**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

🙞∙∙∙☼∙∙∙🙜



**BÀI TẬP LỚN MỞ RỘNG MÔN LẬP TRÌNH NÂNG CAO**

BÁO CÁO:

**LỚP L09 --- NHÓM 5 --- HK242**

**NGÀY NỘP 05/05/2025**

**Giảng viên hướng dẫn: ThS. Trương Tuấn Anh**

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực hiện** | **Mã số sinh viên** | **Điểm số** |
| Nguyễn Xuân Huy Hoàng | 2311070 |  |

**Thành phố Hồ Chí Minh – 202**

**MỤC LỤC**

**Trang**

[I. MỞ ĐẦU 3](#_Toc29156)

[II. NỘI DUNG 6](#_Toc30872)

[Chương 1. DÂN CHỦ VÀ DÂN CHỦ XÃ HỘI CHỦ NGHĨA 6](#_Toc13587)

[1.1. Dân chủ và sự ra đời, phát triển của dân chủ 6](#_Toc32234)

[1.1.1. Quan niệm về dân chủ 6](#_Toc26618)

[1.1.2. Sự ra đời và phát triển của dân chủ 8](#_Toc23996)

[1.2. Dân chủ xã hội chủ nghĩa 10](#_Toc7056)

[1.2.1. Quá trình ra đời của nền dân chủ xã hội chủ nghĩa 10](#_Toc5110)

[1.2.2. Bản chất của nền dân chủ xã hội chủ nghĩa 12](#_Toc10127)

[2.1. Thực trạng phát huy dân chủ xã hội chủ nghĩa ở Việt Nam thời gian qua 17](#_Toc32534)

[2.1.1. Những thành tựu đạt được trong việc phát huy dân chủ xã hội chủ nghĩa ở Việt Nam thời gian qua 17](#_Toc26494)

[2.1.2. Những thách thức hiện tại trong việc phát huy dân chủ xã hội chủ nghĩa ở Việt Nam 18](#_Toc12156)

[2.1.3. Quan điểm phê phán về việc phát huy dân chủ xã hội chủ nghĩa ở Việt Nam 19](#_Toc10575)

[2.2. Giải pháp tiếp tục phát huy dân chủ xã hội chủ nghĩa ở Việt Nam trong thời gian tới 20](#_Toc1570)

[2.2.1. Hoàn thiện hệ thống pháp luật và chính sách 20](#_Toc4534)

[2.2.2. Nâng cao nhận thức và sự tham gia của người dân 22](#_Toc17921)

[2.2.3. Tăng cường minh bạch và trách nhiệm giải trình trong quản lý nhà nước 24](#_Toc11385)

[2.3. Liên hệ phát huy vai trò của sinh viên ở Trường Đại học Bách khoa thành phố Hồ Chí Minh 26](#_Toc5556)

[2.3.1. Thực trạng vai trò của sinh viên trong việc thực hiện dân chủ tại trường 26](#_Toc20494)

[2.3.2. Các hình thức và cơ chế để sinh viên tham gia vào quá trình ra quyết định 35](#_Toc26857)

[2.3.3. Đề xuất giải pháp nâng cao vai trò của sinh viên trong việc phát huy dân chủ xã hội chủ nghĩa tại trường đại học 36](#_Toc2172)

[III. KẾT LUẬN 42](#_Toc30431)

[IV. TÀI LIỆU THAM KHẢO 46](#_Toc721)

# I. MỞ ĐẦU

**1. Tính cấp thiết của đề tài**

Dân chủ là một hình thức tổ chức nhà nước trong đó quyền lực thuộc về nhân dân, được thực hiện trực tiếp hoặc gián tiếp thông qua hệ thống đại diện. Dân chủ không chỉ là một hình thức tổ chức chính trị mà còn là một giá trị xã hội gắn liền với sự tiến bộ của nhân loại. Trong lịch sử phát triển của loài người, dân chủ đã trải qua nhiều hình thức khác nhau, từ nền dân chủ trực tiếp của Hy Lạp cổ đại đến nền dân chủ đại diện ngày nay. Sự phát triển của dân chủ gắn liền với quá trình đấu tranh vì quyền lợi và tự do của con người, với mục tiêu cuối cùng là xây dựng một xã hội công bằng, văn minh, nơi mỗi cá nhân đều có tiếng nói và quyền quyết định đối với các vấn đề chung của đất nước.

Dân chủ xã hội chủ nghĩa là loại hình dân chủ phát triển trên nền tảng chủ nghĩa Mác - Lênin, là hệ thống chính trị đảm bảo quốc gia vận hành theo nguyên tắc “*tất cả quyền lực nhà nước thuộc về nhân dân*”. Trong dân chủ xã hội chủ nghĩa, nhân dân không chỉ tham gia bầu cử mà còn có quyền giám sát, quyết định những vấn đề quan trọng của đất nước. Bản chất của dân chủ xã hội chủ nghĩa thể hiện ở sự kết hợp giữa quyền làm chủ của nhân dân với vai trò lãnh đạo của Đảng Cộng sản. Không giống như các nền dân chủ tư sản, nơi quyền lực có thể bị chi phối bởi các nhóm lợi ích, dân chủ xã hội chủ nghĩa hướng đến việc bảo đảm công bằng xã hội, xóa bỏ áp bức, bóc lột và thực hiện nguyên tắc “*của dân, do dân, vì dân*” một cách thực chất nhất.

Việt Nam thực hiện dân chủ xã hội chủ nghĩa trên cơ sở nhà nước pháp quyền xã hội chủ nghĩa dưới sự lãnh đạo của Đảng Cộng sản Việt Nam. Hiến pháp 2013 quy định “*Nhà nước Cộng hòa xã hội chủ nghĩa Việt Nam là nhà nước pháp quyền xã hội chủ nghĩa của nhân dân, do nhân dân, vì nhân dân*”. Quyền dân chủ được bảo đảm thông qua hệ thống chính trị, bầu cử quốc hội, hội đồng nhân dân các cấp, và những cơ chế giám sát chính trị. Các quyền dân sự, kinh tế, xã hội được bảo đảm qua các chính sách và pháp luật. Ngoài ra, sự phát triển của công nghệ thông tin và truyền thông cũng đóng vai trò quan trọng trong việc mở rộng không gian dân chủ, giúp người dân có thể tham gia đóng góp ý kiến, phản ánh và giám sát hoạt động của chính quyền một cách kịp thời và hiệu quả hơn.

