10727127 資訊三甲 邱佳駿 PL 期末報告

語言，對我們人類來說，是一種使用工具，用來溝通的工具，不同的背景下，會發展出不同的語言，如:英文、中文、法文、閩南語、原住民語……等。而語言本身具有創造性，根據現實狀況，會繁生出新的詞句，比如在 2012 年突然火紅起來的台灣之光-林書豪。一個沒沒無聞的華人，竟然帶領了尼克隊拿到了 7 連勝，幫助該隊伍成功進入季後賽，因為林書豪這樣瘋狂的表現，創造出了一個新的詞—linsanity，表示迅速成為潮流的人。

相信大多數的人從小就是在中文環境下長大，所以對於中文這個語言肯定是非常的熟悉，隨者教育體制的更新，第一外語”英文 ”，成了教育的重點。回想一下過去在學英文的過程以及現在看英文的感受，肯定有很大的不同。以前常常會一直查說 xxx 單字叫什麼、ooo 這個文法要怎麼用啊……等之類的，而現在呢?也許這些事情還是會發生，但發生的頻率肯定沒有以往高，我認為會有這樣的情形，是因為對於此種語言更熟悉及更習慣了。根據以上的論述，我認為會對語言產生影響或會有不同的看法的有幾種原因:

一、使用環境

二、社會發展

三、習慣

我認為程式語言也跟日常對話的語言也有同樣的道理，同樣是個溝通的工具(與機器溝通)，在不同的使用環境下適合不同的語言(APP 設計 => java、深度學習 => python)，因應社會發展程式語言也會不斷的進化(C => C/C++)，自己在平常用最多的就是 C/C++，就會覺得這語言很舒服很舒適，那其他語言平常就比較少在使用，所以當要用的時候腦袋會有架構說要怎麼寫，可是該語言的用法就是不習慣或是會用錯，常常就要一直去上網查用法要怎麼用，所以會比較容易認定其他語言寫起來蠻卡手或不方便的。

接下來就來談談我曾經接觸過的語言以及對它的看法及批判與評價吧!想當然而，第一個要談論的語言就是我最熟悉的 C/C++啦(以下簡稱 C)~~

我認為 C 是非常適合想要深入去了解及寫程式的新手第一個接觸的語言，如果只是想要寫程式不想了解細部的內容的話，學 python 就好了(後面會提到 python)。C 因為是 complier 的關係，所以在執行前任何的語法都不能有錯誤，否則 IDR 會跟你講說他不知道麼個東西是什麼或是它的型別或參數有誤……等之類的 error。C 在任何變數在使用前都一定要宣告一個型別，寫程式的人就會很清楚的知道說這個變數是放什麼樣的東西進去，如 assign 錯誤的話 IDE 也會跟你說好像放錯東西進去了。

再來要說到我覺得 C 最ㄅㄧㄤˋ的東西 ➔ for 迴圈，for 迴圈我認為用起來會比 while 好用一些，因為用迴圈多半會用在陣列，那陣列的 index 值通常就是一個暫時性的變數，跑完迴圈就可以丟了，不用去理會那個變數最後怎麼樣了，for 迴圈時常會跟 vector 做搭配，可以很快設定好 index 初始值及迴

