

TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI

GIÁO TRÌNH

LẬP TRÌNH PYTHON

DÀNH CHO KỸ THUẬT

Tác giả: ThS. Nguyễn Đình Quý

Trường Đại học Xây dựng Hà Nội

HÀ NỘI - 2026

MỤC LỤC

CHƯƠNG 1. Biến, các kiểu dữ liệu và toán tử cơ bản trong Python	2
1.1 Mở đầu	2
1.2 Biến trong Python	2
1.2.1 Bản chất của biến.....	2
1.2.2 Gán giá trị cho biến.....	2
1.2.3 Đặt tên biến	3
1.3 Các kiểu dữ liệu cơ bản trong Python và đặc điểm bộ nhớ	4
1.3.1 Kiểu số nguyên (int)	4
1.3.2 Kiểu số thực (float)	4
1.3.3 Kiểu logic (bool)	5
1.3.4 Kiểu chuỗi (str).....	5
1.3.5 So sánh bộ nhớ và phạm vi biểu diễn của các kiểu dữ liệu cơ bản	5
1.3.6 Nhận xét	5
1.4 Kết luận	6
1.5 Bài tập	6
1.5.1 Bài tập 1: Khai báo biến và tính BMI.....	6
1.5.2 Bài tập 2: Sử dụng toán tử số học	6
1.5.3 Bài tập 3: Toán tử logic và so sánh	6
1.5.4 Bài tập 4: Chuyển đổi kiểu dữ liệu.....	7
1.5.5 Bài tập 5: Xử lý chuỗi	7
CHƯƠNG 2. Kiểu dữ liệu tuần tự.....	8
2.1 Tổng quan về kiểu dữ liệu phức hợp trong Python	8
2.2 Khái niệm kiểu dữ liệu tuần tự	8
2.3 Kiểu dữ liệu String	8
2.3.1 Bản chất của chuỗi trong Python.....	8
2.3.2 Tính bất biến của chuỗi	9

2.3.3 Các thao tác thường gặp với chuỗi	9
2.4 Kiểu dữ liệu List	9
2.4.1 Khái niệm và đặc điểm của List	9
2.4.2 Truy cập và chỉnh sửa phần tử trong List	10
2.4.3 Thêm và xóa phần tử trong List	10
2.5 Kiểu dữ liệu Tuple	10
2.5.1 Khái niệm và bản chất của Tuple.....	10
2.5.2 Truy cập phần tử trong Tuple	10
2.5.3 Tính bất biến và ý nghĩa thực tiễn	11
2.5.4 Kiểu dữ liệu Dictionary.....	11
2.5.5 Bản chất của Dictionary	11
2.5.6 Truy cập và thay đổi dữ liệu trong Dictionary	11
2.5.7 Duyệt Dictionary	12
2.5.8 Ứng dụng thực tế của Dictionary.....	12
2.6 Kiểu dữ liệu Set	12
2.6.1 Khái niệm và đặc điểm của Set	12
2.6.2 Thao tác cơ bản với Set	12
2.6.3 Các phép toán tập hợp	12
2.6.4 Ứng dụng của Set.....	13
2.7 Chỉ mục trong kiểu dữ liệu tuần tự	13
2.7.1 Vai trò của chỉ mục	13
2.7.2 Chỉ mục dương trong Python.....	13
2.7.3 Chỉ mục âm trong Python.....	14
2.7.4 So sánh chỉ mục dương và chỉ mục âm.....	14
2.7.5 Chỉ mục và độ dài dữ liệu	15
2.7.6 Lỗi truy cập ngoài phạm vi chỉ mục.....	15
2.7.7 Chỉ mục trong các kiểu dữ liệu tuần tự khác nhau.....	15
2.8 Duyệt các kiểu dữ liệu tuần tự.....	16

2.9 Cắt lát dữ liệu (Slicing) trong các kiểu dữ liệu tuần tự.....	16
2.9.1 Cú pháp tổng quát của cắt lát	16
2.9.2 Cắt lát cơ bản với chỉ số bắt đầu và kết thúc	16
2.9.3 Cắt lát khi bỏ qua chỉ mục bắt đầu hoặc kết thúc	17
2.9.4 Cắt lát với bước nhảy (step)	17
2.9.5 Cắt lát với chỉ số âm	17
2.9.6 Đảo ngược dữ liệu bằng cách cắt lát	17
2.9.7 Cắt lát với chuỗi	18
2.9.8 Cắt lát với Tuple	18
2.9.9 Những lưu ý quan trọng khi sử dụng cắt lát	18
2.9.10 Ý nghĩa của cắt lát trong lập trình kỹ thuật.....	18
2.10 Bài tập luyện tập	19
2.10.1 Câu hỏi trắc nghiệm	19
2.11 Bài tập thực hành.....	20
CHƯƠNG 3. Nhập xuất dữ liệu trong Python	23
3.1 Xuất dữ liệu ra màn hình	23
3.1.1 Hàm print ()	23
3.1.2 In nhiều dữ liệu cùng lúc	23
3.1.3 Tùy chỉnh định dạng khi in	24
3.1.4 In dữ liệu dạng bảng	24
3.2 Nhập dữ liệu từ bàn phím.....	25
3.2.1 Hàm input ()	25
3.2.2 Nhập và chuyển đổi kiểu dữ liệu	25
3.3 Nhập xuất dữ liệu với tập tin.....	25
3.3.1 Ví dụ tập tin dữ liệu.....	25
3.3.2 Đọc tập tin và xử lý dữ liệu	26
3.3.3 Ghi kết quả ra tập tin	26