Thời gian qua, việc phát huy dân chủ xã hội chủ nghĩa tại Việt Nam đã đạt được nhiều kết quả tích cực. Hội đồng nhân dân các cấp hoạt động hiệu quả, tăng cường giám sát quyền lực nhà nước. Kinh tế thị trường phát triển, nâng cao đời sống nhân dân. Hệ thống pháp luật ngày càng được hoàn thiện, tạo hành lang pháp lý vững chắc để bảo vệ quyền lợi của nhân dân. Các phong trào thi đua yêu nước, hoạt động của các tổ chức chính trị - xã hội cũng góp phần tạo điều kiện để người dân tham gia tích cực vào quá trình quản lý và phát triển đất nước.

Bên cạnh những thành tựu đạt được, vẫn còn một số hạn chế trong việc phát huy dân chủ xã hội chủ nghĩa tại Việt Nam. Quyền giám sát của nhân dân còn hạn chế do thủ tục hành chính rườm rà. Tình trạng quan liêu, tham nhũng vẫn tồn tại, gây ảnh hưởng đến niềm tin của người dân đối với bộ máy nhà nước. Nhiều quyền lợi của người lao động chưa được đảm bảo đầy đủ, đặc biệt trong các lĩnh vực bảo hiểm xã hội, tiền lương, điều kiện làm việc. Việc tiếp cận thông tin của người dân ở một số khu vực còn gặp khó khăn, ảnh hưởng đến sự tham gia của họ trong các hoạt động chính trị - xã hội. Do đó, để tiếp tục phát huy dân chủ xã hội chủ nghĩa một cách hiệu quả, cần có các biện pháp cải cách hành chính mạnh mẽ, nâng cao trách nhiệm của các cơ quan nhà nước, đồng thời tăng cường công khai, minh bạch trong quá trình ra quyết định và thực thi chính sách.

Xuất phát từ tình hình thực tế trên, nhóm chọn đề tài: “***Dân chủ và dân chủ xã hội chủ nghĩa. Phát huy dân chủ xã hội chủ nghĩa ở Việt Nam hiện nay***” để nghiên cứu.

**2. Đối tượng nghiên cứu**

*Thứ nhất,* dân chủ và dân chủ xã hội chủ nghĩa.

*Thứ hai,* phát huy dân chủ xã hội chủ nghĩa ở Việt Nam hiện nay.

**3. Phạm vi nghiên cứu**

Đề tài nghiên cứu phát huy dân chủ xã hội chủ nghĩa Việt Nam hiện nay.

**4. Mục tiêu nghiên cứu**

Mục tiêu nghiên cứu của đề tài:

*Thứ nhất*, làm rõ lý luận cơ bản của chủ nghĩa Mác-Lênin về dân chủ và dân chủ xã hội chủ nghĩa.

*Thứ hai,* đánh giá thực trạng phát huy dân chủ xã hội chủ nghĩa ở Việt Nam thời gian qua.

*Thứ ba,* đề xuất giải pháp tiếp tục phát huy dân chủ xã hội chủ nghĩa ở Việt Nam thời gian thời gian tới.

**5. Phương pháp nghiên cứu**

Đề tài sử dụng phương pháp nghiên cứu của chủ nghĩa duy vật biện chứng và chủ nghĩa duy vật lịch sử. Đồng thời, đề tài sử dụng kết hợp nhiều phương pháp nghiên cứu, trong đó chủ yếu nhất là các phương pháp: [phương pháp thu thập số liệu](https://winerp.vn/phuong-phap-nghien-cuu-khoa-hoc" \l "Phuong_phap_thu_thapso_lieu); [phương pháp phân tích và tổng hợp](https://winerp.vn/phuong-phap-nghien-cuu-khoa-hoc" \l "Phuong_phap_phan_tich_va_tong_thich_hopthuyet); [phương pháp lịch sử](https://winerp.vn/phuong-phap-nghien-cuu-khoa-hoc" \l "Phuong_phap_lich_su) - logic;…

**6. Kết cấu của đề tài**

Ngoài mục lục, phần mở đầu, kết luận và tài liệu tham khảo, đề tài gồm 2 chương:

Chương 1: Dân chủ và dân chủ xã hội chủ nghĩa.

Chương 2: Phát huy dân chủ xã hội chủ nghĩa Việt Nam hiện nay.

# II. NỘI DUNG

# Chương 1. LẬP TRÌNH HƯỚNG ĐỐI TƯƠNG TRONG JAVA VÀ RUBY

# Sơ lược

Lập trình hướng đối tượng (OOP) là một mô hình lập trình dựa trên khái niệm “*đối tượng*” – các thực thể có trạng thái (thuộc tính) và hành vi (phương thức). Java là ngôn ngữ lập trình bậc cao, an toàn về bộ nhớ và được thiết kế chủ yếu theo mô hình đối tượng (class-based OOP). Ruby cũng là ngôn ngữ lập trình hướng đối tượng thuần túy (pure OOP): mọi thứ trong Ruby đều là đối tượng, từ số, chuỗi đến cả lớp và module. Ruby hỗ trợ nhiều mô hình lập trình nhưng đặc biệt nhấn mạnh OOP; mọi giá trị được xử lý dưới dạng object và các đối tượng này đều có thể có phương thức riêng, ví dụ Ruby có sẵn object\_id cho mọi đối tượng.

# Lớp và đối tượng

Trong Java, lớp (class) được khai báo rõ ràng, ví dụ class Cat { ... }, với kiểu dữ liệu tĩnh và cú pháp tương tự C/C++. Mọi mã Java đều phải viết bên trong các lớp, và mỗi dữ liệu (trừ kiểu nguyên thủy như int, float…) là đối tượng. Ngược lại, trong Ruby lớp cũng được định nghĩa như sau:

class Cat

def speak

puts "Meo meo"

end

end

cat = Cat.new

cat.speak # => Meo meo

Trong Ruby, lớp cũng chính là đối tượng (instance của lớp Class). Ví dụ, Cat trong Ruby là một đối tượng, nên có thể bổ sung hoặc thay đổi phương thức một cách động. Ruby cho phép viết mã ngắn gọn hơn – không bắt buộc khai báo kiểu trả về hay kiểu tham số, và nếu muốn có thể bỏ cả dấu ngoặc khi gọi hàm. Trong Java, ngược lại, phải khai báo rõ kiểu biến và phương thức, ví dụ String name = “A”; name.toUpperCase(); và khi gọi phương thức phải thêm dấu ngoặc, ví dụ name.toUpperCase().

