語言學習模式類比於程式學習之批判性分析與初學者速成建議

撰文者: Sunny 給資策會學生的一篇文章

1. 緒論：語言學習與程式學習之類比——批判性考察

對於渴望快速掌握程式設計技能的初學者而言，將學習自然語言的模式與學習程式語言的模式進行類比，似乎具有直觀的吸引力。許多人認為，學習一門新的語言是一項具有挑戰性但可以實現的目標，因此，將程式設計與之類比，可以降低初學者對程式設計複雜性的恐懼感。然而，若不經批判性思考就直接將兩者等同，可能會導致學習策略上的誤導。本報告旨在深入剖析語言學習類比於程式學習的觀點，識別其優勢，更重要的是揭示其局限性。最終目標是結合研究支持的見解，為希望高效學習程式語言的初學者提供具體且可行的建議。

2. 熟悉的回聲：探索語言學習與程式學習之間的共通之處

儘管存在顯著差異，語言學習與程式學習在某些方面確實存在共通之處，理解這些共通點有助於初學者建立初步的學習框架。[1]

- 語法與文法：自然語言如英語或西班牙語有其文法規則，程式語言如Python、JavaScript和HTML也有其特定的語法。[1] 在程式設計中，遺漏一個分號或錯放一個括號可能會導致錯誤，正如句子中錯放一個詞可能會改變其含義一樣。[1] 這種相似性有助於初學者理解精確性和細節在程式設計中的重要性，如同文法錯誤會影響語意理解一樣。
- 詞彙：學習新的語言需要掌握新的詞語、短語和表達方式，同樣地，程式設計也需要學習專門的詞彙——學生需要學習諸如function、for、while和var等術語的正確用法。[1] 這種共通性表明，在語言學習中使用的詞彙記憶策略，例如使用字卡或間隔重複記憶法，或許可以應用於記憶程式設計的關鍵字和函數。
- 練習與重複：無論是語言還是程式設計，熟練掌握都需要持續的練習。[1] 正如孩子們重複單字或短語來內化一種語言一樣，他們也必須反覆編寫和除錯程式碼才能完全掌握程式設計的概念。[1] 這種共通性強調了主動參與和持續努力在兩種學習過程中的重要性。初學者應該被鼓勵定期編寫程式碼，而不僅僅是被動地閱讀相關資料。
- 問題解決能力：學習外語時，學生會面臨諸如造句或理解慣用語等挑戰。類似地，程式設計涉及透過編寫和除錯程式來解決問題。[1] 這兩種學科都能訓練大腦分析情況並提出解決方案。[1] 這種共通性突顯了學習程式設計不僅能獲得技術技能，還能培養分析和解決問題的能力。
- 模式識別：語言涉及句子結構、動詞變位和發音的模式。程式設計也高度依賴於識別演算法和資料結構中的模式。[1] 這種相似性暗示，擅長識別語言模式的學習者可能也具有識別程式碼模式的天賦，這可以在他們的學習過程中加以利用。
- 溝通工具：兩者都作為通用的溝通工具。語言連接不同文化背景的人們，而程式設計則是技術的全球語言，能夠與機器溝通並與世界各地的其他人協作。[1] 雖然溝通的對象不同，但理解這種基本目的可以為學習兩者提供背景和動力。初學者可以將程式設計

視為與電腦「交談」並指示它們執行任務的方式。
- 隨時間演變：自然語言和程式語言都會隨著使用者需求的變化而演變，新的詞彙或功能被引入，而舊的則可能不再使用。[3] 這意味著在兩個領域中，學習都是一個持續的過程。初學者應該為程式語言和技術的不斷發展做好持續學習和適應的準備。
- 創造力與表達：兩者都允許在其各自的框架內進行創造性表達。[3] 正如詩人可以玩弄韻律和節奏一樣，程式設計師可以編寫創意的腳本或有趣的專案，例如編寫遊戲或設計數位藝術。[3] 這種共通性表明，程式設計不僅僅是遵循嚴格的規則，也涉及尋找創新和優雅的解決方案。
- 社群支持：當在社群中進行學習時，學習通常更容易也更有效。[3] 無論是語言學習還是程式設計，社群的互動都能提供知識共享和支持。[3] 初學者應被鼓勵參與程式設計社群，無論線上或線下，以尋求幫助、分享學習經驗並向他人學習。
- 錯誤是過程的一部分：在語言學習和程式設計中，錯誤都是不可避免的，並且在學習和成長中扮演著至關重要的角色。[1] 初學者應該明白，犯錯並不可恥，而是學習和進步的機會。識別並糾正錯誤是兩個領域學習的基礎。