3.4 Các phương thức làm việc với tập tin trong Python	26
3.4.1 close ()	26
3.4.2 fileno ()	27
3.4.3 flush ()	27
3.4.4 isatty ()	27
3.4.5 read (n)	27
3.4.6 readable ()	28
3.4.7 readline (n=-1)	28
3.4.8 readlines (n=-1)	28
3.4.9 seek (offset, from=SEEK_SET)	28
3.4.10 seekable ()	29
3.4.11 tell ()	29
3.4.12 truncate (size=None)	29
3.4.13 writable ()	29
3.4.14 write (s)	30
3.4.15 writelines (lines)	30
3.5 Nhận xét.....	30
3.6 Ứng dụng thực tế.....	30
3.7 Bài tập chương	30
3.7.1 Câu hỏi trắc nghiệm	30
3.7.2 Bài tập thực hành.....	32
3.7.3 Gợi ý.....	33
3.8 Kết luận	33
CHƯƠNG 4. Điều khiển luồng và câu lệnh điều kiện trong Python	34
4.1 Giá trị Boolean và vai trò trong điều khiển luồng.....	34
4.1.1 Khái niệm giá trị Boolean.....	34
4.1.2 Biểu thức logic.....	34

4.2 Các toán tử trả về giá trị Boolean.....	34
4.2.1 Toán tử so sánh.....	34
4.2.2 Toán tử kiểm tra thành viên.....	35
4.2.3 Toán tử so sánh đối tượng	35
4.3 Các giá trị tương đương Boolean trong Python	36
4.3.1 Khái niệm Truthy và Falsy.....	36
4.3.2 Các giá trị được coi là False	36
4.4 Toán tử logic	36
4.4.1 Ba toán tử logic cơ bản	36
4.4.2 Bảng logic.....	36
4.5 Thứ tự ưu tiên của các phép toán logic.....	37
4.6 Các chú ý quan trọng khi làm việc với phép toán logic.....	37
4.7 Câu lệnh if – elif – else.....	38
4.7.1 Câu lệnh if	38
4.7.2 Câu lệnh if – else	38
4.7.3 Câu lệnh if – elif – else	39
4.7.4 Câu lệnh điều kiện lồng nhau	40
4.7.5 Một số lưu ý quan trọng khi sử dụng câu lệnh điều kiện.....	41
4.7.6 Ý nghĩa của câu lệnh điều kiện trong lập trình kỹ thuật	41
4.7.7 Câu lệnh điều kiện lồng nhau	41
4.7.8 Ứng dụng câu lệnh điều kiện trong bài toán kỹ thuật.....	43
4.8 Vòng lặp trong Python.....	45
4.8.1 Vòng lặp for.....	45
4.8.2 Vòng lặp while	48
4.8.3 So sánh vòng lặp for và while	49
4.8.4 Vòng lặp lồng nhau	49
4.8.5 Lệnh break, continue và pass	51
4.8.6 Ứng dụng vòng lặp trong bài toán kỹ thuật.....	52

4.8.7 Xử lý ngoại lệ trong Python (try – except)	53
4.9 Bài tập chương: Câu lệnh điều kiện, vòng lặp và xử lý ngoại lệ.....	57
4.9.1 Câu hỏi trắc nghiệm	57
4.9.2 Bài tập thực hành.....	59
4.9.3 Bài tập tổng hợp	60
4.10 Hàm và Module trong Python	60
4.10.1 Khái niệm về hàm	60
4.10.2 Định nghĩa và gọi hàm	60
4.10.3 Hàm có tham số.....	61
4.10.4 Tham số của hàm trong Python.....	61
4.10.5 Hàm trả về giá trị	64
4.10.6 Phạm vi biến trong hàm.....	65
4.10.7 Hàm vô danh (Lambda function).....	66
4.10.8 Các hàm xử lý tập hợp trong Python	66
4.10.9 List Comprehension.....	72
4.10.10 So sánh các phương pháp xử lý tập hợp.....	76
4.10.11 Khái niệm về Module.....	76
4.10.12 Sử dụng module có sẵn.....	76
4.10.13 Tự xây dựng module.....	76
4.10.14 Ý nghĩa của hàm và module trong lập trình kỹ thuật.....	77
4.10.15 Kết luận.....	77
4.11 Bài tập	77
4.11.1 Bài tập 1: Hàm tính toán cơ bản	77
4.11.2 Bài tập 2: Hàm xử lý chuỗi.....	77
4.11.3 Bài tập 3: Hàm làm việc với List	78
4.11.4 Bài tập 4: Lambda và Higher-order Functions	78
4.11.5 Bài tập 5: Module và Package	78