# Kế thừa (Inheritance)

Java chỉ cho phép kế thừa đơn giữa các lớp (mỗi lớp con chỉ được extends một lớp cha), ngoại trừ việc hỗ trợ đa kế thừa ở mức interfaces. Ví dụ Java:

class Animal { void speak() { System.out.println("Generic sound"); } }

class Dog extends Animal { void speak() { System.out.println("Woof"); } }

Mỗi lớp trong Java (ngoại trừ Object) có duy nhất một lớp cha trực tiếp. Ruby tương tự chỉ hỗ trợ kế thừa lớp đơn: class Dog < Animal để kế thừa từ Animal. Ruby không hỗ trợ đa kế thừa lớp nhưng có cơ chế mixin thông qua module để “hòa trộn” các tính năng (để chia sẻ mã giữa nhiều lớp). Ví dụ Ruby:

module Walkable

def walk; puts "Đi bộ"; end

end

class Animal; end

class Dog < Animal

include Walkable

def speak; puts "Gâu gâu"; end

end

Như vậy, cả Java và Ruby đều theo OOP cổ điển nhưng Ruby linh hoạt hơn về mặt kiểu dữ liệu (không cần khai báo kiểu, mọi thứ đều là đối tượng), còn Java chặt chẽ kiểu tĩnh.

# Đa hình (Polymorphism)

Đây là khả năng cho cùng một phương thức/lệnh được thực thi theo nhiều cách tuỳ thuộc vào đối tượng gọi. Java hỗ trợ đa hình tĩnh (nạp chồng phương thức – method overloading) và đa hình động (ghi đè phương thức – method overriding). Ví dụ, lớp con có thể ghi đè (@Override) phương thức của lớp cha để thay đổi hành vi, và quyết định phương thức cụ thể được gọi được xác định lúc chạy (dynamic dispatch). Java cũng có tính chất compile-time polymorphism (overloading) như ví dụ helper.multiply(int, int) và helper.multiply(double, double) trong tài liệu. Ruby mặc định hỗ trợ đa hình động nhờ duck typing: một đối tượng được xem “đa hình” nếu nó trả lời được lệnh (phương thức) được gọi. Ruby không kiểm tra kiểu tĩnh, miễn là đối tượng hiện tại có phương thức tương ứng thì lệnh gọi được thực thi. Chẳng hạn, hai lớp khác nhau nếu đều có phương thức speak, khi gọi obj.speak sẽ gọi phương thức tương ứng mà không cần quan tâm obj thuộc lớp nào. Cơ chế duck typing phát biểu rằng “nếu nó kêu được như vịt thì nó là vịt” – nghĩa là tập trung vào khả năng đối tượng làm được gì hơn là nó là lớp gì. Điều này khiến Ruby linh hoạt nhưng cũng tiềm ẩn rủi ro lỗi khi đối tượng không có phương thức.

Ví dụ minh hoạ: Sau đây là ví dụ so sánh định nghĩa lớp và kế thừa trong Java và Ruby:

- Java:

public class Animal {

public void speak() { System.out.println("Động vật kêu"); }

}

public class Dog extends Animal {

@Override

public void speak() { System.out.println("Chó sủa"); }

}

- Ruby:

class Animal

def speak; puts "Động vật kêu"; end

end

class Dog < Animal

def speak; puts "Chó sủa"; end

end

dog = Dog.new

dog.speak # => Chó sủa

# Tổng kết

Java và Ruby đều hỗ trợ OOP và chia sẻ các khái niệm cơ bản như lớp, đối tượng, kế thừa và đa hình. Tuy nhiên, Java là ngôn ngữ tĩnh kiểu (statically typed) và ép buộc khai báo kiểu biến cũng như cấu trúc gọn gàng với dấu ngoặc và phân vùng rõ ràng trong lớp. Ruby là ngôn ngữ động kiểu (dynamically typed) thuần OOP: mọi thứ đều là đối tượng, cú pháp đơn giản hơn (không cần khai báo kiểu, cú pháp thân thiện hơn). Java chỉ hỗ trợ kế thừa đơn và đa hình với overloading/overriding điển hình, trong khi Ruby hỗ trợ kế thừa đơn và triển khai đa kế thừa qua mixins, với cơ chế đa hình kiểu “*đáp ứng thông điệp*” (duck typing). Điểm khác biệt nổi bật là Java có kiểu dữ liệu nguyên thủy (primitive) không phải là đối tượng, còn Ruby coi mọi giá trị là đối tượng. Nhìn chung, OOP trong Java nghiêm ngặt hơn, Ruby linh hoạt hơn và có khả năng “*mở rộng*” mã bằng cách sửa đổi lớp tại thời gian chạy.

# Chương 2. LẬP TRÌNH HÀM TRONG HASKELL VÀ PYTHON

Lập trình hàm là mô hình lập trình coi hàm (function) là đơn vị cơ bản, nhấn mạnh tính bất biến (immutability) và hạn chế tác động phụ. Haskell là ngôn ngữ lập trình chỉ hàm (purely functional): mọi hàm trong Haskell đều là hàm thuần (pure function) – không thay đổi trạng thái bên ngoài và không có tác dụng phụ (side effect). Haskell áp dụng đánh giá trì hoãn (lazy evaluation), kiểu tĩnh và đa hình (polymorphic typing), cùng các tính năng như hàm bậc cao (higher-order functions), định nghĩa kiểu theo mẫu (pattern matching) và comprehension cho danh sách. Do hàm tính là bất biến, Haskell không có khái niệm vòng lặp truyền thống; các bài toán lặp được giải bằng đệ quy hoặc các hàm xử lý danh sách (map, fold, v.v.). Ví dụ, để tính dãy Fibonacci trong Haskell ta có thể định nghĩa:

fib 0 = 0

fib 1 = 1

fib n = fib (n-1) + fib (n-2)

Kết quả luôn xác định và không phụ thuộc vào trạng thái bên ngoài. Haskell hỗ trợ liệt kê danh sách và comprehension: ví dụ map (\*2) [1,2,3] trả về [2,4,6] nhờ hàm map áp dụng phép nhân cho mỗi phần tử.