這些共通之處主要集中在語言和程式設計的結構、功能和認知方面。這種類比對於理解基礎概念和一般的學習過程可能有所幫助。透過認識這些共通點，初學者或許可以利用他們在自然語言學習中已有的學習策略來學習程式設計。例如，記憶詞彙的技巧、理解文法結構的方法或定期練習的習慣，都可能適用於程式設計的學習。

3. 分道揚鑣之處：揭示類比的局限性與誤解

儘管存在上述相似之處，但語言學習類比在程式學習中存在顯著的局限性，若不加以批判性思考，可能會導致初學者產生誤解並採取不適當的學習方法，阻礙其速成。[5]

- 溝通目的：自然語言的主要目的是人與人之間的溝通，通常涉及細微的語氣、情境和詮釋。相反，程式語言用於指示電腦，需要精確且明確的指令。[5] 這種根本性的差異意味著學習程式設計需要專注於邏輯精確和明確的指令，這與自然語言溝通中更直觀和依賴情境的方法不同。初學者需要理解，電腦根據嚴格的規則運作，無法從不完整或含糊不清的指令中推斷含義。
- 歧義程度：自然語言本質上包含歧義，依賴於情境和共同理解進行解釋。然而，程式語言的設計是為了消除歧義；任何歧義通常都會導致錯誤。[9] 這種根本性的差異暗示著學習程式設計需要轉向精確和清晰的思考方式，每個指令都必須是明確無誤的。
- 演算法思維與語文流利度：程式設計的核心挑戰通常在於培養演算法思維——將問題分解為邏輯步驟的能力——以及理解底層的計算概念。這與自然語言學習中對詞彙、文法和文化背景的關注點不同。[11] 初學者應該理解，雖然學習程式語言的語法是必要的，但真正的障礙和速成的關鍵在於培養計算思維和設計解決問題的邏輯步驟的能力。這涉及理解資料結構、演算法和控制流程等概念，而不僅僅是學習程式語言的「詞語」和「文法」。
- 回饋機制：程式設計中的回饋通常透過編譯器和錯誤訊息即時且直接地提供，清楚地

指出需要修正的地方。自然語言學習中的回饋可能較慢、不一致且更主觀。[9] 這種差異為程式設計的快速學習提供了顯著的優勢。初學者應該學會有效地利用程式設計中這些即時回饋機制來快速識別和糾正錯誤，從而加速他們的學習。

- 文化背景：自然語言與文化細微差別、慣用語和社會背景密切相關，而程式語言則在很大程度上缺乏這些。[8] 雖然可能存在程式設計範式或社群最佳實踐，但它們並不具有與人類語言相同的文化份量。初學者應該認識到，雖然理解程式設計社群和最佳實踐是有益的，但自然語言學習中那種深入的文化沉浸方面在程式設計中沒有直接的對應。程式設計的重點主要是邏輯和功能，而不是文化理解。
- 沒有口語和聽力成分：程式語言主要以書面形式存在；程式設計的核心任務不涉及顯著的口語或聽力成分。這與自然語言形成鮮明對比，在自然語言中，口語和聽力對於溝通至關重要。[12] 這對於某些可能覺得語言學習的口語方面具有挑戰性的初學者來說可能是一個優勢。然而，這也意味著學習程式設計很大程度上依賴於閱讀理解能力，以理解程式碼、文件和線上資源。
- 精通的可能性：雖然由於演變，兩個領域都需要持續學習，但有人認為，程式語言可能比不斷演變且範圍廣泛的自然語言更容易完全精通。[12] 初學者可能會覺得，雖然程式設計領域廣闊，但單個程式語言可能具有更明確的精通範圍，這可以為他們在學習過程中提供可實現的目標感。

語言學習類比在考慮溝通的基本目的和性質時顯得不足。程式設計是關於精確的指令執行，而自然語言是關於細微的人際互動。這種差異意味著學習策略應優先考慮程式設計中的邏輯思維、問題分解和對細節的關注，而不是像語言學習那樣僅僅關注詞彙和語法習得。此外，程式設計中透過編譯器和錯誤訊息提供的即時回饋迴路，相較於語言學習中經常延遲和主觀的回饋，為快速學習提供了顯著的優勢。初學者應學習有效地利用這些工具。

4. 程式設計速成原則：綜合有效的學習策略

基於對語言學習類比的分析以及從加速程式學習資源中獲得的見解，以下為旨在快速學習的初學者概述一套具體的原則。[13]