CHƯƠNG 5. Lập trình hướng đối tượng trong Python.....	79
5.1 Giới thiệu về lập trình hướng đối tượng.....	79
5.1.1 Khái niệm lập trình hướng đối tượng	79
5.1.2 Lợi ích của lập trình hướng đối tượng	79
5.1.3 Các khái niệm cơ bản trong OOP	80
5.2 Lớp và đối tượng trong Python.....	81
5.2.1 Định nghĩa lớp	81
5.2.2 Tạo đối tượng từ lớp	81
5.2.3 Thuộc tính của đối tượng.....	82
5.3 Phương thức khởi tạo - Constructor.....	83
5.3.1 Phương thức <code>__init__()</code>	83
5.3.2 Giá trị mặc định cho tham số.....	85
5.4 Phương thức của lớp	85
5.4.1 Định nghĩa phương thức	85
5.4.2 Phương thức với giá trị trả về	86
5.5 Tính đóng gói (Encapsulation)	87
5.5.1 Khái niệm đóng gói.....	87
5.5.2 Thuộc tính và phương thức riêng tư	87
5.5.3 Getter và Setter	89
5.5.4 Property decorator	90
5.6 Tính kế thừa (Inheritance).....	90
5.6.1 Khái niệm kế thừa.....	90
5.6.2 Cú pháp kế thừa.....	91
5.6.3 Ví dụ kế thừa cơ bản.....	91
5.6.4 Phương thức <code>super()</code>	92
5.6.5 Kế thừa nhiều cấp	92
5.6.6 Kiểm tra quan hệ kế thừa.....	93

5.7 Tính đa hình (Polymorphism)	94
5.7.1 Khái niệm đa hình.....	94
5.7.2 Đa hình thông qua ghi đè phương thức	94
5.7.3 Đa hình với các toán tử	95
5.8 Các phương thức đặc biệt (Magic Methods).....	96
5.8.1 Giới thiệu về magic methods	96
5.8.2 Một số magic methods thường dùng	97
5.8.3 Ví dụ sử dụng magic methods	97
5.9 Thuộc tính và phương thức lớp (Class Attributes and Methods)	98
5.9.1 Thuộc tính lớp.....	98
5.9.2 Phương thức lớp	99
5.9.3 Phương thức tĩnh	100
5.10 Ứng dụng OOP trong bài toán thực tế	101
5.10.1 Hệ thống quản lý thư viện	101
5.11 Bài tập thực hành.....	104
5.11.1 Bài tập cơ bản	104
5.11.2 Bài tập nâng cao	104
5.12 Tổng kết chương.....	105
5.13 Bài tập	105
5.13.1 Bài tập 1: Class Sinh viên	105
5.13.2 Bài tập 2: Class Hình học	106
5.13.3 Bài tập 3: Class TaiKhoan Ngân hàng	106
CHƯƠNG 6. Thư viện NumPy trong Kỹ thuật	107
6.1 Giới thiệu về NumPy	107
6.1.1 Tại sao NumPy quan trọng trong kỹ thuật	107
6.1.2 Cài đặt và import NumPy	107
6.2 Mảng NumPy - ndarray	108
6.2.1 Khái niệm mảng đa chiều	108

6.2.2 Tạo mảng NumPy	108
6.2.3 Thuộc tính của mảng NumPy	111
6.2.4 Kiểu dữ liệu trong NumPy	112
6.3 Chỉ mục và cắt lát mảng NumPy	112
6.3.1 Chỉ mục cơ bản	113
6.3.2 Cắt lát (Slicing)	114
6.3.3 Chỉ mục nâng cao	116
6.4 Các phép toán trên ma trận.....	119
6.4.1 Phép toán từng phần tử (Element-wise Operations).....	119
6.4.2 Phép toán với scalar	120
6.4.3 Nhân ma trận (Matrix Multiplication)	121
6.4.4 Chuyển vị ma trận (Transpose)	122
6.4.5 Các phép toán ma trận nâng cao	123
6.4.6 Broadcasting - Cơ chế mở rộng tự động.....	126
6.5 Các phép xử lý và thống kê trên ma trận	127
6.5.1 Các hàm thống kê cơ bản.....	127
6.5.2 Chỉ số của giá trị min/max.....	128
6.5.3 Phương sai và độ lệch chuẩn	129
6.5.4 Tích lũy và tích lũy theo chiều.....	129
6.5.5 Sắp xếp mảng	130
6.5.6 Loại bỏ trùng lặp và tìm giá trị duy nhất.....	131
6.5.7 Các hàm toán học phổ biến	131
6.5.8 Hàm lượng giác	132
6.5.9 Chuẩn hóa dữ liệu	133
6.5.10 Reshape và thay đổi cấu trúc mảng	134
6.5.11 Nối và chia mảng	135
6.6 Ứng dụng NumPy trong kỹ thuật	136
6.6.1 Xử lý tín hiệu	136