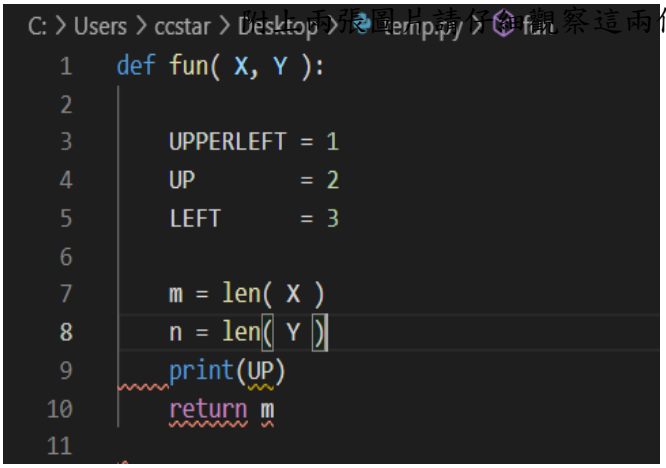
圈條件，當存取 vector 發生一些問題時，直接印出 index 值及該位置的東西就蠻好去 debug。談到 C 的 for 迴圈，就必須跟 python 的 for 迴圈來做個比較，我直接開門見山的說，我認為 python 的 for 迴圈真的很爛很難用，就是因為 python 設計的太方便了，有些東西簡化的太過了，python 使用 for 迴圈的用法是 for i in something:，這個 i 我把它稱為是變身超人，為什麼這樣稱呼呢?因為它會根據 something 的裡面存的型別去做改變，假如 something 是[1,2]那 i 的型別就會是 int，若 something 是[“TOM”,”JACK”]，i 的型別就會是 string，把 for 稍微小改一下變成 for i in range(something): i 就會變成 index 的狀況，也就是 int。XX(這個是髒話也可以稱為語助詞，請自行帶入。以下也會繼續出現 XX，就請自行轉換吧~~)，這樣變來變去誰受的了啊!! 假如我要取代某個特定位置，或是只要查看前面幾個的東西，還不是需要 index。那假如 something 是個二維陣列，i 取出來是個 1 為陣列，那我要怎麼知道它是二維陣列的第幾個陣列，啊還不是要用到 index，那為什麼不用 idex 就好了，所以現在基本上我在用 python 的迴圈時，都會用 for i in range(something): 的形式，好把 i 的型別給定清楚，不讓他當變身超人，方便我比較好閱讀及讀懂程式碼。再來談談 C 我覺得有些部分可以做改進的地方，第一個地方就是每一行指令也就是每一行程式碼最後都要加一個分號(;)，我的想法是:同一行加分號就算了(如: for 迴圈)因為要把指令給切開，但基本上我們寫的程式碼幾乎都是一行一個指令，這點不得不說 python 最到了，挺好挺好。正常人不會把 int a = 10 ; int b = 5 ; 這兩個指令寫成同一行吧!再來第二個我覺得可以改進的地方是有關 function 的部分，以同樣層級來看 function，後宣告的 function 可以使用前面宣告過的任何 function，但前面的 function 不能使用任何比他還要後面宣告的 function，沒有不好，但可以更好。因為在有些特殊的情況下會需要用到前面的 function，這時候只好另闢另一條路了。

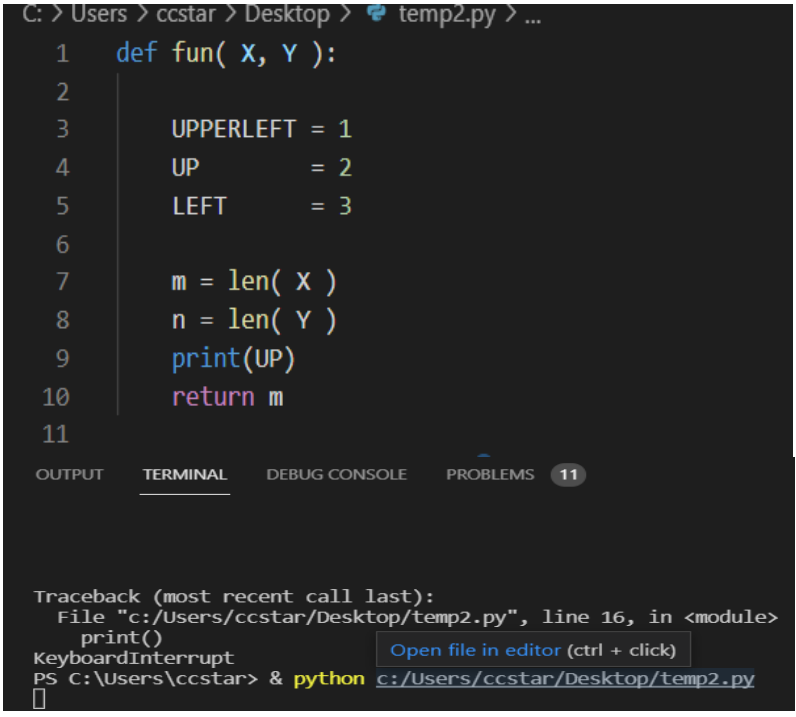
接者來聊聊第二個程式語言吧~~人人都在自學的語言，我稱為因為太過方便而讓人隨便的語言-python。

