的技術
  - 使用好的樣式編寫簡單的程式碼
    - 讓「正常」流程明顯地實穿你的程式碼。
      - 例： if-then-else 結構，必須要把順序固定起來，比如先判斷正常的情況，再判斷錯誤的情況，之後的相同結構也都必須固定同樣的流程！反之亦然。
    - 避免過多的巢狀式敘述。儘量做到單進單出。
    - 要謹慎的最佳化程式碼，防止它不再清晰的表達基礎的演算法。



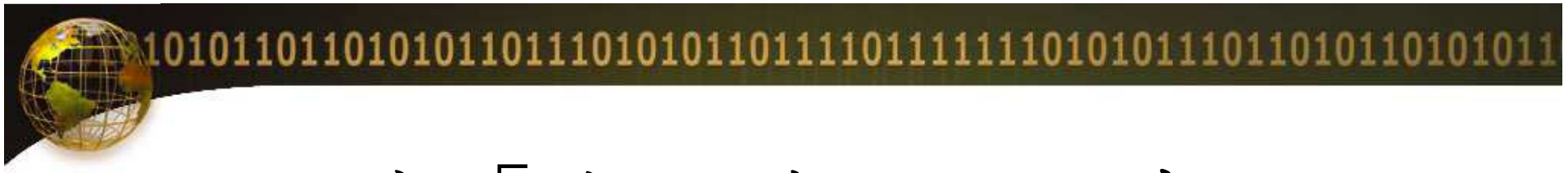
編寫「自文件化」程式碼

- 選擇有意義的名稱
- 分解為原子函數
  - 儘量每個函數只作一件事
  - 減少任何出人意料之外的副作用
  - 保持簡短、乾淨的而易於理解的程式內容。
- 選擇描述性的型別
  - 如果有一個永不改變的數值，那就定義成 const
  - 如果有一個數字永遠是正數，請定義成無符號型別。
  - 使用列舉來描述一組相關的值。
  - 選擇適當的型別。
    - 例： C++ 中，把實際的值放在 size_t 之中，而指標值則是放在 ptfdiff_t 之中。



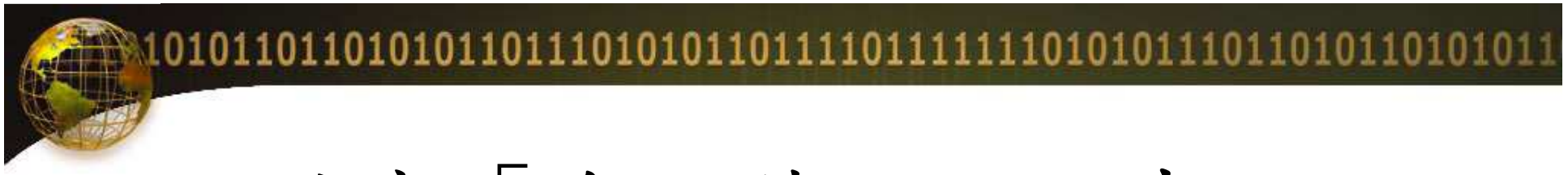
編寫「自文件化」程式碼

- 命名常數
  - 例： if (temperature == 100) , 100 應該可以用 boiling_point 來宣告！
- 強調重要的程式碼
  - 在類別按一定順序來進行宣告。
    - 公開的資訊應放在最前頭。
    - 私有的資訊放在最後，因為，跟使用者沒太多關係！
  - 盡可能隱藏所有不重要的資訊
  - 不要隱藏重要的程式碼。
    - 每一行程式碼，只放一行敘述，不要多行敘述擠成一行。
  - 限制巢狀敘述的數量



