**為什麼要學習程式語言？**

從理論上的學習可以幫助人們設計出更好用的程式語言。

學習使用實際的程式語言可以寫出解決問題的程式。

**和電腦溝通**

進行的工作或是要解決的問題必須有很明確的描述，同時也有預期達成的結果。

把工作進行的程序詳細的寫出來。

把程序轉化成以程序語言寫成的程式（Program），就可以交由電腦來幫我們完成所要進行的工作。

**語言的抽象化**

以機器語言層次的程式來處理比較簡單的語言寫出來的程式。

用一種比較平易親和的程式語言來撰寫程式。

轉換成機器語言，交由CPU來執行。

**程式語言與機器語言之間的轉譯**

以機器語言存在的系統程式能做程式語言與機器語言之間的轉譯。

這一類的程式被稱為【編譯器】（Compiler）或是【直譯器】（Interpreter）。

Fortran、Cobol、Pascal、C、C++、Basic、Java、Ada等都是有名的程式語言。

也有自然語言（Natural Language）方面的研發，希望能把與電腦之間的溝通更進一步地簡化。

**程式語言發展史**

程式語言的演進已經有大約半世紀的歷史，從這一段演進的歷史，我們可以大約的想像出程式語言在資訊科學領域中的重要地位。

1.在西元1950年中期就有高階的程式語言出現，例如Fortran、APL。

2.到了1960年代和1970年代初期，ALGOL的發展對程式語言產生了很大的影響。

3.1980年代是物件導向程式語言盛行的時期，雖然物件導向的觀念很早就有了，真正把它應用在程式開發上還是由程式語言的發展。

4.1990年代軟體工具的運用普及，很多程式語言稱為整合性開發工具的一部分。

**程式語言的組成**

一個程式語言最主要的組成是其語法（Syntax）與語意（Semantics）。

語法代表程式語言所提供的語言，會決定寫出來程式的外觀。

語意代表各種語法所描述的功能，以及被執行時產生的作用。

**簡單的程式實例**

X:=Y；

X和y代表程式變數（program variable）的名稱。

在語法上我們使用了指定（Assignment）的描述，即符號【:=】。

在語意上表示將Y的值指定給X。

執行的時候，系統會被儲存在Y位置的值寫入X位置所對應的儲存空間，也就是記憶體中特定的位置。

**程式語言文法主要的成分**

終結符號（Terminal symbol）