首先來談談 python 的第一個特色:變數不用先宣告就可以直接使用，我認為看似是他的優勢但可能是他的劣勢，隨便舉幾個例子來說，假設寫了一之 5000 行的程式好了，使用的變數相對的也可能會比較多，在寫到一個段落時需要一個變數，但使用的那個變數的名字好死不死你以為之前沒使用過，但那個變數剛好是個全域變數，這樣整隻程式不就可能出了個大問題嗎?在一個例子，在 for 迴圈中去 assign 一個值，阿 for 迴圈那個 i 就會根據 something 的型態做轉變，當 IDE 告訴你要錯時，還要去好好地確認 something 到底是什麼型別，當初把所有的型別以及宣告給做好不就完事了嗎?就不會產生出這些問題啦。

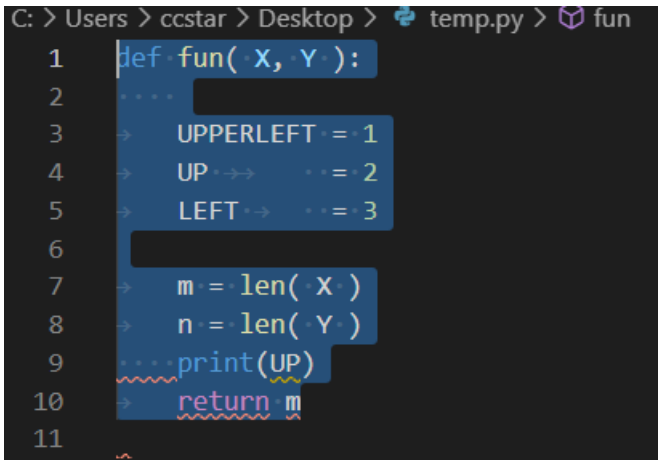
再來說說 python 的第二個特色:縮排。Python 不像其他常見的語言如 C、

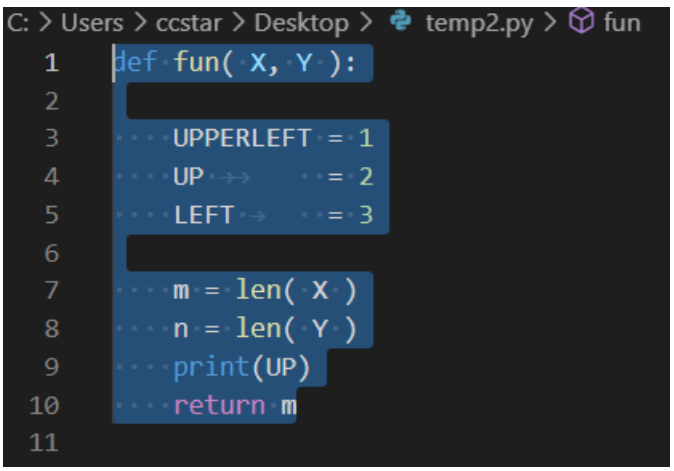
Java 的 function、迴圈或 if 使用中括號 “{ }” 來代表開頭以及結束，也可以說用中括號來表示他們的勢力範圍在哪裡，而 python 就是用縮排來表示勢力範圍在哪，還是一樣那句話，因為太過方便而讓人隨便，雖然是可以少打一點符號沒有錯，但是少了一點程式的可讀性，同樣地，批判總要舉個例子。以現在正在寫 PL project 來說，裡面用到了許多層的 if，假設說某個層狀結構的 if 有層這個多，同樣層級的有 10 多個，當在 debug 時，還要先確認說這個東西的勢力範圍在哪，在修程式碼的時候可能大半時間都會花在那上面。因為 python 是靠縮排來分辨勢力範圍的，所以一定要使用 tab 才行，不能用空白鍵去對其。再來我要來分享一下前陣子寫演算法的 project 遇到的不有趣事情，附上兩張圖片請仔細觀察這兩個圖的差別。