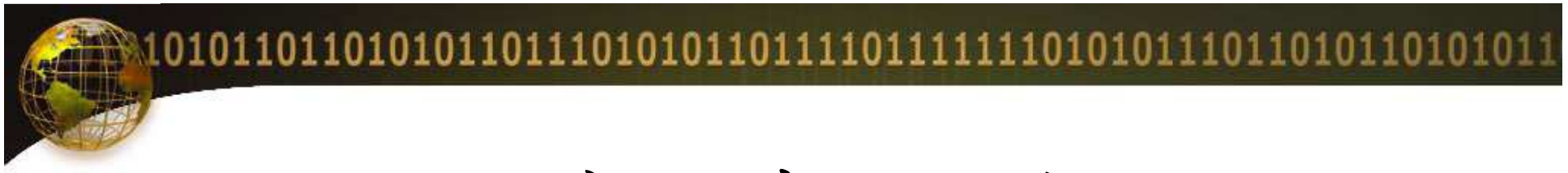
編寫「自文件化」程式碼

- 分組相關資訊
  - 將所有相關的資訊放在同一個地方。
  - 盡可能地透過語言的作法，將物件分類。例：C++ 可以使用 名稱空間，Java 則使用 package！
- 提供檔頭
  - 在檔案的頂部放置一個註解區塊，用來描述檔案的內容以及該檔案所屬的專案。
  - 檔案的頂部亦可放入版權的宣告，以提醒使用者相關的使用權利。



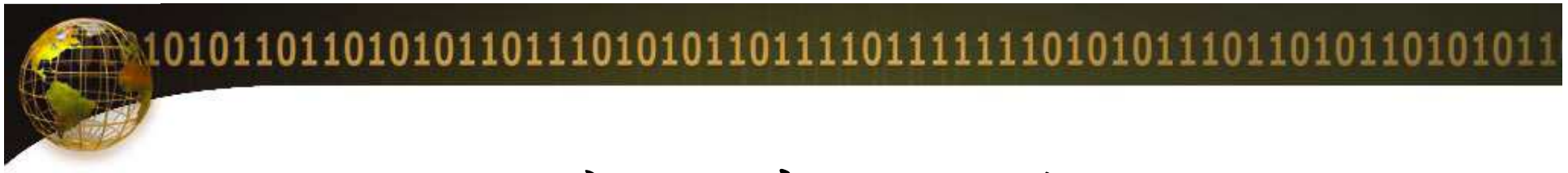
編寫「自文件化」程式碼

- 恰當的處理錯誤
  - 例：在讀寫檔案的時候，應該要先寫一段程式碼來偵測是否有任何讀寫上的問題！
  - 錯誤本身就是「該處理程式碼的問題是什麼」的精確描述。
- 編寫有意義的註解
  - 在編寫出最清晰的程式碼之後(表示已經不寫註解了)，其餘的資訊才使用註解表示。



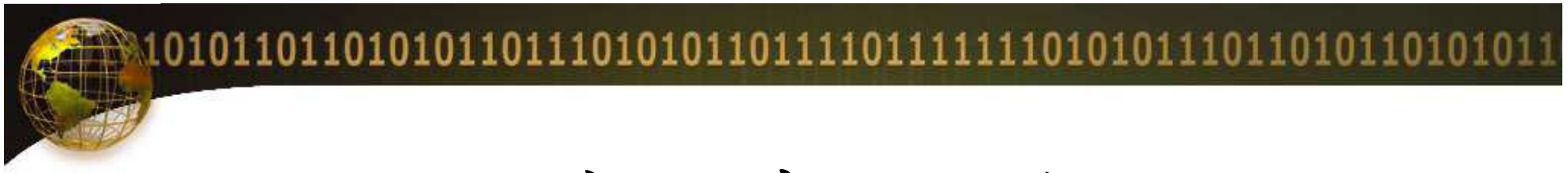
編寫程式碼註解

- 什麼是註解？
  - 編譯器不看的東西
  - 註解通常是對其所處位置的程式碼做解釋
  - 註解的目標是人，不是電腦
  - 程式碼註解不是你應當放在程式碼中的唯一文件。
  - 作為負責的程式設計師，寫註解是義務也是責任。