非終結符號（Non-terminal symbol）

產生規則（Production）

目標符號（Goal symbol）

目標符號 非終結符號 終結符號

<句子> := <主詞>+<動詞>+<輔語>+。

<主詞> := 老師|學生

<動詞> := 教學|學習

<輔語> := 認識|鬆散

終結符號

-------------------------

產生規則

**貝諾爾格式（BNF）Back Naur Form**

文法：

<goal> := [x]y{<money>}

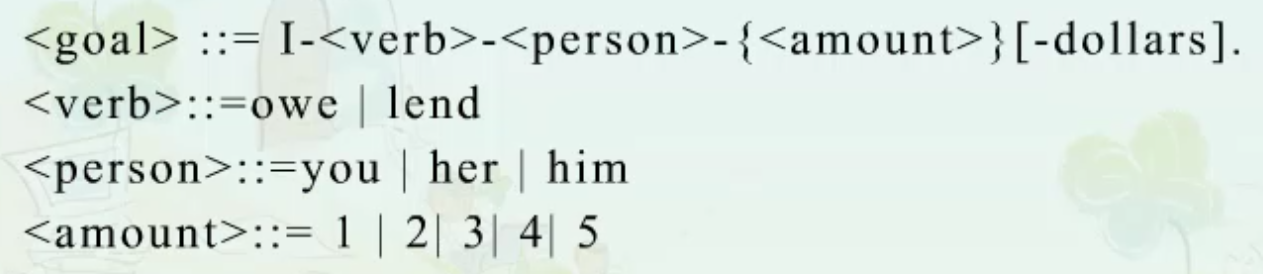
<money> := 10 | 20 | 30

1.xy10

xy可以對應到第一個文法規則，[x]y

10可以對應到第二個文法規則<money>可以被10、20、30完成取代。

~~2.而y20x則不行~~



**程式語言採用的方法論（Paradigm）**

通常每一種語言的語法都會有差異，所以從程式語言所採用的方法論（Paradigm）來做分類，比較容易找到同一類的語言。

常見的程式語言方法論有4種：

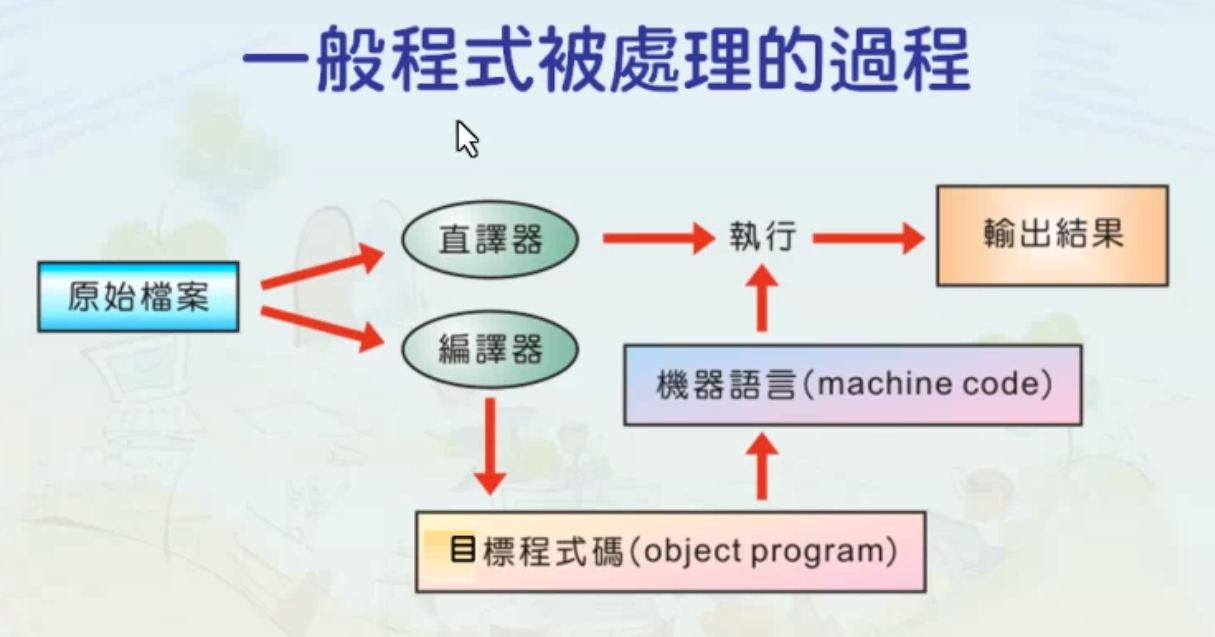
1. 命令式的（Imperative）：Pascal、C
2. 函數式的（Functional）：以函式的呼叫與傳回參數的方式來描述運算
3. 邏輯式（logic）：利用語法來描述試試（Facts）以及推理的規則（Rules），程式執行時就以邏輯的理論為根本，以事實為基礎，用規則來推理出更多的結論。
4. 物件導向的（object-oriented）：①封裝（encapsulation） ②繼承（inheritance） ③多元性（polymorphism）

這些方法論決定了程序語言的運算模型（model of computation）

宣告式的語言不必詳細描述運算的步驟，像資料庫系統常用的結構化查詢語言（SQL, Structured Query Language）就是一種宣告式的語言。

程序式的語言層一步一步地描述運算的過程，像 Pascal、C、C++ 等就可算是程序式的程式語言，這一類的語言能將各種運算描述的非常清楚，具有運算上的完整性（Computational-completeness）。