觀看 1~10 行以肉眼來看明明兩個程式碼都長一樣啊，程式碼是從別的地方複製過來並刪減過的，左邊是未經過加工的，右邊是有加工過的，那時候我就很納悶，XX 明明就複製過來的，我為什麼不能用 print 指令，怎麼想怎麼看都覺得程式碼沒有錯啊，哪裡有問題阿!沒問題阿!我程式能力沒這麼爛吧!好囉~~接下來要公布這兩個程式碼哪裡不一樣囉~~





如上面兩張圖，指令的前面長得不一樣，左邊那張指令的前面是個箭頭，只有print()前面是圓點，除了 print()指令外，其他都是原封不動複製過來的，而右邊那張圖指令的前面全部都是圓點，是程式碼複製完後，進行重新縮排。左邊那張因為縮排方式不一樣，所以 print()指令會有問題，右邊縮排方式相同，所以不會產生問題。這時，我心裡浮現出許多的 OS: XX，看起來不就沒問題，該對其的部分就對其啦，XX，真的不信有哪個神人可以一看就看到這個錯誤，氣死，浪費了一大堆的時間在找這個錯誤。以上就是我對 python 的縮排小小的分享。前面說了這麼多 Python 的小缺陷，現在來說說它的一些優點吧，因為 Python 是個 interpreter，所以當發生 bug 時，不會阻止前面程式碼的執行。再來，是我覺得 python 最強也最優秀的地方，就是 python 的套件非常的多，需要什麼東西，直接先 install 好 package，然後再 import 進來，接著程式碼就被濃縮成幾行而已。

接著來說說標記式語言吧，我認為標記式語言不算是個邏輯語言，它算是個裝飾語言，用來描述物件的表態，專門點綴用的，讓外表看起來比較漂亮一些。如果只有標記式語言寫出來的東西只能說是一張畫，一個在厲害再漂亮的靜態東西，還是比不過，稍微醜陋一點的動態效果，要有互動的效果存在還是必須仰賴後端的邏輯語言，如網頁需要 JavaScripe，Andriod App 需要 Java 一樣，應該說標記式語言適合與 O O 的環境做搭配，達到相輔相乘的效果。

我自己對蠻多領域都有多少涉略一些，如運動、遊戲、動漫……等，但說起要跟程式語言做結合的話，我會想要結合桌遊這一塊。桌遊其實在台灣發展有一段時間了，桌遊店也開了不少，可是網路上的線上桌遊似乎就沒有發展起來。想想現在疫情嚴重，是不是發展個線上桌遊是個好時機呢?於是乎我上網查了一下有沒有人在設計線上桌遊這一塊，發現了兩款線上桌遊平台，第一款是 steam 的 tabletop simulator，第二款是網頁遊玩的 Board Game Arena。
我認為線上桌遊在程式設計上最重要的一點是要把 OO 規畫得很好，再來就是要程式運作的 smoothly，線上的使用者會比一般聚會的玩家還要來的沒耐心，一個卡頓或畫面不順暢就會造成使用者不願意買單 。如果是我去設計線上桌遊的話，我會想用 Java 當作主體，畢竟 Java 是個很重物件導向的語言，然後去研究看看能不能把 C 跟 Java 做結合，因為 C 是程式語言中執行速度最快的，在相同的演算法下，C 基本上會是最優的，然後再結合一些標記式語言來美化一下美術效果。如果要我選擇第一款要設計的線上桌遊的話，我會想要選狼人殺。第一個原因，說起桌遊，一定沒有人不知道狼人殺這個經典之作，第二、目前市面上的網路狼人殺都採取語音限時模式，缺少了面殺的既視感以及完整的語言論述空間。第三、整體的遊玩規則簡單，物件需求不需要太多，應該不用花太多時間就可完成。第四、中原資工的學生在嘴砲的能力上需要訓練一番，也許我們的邏輯清晰，可以盤出許多論點出來，但大多時候就是缺少把腦

袋中的想法講出來以及說服別人的能力。在真正設計的時候除了基本規則外，把限定時間這個線上 APP 狼人殺都有的通病給拿掉，再來就是搭配攝影機，享受面殺的快感。寫到這時，我就想到何不在 google meet 的條件下，直接在上面外掛這個功能就好了呢!在現有的資源下，增加我們的需求在上面不就是最快的道路嗎?