- 專注於基礎：在深入研究進階主題或框架之前，優先建立對核心程式設計概念的紮實理解。[13] 紮實的基礎知識，如變數、資料類型、控制結構和函數，為理解更複雜的主題提供了必要的基石，並防止初學者在以後的學習過程中迷失方向或養成不良習慣。
- 從實作中學習（專案式學習）：從一開始就強調主動編寫程式碼和建立實際專案，這是鞏固學習最有效的方法。[13] 透過建立專案來實際應用知識，比被動閱讀或觀看教學影片更能加深理解，並培養解決問題的能力。初學者應選擇自己真正感興趣的專案，以保持學習動力。
- 持續練習：規律的程式設計練習，即使時間不長，也比零星的長時間練習更有效，因為持續性是保持資訊和建立動力的關鍵。[1] 每天撥出專門的時間進行程式設計，有助於建立學習的慣性並防止遺忘先前學過的概念。
- 尋求協助並參與社群：遇到困難時不要猶豫尋求協助，並積極參與線上論壇或社群。[13]

向更有經驗的程式設計師或同儕學習者尋求指導，可以快速解決障礙並提供新的視角。

- 有效利用線上資源：充分利用豐富的免費和付費線上資源，包括教學影片、課程和文件。[13] 網路提供了大量的程式學習資源。初學者應探索不同的平台，找到最適合自己學習風格和步調的資源。學習導航和使用官方文件也是長期成功的關鍵技能。
- 盡早學習除錯：從一開始就培養除錯技能對於有效地識別和解決程式碼中的錯誤至關重要。[1] 除錯是程式設計過程中不可或缺的一部分。初學者應學習如何解釋錯誤訊息、使用除錯工具並系統地排除程式碼中的錯誤。將錯誤視為學習機會而非挫折的來源，對於速成至關重要。
- 先規劃再編碼：在編寫程式碼之前，特別是對於更複雜的任務，花時間思考問題並概述步驟或邏輯。[24] 在編寫程式碼之前花幾分鐘規劃程式的結構和流程，可以從長遠來看節省大量的時間和精力，並防止迷失方向或編寫效率低下的程式碼。
- 將錯誤視為學習機會：鼓勵初學者將錯誤和bug視為學習過程中自然的一部分，以及成長和更深入理解的寶貴機會。[1] 初學者應被鼓勵進行實驗、犯錯並從中學習。識別程式碼無法正常運作的原因並找到解決方案的過程，可以更深入地理解程式設計概念。
- 初期專注於一種語言：強烈建議初學者將最初的學習精力集中於精通一種程式語言，然後再嘗試同時學習多種語言。[25] 專注於單一語言有助於在核心程式設計概念上建立紮實的基礎，而不會因不同的語法和範式而感到困惑。一旦他們對一種語言有了紮實的理解，學習其他語言就會容易得多。

「從實作中學習」的原則對學習的速度和深度有直接的因果關係，因為它迫使學習者主動參與教材，從而更好地記住知識並培養對程式設計速成至關重要的實際問題解決能力。被動地閱讀或觀看教學影片無法像積極嘗試建立專案那樣吸引學習者。當初學者在進行專案時遇到問題時，他們會被迫應用所學的知識、研究解決方案並嘗試程式碼，這比僅僅記憶概念更能深入理解並更快地掌握技能。此外，強調尋求協助和參與程式設計社群表明，雖然程式設計學習通常是個人追求，但透過社交互動和協作解決問題可以顯著增強學習效果。這與語言習得的社交方面相似，並突顯了同儕支持和指導在實現速成方面的價值。當初學者遇到困難或感到不知所措時，與其他學習者和經驗豐富的程式設計師聯繫的能力可以提供指導、不同的視角和動力。這種社交支持網絡可以幫助更快地克服學習障礙，並在程式設計社群中培養歸屬感，鼓勵持續學習和進步。

5. 選擇正確的起點：為快速學習選擇適合初學者的程式語言

討論適合希望快速學習的初學者的程式語言，考慮的因素包括語法的易用性、學習資源的可用性、社群支持以及初始專案的多功能性。[37]

- **Python**：強調Python因其語法類似英語而廣泛認可的易讀性、簡化許多程式設計任務的豐富函式庫，以及其在網路開發、資料科學和腳本編寫等各個領域的廣泛應用，使其成為速成的絕佳入門語言。[15] Python清晰直觀的語法使初學者能夠專注於基本的程式設計概念，而不會被複雜或晦澀的語言規則所困擾。其龐大的函式庫和框架生態系