6.6.2 Xử lý dữ liệu cảm biến	138
6.6.3 Tính toán ma trận cho kết cấu.....	139
6.6.4 Mô phỏng và tính toán số	140
6.6.5 Xử lý ma trận hình ảnh.....	141
6.6.6 Phân tích thống kê dữ liệu thực nghiệm	142
6.6.7 Hồi quy tuyến tính	142
6.7 Bài tập thực hành	143
CHƯƠNG 7. Trực quan hóa dữ liệu - Visualization	148
7.1 Giới thiệu về Visualization trong Data Science.....	148
7.1.1 Vai trò của Visualization trong Machine Learning	148
7.1.2 Thư viện Visualization trong Python.....	148
7.2 Cài đặt và cấu hình	149
7.2.1 Cài đặt thư viện	149
7.2.2 Import và cấu hình	149
7.2.3 Lưu hình ảnh	149
7.3 Matplotlib: Nền tảng Visualization	149
7.3.1 Cấu trúc cơ bản: Figure và Axes	149
7.3.2 Line Plot - Đồ thị đường.....	150
7.3.3 Scatter Plot - Biểu đồ phân tán	151
7.3.4 Histogram và Distribution Plot	152
7.3.5 Bar Chart - Biểu đồ cột.....	153
7.3.6 Subplots - Vẽ nhiều biểu đồ	154
7.4 Seaborn: Statistical Visualization	156
7.4.1 Boxplot - Phát hiện Outliers.....	156
7.4.2 Violin Plot - Phân phối chi tiết	158
7.4.3 Heatmap - Correlation Matrix	158
7.4.4 Pairplot - Quan hệ giữa nhiều biến	159
7.4.5 Distribution Plot với KDE	161

7.5 Visualization cho Machine Learning	162
7.5.1 Confusion Matrix	162
7.5.2 ROC Curve và AUC	163
7.5.3 Learning Curves - Phát hiện Overfitting	165
7.5.4 Feature Importance	168
7.5.5 Precision-Recall Curve	169
7.6 Visualization cho Deep Learning.....	170
7.6.1 Training History Visualization.....	170
7.6.2 Visualizing Neural Network Architecture	172
7.6.3 Activation Maps Visualization.....	174
7.7 Best Practices trong Data Visualization	175
7.7.1 Nguyên tắc thiết kế biểu đồ hiệu quả.....	175
7.7.2 Code Template tái sử dụng.....	176
7.8 Bài tập thực hành	177
7.9 Kết luận	179
CHƯƠNG 8. Tính toán Khoa học với Thư viện SciPy.....	180
8.1 Giới thiệu về SciPy.....	180
8.1.1 Cài đặt và Import.....	180
8.1.2 Các Hàm Cơ bản và Hằng số Toán học.....	180
8.2 Tổng quan Các Gói của SciPy.....	182
8.2.1 scipy.integrate - Tích phân và Phương trình Vi phân	182
8.2.2 scipy.optimize - Tối ưu hóa và Tìm nghiệm	182
8.2.3 scipy.interpolate - Nội suy và Xấp xỉ.....	182
8.2.4 scipy.linalg - Đại số Tuyến tính.....	182
8.2.5 scipy.signal - Xử lý Tín hiệu.....	182
8.2.6 scipy.stats - Thống kê và Xác suất	182
8.2.7 scipy.sparse - Ma trận Thưa	183
8.2.8 scipy.fft - Biến đổi Fourier Nhanh	183

8.3 Tính Tích phân với SciPy	183
8.3.1 Cơ sở Lý thuyết Tích phân Số	183
8.3.2 Tích phân Một biến với quad	183
8.3.3 Tích phân Bội	184
8.3.4 Ứng dụng: Tính Năng lượng và Công	185
8.4 Giải Phương trình Vi phân với SciPy	186
8.4.1 Phương trình Vi phân Thường (ODE)	186
8.4.2 Phương pháp Số Giải ODE	187
8.4.3 Sử dụng solve_ivp	187
8.4.4 Hệ Phương trình Vi phân	188
8.4.5 Ứng dụng: Mô hình Lotka-Volterra (Săn mồi - Con mồi)	189
8.5 Tìm Nghiệm Phương trình với SciPy	191
8.5.1 Cơ sở Lý thuyết	191
8.5.2 Phương pháp Chia đôi (Bisection Method)	191
8.5.3 Phương pháp Newton-Raphson và fsolve	193
8.5.4 Giải Hệ Phương trình Phi tuyến	194
8.5.5 Ứng dụng: Tính Điểm Cân bằng trong Phản ứng Hóa học	196
8.6 Nội suy Dữ liệu với SciPy	197
8.6.1 Cơ sở Lý thuyết Nội suy	197
8.6.2 Nội suy Tuyến tính	198
8.6.3 Nội suy Đa thức và Spline	199
8.6.4 Nội suy Hai chiều	200
8.6.5 Ứng dụng: Xử lý Dữ liệu Thiếu	202
8.7 Bài tập Thực hành	203
8.8 Kết luận	206
CHƯƠNG 9. Lời Giải Bài Tập.....	208
9.1 Chương 1: Biến, Toán tử, Kiểu dữ liệu	208
9.1.1 Lời giải Bài tập 1	208