**Java在寫程式的時候還是需要一步一步把運算寫清楚，屬於程序式的程式語言。**



**直譯**

原始程式碼為直譯器（Interpreter）的輸入。

每譯完一個指令就馬上執行，然後再處理下一個指令。

直譯的方式感覺上效率比較高，但實際上不見得。

**編譯**

原始程式碼為編譯器（Compiler）的輸入。

全部轉換成目標程式（object program）。這個過程叫做編譯時期（Compile-time）。

目標程式可以直接對應成機器語言在電腦上執行，這個過程叫做執行時期（Run-time）。

**直譯 vs 編譯**

通常編譯時期會花掉一些時間，但是編譯完之後的程式執行起來會比直譯的情況有效率。

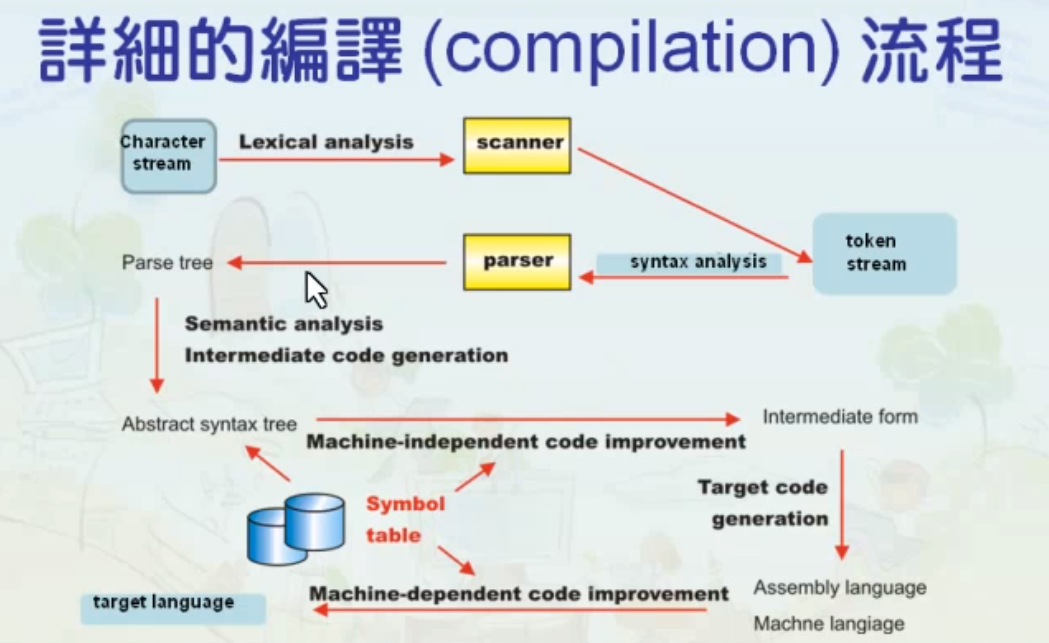
一旦程式除錯完畢，只需要編譯過一次，就可反復執行。

直譯程式每次執行都要再直譯一次。

目前大多數的程式語言在處理上主要仍採用編譯的方式。

**名稱的連接（繫接）**

詳細的編譯（compilation）流程



1. character stream: 字元串流
2. lexical analysis：句法分析
3. scanner：掃描
4. token stream：符號串流
5. syntax analysis：語法分析
6. parser：解析器
7. parser：資料結構
8. Semantic analysis：語義分析
9. Intermediate code generation：中間碼
10. Abstract syntax tree：語法樹
11. Machine-independent code improvement：執行中間的格式，通過code improvement改善程式的效率
12. Symbol table：符號表格

**繫結（binding）發生的時機**

語言設計的時候

語言製作的時候

寫程式的時候

編譯時期

連接時期

載入時期（load time）

執行時期（run time）

靜態繫結（static binding）表示繫結發生在執行時期之前。

假如繫結發生在執行時期則稱為動態繫結（dynamic binding）。

繫結的時間（binding times）在程式語言的設計上是很重要的問題，提早的繫結時間（early binding times）有助於提升程式效能，延續的繫結時間（late binding times）則容許比較大的彈性。