Python không phải là ngôn ngữ hàm thuần, nhưng là ngôn ngữ đa mô hình (multi-paradigm): ngoài OOP và thủ tục, Python hỗ trợ lập trình hàm ở mức độ nhất định. Ví dụ, Python có các hàm bậc cao như map(), filter(), reduce(), biểu thức lambda và đặc biệt là comprehension để xử lý danh sách. So với Haskell, Python là động kiểu (dynamically typed) và cho phép sử dụng biến mutable (có thể thay đổi) như list và dict. Python ủng hộ phong cách hàm thông qua việc cho phép định nghĩa hàm trong hàm, hàm trả về hàm và sử dụng lambda, nhưng không bắt buộc tính thuần. Python có thể viết chương trình theo phong cách hàm (functional style), nhưng thông thường vẫn cho phép tác dụng phụ (ví dụ in ra màn hình hoặc thay đổi danh sách). Tài liệu Python nêu rõ rằng Python hỗ trợ nhiều cách tiếp cận, bao gồm cả lập trình hàm, nhưng “*các chương trình Python viết theo phong cách hàm thường sẽ không tránh hoàn toàn tất cả các tác dụng phụ*”.

# 2.1. Tính bất biến (immutability)​

Trong Haskell, tất cả giá trị là bất biến (immutable): một khi định nghĩa giá trị thì không bao giờ thay đổi. Python thì mặc định là mutable với các cấu trúc dữ liệu như list, dict. Tuy nhiên, Python có một số kiểu bất biến như tuple, string, số. Việc tránh side effect trong Python đòi hỏi quy ước của người lập trình. Tài liệu cho biết: “Hàm mà không có tác dụng phụ được gọi là hàm thuần (pure function). Việc tránh tác dụng phụ đồng nghĩa với không sử dụng cấu trúc dữ liệu bị cập nhật khi chạy chương trình”.

# 2.2. Đệ quy

Haskell dùng đệ quy rộng rãi thay thế cho vòng lặp, do không có cấu trúc lặp đơn như for/while. Mọi bài toán cần duyệt hoặc tính toán lặp đều có thể viết đệ quy hoặc dùng các hàm trừu tượng như foldr, zipWith. Python cũng hỗ trợ gọi đệ quy, nhưng không có tối ưu đuôi (tail-call optimization) nên đệ quy sâu dễ gây tràn ngăn xếp (stack overflow). Do đó, trong Python thường ưu tiên dùng vòng lặp hoặc comprehension. Ví dụ, phép tính giai thừa (factorial) có thể viết đệ quy trong cả hai ngôn ngữ:

- Haskell:

fact 0 = 1

fact n = n \* fact (n-1)

- Python:

def fact(n):

if n == 0: return 1

return n \* fact(n-1)

# 2.3. Hàm thuần (pure) và xử lý danh sách

Haskell ép kiểu thuần ngay từ ngữ nghĩa ngôn ngữ; mọi hàm không gây thay đổi trạng thái bên ngoài và các cạnh IO được cô lập (qua monad). Python không ép buộc thuần, nhưng có thể viết hàm thuần. Ví dụ, ta có thể dùng Python để tính tích của các phần tử trong danh sách mà không cần biến trung gian:

from functools import reduce

import operator

product = reduce(operator.mul, [1,2,3,4], 1) # = 24

Danh sách trong Haskell hỗ trợ các thao tác như comprehension (ví dụ [x\*2 | x <- [1,2,3]]) tương tự list comprehension của Python ([x\*2 for x in [1,2,3]]). Các hàm map, filter trong Python cũng tương đương các thao tác map/filter của Haskell, tuy Python trả về iterator và thường sử dụng list() để kết quả.

Ví dụ minh hoạ xử lý danh sách: Ví dụ chuyển đổi danh sách số:

- Haskell:

doubleList xs = [x\*2 | x <- xs]

-- doubleList [1,2,3] = [2,4,6]

- Python:

def double\_list(xs):

return [x\*2 for x in xs]

# double\_list([1,2,3]) = [2, 4, 6]

# 2.4. Tổng kết

Haskell và Python thể hiện hai cực trong phong cách hàm. Haskell là ngôn ngữ chức năng thuần túy, với tính bất biến mặc định và đệ quy là công cụ lặp chính, phù hợp cho các bài toán toán học và xử lý dữ liệu lớn theo kiểu hàm. Python tuy không phải ngôn ngữ thuần hàm, nhưng hỗ trợ một số yếu tố của lập trình hàm. Python linh hoạt hơn trong sử dụng nhiều mô hình khác nhau và phổ biến trong phát triển phần mềm đa mục đích. Tuy nhiên, Haskell dễ bảo đảm tính đúng đắn (không có tác dụng phụ) và tối ưu hoá chương trình hàm, trong khi Python thường thực hiện xử lý hàm ở mức sách lược (lambda, comprehension) mà vẫn có khả năng kết hợp với thao tác thay đổi trạng thái khi cần. Tóm lại, Haskell nổi trội trong lập trình thuần hàm và an toàn kiểu, còn Python là ngôn ngữ phổ quát, cung cấp nhiều tính năng hàm mà vẫn duy trì tính linh hoạt của ngôn ngữ động.

# Chương 3. CÁC HƯỚNG LẬP TRÌNH MỚI:

# DATA WRANGLING VÀ SMART CONTRACTS

# 3.1. Xử lý dữ liệu (Data Wrangly)

Định nghĩa: Xử lý dữ liệu (data wrangling) là quá trình thu thập, làm sạch và chuyển đổi dữ liệu thô (raw data) thành định dạng thích hợp để phân tích và trực quan hóa. Theo đó, dữ liệu sau khi wrangling sẽ dễ hiểu hơn, chất lượng tốt hơn và đáp ứng nhanh hơn cho mục đích ra quyết định. Quá trình này bao gồm các bước như chuẩn hoá tên cột, điền giá trị thiếu, loại bỏ dữ liệu nhiễu, chuyển đổi kiểu dữ liệu, kết hợp nhiều nguồn dữ liệu, v.v. Mục tiêu là chuẩn bị dữ liệu sao cho sạch, đồng nhất và sẵn sàng cho phân tích (bao gồm thống kê, học máy, báo cáo).