9.1.2 Lời giải Bài tập 2	208
9.1.3 Lời giải Bài tập 3	209
9.1.4 Lời giải Bài tập 4	210
9.1.5 Lời giải Bài tập 5	210
9.2 Chương 2: Cấu trúc Dữ liệu.....	211
9.2.1 Lời giải Bài tập 1	211
9.2.2 Lời giải Bài tập 3	212
9.2.3 Lời giải Bài tập 5	212
9.3 Chương 3: Nhập Xuất Dữ liệu.....	213
9.3.1 Lời giải Bài tập 2	213
9.3.2 Lời giải Bài tập 5	214
9.3.3 Lời giải Bài tập 8	214
9.4 Chương 4: Điều kiện và Vòng lặp	215
9.4.1 Lời giải Bài tập 1	215
9.4.2 Lời giải Bài tập 2	216
9.5 Chương 5: Hàm và Module	217
9.5.1 Lời giải Bài tập 1	217
9.5.2 Lời giải Bài tập 2	218
9.5.3 Lời giải Bài tập 3	219
9.5.4 Lời giải Bài tập 4	220
9.5.5 Lời giải Bài tập 5	221
9.6 Chương 6: Lập trình Hướng Đối tượng	223
9.6.1 Lời giải Bài tập 1	223
9.6.2 Lời giải Bài tập 2	224
9.6.3 Lời giải Bài tập 3	225
9.7 Chương 7: NumPy trong Kỹ thuật.....	228
9.7.1 Lời giải Bài tập 1	228
9.7.2 Lời giải Bài tập 2	229

9.8 Chương 8: Visualization	230
9.8.1 Lời giải Bài tập 1	230
9.8.2 Lời giải Bài tập 2	230
9.9 Chương 9: SciPy	231
9.9.1 Lời giải Bài tập 1	231
9.9.2 Lời giải Bài tập 4	233
9.10 Lưu ý chung	234

Giới thiệu

Trong bối cảnh chuyển đổi số mạnh mẽ hiện nay, lập trình đã trở thành một công cụ nền tảng không chỉ đối với các nhà khoa học máy tính mà còn đối với kỹ sư và nhà nghiên cứu trong hầu hết các lĩnh vực khoa học – kỹ thuật. Khả năng mô hình hóa, mô phỏng, xử lý dữ liệu và tự động hóa bằng máy tính đóng vai trò then chốt trong việc giải quyết các bài toán thực tiễn ngày càng phức tạp. Vì vậy, việc trang bị cho sinh viên những kiến thức và kỹ năng lập trình ngay từ giai đoạn đào tạo đại học là một yêu cầu tất yếu.

Python là một trong những ngôn ngữ lập trình phổ biến nhất hiện nay trong lĩnh vực khoa học và kỹ thuật nhờ cú pháp đơn giản, dễ tiếp cận, khả năng biểu đạt cao và hệ sinh thái thư viện phong phú. Python cho phép người học tập trung vào tư duy thuật toán và mô hình tính toán thay vì bị chi phối bởi các chi tiết cú pháp phức tạp. Đặc biệt, với các thư viện mạnh như NumPy, SciPy và các công cụ trực quan hóa dữ liệu, Python đã trở thành ngôn ngữ tiêu chuẩn trong tính toán khoa học, phân tích dữ liệu và mô phỏng kỹ thuật.

Giáo trình *Lập trình Python dành cho kỹ thuật* được biên soạn nhằm phục vụ giảng dạy và học tập cho sinh viên ngành Khoa học Máy tính, Trường Đại học Xây dựng Hà Nội. Nội dung giáo trình được xây dựng theo hướng kết hợp chặt chẽ giữa lý thuyết và thực hành, từ những khái niệm cơ bản của lập trình Python đến các ứng dụng trong tính toán số, xử lý dữ liệu và mô hình hóa các bài toán kỹ thuật. Giáo trình không chỉ giúp sinh viên nắm vững cú pháp và cấu trúc của ngôn ngữ Python mà còn hình thành tư duy lập trình và khả năng áp dụng Python như một công cụ giải quyết bài toán thực tế.

Bên cạnh việc trình bày các kiến thức nền tảng như kiểu dữ liệu, điều khiển luồng, hàm và module, giáo trình còn giới thiệu các thư viện quan trọng phục vụ cho tính toán khoa học và kỹ thuật. Các ví dụ minh họa và bài tập được thiết kế theo hướng gần gũi với các bài toán thực tế, giúp sinh viên từng bước làm quen với việc xây dựng mô hình tính toán, thực hiện tính toán số và trực quan hóa kết quả.