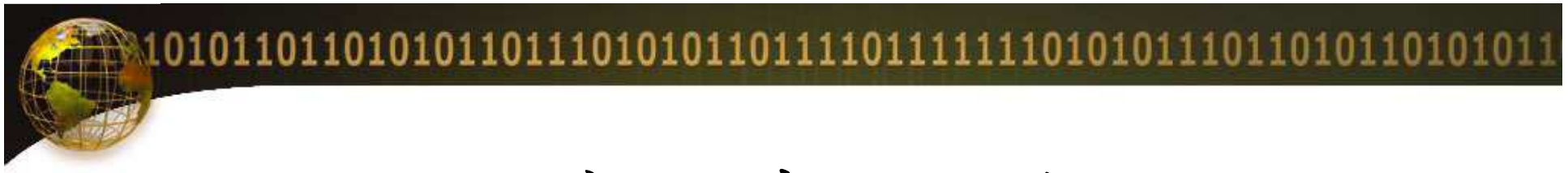
編寫程式碼註解

- 多少註解才是恰當的？
  - 寫註解的重點在於註解的品質上。
  - 只有能在為程式碼增添色彩的時候，才編寫註解。
  - 看程式碼的人，也會看註解！儘量以自文件式的程式碼來說明程式，而不是大量的註解。
  - 編寫得非常好的程式，應該是不用寫註解的。



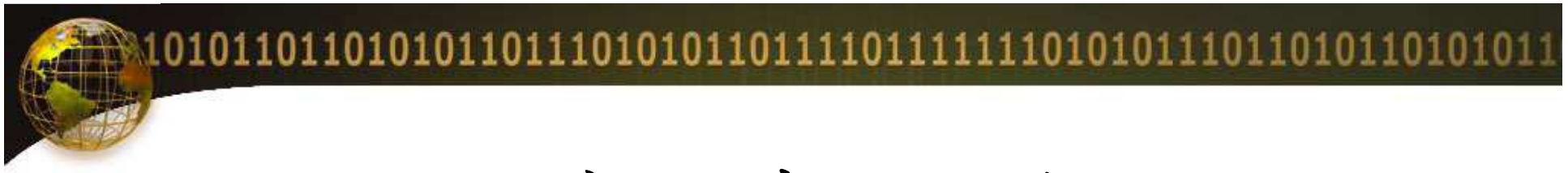
編寫程式碼註解

- 註解裡該寫些什麼？
  - 解釋為何，而非如何
    - 註解應該明確告知寫這一段程式碼的目的是要做何用途！
    - 因為程式的演算內容會改變，而目的卻從不改變！
  - 不要描述程式碼
    - 尤其是不要用英文重複描述程式碼！如果真的是這樣，那應該考慮程式內容重寫！
    - 如果你在大量的解釋程式碼是如何進行演算法的步碼，則必須要快點停下來，重新好好思考程式碼的內容。



編寫程式碼註解

- 確保註解是有用的內容
  - 記錄意想不到的內容。
    - 如果程式碼內容是不常見的、或是常令人意想不到的，則必須要寫上註解。
    - 涵義明顯的程式碼是不用寫註解的。
  - 描述事實
    - 由於程式碼常修改，最後有可能程式碼已經在做別的事，而註解仍停留在早先的狀況，即會發生註解是在欺騙讀者的狀況！
  - 寫有意義的事
    - 不要在註解裡開玩笑，或是使用不雅的文字或是自以為是幽默的描述語句。

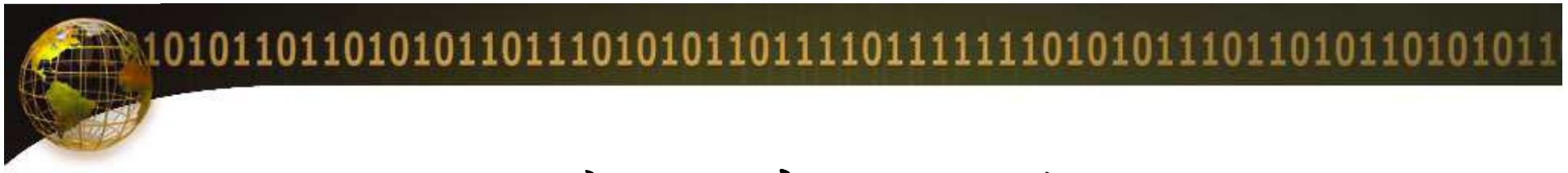


編寫程式碼註解

- 清晰明瞭
  - 註解的作用在於標註和解釋程式碼，所以，儘量具體描述程式碼的目的與意義！
- 容易理解
  - 註解中不必一定要寫出完整的而語法正確的英文句子，但是，一定是必須可讀的！
  - 儘量不要只有使用單字來寫註解。