如果要我設計一個語言出來的話，第一個要求就是一定要有型別及宣告，第二、一行就是一個指令，不用分號作結束(除了特殊的地方外 如:for 迴圈)。第三、變數名稱可以是中文。第四、這個語言需要有特殊的 IDE，這個 IDE 可以幫我想合適的變數名稱，給定一些對這個名稱的形容詞或特徵，IDE 就要生一個符合這個條件的名稱，因為有時候我會卡在不知道該如何取名變數名稱，然後這個 IDE 還要有具象化的功能，比如說在寫 Project1 和 Project2 時，它可以幫我把樹給畫出來，以及 stack、queue、vector 裡面每一步給畫出來，並搭配說這是第幾行的時候以及什麼資料，這樣在 debug 時可能可以輕鬆很多，如到後面會有許多 function，那這個 IDE 可以很有效地在目錄上分類，在，然後它可以幫我自動整理程式碼，在不影響程式的狀況下，可以幫我盡可能地照著目錄去做排序，第五、這個 IDE 以及語言可以像 word 一樣進行文字底色的變換，但儘管變數名稱相同但底色不同的話就是不同，舉例: int ball 和 int ball 是不同的，這樣變數名稱就可以取得很廣。第六、支援語音輸入，也支援語音註解，增加程式撰寫速度以及以後回頭看程式時可以馬上了解自己在寫什麼。第七、這個語言一邊寫的同時一邊就會開始除錯，提早告訴使用者說哪個部分可能會遇到問題。然後我這個理想的語言會想用在物件導向的程式設計上，有視覺化的關係，應該用起來會更加方便吧~~

近年來，科技進步迅速，人人學習程式設計成為一大潮流，就連教育體制都開始讓程式設計成為必學科目之一。老實說，我非常不認同把程式設計這門課納入 12 年國教的新課綱中，以我所看到的，選擇要讀資工系是自己的意願這是肯定的，但真正願意花時間去寫去學程式的又有多少呢?，這時讓教育強迫每個人都要去接觸程式，我認為多數的人這段學習的歷程是多餘的，也許學程式就跟自己的性向互斥，那為何還要花時間去學呢?科技這波潮流肯定會帶來許多的衝擊，我認為這波潮流確實可以促使全民寫程式，但是卻不是全民都可以學程式。其實，這個現象在網路上尋找資源的時候就看的出來，比如說我現在想要去學習一個新的程式語言好了，查到的前幾筆資料肯定都是罐頭式教學。你要安裝什麼 IDE，然後變數怎麼宣告，怎麼印出 hello word，迴圈怎麼使用，function 怎麼用……等，幾乎都是這些超級皮毛的解說。

最後來分享一下寫程式的心得，會寫程式真的不是短短幾天就會的東西，寫程式是需要花時間去堆疊出來的，需要一個人沉靜在自己的世界中，不斷的與自己對話，有些熱血番的動漫不是都會去講述說進到某個領域裡面嗎?如 網 x 王子、影 x 籃 x 員……等，我真的覺得是真的，一旦進入到那個狀態中，回

過神來，時間不知不覺就過得好快。每一堂課所給的 project 肯定都能從中學習到一些東西，比如要放個東西或查找東西，潛意識就會說:要不要用 hash Table，這個都已經快變成自然反應了。既然我的青春年華已經選擇了資工系這條路，也剛好乘上了這波科技風潮，就好好地把自己的時間用力的砸在程式上面。幹這一行不能不斷更新自己的能力與技術是不行的，很容易會被潮流快速的拋在腦後，況且人生就這麼的一次(life only once)，青春就這麼一段，一旦過了，就再也不能回頭去補救了。