Giáo trình được biên soạn với mục tiêu trở thành tài liệu học tập chính thức cho học phần lập trình Python, đồng thời cũng có thể được sử dụng như tài liệu tham khảo cho sinh viên, học viên cao học và những người làm việc trong lĩnh vực khoa học – kỹ thuật có nhu cầu tiếp cận và ứng dụng Python trong công việc nghiên cứu và thực tiễn.

Tác giả xin chân thành cảm ơn các tài liệu, giáo trình và nguồn học thuật trong và ngoài nước đã được tham khảo trong quá trình biên soạn. Mặc dù đã có nhiều cố gắng, giáo trình khó tránh khỏi những thiếu sót, rất mong nhận được các ý kiến đóng góp của đồng nghiệp và người học để tài liệu ngày càng được hoàn thiện hơn.

CHƯƠNG 1. Biến, các kiểu dữ liệu và toán tử cơ bản trong Python

1.1 Mở đầu

Trong mọi ngôn ngữ lập trình, dữ liệu là thành phần trung tâm quyết định cách chương trình hoạt động và giải quyết bài toán. Để làm việc với dữ liệu, người lập trình cần hiểu rõ ba khái niệm nền tảng: biến, kiểu dữ liệu và toán tử. Đây là những viên gạch đầu tiên để xây dựng nên bất kỳ chương trình nào, từ những đoạn mã đơn giản cho đến các hệ thống phần mềm phức tạp trong kỹ thuật và khoa học.

Python là một ngôn ngữ lập trình bậc cao, được thiết kế với mục tiêu đơn giản, dễ đọc và dễ học. Khác với nhiều ngôn ngữ lập trình truyền thống, Python cho phép người học tiếp cận lập trình mà không cần khai báo kiểu dữ liệu một cách tường minh. Tuy nhiên, sự linh hoạt này chỉ thực sự phát huy hiệu quả khi người học hiểu rõ bản chất của các kiểu dữ liệu và cách Python xử lý chúng trong bộ nhớ. Vì vậy, chương này tập trung trình bày một cách chi tiết và có hệ thống các khái niệm cơ bản nhất của Python, làm nền tảng cho toàn bộ nội dung phía sau của giáo trình.

1.2 Biến trong Python

1.2.1 Bản chất của biến

Trong Python, biến không đơn thuần là một vùng nhớ như cách hiểu trong một số ngôn ngữ lập trình khác. Thực chất, biến là một tên gọi được dùng để tham chiếu đến một đối tượng tồn tại trong bộ nhớ. Khi một giá trị được gán cho biến, Python sẽ tạo ra một đối tượng tương ứng trong bộ nhớ và liên kết tên biến với đối tượng đó.

Cách tiếp cận này giúp Python rất linh hoạt, cho phép một biến có thể tham chiếu đến các đối tượng có kiểu dữ liệu khác nhau tại các thời điểm khác nhau trong quá trình thực thi chương trình. Điều này đặc biệt thuận tiện cho việc thử nghiệm, tính toán nhanh và mô phỏng các bài toán kỹ thuật.

1.2.2 Gán giá trị cho biến

Việc tạo biến trong Python được thực hiện thông qua phép gán bằng toán tử `=`. Không cần bất kỳ khai báo nào trước đó, biến sẽ tự động được tạo ra ngay tại thời điểm gán giá trị.

```
x = 10
y = 3.5
message = "Hello Python"
```

Trong ví dụ trên, Python tự động nhận biết rằng `x` là một số nguyên, `y` là một số thực và `message` là một chuỗi ký tự. Cơ chế này được gọi là **suy luận kiểu dữ liệu** (type inference).

1.2.3 Đặt tên biến

Tên biến trong Python không chỉ có ý nghĩa kỹ thuật mà còn ảnh hưởng trực tiếp đến khả năng đọc hiểu và bảo trì chương trình. Một tên biến tốt cần phản ánh đúng ý nghĩa của dữ liệu mà nó lưu trữ. Python cho phép sử dụng chữ cái, chữ số và dấu gạch dưới trong tên biến, đồng thời phân biệt chữ hoa và chữ thường.

```
student_name = "An"
total_score = 85
```

Việc sử dụng những tên biến có ý nghĩa rõ ràng là một thói quen quan trọng trong lập trình, đặc biệt trong các dự án kỹ thuật và nghiên cứu có quy mô lớn.

Bên cạnh các quy tắc đặt tên biến hợp lệ, người học cũng cần nắm rõ những trường hợp đặt tên biến sai thường gặp để tránh lỗi cú pháp ngay từ giai đoạn viết chương trình. Trong Python, các lỗi này sẽ được phát hiện ngay khi chương trình được thực thi, gây gián đoạn quá trình chạy chương trình.

Một lỗi phổ biến là đặt tên biến bắt đầu bằng chữ số. Python không cho phép điều này vì sẽ gây nhầm lẫn với các hằng số số học.

```
2value = 10
```

Đoạn mã trên là không hợp lệ vì tên biến không được phép bắt đầu bằng chữ số. Python sẽ báo lỗi cú pháp ngay khi biên dịch.