Công nghệ và ngôn ngữ: Ngôn ngữ và thư viện phổ biến cho data wrangling là Python và R:

- Python: Với thư viện mạnh như Pandas (xử lý bảng dữ liệu), NumPy (tính toán số học) và OpenRefine (công cụ GUI), Python rất phù hợp xử lý và biến đổi tập dữ liệu lớn. Ví dụ, Pandas cung cấp các hàm dropna(), merge(), groupby(), giúp làm sạch và gộp dữ liệu đơn giản.

- R: Thư viện tidyverse (như dplyr, tidyr) cung cấp các hàm filter(), select(), group\_by(), summarise(),… rất mạnh trong phân tích và tổng hợp dữ liệu. R được ưa chuộng trong cộng đồng thống kê và học thuật do cú pháp ngắn gọn cho xử lý bảng.

- SQL: Ngôn ngữ truy vấn cơ sở dữ liệu SQL cũng thường được dùng để wrangle dữ liệu ngay tại nguồn, đặc biệt với dữ liệu trong kho SQL. Người làm dữ liệu dùng các câu lệnh SELECT, WHERE, JOIN, GROUP BY để trích xuất và xử lý dữ liệu trước khi đưa vào Python/R.

- Công cụ khác: Ngoài ra còn có các công cụ ETL (Extract-Transform-Load) và nền tảng tích hợp như Apache Spark (xử lý dữ liệu phân tán), Apache NiFi, Microsoft Power BI, Alteryx,… hỗ trợ xử lý dữ liệu số lượng lớn một cách tự động hoá.

Ví dụ minh hoạ: Giả sử có tập tin CSV chứa thông tin bán hàng, ta có thể dùng Python (Pandas) để đọc và làm sạch:

import pandas as pd

df = pd.read\_csv('sales.csv') # Đọc dữ liệu từ file

df = df.dropna() # Xóa các dòng có giá trị thiếu

df['Date'] = pd.to\_datetime(df['Date']) # Chuyển cột ngày về kiểu DateTime

df\_grouped = df.groupby('Region')['Revenue'].sum() # Tổng doanh thu theo khu vực

Đoạn mã trên sẽ trả về tổng doanh thu (sum()) cho mỗi Region, sau khi đã xoá những dòng thiếu dữ liệu và chuẩn hoá định dạng ngày. Tương tự, bằng R/dplyr ta có thể viết:

library(dplyr)

df <- read.csv("sales.csv") %>%

filter(!is.na(Revenue)) %>% # Loại bỏ giá trị NA

mutate(Date = as.Date(Date, "%Y-%m-%d")) %>%

group\_by(Region) %>%

summarise(TotalRevenue = sum(Revenue))

Nhận xét: Data wrangling là bước thiết yếu trong khoa học dữ liệu và phân tích thông tin. Các kỹ năng như làm sạch, hợp nhất dữ liệu, chuyển đổi định dạng được hỗ trợ mạnh mẽ bởi công cụ và thư viện trên Python và R. Thực tế ứng dụng rộng rãi: từ phân tích kinh doanh (BI), báo cáo (ví dụ Excel, Power BI), đến tiền xử lý cho máy học và trí tuệ nhân tạo. Một điểm cần lưu ý là data wrangling thường tiêu tốn nhiều thời gian trong dự án, nên sử dụng công cụ phù hợp (code hay GUI) để tối ưu hoá công việc.

# 3.2. Smart Contracts

Định nghĩa: Hợp đồng thông minh là chương trình máy tính được triển khai và thực thi trên nền tảng blockchain, tự động thi hành các điều khoản khi các điều kiện xác định được thoả mãn. Smart contract cho phép nhiều bên thực hiện giao dịch một cách giá trị ràng buộc và không cần bên trung gian, đảm bảo tính minh bạch và không thể thay đổi sau khi được ghi lên blockchain. Tính “*thông minh*” ở đây là tự động hoá quy trình: ví dụ, khi một hợp đồng vay tiền nhận đủ xác nhận từ các bên, nó có thể tự giải ngân và ghi nhận trên chuỗi mà không cần ngân hàng hay luật sư can thiệp.

Ngôn ngữ và công nghệ:

- Ethereum / Solidity: Ethereum là nền tảng điển hình cho smart contract; ngôn ngữ chính là Solidity – một ngôn ngữ kiểu C/JavaScript dành riêng để viết hợp đồng trên Máy ảo Ethereum (EVM). Solidity hỗ trợ kiểu tĩnh, cú pháp tương tự JS và có sẵn nhiều tính năng phù hợp viết contract.

- Vyper: Là ngôn ngữ thứ hai phổ biến cho EVM, có cú pháp giống Python và nhấn mạnh tính an toàn (booleans, bignum…). Vyper loại bỏ các tính năng phức tạp (như kế thừa, vòng lặp) để giảm lỗ hổng bảo mật.

- Rust: Các blockchain mới như Solana, Polkadot sử dụng ngôn ngữ Rust để viết hợp đồng (WebAssembly). Rust được đánh giá bảo mật cao, hiệu quả và hỗ trợ viết smart contract nhẹ và nhanh. Ví dụ, trên Solana developer dùng Rust viết “*program*” (smart contract) hoạt động trên chuỗi.

- Ngôn ngữ khác: Một số blockchain còn sử dụng Go hoặc Java (ví dụ Hyperledger Fabric cho “*chaincode*”), Haskell (Cardano có Plutus viết bằng Haskell), hoặc ngôn ngữ riêng như Move (Meta Libra).

- Công nghệ nền: Ngoài Ethereum, có các nền tảng blockchain hỗ trợ smart contract như Binance Smart Chain, Avalanche (EVM), Algorand, Near… Người phát triển dùng IDE và ví, ví dụ Remix IDE, Truffle framework, Hardhat, cùng gói thư viện tương ứng cho từng blockchain.