統為各種任務提供了現成的工具，使初學者能夠相對快速地建立有意義的專案。強大的社群支持也確保了新學習者可以獲得充足的資源和協助。

- **JavaScript**：討論JavaScript在前端網路開發中的關鍵作用、其在後端開發（Node.js）中日益增長的使用，以及其相對較低的入門門檻，特別是當與HTML和CSS結合用於視覺和互動專案時，可以為希望在網路相關領域速成的初學者提供即時的回饋。[1] JavaScript與瀏覽器的直接整合意味著初學者可以立即看到程式碼的執行結果，這可以極大地激發學習動力。其在網路開發中的廣泛使用也確保了大量的學習資源和強大的社群。雖然它有其複雜性，但JavaScript的初始步驟，特別是對於前端開發，通常被認為對初學者來說是容易上手的。
- **HTML 和 CSS**：簡要提及HTML（超文本標記語言）用於構建網頁內容，CSS（層疊樣式表）用於設定網頁樣式，作為經常與JavaScript一起學習的基礎技術，它們具有相對較淺的學習曲線，為初學者提供了快速進入視覺化網路開發的途徑。[1] 雖然從技術上講它們是標記和樣式語言，而不是完整的程式語言，但HTML和CSS對於任何對網路開發感興趣的人來說都是必不可少的。它們簡單的語法和即時的視覺回饋使它們成為初學者快速創建和設定網頁樣式的一個良好的起點，並在早期就能獲得成就感。

語言/技術	易學程度	主要應用	初始學習曲線
Python	非常容易	網路開發、資料科學、腳本編寫、自動化	淺
JavaScript	相對容易	前端和後端網路開發	中等
HTML	容易	構建網頁內容	非常淺
CSS	容易	設定網頁樣式	淺

多個資源一致推薦Python和JavaScript（通常與網路開發的HTML和CSS一起）作為初學者的理想入門語言，這強烈表明社群對其適合速成學習的共識，這歸因於它們對初學者友好的語法、豐富的資源和多功能性。選擇第一門程式語言可能會顯著影響初學者的動機和學習軌跡。從與他們興趣相符的語言開始（例如，對網路開發感興趣的選擇JavaScript，對通用程式設計感興趣的選擇Python），可以使學習過程更具吸引力，並提高他們堅持下去的可能性，最終實現更快的進步和更大的速成成功。

6. 從實作中學習：實務專案在加速技能發展中不可或缺的作用

強調實務專案在程式設計學習旅程中絕對關鍵的作用，詳細解釋它們如何促進更深入的理解並顯著加速旨在速成的初學者的技能習得。[29]

- 主動應用與更深層次的記憶：專案需要主動應用所學的概念，迫使初學者超越被動地吸收資訊，直接參與程式碼。這種主動參與比僅僅閱讀或觀看教學影片更能顯著地提升知識的記憶。[13] 當初學者進行專案時，他們不僅僅是孤立地記憶語法或概念，而是積極地使用這些元素來解決問題並建立具體的成果。這種應用過程以被動學習無法比擬的方式加強了他們的理解，從而產生更穩固和持久的知識。
- 培養必要的解決問題能力：程式設計從根本上來說就是解決問題。專案提供實際情境，要求初學者將複雜的問題分解為更小、更易於管理的步驟，設計邏輯解決方案，並將其以程式碼的形式實現。這種迭代過程對於培養程式設計熟練度核心的批判性思維和解決問題能力至關重要。[1] 教學影片通常提供逐步的指示，但專案需要學習者獨立思考並提出自己的解決方案。這培養了創造力、面對挑戰的韌性以及自信地處理新問題的能力。
- 建立展示技能的有形作品集：完成的專案，即使是簡單的專案，也為初學者的技能和能力提供了有形的證據。這個作品集在尋求回饋、申請實習或最終尋找該領域的工作時非常寶貴。它展示了超越理論知識的實務經驗。[15] 雇主和合作者更感興趣的是了解程式設計師實際能做什麼，而不僅僅是聽他們說學了什麼。專案作品集提供了初學者程式設計能力、解決問題的方法以及建立功能性應用程式的能力的具體範例。
- 提升參與度、動機和成就感：從事個人感興趣或解決實際問題的專案可以顯著提高初學者的參與度和動機。完成專案並看到它運作所帶來的滿足感提供了一種強大的成就感，激勵他們進一步學習並堅持克服挑戰。[15] 當初學者對他們的專案投入時，他們更有可能花費必要的時間和精力來學習所需的技能並克服障礙。一個可運行的專案的具體成果帶來了強烈的成就感，這加強了他們的學習並激勵他們處理更雄心勃勃的專案。