Một lỗi thường gặp khác là sử dụng các ký tự đặc biệt như dấu gạch ngang hoặc khoảng trắng trong tên biến.

```
student-name = "An"
total score = 100
```

Trong trường hợp này, dấu gạch ngang bị Python hiểu là toán tử trừ, còn khoảng trắng được coi là dấu phân tách giữa các từ khóa, do đó đều dẫn đến lỗi cú pháp.

Ngoài ra, việc sử dụng từ khóa (keyword) của Python làm tên biến cũng là một lỗi nghiêm trọng. Các từ khóa này đã được Python định nghĩa sẵn cho các mục đích cú pháp đặc biệt.

```
for = 5
class == "CS"
```

Những tên biến trên là không hợp lệ vì `for` và `class` là các từ khóa của ngôn ngữ Python. Để tránh lỗi này, người học nên lựa chọn những tên biến có ý nghĩa, không trùng

với từ khóa và tuân thủ đầy đủ các quy tắc đặt tên.

Việc hiểu rõ các cách đặt tên biến sai không chỉ giúp tránh lỗi cú pháp mà còn hình thành thói quen lập trình đúng đắn, đặc biệt quan trọng trong các dự án kỹ thuật và nghiên cứu có quy mô lớn.

1.3 Các kiểu dữ liệu cơ bản trong Python và đặc điểm bộ nhớ

Trong quá trình lập trình, ngoài việc hiểu ý nghĩa logic của dữ liệu, người lập trình kỹ thuật cần quan tâm đến cách dữ liệu được lưu trữ trong bộ nhớ và phạm vi biểu diễn của chúng. Điều này đặc biệt quan trọng khi xử lý dữ liệu lớn, tính toán khoa học hoặc các bài toán yêu cầu hiệu năng cao.

Python là ngôn ngữ lập trình bậc cao với cơ chế quản lý bộ nhớ tự động. Không giống như các ngôn ngữ bậc thấp, người lập trình Python không cần trực tiếp cấp phát hay giải phóng bộ nhớ. Tuy nhiên, mỗi kiểu dữ liệu trong Python vẫn chiếm một lượng bộ nhớ nhất định và có phạm vi biểu diễn riêng, điều mà người học cần nắm rõ để sử dụng hiệu quả.

1.3.1 Kiểu số nguyên (`int`)

Kiểu số nguyên trong Python được biểu diễn bằng `int`. Khác với các ngôn ngữ như C hoặc C++, Python không giới hạn số nguyên trong một số bit cố định (ví dụ 32-bit hay 64-bit). Thay vào đó, Python sử dụng cơ chế **số nguyên có độ chính xác tùy ý** (arbitrary precision).

Điều này có nghĩa là phạm vi biểu diễn của số nguyên trong Python chỉ bị giới hạn bởi dung lượng bộ nhớ của hệ thống. Tuy nhiên, đi kèm với sự linh hoạt đó là chi phí bộ nhớ lớn hơn.

```
a = 10  
b = 10**100
```

Về mặt bộ nhớ, một số nguyên trong Python không chỉ lưu giá trị mà còn lưu thêm thông tin quản lý đối tượng. Trên hệ thống 64-bit, một số nguyên nhỏ thường chiếm khoảng **28 byte** bộ nhớ, và sẽ tăng thêm khi giá trị lớn hơn.

1.3.2 Kiểu số thực (`float`)

Kiểu số thực trong Python được biểu diễn bằng `float`, tuân theo chuẩn IEEE 754 dạng **double-precision**. Mỗi số thực chiếm **64 bit** (8 byte) để lưu giá trị dấu phẩy động.

```
x = 3.1415926535
```

Phạm vi biểu diễn xấp xỉ của kiểu `float` là từ:

$$\pm 1.8 \times 10^{308}$$

với độ chính xác khoảng **15–16 chữ số thập phân**. Do đặc điểm của biểu diễn dấu phẩy

dòng, các phép toán với `float` có thể phát sinh sai số làm tròn, điều này cần được lưu ý trong các bài toán kỹ thuật chính xác cao.

1.3.3 Kiểu logic (`bool`)

Kiểu logic `bool` chỉ có hai giá trị là `True` và `False`. Về mặt khái niệm, kiểu này chỉ cần 1 bit để biểu diễn. Tuy nhiên, trong Python, `bool` là một đối tượng hoàn chỉnh, do đó nó chiếm nhiều bộ nhớ hơn so với biểu diễn lý thuyết.

```
flag = True
```

Trên hệ thống 64-bit, một giá trị `bool` thường chiếm khoảng **28 byte**. Mặc dù chi phí bộ nhớ lớn, kiểu `bool` đóng vai trò không thể thiếu trong điều khiển luồng chương trình và biểu thức điều kiện.