– 避免分心的描述方式

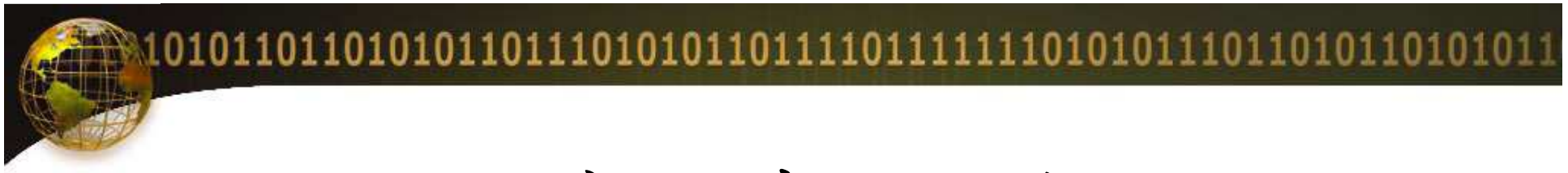
- 講過去的事情
- 不要把需要剔除的程式碼，放在註解中，會造成認知上的混淆！



編寫程式碼註解

- 避免使用ASCII 藝術圖形，例： ^^^ 來突顯上一行的某一段程式碼！
- 避免無意義的程式碼區塊結尾，例： 在 if 敘述之後，加上 // end if (a<1) ！

- 註解之美
  - 一致性
    - 所有的註解都應該要清晰明瞭，前後一致！
    - 當註解方式決定之後，應要使團隊所有成員遵守相同的註解規定。



編寫程式碼註解

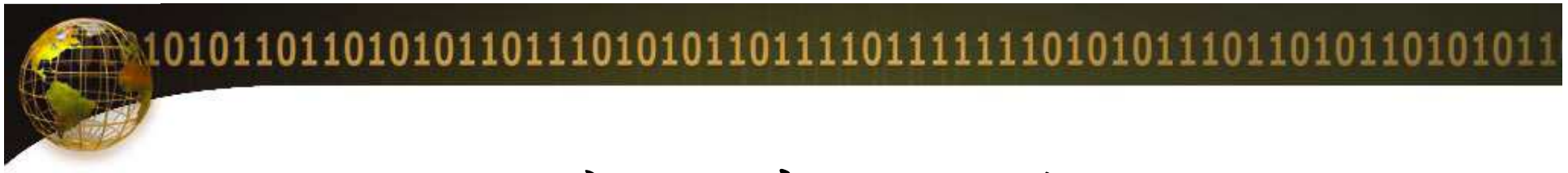
- 清晰的塊狀註解

```
/*
 * This is much more readable
 * as a block comment in the midst
 * of a whole pile of code
 */
```