Ứng dụng thực tế: Smart contract đang được ứng dụng nhiều trong DeFi (tài chính phi tập trung): vay mượn, cho vay, trao đổi tiền điện tử tự động (Uniswap, Aave, MakerDAO,…), phát hành token & NFT (non-fungible tokens), bảo hiểm tự động, chuỗi cung ứng truy xuất nguồn gốc, bầu cử điện tử, v.v. Những “hợp đồng” này đảm bảo quyền lợi bằng toán học và blockchain, không cần bên thứ ba như ngân hàng hoặc tổ chức trung gian. Theo Chainlink, các use case điển hình bao gồm từ tài chính (DeFi), game tài chính ảo, NFT đến cung cấp tính đảm bảo (trust-minimized guarantees) và quyền sở hữu tài sản số cho người dùng. Smart contract giúp tạo ra “*ngành tài chính minh bạch hơn, những hợp đồng bất biến và quyền sở hữu kỹ thuật số*”.

Ví dụ minh hoạ: Dưới đây là ví dụ cơ bản về hợp đồng thông minh bằng Solidity (Ethereum):

pragma solidity ^0.8.0;

contract HelloWorld {

string public greet = "Xin chao, the gioi!";

}

Hợp đồng này lưu trữ một chuỗi và cho phép truy vấn từ bên ngoài. Một khi triển khai lên blockchain, mã này không thể thay đổi và mọi người đều có thể truy vấn greet. Cơ chế tự động hoá của smart contract thể hiện: khi có ai tương tác gửi giao dịch (ví dụ cập nhật biến), nó sẽ chạy code này một cách bất biến.

Nhận xét tổng kết: Smart contract đại diện cho xu hướng mới trong phát triển ứng dụng: lập trình tập trung vào logic giao dịch cho blockchain. Ngôn ngữ phát triển contract thường là ngôn ngữ thiết kế riêng cho blockchain (như Solidity, Vyper) hoặc ngôn ngữ cấp cao có biên dịch xuống VM (Rust, Go, v.v.). Công nghệ này mở ra nhiều ứng dụng đột phá trong các lĩnh vực tài chính, pháp lý, logistics, góp phần tạo hệ thống phi tập trung, giảm chi phí trung gian và tăng tính minh bạch. Tuy nhiên, lập trình smart contract yêu cầu chú ý bảo mật cực cao (do mã sau deploy không thể sửa đổi) và thường bị giới hạn các tính năng của ngôn ngữ để tránh rủi ro (ví dụ Vyper thiếu vòng lặp).

# Chương 4. NGÔN NGỮ LẬP TRÌNH GO (GOLANG)

# 4.1. Lịch sử phát triển

Go (còn gọi là Golang) là ngôn ngữ lập trình mã nguồn mở được phát triển bởi Google, do Robert Griesemer, Rob Pike và Ken Thompson tạo ra từ năm 2007 và công bố chính thức năm 2009. Ngôn ngữ này được thiết kế với mục tiêu đơn giản, hiệu quả, và hỗ trợ tốt cho xây dựng phần mềm quy mô lớn. Phiên bản 1.0 của Go được phát hành vào năm 2012 và kể từ đó liên tục được cập nhật (hiện đã có các phiên bản 1.18, 1.19… với bổ sung như generic năm 2022). Go hướng tới giải quyết nhu cầu của lập trình hệ thống và phát triển mạng (network), với cú pháp gọn gàng, hiệu năng cao và thư viện phong phú.

# 4.2. Đặc điểm chính

Kiểu tĩnh, biên dịch nhanh: Go là ngôn ngữ tĩnh kiểu (static typing) và biên dịch (compiled) – mỗi chương trình Go được biên dịch thành mã máy nhanh chóng. Cú pháp Go chịu ảnh hưởng của C nhưng được rút gọn, loại bỏ nhiều yếu tố phức tạp (không có Generic trước 2022, không có kế thừa lớp, sử dụng interface định dạng động). Go hỗ trợ suy diễn kiểu (type inference) ở một mức giới hạn (dùng :=), giúp viết ngắn gọn hơn nhưng vẫn đảm bảo an toàn kiểu tại biên dịch.

Hỗ trợ concurrency: Một điểm nổi bật của Go là cơ chế xử lý đồng thời đơn giản nhưng mạnh mẽ. Go có goroutine – các tác vụ đồng thời nhẹ chi phí thấp – và kênh (channel) để giao tiếp giữa các goroutine. Ví dụ, ta chỉ cần dùng từ khoá go trước lời gọi hàm để chạy nó trên goroutine mới. Go được thiết kế để dễ dàng viết code đồng thời mà tránh được rắc rối quản lý luồng (thread) thủ công. Điều này giúp Go rất phù hợp cho các ứng dụng mạng, dịch vụ web song song, hệ thống phân tán, nơi cần xử lý nhiều tác vụ cùng lúc với hiệu năng cao.

Quản lý bộ nhớ: Go tích hợp sẵn garbage collector (GC) để tự động giải phóng bộ nhớ không dùng nữa, giảm thiểu lỗi tràn bộ nhớ so với C/C++. Việc quản lý bộ nhớ tự động giúp tập trung vào logic nghiệp vụ mà không lo quá nhiều về leak bộ nhớ.

Thư viện tiêu chuẩn và công cụ: Go có thư viện chuẩn phong phú (net/http, encoding/json, database/sql, v.v.), công cụ go fmt tự động định dạng mã, hệ thống gói (package) dễ sử dụng và công cụ quản lý dependency (go.mod). Ngôn ngữ khuyến khích code gọn gàng, dễ đọc và tối giản cú pháp (loại bỏ biểu tượng “;” ở cuối dòng, coi newline là kết thúc lệnh, v.v.).

Ứng dụng: Nhờ các đặc điểm trên, Go được sử dụng rộng rãi để xây dựng dịch vụ web hiệu năng cao, microservices, công cụ DevOps, hệ thống Cloud (Docker, Kubernetes viết chủ yếu bằng Go), các ứng dụng yêu cầu quy mô lớn và khả năng mở rộng cao.

