* 1. **Khái niệm phần mềm mã nguồn mở và phần mềm mã nguồn đóng**
* Phần mềm mã nguồn mở (Open Source Software - OSS): Là phần mềm mà mã nguồn được công khai, cho phép mọi người tự do sử dụng, sửa đổi và chia sẻ. Ví dụ phần mềm mã nguồn mở: Linux, Ubuntu, Apache TomCat ,…
* Phần mềm mã nguồn đóng (Proprietary Software): Là phần mềm được tạo ra bởi mã nguồn độc quyền được bảo vệ chặt chẽ bởi đơn vị nhà phát hành. Chỉ có người xây dựng mới được phép truy cập để sao chép, sửa đổi. Ví dụ phần mềm mã nguồn đóng: Microsoft Windows, Adobe Photoshop. Google Earth
* Các thuật ngữ liên quan:
* GPL v3.0 (General Public License): Giấy phép yêu cầu yêu cầu bất kỳ phần mềm nào được phân phối lại hoặc sửa đổi phải công khai mã nguồn, đảm bảo mọi phiên bản đều là mã nguồn mở. Sử dụng giấy phép này khi bạn muốn bảo vệ quyền tự do của người dùng và đảm bảo tất cả các phiên bản sửa đổi và phân phối lại vẫn là mã nguồn mở.
* MIT License: Cho phép sử dụng, chỉnh sửa, phân phối mã nguồn miễn là giữ nguyên thông báo bản quyền, không yêu cầu công khai mã nguồn đã sửa đổi. Sử dụng giấy phép khi bạn muốn cung cấp phần mềm với ít ràng buộc nhất, cho phép sử dụng rộng rãi, kể cả trong các dự án thương mại.
* Apache License: Cho phép sử dụng, sửa đổi, và phân phối phần mềm, kể cả trong các sản phẩm thương mại. Sử dụng giấy phép này khi bạn muốn bảo vệ quyền sáng chế và cho phép sử dụng phần mềm trong các sản phẩm thương mại mà không cần công khai mã nguồn đã sửa đổi.
  1. **Định nghĩa phần mềm mã nguồn mở theo tiêu chí OSI**
* Theo Open Source Initiative (OSI), phần mềm mã nguồn mở phải đáp ứng các tiêu chí sau:
* Phân phối tự do: Phần mềm có thể được bán hoặc phân phối lại mà không bị hạn chế.
* Mã nguồn: Mã nguồn phải được bao gồm trong phân phối hoặc dễ dàng truy cập.
* Tác phẩm tái sinh: Phải cho phép chỉnh sửa và phân phối lại các phiên bản chỉnh sửa.
* Tính toàn vẹn của mã nguồn tác giả: Có thể hạn chế phân phối mã nguồn đã chỉnh sửa chỉ nếu việc đó cho phép phân phối các tập tin bản vá.
* Không phân biệt đối xử với cá nhân hoặc nhóm.
* Không phân biệt đối xử đối với lĩnh vực ứng dụng.
* Phân phối giấy phép: Các quyền đính kèm vào chương trình phải được áp dụng cho tất cả những ai mà chương trình được phân phối đến mà không cần ký thêm giấy phép.
* Giấy phép không cụ thể cho một sản phẩm.
* Giấy phép không hạn chế phần mềm khác.
* Giấy phép phải trung lập về công nghệ.
  1. **Định nghĩa phần mềm tự do, phần mềm miễn phí và phần mềm trả một phần**
* Phần mềm tự do (Free Software): Phần mềm tự do là phần mềm mà người dùng có quyền tự do chạy, nghiên cứu, sửa đổi và phân phối lại phần mềm. Được đảm bảo bởi các giấy phép phần mềm tự do như GPL, MIT, Apache,..
* Phần mềm miễn phí (Freeware): Là phần mềm có bản quyền được phân phối miễn phí. Người dùng chỉ sử dụng, không có quyền sửa đổi hay phân phối lại.
* Phần mềm trả một phần (Freemium Software): Là một loại phần mềm được cấp miễn phí cho người dùng nhưng ở một định dạng hạn chế. Phiên bản đầy đủ của phần mềm sẽ được cấp sau một thời gian dùng thử hoặc phiên bản dùng thử sẽ được cấp với một số tính năng bị vô hiệu hóa.