讓 /* 與 */ 都各自佔用一行

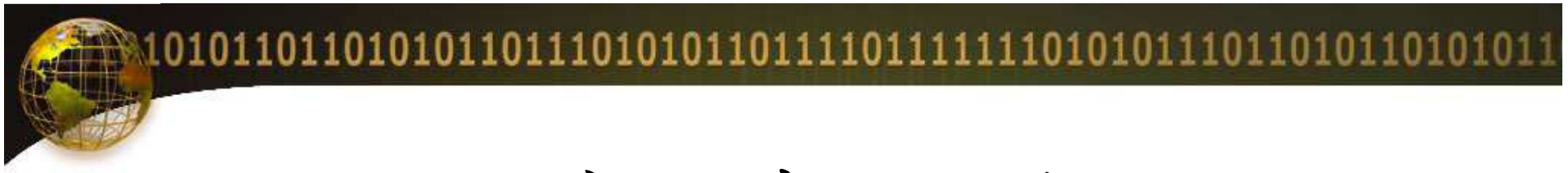
星號就讓他們在左邊對齊

註解的內容清楚交待程式碼的內容爲何目的，而不是做法



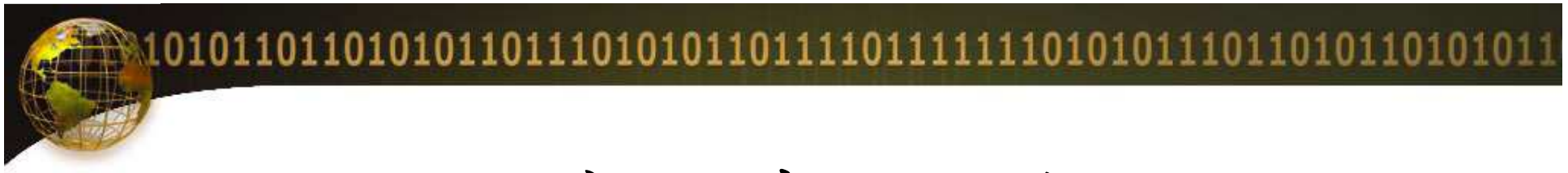
編寫程式碼註解

- 內縮註解
  - 註解不應打斷程式碼或者打斷邏輯的流程。
  - 讓註解的內縮位置與周圍的程式碼保持一致。
- 行尾註解
  - 大多數的註解都是單獨地放在各自的行上。
  - 但是有時較短的單行註解可以跟在程式碼敘述的後面。
- 協助讀者閱讀程式碼
  - 註解通常在它描述的程式碼上方，而不是下方！這跟閱讀習慣有關。
  - 善用空白行可讓程式碼看起來跟文章有相同段落的感覺。



編寫程式碼註解

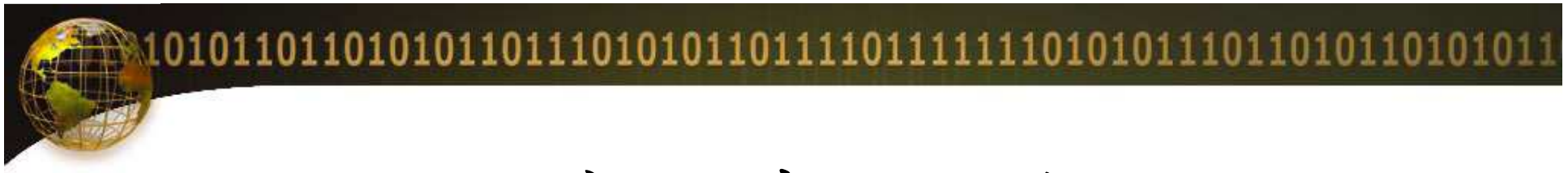
- 選擇維護成本低的註解風格
  - 寫註解非常花時間，應該要選用精簡的風格來降低所消耗的時間。
  - 在美觀的原始碼和維護成本之間，取得平衡。(因為有些程式設計師會把註解編排得十分華麗，例如：左右邊都有星號，而且都有對齊)
- 分隔板
  - 註解經常被用作不同程式碼之間的「分隔板」
  - 程式設計師常用不同的方案來區分主要的註解與次要的註解。



編寫程式碼註解

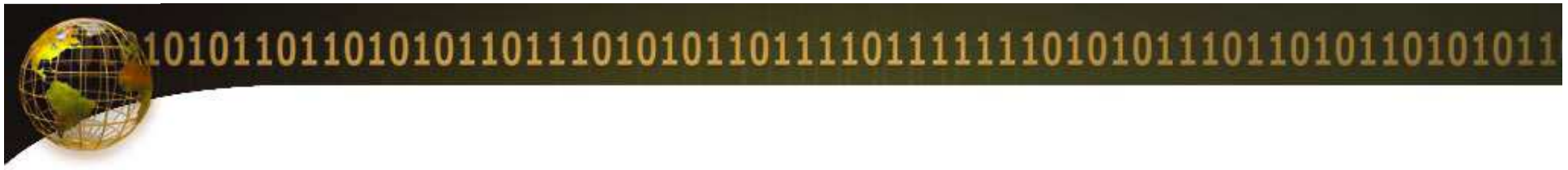
- 選擇適當的「分隔板」可以協助使用者快速瀏覽檔案內容。
- 函數與函數之間，則不須要使用大量的「分隔板」，空白幾行即有效果！

```
/*****************************
 * Class foo implementation
 *****************************/
```