Ví dụ chương trình cơ bản: Dưới đây là ví dụ in Hello, World! bằng Go:

package main

import "fmt"

func main() {

fmt.Println("Hello, World!")

}

Giải thích: package main định nghĩa gói chính, hàm main() là điểm vào (entry point). fmt.Println in chuỗi ra đầu ra chuẩn. Khi biên dịch và chạy, chương trình sẽ hiển thị Hello, World!. Ví dụ này cho thấy cú pháp Go tương tự C nhưng không cần dấu chấm phẩy và có sẵn thư viện fmt in/uốn chuỗi.

# 4.3. Tổng kết

Go (Golang) là ngôn ngữ hướng vào sự đơn giản và hiệu suất. Các đặc điểm nổi bật là tĩnh kiểu, đa luồng nhẹ nhàng (goroutines), và thư viện mạng mạnh. Không có một số tính năng phức tạp như generic (đến Go1.18 mới có), kế thừa lớp, hoặc overloading làm code dễ đọc hơn. Go phù hợp cho lập trình hệ thống, web services, và những ứng dụng cần xử lý đồng thời cao. Ngành CNTT ngày nay sử dụng Go nhiều trong môi trường đám mây và phân tán; ví dụ Docker, Kubernetes đều viết bằng Go. Với cú pháp đơn giản và khả năng biên dịch nhanh, Go hỗ trợ người lập trình tập trung vào nghiệp vụ chính mà không bị ràng buộc bởi chi tiết máy móc. Theo GeeksforGeeks, Go được phát triển tại Google năm 2009 để tạo ra ngôn ngữ mới “*đơn giản, hiệu quả và dễ học*” cho các hệ thống quy mô lớn.

# Chương 5. LẬP TRÌNH TETRIS IN TERMINAL BẰNG C++

# 5.1. Giới thiệu

Game Tetris được xây dựng để chạy trong terminal, cho phép người chơi điều khiển các khối rơi (Tetromino) bằng bàn phím. Dự án được tổ chức rõ ràng theo mô hình hướng đối tượng (OOP), chia thành nhiều lớp và file nguồn.

project-root/

├── include/ # Thư mục chứa các file header (khai báo lớp)

│ ├── Board.h # Lớp quản lý bảng trò chơi

│ ├── ConstVariable.h # Các hằng số (chiều cao, chiều rộng,sắc,...)

│ ├── main.h # Khai báo hàm main và các tiện ích

│ ├── Tetris.h # Lớp điều phối trò chơi (logic chính)

│ └── Tetromino.h # Lớp đại diện cho các khối tetromino

├── src/ # Thư mục chứa các file hiện thực

│ ├── Board.cpp

│ ├── Tetris.cpp

│ └── Tetromino.cpp

├── main.cpp # Điểm bắt đầu chương trình, tạo và chạy game

├── tetris.exe # File thực thi (tạo sau khi biên dịch)

├── run.bat # Script chạy trên Windows để biên dịch game

Link github: (Có video demo)

# 5.2. Cách cài đặt

## 5.2.1. Yêu cầu hệ thống

- Trình biên dịch hỗ trợ g++

- Terminal hỗ trợ ANSI escape code (màu sắc, ký tự)

## 5.2.2. Cách cài đặt và chạy

- Clone hoặc tải project (trường hợp chưa có sẵn mã nguồn): có thể tải toàn bộ dự án về máy (dưới dạng .zip) hoặc dùng lệnh:

git clone <link-git>

cd <tên-folder>

- Biên dịch: Đối với Windows (Command Prompt hoặc PowerShell) chạy file run.bat. Đối với hệ điều hành Linux do thư viện #include<conio.h> chỉ hỗ trợ trên Windows có thể dùng file tetris.exe đã được chạy sẵn.

# 5.3. Cách chơi

- Chạy file tetris.exe và trò chơi sẽ yêu cầu người dùng nhập tên (số ký tự phải khác 0 và không quá 10).

- Các phím điều khiển

|  |  |
| --- | --- |
| Phím nhấn | Hành động |
| A, a, hoặc ◄(phím mũi tên qua trái) | Di chuyển khối sang **trái** |
| D, d, hoặc ►(phím mũi tên qua phải) | Di chuyển khối sang **phải** |
| S, s, hoặc ▼ | Làm khối **rơi** nhanh hơn (rơi mềm) |
| W, w, hoặc ▲ | **Xoay** khối (xoay 90 độ) |
| SPACE (phím cách) | **Thả** nhanh khối (rơi tức thì) |
| P, p | **Tạm dừng** trò chơi |
| R, r | **Tiếp tục** trò chơi sau khi dừng |

- Mục tiêu

+ Di chuyển và xoay các khối sao cho chúng xếp đầy một hàng ngang.

+ Khi một hàng được lấp đầy, nó sẽ biến mất và bạn sẽ được cộng điểm.

+ Đạt điểm càng nhiều càng tốt, trò chơi kết thúc khi xảy ra tràn số.

# 5.4. Giải thích code

## 5.4.1. ConstVariable.h:

#ifndef CONSTVARIABLE\_H

#define CONSTVARIABLE\_H

#include "main.h"

using namespace std;

const int WIDTH = 20;

const int HEIGHT = 20;

#endif

Đoạn code trên là file header định nghĩa các hằng số WIDTH và HEIGHT dùng cho trò chơi Tetris (kích thước bảng là 20x20).

## 5.4.2. Tetromino.h

#ifndef TETROMINO\_H

#define TETROMINO\_H

#include "ConstVariable.h"

enum TetrominoType { I, O, T, L, J, S, Z };

class Tetromino {

private:

VectorMatrix shape;

int color;

int x;

int y;

public:

Tetromino();

int getColor() const;

void setColor(int color);

int getX() const;

void setX(int x);

int getY() const;

void setY(int y);

VectorMatrix &getShape();

void setShape(VectorMatrix update);

};

#endif

- Enum TetrominoType { I, O, T, L, J, S, Z };: Liệt kê các loại khối Tetris.

- Class Tetromino: Định nghĩa lớp chứa:

+ shape: Ma trận biểu diễn hình dạng khối.

+ color: Màu sắc của khối.

+ x, y: Tọa độ hiện tại của khối trên bảng (x là cột, y là hàng).