**懸置的引用（dangling reference）**

1.程式中所引用的物件在執行時期會產生，物件的內涵與其名稱會繫結（bind）在一起。

2.在這個過程中，除了物件本身的建立（creation）之外，物件透過繫結（bind）得到了名稱。

3.執行過程中現有的繫結可能會改變，等到不再用到該物件時，繫結與物件都會被移除。這就是繫結與物件的生命週期（lifetime）。

4.假如物件已經不存在，而仍有繫結到此物件的名稱在使用，我們就把這樣的名稱叫做懸置的引用（dangling reference）。

**物件生命週期儲存空間的配置方式**

靜態的物件（static objects）

堆疊物件（stack objects）

堆積物件（heap objects）

**副程式的靜態配置**

暫存區（temporaries）

區域變數（local variable）

一般記錄（bookkeeping）

回傳的位址（return address）

參數（arguments）與回傳資料

**範圍法則（scope rule）**

1.某個名稱的繫結（binding）只在程式的某個範圍（scope）內才有效，這是繫結的範圍（scope of binding）的由來。

2.通常繫結的範圍是在編譯時期就決定的，所以屬於靜態的。

3.有些語言支援動態範圍（dynamic scope），則繫結會與執行時期的流程控制有關。

4.在程式執行的過程中，有效繫結（active bindig）形成目前的引用環境（referencing environment）。

分開編譯（separate compilation）的觀念

1. 大型的程式通常都要慢慢地擴充與測試，假如要一次編譯將會花費很多的時間。
2. 因此大多數的程式語言都會支援所謂的分開編譯（separate compilation）的功能。
3. 編譯的單位常稱為模組（module），C語言早期將宣告的部分放在【.h】的header file中，讓其他的檔案也能共用，但是當header file改變時將需要重新編譯共用該檔案的所有程式。
4. Java以包裹（package）來代替模組的觀念，除了公用類別之外，Java類別只能讓相同包裹中的其他類別引用。

**用程式語言描述資料**

資訊本身可以分成數種資料形態（data types），不同的資料形態有不一樣的格式與儲存方式。

寫程式的時候要依照資訊的特性來決定資訊所屬的資料形態。

資料形態也可以依需要組合成資料結構（data structure），在各種場合中運用。

**基本的資料形態**

一般程式語言寫出來的程式處理的最基本的資料常稱為：

1. 基本值（Literals）
2. 整數值（Integer literals）
3. 浮點數（Floating-Point literals）
4. 布林值（Boolean literals）
5. 字元值（Character literals）
6. 字串值（String literals）