  1. **Giấy phép sử dụng phần mềm mã nguồn mở, lợi ích và các yếu tố tác động đến sự tự do phần mềm mã nguồn mở**
* Giấy phép phần mềm mã nguồn mở : là một loại giấy phép cho phần mềm máy tính và các sản phẩm khác, cho phép mã nguồn – bản thiết kế được sử dụng, sửa đổi hoặc chia sẻ theo các điều khoản và điều kiện được xác định. Ví dụ: GPL, MIT, Apache.
* Lợi ích:
* Tự do sử dụng và phân phối
* Tính hợp pháp và bản quyền rõ ràng
* Tự do sửa đổi và cải tiến
* Các yếu tố tác động đến sự tự do phần mềm mã nguồn mở:
* Giấy phép: Một số giấy phép có thể hạn chế tự do sửa đổi và phân phối lại.
* Phụ thuộc vào cộng đồng: Nếu cộng đồng không đủ mạnh, phần mềm có thể bị bỏ rơi.
* Yếu tố kinh doanh: Một số công ty có thể hạn chế tính tự do để bảo vệ lợi ích kinh doanh.
  1. **Ưu điểm và nhược điểm hệ thống mã nguồn mở**
* Ưu điểm:
* Hoạt động tự do và linh hoạt : Người dùng tự do điều chỉnh sao cho phù hợp với nhu cầu sử dụng
* Tiết kiệm chi phí: Giảm chi phí bản quyền phần mềm.
* Cộng đồng hỗ trợ: Có sự hỗ trợ từ cộng đồng người dùng và nhà phát triển rộng lớn.
* Nhược điểm:
* Tính tương thích : Có thể gặp khó khăn trong việc tương thích với phần mềm hoặc phần cứng khác.
* Nguy cơ tấn công cao : Các đoạn mã nguồn được công khai, tạo điều kiện cho hacker tận dụng lỗ hổng trước khi chúng được người dùng phát hiện
  1. **Cách khắc phục nhược điểm của hệ thống mã nguồn mở**
* Tăng cường bảo mật : Thực hiện kiểm tra bảo mật định kỳ để phát hiện và khắc phục lỗ hổng
* Cải thiện hỗ trợ : Cung cấp hướng dẫn rõ ràng và dễ hiểu về cách sử dụng. Tạo diễn đàn hoặc nhóm trò chuyện để người dùng có thể tương tác và hỗ trợ lẫn nhau
* Đảm bảo tính bền vững : Xây dựng cộng đồng người dùng và nhà phát triển tích cực tham gia vào dự án. Xác định các mục tiêu và kế hoạch cụ thể cho sự phát triển và bảo trì lâu dài của phần mềm.
  1. **Phần mềm miễn phí dạng dịch vụ, ưu và nhược điểm, ví dụ**
* Phần mềm miễn phí dạng dịch vụ (SaaS) là mô hình cung cấp phần mềm qua internet, nơi người dùng có thể truy cập và sử dụng phần mềm mà không cần phải cài đặt.
* Ưu điểm:
* Khả năng tiếp cận: Người dùng có thể tiếp cận các tính năng cơ bản miễn phí.
* Tăng số lượng người dùng: Dễ dàng thu hút người dùng thử nghiệm dịch vụ.
* Khả năng nâng cấp: Người dùng có thể dễ dàng nâng cấp lên các tính năng cao cấp khi cần thiết.
* Nhược điểm:
* Hạn chế tính năng: Các tính năng cao cấp thường bị khóa, chỉ có trong phiên bản trả phí.
* Chi phí ẩn: Người dùng có thể gặp chi phí cao nếu muốn sử dụng đầy đủ các tính năng.
* Phụ thuộc vào nhà cung cấp: Người dùng bị phụ thuộc vào dịch vụ của nhà cung cấp.
* Ví dụ phần mềm miễn phí dạng dịch vụ:
* Google Workspaces : Cung cấp các ứng dụng văn phòng như Gmail, Google Docs, Google Sheets miễn phí và trả phí.
* Dropbox: Cung cấp dung lượng lưu trữ miễn phí, người dùng có thể trả phí để tăng dung lượng và truy cập các tính năng bổ sung.
  1. List
* List comprehension
* Iterators, Generators
  1. OOP mức thông hiểu
  2. Higher Order Function Python : lambda, map, filter, reduce
  3. MRO