- Các hàm thành viên

+ Constructor: Đặt vị trí khởi đầu của khối tại giữa đỉnh bảng.

+ get/setX, get/setY: Lấy hoặc cập nhật vị trí.

+ get/setColor: Lấy hoặc đặt màu sắc.

+ get/setShape: Lấy hoặc cập nhật hình dạng khối.

## 5.4.3. Board.h

#ifndef BOARD\_H

#define BOARD\_H

#include "ConstVariable.h"

class Board {

private:

VectorMatrix board;

int row;

int col;

public:

Board();

void setBoardXY(int y, int x, int val);

int getBoardXY(int y, int x);

void setBoard(VectorMatrix tmp);

void RemoveRow(int k);

void resetRow(int k);

VectorMatrix &getBoard();

};

#endif

Lớp Board đại diện cho bảng trò chơi Tetris – nơi các khối Tetromino di chuyển và rơi xuống.

Thành phần dữ liệu:

- VectorMatrix board: Ma trận 2D lưu trạng thái của bảng chơi, bao gồm viền và khối.

- row, col: Số dòng và cột của bảng (bao gồm viền, nên là HEIGHT + 2, WIDTH + 2).

Các phương thức:

- Board(): Hàm khởi tạo, tạo bảng chơi với giá trị ban đầu là 0.

- setBoardXY(int x, int y, int val): Gán giá trị val vào vị trí (x, y) trên bảng.

- getBoardXY(int x, int y): Trả về giá trị tại vị trí (x, y) (0 nếu ngoài phạm vi).

- setBoard(VectorMatrix tmp): Gán toàn bộ bảng bằng một ma trận mới.

- RemoveRow(int k): Di chuyển dòng trên xuống dòng k (để làm hiệu ứng xóa hàng).

- resetRow(int k): Xóa toàn bộ dòng k (đặt tất cả ô về 0).

- getBoard(): Trả về tham chiếu đến ma trận bảng để truy cập trực tiếp.

## 5.4.4. Tetris.h

#ifndef TETRIS\_H

#define TETRIS\_H

#include "Tetromino.h"

#include "Board.h"

class Tetris {

private:

Board gameBoard;

Tetromino current;

Tetromino nextT;

string warning = "\033[36mYour name should not have more than 10 characters\033[0m";

int score = 0;

int updateScoreSuccess = 0;

bool paused = 0;

string name = "";

string hello = "Hello ";

int nameIter = 5;

int width\_score = 16;

public:

Tetris();

void Playing();

void InsertName();

void spawnTetromino();

void spawnNextTetromino();

void handleInput();

void merge();

void handlePause();

bool collides(int x, int y);

void clearLines();

void clearInputBuffer();

void rotate();

string centerWithColor(const string& content, int totalWidth);

void outsider(int i);

void draw();

};

#endif

Lớp này điều phối toàn bộ logic game:

- Điều khiển khối Tetromino hiện tại và kế tiếp.

- Kiểm tra va chạm, gộp khối, xóa dòng.

- Xử lý đầu vào từ người chơi.

- Vẽ game trên terminal.

Giải thích các hàm

|  |  |
| --- | --- |
| Hàm | Mục đích |
| Tetris() | Khởi tạo game, bảng, khối hiện tại/kế tiếp. |
| Playing() | Hàm chính để chơi game, thực hiện vòng lặp cập nhật màn hình, xử lý đầu vào, logic rơi khối. |
| InsertName() | Nhập tên người chơi, kiểm tra độ dài và hiển thị cảnh báo nếu cần. |
| spawnTetromino() | Tạo khối đầu tiên cho trò chơi. |
| spawnNextTetromino() | Tạo khối tiếp theo (hiển thị trong khung phụ bên cạnh). |
| handleInput() | Xử lý các phím bấm điều khiển (trái, phải, xoay, tạm dừng,...). |
| merge() | Gộp khối hiện tại vào bảng khi chạm đáy hoặc va chạm. |
| handlePause() | Dừng hoặc tiếp tục game khi người chơi nhấn p / r. |
| collides(int x, int y) | Kiểm tra nếu khối sẽ va chạm nếu di chuyển thêm (x, y). |
| clearLines() | Xóa các dòng đã đầy và cập nhật điểm. |
| clearInputBuffer() | Xóa các phím dư bị người chơi giữ lâu. |
| rotate() | Xoay khối hiện tại nếu có thể. |
| centerWithColor() | Căn giữa và tô màu chuỗi để hiển thị ở khung bên cạnh. |
| outsider(int i) | Hiển thị khung ngoài bên cạnh ô chơi (score, next,...). |
| draw() | Vẽ lại toàn bộ màn hình bao gồm bảng chơi, score, khối kế tiếp. |

Hàm Playing

void Tetris::Playing() {

while (true) {

draw(); // Vẽ màn hình game hiện tại.

handleInput(); // Lấy phím người chơi và xử lý.

handlePause(); // Nếu đang tạm dừng, không xử lý gì thêm.

// Nếu không có va chạm khi đi xuống, thì cho khối rơi thêm.

if (!collides(0, 1))

current.setY(current.getY() + 1);

else {

merge(); // Gộp khối vào bảng.

clearLines(); // Xóa hàng nếu đầy.

clearInputBuffer(); // Xóa phím bị giữ liên tục.

current = nextT; // Khối hiện tại = khối kế tiếp.

spawnNextTetromino(); // Tạo khối kế tiếp mới.

// Kiểm tra nếu khối mới sinh ra đã va chạm => hết chỗ => game over.

if (collides(0, 0)) {

cout << "Game Over!\n";

break;

}

}

clearInputBuffer(); // Lặp lại để chắc chắn input sạch.

std::this\_thread::sleep\_for(std::chrono::milliseconds(300));

// Delay 300ms giữa mỗi lần vẽ, tạo hiệu ứng rơi từ từ.

}

}

Hàm Playing() là vòng lặp game loop, liên tục cập nhật trạng thái trò chơi cho đến khi game kết thúc (va chạm ở vị trí sinh khối). Mỗi vòng lặp:

- Vẽ game, nhận đầu vào, xử lý logic rơi, va chạm, và sinh khối mới.

- Thêm hiệu ứng thời gian chờ để khối rơi không quá nhanh.