提供各種適合初學者的專案想法，按複雜程度和潛在的程式語言進行分類（例如，非常簡單：計算機、單位轉換器；中等：待辦事項清單、猜數字遊戲；稍複雜：基本網站、簡單的遊戲如井字遊戲）。參考提供的資料以獲取具體範例。[31] 提供具體且多樣化的專案想法有助於初學者克服「我該從哪裡開始？」的困境。按難度對其進行分類，使學習者能夠選擇適合其當前技能水平的專案，並隨著進步逐步增加複雜性。

解釋專案式學習通常如何反映真實世界的軟體開發情境，培養寶貴的技能，例如分解需求、規劃實施、管理範圍、測試，甚至在團隊合作時培養協作能力（強烈建議）。[29] 透過參與專案，初學者開始了解軟體開發生命週期，即使是在小規模上。他們學習思考需求、規劃方法、管理專案範圍、測試程式碼中的錯誤，以及在團隊合作的情況下進行協作，所有這些都是軟體開發職業的必要技能。

主動參與和專案式學習中固有的問題解決能力，導致程式設計技能的習得速度和深度遠超被動學習，使其成為旨在速成的初學者最有效的策略。當初學者積極從事專案時，他們會遇

到實際的挑戰，需要他們應用知識、研究解決方案並實驗程式碼。這種主動解決問題的過程不僅鞏固了他們的理解，還培養了批判性思維和除錯等關鍵技能，這些技能對於快速成為熟練的程式設計師至關重要。此外，從與個人興趣相關的專案開始不僅可以提高動機，還可以培養與學習材料更深層次的聯繫，使學習過程更具樂趣，並帶來更好的知識保留和更高的速成可能性。當初學者對他們正在進行的專案充滿熱情時，他們更有可能投入必要的時間和精力來克服挑戰並學習新技能。這種內在動機創造了一個更積極有效的學習環境，從而加快了他們的進度。

7. 避開雷區：避免早期程式學習中的常見錯誤和陷阱

基於對程式設計初學者常見錯誤的研究，找出可能嚴重阻礙進度的關鍵陷阱，並提供可行的策略來避免這些錯誤，從而加速旨在速成的初學者的學習過程。[24]

- 跳過基礎：在沒有紮實理解核心程式設計概念（如變數、資料類型、控制流程和函數）的情況下，就急於學習進階框架或嘗試建立複雜的應用程式。[24] 避免策略：強調在深入研究更專業的主題之前，花足夠的時間徹底學習和練習所選語言的基礎知識。鼓勵初學者完成基本練習和小型的程式設計挑戰，以鞏固他們對這些核心概念的理解。
- 沒有計畫就編碼：在沒有事先思考問題、概述步驟或規劃程式的整體結構的情況下，就直接開始編寫程式碼。[24] 避免策略：建議初學者在編寫任何程式碼之前花幾分鐘時間清楚地定義他們試圖解決的問題，並概述他們需要實施的邏輯步驟或演算法。對於更複雜的任務，建議使用虛擬碼或流程圖來幫助在編碼之前可視化解決方案。
- 不分解問題：試圖將大型複雜的問題作為一個單一的整體來處理，而不是將它們分解為更小、更易於管理且更容易解決和實施的子問題。[1] 避免策略：教導初學者將複雜問題分解為更小、獨立的部分的策略。鼓勵他們一次專注於解決一個子問題，然後將解決方案組合起來以解決整體問題。這種方法使編碼不那麼令人畏懼，也更易於管理。
- 編寫難以閱讀的程式碼：使用不明確的變數名稱、不一致的格式或忽略添加註釋來解釋程式碼的目的和邏輯。[24] 避免策略：指導初學者使用描述性且有意義的變數名稱、遵守一致的格式慣例（例如，使用程式碼格式化工具）以及添加註釋來解釋程式碼中複雜或不明顯的部分，從而強調編寫清晰易讀程式碼的重要性。強調程式碼被閱讀的次數比被編寫的次數更多。
- 複製貼上程式碼而不理解：盲目地複製貼上網路資源中的程式碼片段，而不花時間理解它們的工作原理或它們是否適合他們特定的問題。[24] 避免策略：建議初學者將任何借用的程式碼視為學習機會。鼓勵他們仔細檢查程式碼，嘗試理解其邏輯，並根據自己的特定需求進行調整。建議用他們自己的風格重寫程式碼以加強理解。
- 過度複雜化解決方案：初學者傾向於編寫過於複雜或冗餘的程式碼，而更簡單、更直接的方法也能達到相同的結果。[24] 避免策略：鼓勵初學者專注於程式碼的清晰性和簡潔性。建議他們尋求最直接且易於理解的問題解決方案，而不是試圖實施過於複雜或巧妙的程式碼。
- 忽略錯誤訊息：忽略或不理會編譯器或直譯器顯示的錯誤訊息，而不仔細閱讀並試圖理解它們的含義。[24] 避免策略：教導初學者將錯誤訊息視為寶貴的線索，這些線索提供