**Lab1 :**

sanpham = [

    {'id' : 1, 'tensanpham' : 'AO1', 'gia' : 10000},

    {'id' : 2, 'tensanpham' : 'AO2', 'gia' : 10000},

    {'id' : 3, 'tensanpham' : 'AO3', 'gia' : 10000},

    {'id' : 4, 'tensanpham' : 'AO4', 'gia' : 100000},

    {'id' : 5, 'tensanpham' : 'AO5', 'gia' : 100000},

    {'id' : 6, 'tensanpham' : 'AO6', 'gia' : 100},

    {'id' : 7, 'tensanpham' : 'AO7', 'gia' : 1000}

]

tong = 0

for i in sanpham:

    tong += i['gia']

print(tong)

x = sum(map(lambda *a* : *a*['gia'],sanpham))

print(x)

text = sum(map(lambda *x* : float(*x*['gia']) , sanpham))

print(text)

a = (1,2,3,4,5,6,7,8,9,10)

kq = [i \* 2 if i % 2 == 0 else i \* 3 for i in a]

print(list(kq))

a = [1,2,3,4,5,6,7]

b = [2,5,6,4,1]

result = []

for i in b:

    kq = a.index(i)

    result.append((i,kq))

print(result)

**Lab2:**

def input\_movie\_data():

    title = input("Nhập tiêu đề của phim: ")

    rates = input("Nhập số lượng đánh giá phim : ")

    ratings = []

    for i in range(3):

        rating = float(input(f"Nhập điểm đánh giá thứ {i + 1} của phim: "))

        ratings.append(rating)

    return rates, ratings, title

def calculate\_average\_rating(*ratings*):

    return sum(*ratings*) / len(*ratings*)

def print\_movie\_info(*rates*, *title*, *num\_ratings*, *avg\_rating*):

    print(f"rates = {*rates*}, title = {*title*}, ratings = {*num\_ratings*}, score = {*avg\_rating*:.2f}")

def main():

    movie\_list = []

    num\_movies = int(input("Nhập số lượng phim bạn muốn quản lý : "))

    for i in range(num\_movies):

        print(f"\nNhập thông tin cho phim {i + 1}")

        rates, ratings, title = input\_movie\_data()

        num\_ratings = len(ratings)

        avg\_rating = calculate\_average\_rating(ratings)

        movie\_list.append((rates, title, num\_ratings, avg\_rating))

    print("\nThông tin các phim : ")

    for movie in movie\_list:

        print\_movie\_info(\*movie)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

class MovieRatings:

    def rate\_movies(*scanner*):

        for line in *scanner*:

            line = line.strip()

            rates = line.split()

            num\_ratings = int(rates[0])

            movie\_ratings = [float(rating) for rating in rates[1 : ] if rating.replace('.', '', 1).isdigit()]

            title\_index = 1 + len(movie\_ratings)

            title = ' '.join(rates[title\_index : ])

            avg\_rating = sum(movie\_ratings) / len(movie\_ratings)

            print(f"title = {title}, ratings = {num\_ratings}, score = {avg\_rating:.2f}")

if \_\_name\_\_ == '\_\_main\_\_':

    input\_data = [

        "4 9.2 9 8 9.5 Seven",

        "5 8.2 9.5 7 10.0 Men",

        "6 5.7 6 9 9.3 Club Streets"

    ]

    scanner = iter(input\_data)

    MovieRatings.rate\_movies(scanner)

**Lab3:**

class Manufacturer:

    def \_\_init\_\_(*self*, *name*, *location*):

*self*.name = *name*

*self*.location = *location*

    def mota(*self*):

        return f"{*self*.name}, Location  : {*self*.location}"

class Device:

    def \_\_init\_\_(*self*, *name*, *price*, *manufacturer*):