編寫程式碼註解

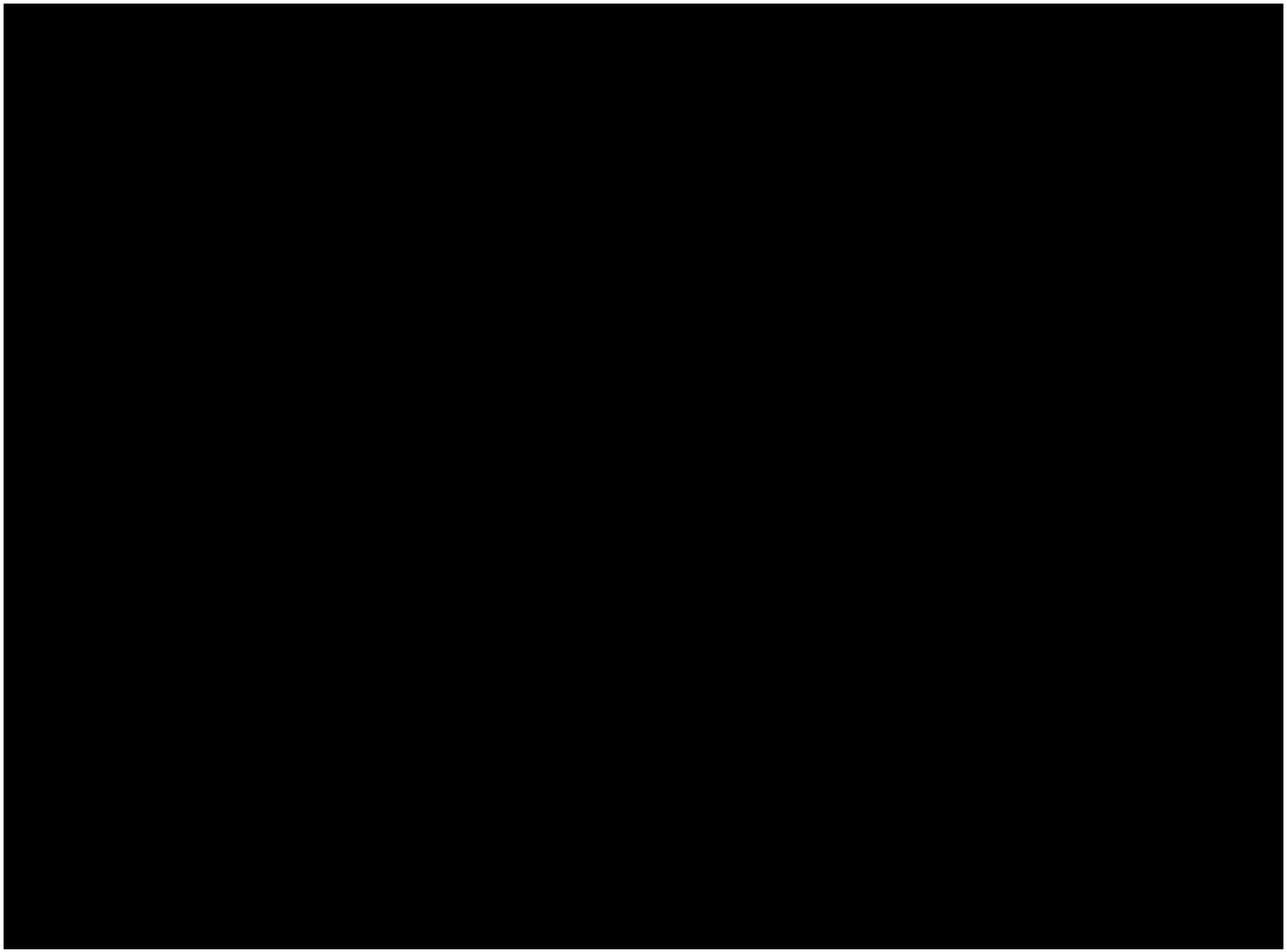
- 標註
  - 利用註解來標定哪些地方是要修改的，哪些地方則是尚未完成。
    - 例： //FIXME 表示該行程式有問題
    - 例：//TODO 則表示程式還沒寫完
- 文件檔標頭
  - 所有原始檔案都應該以描述其內容的註解區塊作為開頭。
    - 內容包含：程式碼的概述、前言，但不是包含所有函數、類別、全域變數等內容。
  - 檔案標頭註解的內容，不應是過期的資訊。
  - 不需要包含每次修改程式的記錄。