了有關程式碼中發生錯誤的具體資訊。鼓勵他們仔細閱讀錯誤訊息，必要時研究其含義，並使用它們來指導他們的除錯工作。

- 陷入「教學影片地獄」：花費過多時間被動地觀看教學影片或閱讀文件，而沒有主動編寫程式碼並透過建立自己的專案來應用所學知識。[24] 避免策略：建議初學者在從教學影片中學習和積極從事自己的專案之間取得平衡。鼓勵他們花更多時間編寫程式碼和進行實驗，而不是被動地消費內容。建議設定他們想要建立的具體目標，並根據需要使用教學影片來學習必要的技能。
- 試圖一次學習所有內容：嘗試同時學習多種程式語言、框架或技術，導致不知所措且對任何一個領域都沒有深入的理解。[25] 避免策略：強烈建議初學者將最初的學習精力集中於精通一種程式語言及其核心概念，然後再嘗試擴展到其他語言或技術。強調在一種語言上打下堅實的基礎將使以後學習其他語言更容易。
- 不定期測試程式碼：直到專案快結束時才測試程式碼，這可能導致大量累積的bug難以識別和修復。[26] 避免策略：鼓勵初學者養成在編寫程式碼時經常測試程式碼的習慣。建議編寫小段程式碼並立即測試，以確保它們按預期工作。介紹單元測試的概念，以便更系統地測試個別組件。

許多初學者常犯的錯誤都圍繞著缺乏主動參與學習過程以及傾向於被動地吸收資訊。這突顯了強調實作練習和專案式學習作為速成最有效策略的重要性。初學者經常陷入認為僅僅觀看教學影片或閱讀程式設計相關資料就足以學習的陷阱。然而，真正的理解和技能發展來自於透過編碼、實驗和解決問題來主動應用知識。認識並積極克服這些被動學習的傾向對於加速進度至關重要。避免這些常見的陷阱需要初學者對自己的學習過程有元認知意識，並有意識地採取更有效的學習習慣。這包括積極主動地規劃、尋求回饋、擁抱挑戰以及將錯誤視為成長的機會，這與教育中的自我調節學習和成長心態原則一致。透過了解阻礙初學者程式設計師的常見錯誤，學習者可以做出有意識的選擇，採取更有效的學習策略。這涉及主導自己的學習、注意自己的方法並積極尋找改進學習習慣的方法，最終實現更有效和成功的程式設計速成之路。

8. 結論：規劃有效率的程式學習之路

總結報告的主要見解，重申語言學習與程式學習之間微妙的關係。承認類比最初的直觀吸引力，但再次強調關鍵的差異，特別是在溝通目的和程式設計中演算法思維的重要性，這需要為速成量身定制的學習策略。重申初學者速成程式設計的核心原則：優先建立紮實的基礎、透過專案式工作主動學習、持續練習、不要猶豫尋求社群和資源的幫助、盡早培養除錯技能、在編碼前規劃、擁抱錯誤作為寶貴的學習機會，並將最初的精力集中於掌握一種像Python或JavaScript這樣對初學者友好的語言。再次強調實務專案在鞏固學習、培養關鍵的解決問題能力以及建立有形的作品集以展示其能力方面不可或缺的作用。強調選擇與他們的興趣相符的專案，以在整個學習過程中保持動力和參與度。最後，以鼓勵和賦予力量的信息結束，強調雖然學習程式設計需要奉獻精神、毅力和擁抱挑戰的意願，但透過遵循這些研究支持的原則並積極避免常見的陷阱，他們可以顯著加速他們的學習過程，並成功踏上