*self*.name = *name*

*self*.price = *price*

*self*.manufacturer = *manufacturer*

    def mota(*self*):

        return (f"Phone Name : {*self*.name}, Price: {*self*.price}, " +

                f"From: {*self*.manufacturer.mota()}")

device1 = Device("Apple", 32000, Manufacturer("Apple","USA"))

print(device1.mota())

**Lab4\_1:**

class SubjectMapping:

    def \_\_init\_\_(*self*, *list\_A*, *list\_B*):

*self*.list\_A = *list\_A*

*self*.list\_B = *list\_B*

*self*.mamonHoc = {

            "Lập trình Python": 1,

            "Hệ thống thông minh": 2,

            "Nguyên lý ngôn ngữ lập trình": 3,

            "Học máy": 4

        }

*self*.manganh = {

            1: "CNTT",

            2: "KHDL",

            3: "KTPM",

            4: "TTNT"

        }

    def cauA(*self*):

        for monHoc, maMon in zip (*self*.list\_A, *self*.list\_B):

            print("Môn học: ", monHoc, "- Mã môn: ", maMon)

    def cauB(*self*):

        return [(mamonHoc, *self*.manganh[manganh]) for mamonHoc, manganh in *self*.mamonHoc.items()]

if \_\_name\_\_ == "\_\_main\_\_":

    list\_A = ["Lập trình trực quan", "Cấu trúc dữ liệu Python", "Khai Phá dữ liệu", "Chuyên đề phân tích dữ liệu"]

    list\_B = [101, 211, 331, 422]

    # câu 1 A

    subjects = SubjectMapping(list\_A, list\_B)

    print("\nMapping list\_MonHoc với list\_Year:")

    subjects.cauA()

    # Câu 1 B

    print("\nMapping mamonHoc với manganh:")

    map = subjects.cauB()

    for mamonHoc, manganh in map:

        print(f"{mamonHoc} : {manganh}")

**Lab4\_2a:**

class CONGTY:

    def \_\_init\_\_(*self*, *macongty*, *makhuvuc*):

*self*.macongty = *macongty*

*self*.makhuvuc = *makhuvuc*

    def mota(*self*):

        return f"{*self*.macongty}, Mã khu vực: {*self*.makhuvuc}"

class NHANVIEN:

    def \_\_init\_\_(*self*, *name*, *age*, *congty*):

*self*.name = *name*

*self*.age = *age*

*self*.congty = *congty*

    def mota(*self*):

        return (f"Employee Name: {*self*.name}, Age : {*self*.age}, " +

                f"Company : {*self*.congty.mota()}")

# Example usage

nhanvien1 = NHANVIEN("Nguyen Van A", 25, CONGTY(1, "A"))

print(nhanvien1.mota())

**Lab4\_2b :**

from abc import ABC, abstractmethod

class NhanSu(ABC):

    def \_\_init\_\_(*self*, *ten*: str, *namsinh*: int):

*self*.ten = *ten*

*self*.namsinh = *namsinh*

    @abstractmethod

    def thongtin(*self*):

        pass

    @abstractmethod

    def get\_role(*self*):

        pass

class SinhVien(NhanSu):

    def \_\_init\_\_(*self*, *ten*: str, *namsinh*: int, *diem*: str):

        super().\_\_init\_\_(*ten*, *namsinh*)

*self*.diem = *diem*

    def thongtin(*self*):

        return f"Sinh viên: {*self*.ten}, Năm sinh: {*self*.namsinh}, Điểm: {*self*.diem}"

    def get\_role(*self*):

        return "sinhvien"

class BacSi(NhanSu):

    def \_\_init\_\_(*self*, *ten*: str, *namsinh*: int, *chuyenkhoa*: str):

        super().\_\_init\_\_(*ten*, *namsinh*)

*self*.chuyenkhoa = *chuyenkhoa*

    def thongtin(*self*):

        return f"Bác sĩ: {*self*.ten}, Năm sinh: {*self*.namsinh}, Chuyên khoa: {*self*.chuyenkhoa}"

    def get\_role(*self*):

        return "bacsi"

class GiangVien(NhanSu):

    def \_\_init\_\_(*self*, *ten*: str, *namsinh*: int, *chuyenmon*: str):

        super().\_\_init\_\_(*ten*, *namsinh*)