總　結

- 為使得程式碼易於開發、維護，遵守編排風格是必要的，而且也是必須的。
- 常見的抱怨：「程式碼看不懂」，其實是在講程式碼很難閱讀！

- antallen@gmail.com



學歷、專業力、學習力

校園巡迴講座 活動問卷

感謝您參加本場講座活動，敬請您提供以下寶貴意見，以供未來活動改進參考。

問卷題目

1.講座活動滿意度調查：

(1) 講座內容規劃
□非常滿意 □滿意 □尚可 □有點不滿意 □不滿意 □非常不滿意

(2) 講師表達方式
□非常滿意 □滿意 □尚可 □有點不滿意 □不滿意 □非常不滿意

寫 □職涯探索

_____)

：_____

關資訊？

腦動畫

技能軟體 □電腦技能認證考試

免費贈送學生體驗90天

聯成電腦數位學苑行動線上學習
享有線上所有的電腦軟體技能教學影音課程

數位學苑 帳號申請表

本人同意提供以下個人基本資料，作為申請「聯成電腦數位學苑線上學習」帳號之用。

申請學生親自簽名：_____ (中文字體端正勿潦草)

學校名稱：_____ (中文字體端正勿潦草)

學系年級：_____

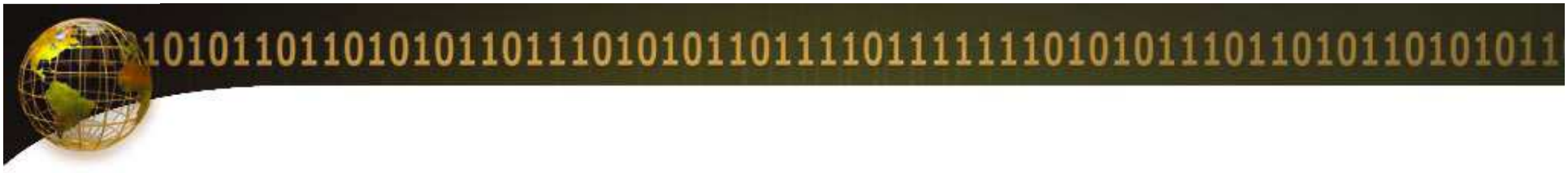
學　制：□五專 □二專 □四技 □二技 □四年制大學 □碩士班 □博士班

畢業年月：　　年　　月

行動電話：_____ (號碼字體端正勿潦草)

E-mail信箱：_____ (英文字體端正勿潦草)

註：1. **本帳號申請表，請務必親自簽名，並逐欄填寫完整；提供個人專屬行動電話與E-mail信箱，以利收受帳號/密碼。**
2. 本表經申請核定後，**將開立有效期90天**「聯成數位學苑線上所有的電腦軟體技能教學影音課程」學習帳號。
3. 帳號開通後，為了解同學線上電腦教學影音使用情形，未來將有客服人員致電關懷詢問改進意見。



Q & A

聯成電腦 校園巡迴講座

青年夢想微笑曲線

青年夢想放大
人生自由揮灑

簡 報 結 束