程式設計這個充滿回報的旅程，實現他們的速成目標。

reference

1. How Learning to Code Is Similar to Learning a Foreign Language ..., https://www.cococoders.com/resources-for-parents/how-learning-to-code-is-similar-to-learning-a-foreign-language
2. 7 ways learning to code is like learning a language - Educative.io, https://www.educative.io/blog/similarities-learning-language-coding
3. Programing Languages and Human Languages - Day Translations ..., https://www.daytranslations.com/blog/the-language-classroom-the-surprising-similarities-between-programming-languages-and-human-languages/
4. Bridging the Gap: How Learning a Foreign Language Can Enhance Your Programming Skills - Data Column | Institute for Advanced Analytics, https://datacolumn.iaa.ncsu.edu/blog/2024/02/28/bridging-the-gap-how-learning-a-foreign-language-can-enhance-your-programming-skills/
5. Are programming languages actually languages? - Lucid Software, https://lucid.co/techblog/2021/12/21/are-programming-languages-actually-languages
6. What's a good Programming Metaphor? [closed] - Software Engineering Stack Exchange, https://softwareengineering.stackexchange.com/questions/2410/whats-a-good-programming-metaphor
7. The Wrong Way to Learn How to Code | by Graham Sahagian | The ..., https://medium.com/swlh/learning-to-code-is-nothing-like-learning-a-foreign-language-3bfb7c29a340
8. Coding or Foreign Language: What Should Be Taught?, https://www.unitedlanguagegroup.com/learn/global-news/teaching-coding-foreign-language
9. Ask HN: Is coding like speaking another language? | Hacker News, https://news.ycombinator.com/item?id=15009377
10. What's more difficult, learning to program or learning a new spoken language? - Reddit, https://www.reddit.com/r/learnprogramming/comments/173fhnk/whats_more_difficult_learning_to_program_or/
11. Difference between learning programming and learning a language? - Reddit, https://www.reddit.com/r/learnprogramming/comments/18f92rc/difference_between_learning_programming_and/
12. Is learning how to program like learning a foreign language ..., https://latex-ninja.com/2019/07/28/is-learning-how-to-program-like-learning-a-foreign-language/
13. 7 Tips and Tricks to Learn Programming Faster - GeeksforGeeks, https://www.geeksforgeeks.org/7-tips-and-tricks-to-learn-programming-faster/
14. 7 Critical Tips to Learn Programming Faster - #3 Will Land You a Job - Coding Dojo, https://www.codingdojo.com/blog/7-tips-learn-programming-faster

15. How To Learn Programming From Scratch [2025 Guide] - Springboard, https://www.springboard.com/blog/software-engineering/how-to-learn-programming/
16. How to Learn to Code, Fast | Codementor, https://www.codementor.io/learn-programming/how-to-learn-to-code-fast
17. daily.dev, https://daily.dev/blog/beginners-guide-how-to-start-learning-coding-from-scratch#:~:text=Codecademy%3A%20Offers%20free%20courses%20across,HTML%2C%20CSS%2C%20and%20JavaScript.
18. Beginner's Guide: How to Start Learning Coding from Scratch - Daily.dev, https://daily.dev/blog/beginners-guide-how-to-start-learning-coding-from-scratch
19. Codecademy: Learn to Code - for Free, https://www.codecademy.com/
20. STEMscopes Coding Education - Accelerate Learning, https://acceleratelearning.com/technology/stemscopes-coding/
21. One Month: Learn to Code | Online Coding Courses, https://onemonth.com/
22. Accelerated Computing - Training | NVIDIA Developer, https://developer.nvidia.com/accelerated-computing-training
23. Amazon.com: Accelerated C++: Practical Programming by Example, https://www.amazon.com/Accelerated-C-Practical-Programming-Example/dp/0201703 53X
24. Common Mistakes Beginners Make in Coding and How to Avoid Them - DEV Community, https://dev.to/balrajola/common-mistakes-beginners-make-in-coding-and-how-to-avoid-them-4loc
25. Common Mistakes to Avoid When Learning to Code: A Comprehensive Guide, https://algocademy.com/blog/common-mistakes-to-avoid-when-learning-to-code-a-comprehensive-guide/
26. Beginner Coding Mistakes and How to Avoid Them - SkillReactor Blog, https://www.skillreactor.io/blog/beginner-coding-mistakes-and-how-to-avoid-them/
27. Mistakes to Avoid as a New Programmer: Common Pitfalls That Can Hinder Your Learning Progress - DEV Community, https://dev.to/scofieldidehen/mistakes-to-avoid-as-a-new-programmer-common-pitfalls-that-can-hinder-your-learning-progress-370o
28. Mistakes New Coders Make and How to Overcome Them - AlgoCademy, https://algocademy.com/blog/mistakes-new-coders-make-and-how-to-overcome-them/
29. Project-Based Learning: Real-World Coding Challenges, https://www.nucamp.co/blog/coding-bootcamp-job-hunting-projectbased-learning-realworld-coding-challenges
30. The Best Method To Learning Programming | Project-Based Learning | by Shaun Smerling, https://smerling.medium.com/the-best-method-to-learning-programming-project-based-learning-6ebaaffed22e