*self*.chuyenmon = *chuyenmon*

    def thongtin(*self*):

        return f"Giảng viên: {*self*.ten}, Năm sinh: {*self*.namsinh}, Chuyên môn: {*self*.chuyenmon}"

    def get\_role(*self*):

        return "giangvien"

class DayNS:

    def \_\_init\_\_(*self*, *ten*: str):

*self*.ten = *ten*

*self*.danh\_sach\_nhan\_su = []

    def themNhanSu(*self*, *nhansu*: NhanSu):

*self*.danh\_sach\_nhan\_su.append(*nhansu*)

    def thongtin(*self*):

        result = f"Tên dãy NS: {*self*.ten}\n"

        for nhansu in *self*.danh\_sach\_nhan\_su:

            result += nhansu.thongtin() + "\n"

        return result

    def demBacSi(*self*):

        return sum(1 for nhansu in *self*.danh\_sach\_nhan\_su if nhansu.get\_role() == "bacsi")

    def sapxepNamSinh(*self*):

*self*.danh\_sach\_nhan\_su.sort(*key*=lambda *nhansu*: *nhansu*.namsinh)

    def trungbinhNamSinh(*self*):

        giangvien\_list = [nhansu.namsinh for nhansu in *self*.danh\_sach\_nhan\_su if nhansu.get\_role() == "giangvien"]

        return sum(giangvien\_list) / len(giangvien\_list) if giangvien\_list else 0

# Example usage

dayns = DayNS(*ten*="Khoa CNTT")

dayns.themNhanSu(SinhVien(*ten*="Sinh Vien A", *namsinh*=2000, *diem*="A"))

dayns.themNhanSu(GiangVien(*ten*="Giang Vien B", *namsinh*=1975, *chuyenmon*="CNTT"))

dayns.themNhanSu(GiangVien(*ten*="Giang Vien C", *namsinh*=1980, *chuyenmon*="Math"))

dayns.themNhanSu(BacSi(*ten*="Bac Si D", *namsinh*=1965, *chuyenkhoa*="Tim mach"))

dayns.themNhanSu(BacSi(*ten*="Bac Si E", *namsinh*=1970, *chuyenkhoa*="Than kinh"))

print(dayns.thongtin())

print(f"Số lượng bác sĩ: {dayns.demBacSi()}")

dayns.sapxepNamSinh()

print("Sau khi sắp xếp theo năm sinh:")

print(dayns.thongtin())

print(f"Trung bình năm sinh của Giảng Viên: {dayns.trungbinhNamSinh()}")

**Lab5:**

class Person:

    def \_\_init\_\_(*self*, *name*, *yob*):

*self*.name = name

*self*.yob = yob

    def describe(*self*):

        pass

    def get\_role(*self*):

        return "person"

class Student(*Person*):

    def \_\_init\_\_(*self*, *name*, *yob*, *grade*):

*super*().\_\_init\_\_(name, yob)

*self*.grade = grade

    def describe(*self*):

        print(f"Student: {*self*.name}, Year of Birth: {*self*.yob}, Grade: {*self*.grade}")

    def get\_role(*self*):

        return "student"

class Teacher(*Person*):

    def \_\_init\_\_(*self*, *name*, *yob*, *subject*):

*super*().\_\_init\_\_(name, yob)

*self*.subject = subject

    def describe(*self*):

        print(f"Teacher: {*self*.name}, Year of Birth: {*self*.yob}, Subject: {*self*.subject}")

    def get\_role(*self*):

        return "teacher"

class Doctor(*Person*):

    def \_\_init\_\_(*self*, *name*, *yob*, *specialist*):

*super*().\_\_init\_\_(name, yob)

*self*.specialist = specialist

    def describe(*self*):

        print(f"Doctor: {*self*.name}, Year of Birth: {*self*.yob}, Specialist: {*self*.specialist}")

    def get\_role(*self*):

        return "doctor"

class Ward:

    def \_\_init\_\_(*self*, *ward\_name*):

*self*.ward\_name = ward\_name

*self*.people = []

    def addPerson(*self*, *person*):