基本值可以指定成變數（variable）的值，該變數就具有基本值所屬的資料形態。

例如變數x=20，x是程式變數，20是整數值，則x的資料形態就是整數資料形態（integer data type）。

**認識運算子（operator）**

各種資料形態都有一些相關的運算子，用啦描述數值的運算。

常見的加（+）、減（-）、乘（\*）、除（/）和取餘數的算數運算子。

布林值可以使用布林運算，也就是邏輯運算。

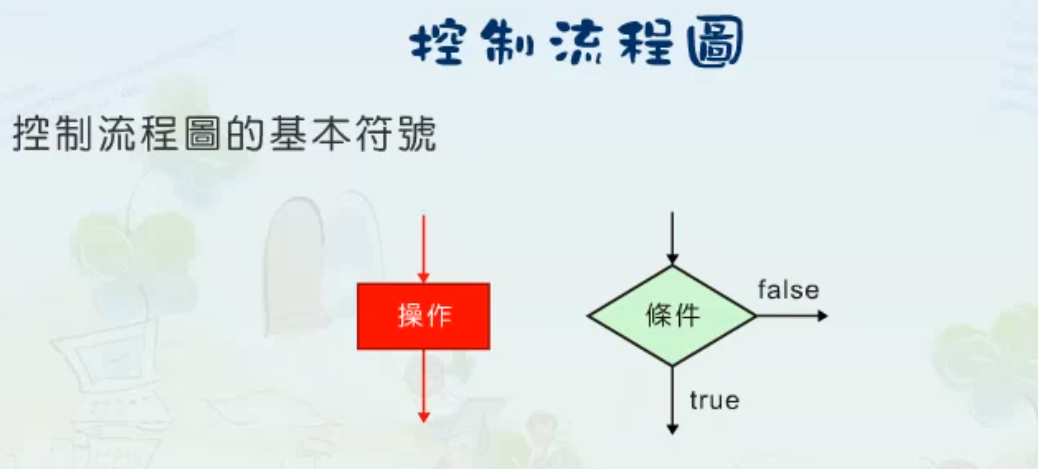
**程式中資料的使用**

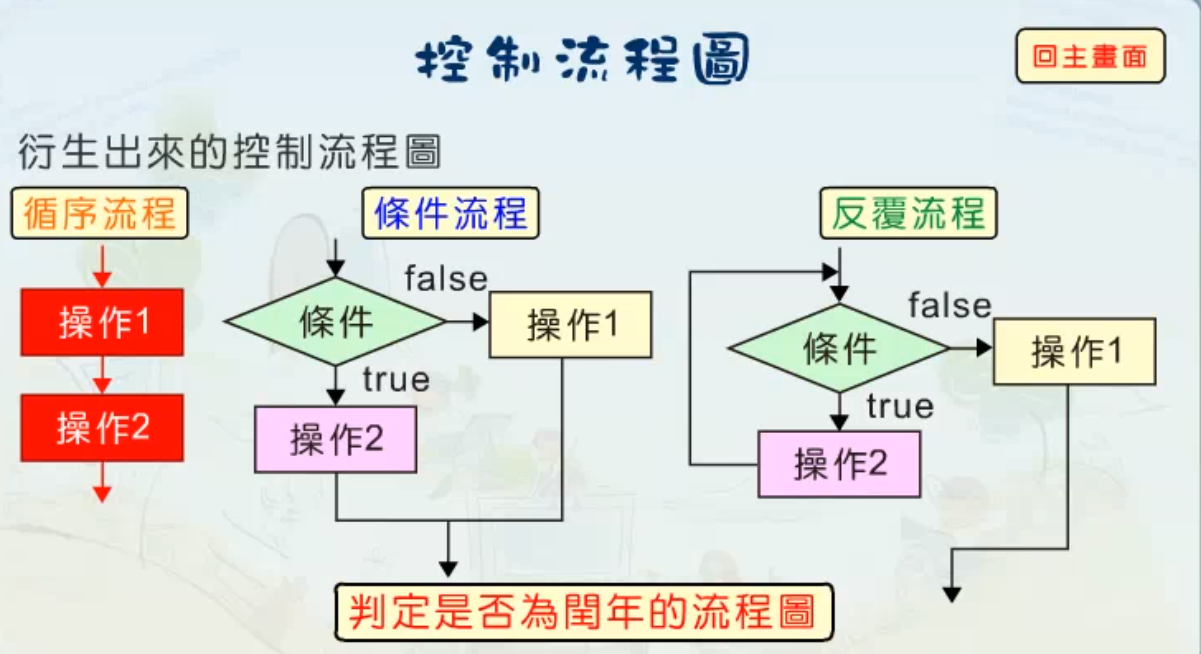
常熟（constant）：如圓周率、在程式執行的過程中不會改變

變數（variable）：在程式執行的過程中會改變，可以看成一個儲存資料的空間，裡面儲存的值會改變

**控制結構**

1. 順序（sequencing）
2. 選擇（selection）
3. 反復（iteration）
4. 遞迴（recursion）
5. 程序抽象化（procedural abstraction）
6. 並行（concurrency）





**程式開發的環境**

1.做簡單的語法上的檢查。

2.顯示提示文字幫助程式設計者找到要使用的語法。

3.讓使用者在程式中快速地來回瀏覽，找到需要修改或新增內容的地方。

4.程式組態的管理(configuration management) ，例如各程式模組的相關性與編譯的先後記錄。

**第一章總結：**

1.程式語言的功能：我們人與電腦溝通的媒介，程式語言是我們跟電腦溝通的工具

2.程式語言的理論基礎：程式語言要經過處理之後才能讓電腦依照邏輯工作。

3.程式語言的要領：程式語言最重要的就是它的語法和語意。