31. 11 Best Coding Projects for Newbies + Beginners - Codecademy, https://www.codecademy.com/resources/blog/coding-projects-for-beginners/
32. Why everyone tells you build projects | by Mukwende - Medium, https://medium.com/@Mukwende16/why-everyone-tells-you-build-projects-e7b23ae61c24
33. 13 Coding Projects and Programming Ideas for Beginners, https://www.springboard.com/blog/software-engineering/coding-project-ideas/
34. Top 10 Coding Projects For Beginners - GeeksforGeeks, https://www.geeksforgeeks.org/coding-projects-for-beginners/
35. Common Pitfalls to Avoid When Learning to Code | HackerNoon, https://hackernoon.com/common-pitfalls-to-avoid-when-learning-to-code
36. 7 Common Mistakes That Beginners Should Avoid While Learning to Code - Medium, https://medium.com/@anuupadhyay1994/7-common-mistakes-that-beginners-should-avoid-while-learning-to-code-b6f559062616
37. The 8 Easiest Programming Languages to Learn | App Academy, https://www.appacademy.io/blog/easy-programming-langauges-to-learn
38. Is there a 'wrong' way to learn programming? What was your biggest mistake? - Reddit, https://www.reddit.com/r/learnprogramming/comments/1fprcfi/is_there_a_wrong_way_to_learn_programming_what/
39. Mistakes Self-Taught Programmers Make When Learning to code (and how to avoid them), https://dev.to/juliafmorgado/mistakes-self-taught-programmers-make-when-learning-to-code-and-how-to-avoid-them-5hf5
40. 10 Hardest and Easiest Programming Languages in 2025 - GUVI Blogs, https://www.guvi.in/blog/easiest-programming-languages-to-hardest-ranked/
41. Easiest Programming Languages to Learn for Beginners | Coding ..., https://www.codingtemple.com/blog/easiest-programming-languages-to-learn/
42. The 11 Easiest Programming Languages To Learn According To Developers - Codecademy, https://www.codecademy.com/resources/blog/easiest-programming-languages-to-learn/
43. What is the best coding language to learn as a beginner?? - Career Village, https://www.careervillage.org/questions/810876/what-is-the-best-coding-language-to-learn-as-a-beginner
44. What's the Best Programming Language to Learn First? It Depends - SmartBear, https://smartbear.com/blog/best-programming-language-to-learn-first/
45. The Best Programming Languages To Learn First As A Beginner ..., https://zerotomastery.io/blog/best-beginner-programming-language/
46. Which Programming Language Should I Learn First in 2024? - SitePoint, https://www.sitepoint.com/which-programming-language-should-i-learn-first/
47. 9 Fun Coding Projects for Beginners | BestColleges, https://www.bestcolleges.com/bootcamps/guides/fun-coding-projects-beginners/

48. The Importance of Project-Based Learning Explained | Blogs - Royal Public Schools, https://royaltx.org/why-is-project-based-learning-important/
49. 7 Beginner Projects Every Aspiring Programmer Should Make - CodeOp, https://codeop.tech/blog/beginner-projects-for-programmers/
50. 10 Beginner Coding Projects for Easy Learning, https://www.codingdojo.com/blog/beginner-coding-projects
51. Top 10 Coding Projects for Beginners - Inspirit AI, https://www.inspiritai.com/blogs/ai-blog/coding-projects-for-beginners
52. Top 10 Beginner Programmer Mistakes and How to Fix them | by NALSengineering, https://medium.com/@NALSengineering/top-10-beginner-programmer-mistakes-and-how-to-fix-them-1e11c307cbc6
53. The 6 Mistakes I Made When Learning To Code (And How To Get Past Them), https://zerotomastery.io/blog/mistakes-i-made-when-learning-to-code/
54. www.skillreactor.io, https://www.skillreactor.io/blog/beginner-coding-mistakes-and-how-to-avoid-them/#:~:text=Take%20your%20time%20to%20master,more%20proficient%20and%20confident%20coder.
55. Learning Mistakes to Avoid As a Software Developer | carlos schults / blog, https://carlosschults.net/en/learning-mistakes