*self*.people.append(person)

    def describe(*self*):

        print(f"Ward: {*self*.ward\_name}")

        for person in *self*.people:

            person.describe()

    def countDoctor(*self*):

        return sum(1 for person in *self*.people if person.get\_role() == "doctor")

    def sortAge(*self*):

*self*.people.sort(*key*=lambda *person*: person.yob)

    def aveTeacherYearOfBirth(*self*):

        teacher\_yobs = [person.yob for person in *self*.people if person.get\_role() == "teacher"]

        return sum(teacher\_yobs) / len(teacher\_yobs) if teacher\_yobs else 0

    def removePerson(*self*, *name*):

*self*.people = [person for person in *self*.people if person.name != name]

    def findOldestPerson(*self*):

        oldest\_person = min(*self*.people, *key*=lambda *person*: person.yob, *default*=None)

        if oldest\_person:

            oldest\_person.describe()

    def findPersonBySpecialist(*self*, *specialist*):

        for person in *self*.people:

            if person.get\_role() == "doctor" and person.specialist == specialist:

                person.describe()

    def findStudentsByGrade(*self*, *grade*):

        for person in *self*.people:

            if person.get\_role() == "student" and person.grade == grade:

                person.describe()

    def updateTeacherSubject(*self*, *name*, *new\_subject*):

        for person in *self*.people:

            if person.get\_role() == "teacher" and person.name == name:

                person.subject = new\_subject

    def findPersonByName(*self*, *name*):

        for person in *self*.people:

            if person.name == name:

                person.describe()

    def aveAgeByOccupation(*self*, *occupation*):

        current\_year = 2024

        ages = []

        for person in *self*.people:

            if occupation == "student" and person.get\_role() == "student":

                ages.append(current\_year - person.yob)

            elif occupation == "teacher" and person.get\_role() == "teacher":

                ages.append(current\_year - person.yob)

            elif occupation == "doctor" and person.get\_role() == "doctor":

                ages.append(current\_year - person.yob)

        return sum(ages) / len(ages) if ages else 0

ward = Ward("A1")

ward.addPerson(Student("Alice", 2005, "10A"))

ward.addPerson(Teacher("Mr. Smith", 1980, "Math"))

ward.addPerson(Teacher("Ms. Johnson", 1975, "History"))

ward.addPerson(Doctor("Dr. Brown", 1970, "Cardiology"))

ward.addPerson(Doctor("Dr. Green", 1965, "Neurology"))

ward.describe()

print("\nSố lượng bác sĩ:", ward.countDoctor())

ward.sortAge()

print("\nDanh sách sau khi sắp xếp theo tuổi tăng dần:")

ward.describe()

print("\nTrung bình năm sinh của giáo viên:", ward.aveTeacherYearOfBirth())

print("\nDanh sách sau khi xóa Ms. Johnson:")

ward.removePerson("Ms. Johnson")

ward.describe()

**Lab6 :**

class MyMap:

    def \_\_init\_\_(*self*, *number*):

*self*.list = []

*self*.\_\_mapAdd(*number*)

    def mapAdd(*self*, *number*):

        for item in *number*:

*self*.list.append(item)

    \_\_mapAdd = mapAdd

class MapSubClass(MyMap):

    def mapAdd(*self*, *keys*, *values*):

        formatted\_values = []

        for num, value in zip(*self*.list, *values*):

            if num - 1 < len(*keys*):

                formatted\_values.append(f"{num}-{*keys*[num - 1]}")

            else:

                formatted\_values.append(f"{num}-zero")

        return formatted\_values

def main():

    number\_list = [1, 2, 3, 4, 5]

    map\_instance = MapSubClass(number\_list)

    keys = ['one', 'two', 'three']

    values = [1, 2, 3, 4, 5]

    print("Before Map list = ", number\_list)

    formatted\_values = map\_instance.mapAdd(keys, values)

    print("Map list = ", formatted\_values)

    for formatted\_value in formatted\_values:

        map\_instance.list.append(formatted\_value)

    print("After Map list = ", map\_instance.list)

if \_\_name\_\_ == "\_\_main\_\_":

    main()