1.3.4 Kiểu chuỗi (`str`)

Kiểu chuỗi `str` dùng để lưu trữ dữ liệu văn bản. Chuỗi trong Python là một dãy các ký tự Unicode, mỗi ký tự có thể chiếm từ 1 đến 4 byte tùy theo mã Unicode.

```
text = "Python"
```

Dung lượng bộ nhớ của một chuỗi phụ thuộc vào:

- Số lượng ký tự
- Loại ký tự (ASCII hay Unicode mở rộng)
- Thông tin quản lý đối tượng

Do đó, một chuỗi ngắn cũng có thể chiếm vài chục byte bộ nhớ. Chuỗi trong Python là **bất biến** (immutable), nghĩa là mỗi lần thay đổi nội dung, Python sẽ tạo ra một đối tượng mới trong bộ nhớ.

1.3.5 So sánh bộ nhớ và phạm vi biểu diễn của các kiểu dữ liệu cơ bản

Bảng 1.1 tóm tắt đặc điểm bộ nhớ và phạm vi biểu diễn của các kiểu dữ liệu cơ bản trong Python.

Bảng 1.1: So sánh các kiểu dữ liệu cơ bản trong Python

Kiểu dữ liệu	Bộ nhớ (xấp xỉ)	Phạm vi biểu diễn	Ghi chú
<code>int</code>	~28 byte	Không giới hạn	Tăng theo độ lớn giá trị
<code>float</code>	8 byte	$\pm 1.8 \times 10^{308}$	Sai số làm tròn
<code>bool</code>	~28 byte	<code>True / False</code>	Kiểu logic
<code>str</code>	Phụ thuộc độ dài	Unicode	Bất biến

1.3.6 Nhận xét

Từ bảng so sánh có thể thấy rằng Python đánh đổi hiệu năng và bộ nhớ để đổi lấy tính linh hoạt và dễ sử dụng. Đối với các bài toán kỹ thuật có quy mô lớn hoặc yêu cầu tối ưu

tài nguyên, người lập trình cần cân nhắc lựa chọn kiểu dữ liệu phù hợp, cũng như sử dụng các thư viện chuyên dụng như NumPy để tối ưu bộ nhớ và tốc độ tính toán.

1.4 Kết luận

Việc hiểu rõ cách Python lưu trữ dữ liệu trong bộ nhớ và phạm vi biểu diễn của từng kiểu dữ liệu giúp người học sử dụng ngôn ngữ này một cách hiệu quả và khoa học hơn. Đây là kiến thức nền tảng quan trọng để tiếp cận các nội dung nâng cao như cấu trúc dữ liệu, xử lý mảng lớn và tính toán số trong các chương tiếp theo.

1.5 Bài tập

1.5.1 Bài tập 1: Khai báo biến và tính BMI

Viết chương trình nhập vào chiều cao (m) và cân nặng (kg) của một người, sau đó tính chỉ số BMI theo công thức:

$$BMI = \frac{\text{Cân nặng (kg)}}{\text{Chiều cao (m)}^2}$$

Chương trình cần:

- Sử dụng biến để lưu chiều cao, cân nặng và BMI
- In kết quả BMI với 2 chữ số thập phân
- Đưa ra đánh giá dựa trên BMI:
 - $BMI < 18.5$: Thiếu cân
 - $18.5 \leq BMI < 25$: Bình thường
 - $25 \leq BMI < 30$: Thừa cân
 - $BMI \geq 30$: Béo phì

1.5.2 Bài tập 2: Sử dụng toán tử số học

Viết chương trình tính diện tích và chu vi của:

1. Hình chữ nhật: nhập chiều dài và chiều rộng
2. Hình tròn: nhập bán kính (sử dụng $\pi = 3.14159$)

Yêu cầu: Sử dụng các toán tử số học phù hợp và in kết quả với định dạng rõ ràng.

1.5.3 Bài tập 3: Toán tử logic và so sánh

Viết chương trình nhập vào một số nguyên, sau đó kiểm tra và in ra:

- Số đó có phải là số chẵn không?
- Số đó có phải là số dương không?
- Số đó có phải vừa chẵn vừa dương không?

Yêu cầu: Sử dụng toán tử so sánh và toán tử logic (and, or, not).

1.5.4 Bài tập 4: Chuyển đổi kiểu dữ liệu

Viết chương trình chuyển đổi nhiệt độ giữa các đơn vị:

- Celsius sang Fahrenheit: $F = C \times \frac{9}{5} + 32$
- Fahrenheit sang Celsius: $C = (F - 32) \times \frac{5}{9}$
- Celsius sang Kelvin: $K = C + 273.15$

Yêu cầu: Nhập vào giá trị và đơn vị ban đầu, sau đó chuyển đổi sang các đơn vị còn lại.

1.5.5 Bài tập 5: Xử lý chuỗi

Viết chương trình nhập vào một chuỗi ký tự từ bàn phím, sau đó:

1. Đếm số ký tự, số từ trong chuỗi
2. Chuyển đổi chuỗi sang chữ hoa và chữ thường
3. Kiểm tra chuỗi có phải là palindrome không (đọc xuôi và ngược giống nhau)
4. Thay thế tất cả khoảng trắng bằng dấu gạch dưới

Ghi chú: Xem lời giải chi tiết tại Phụ lục - Lời giải bài tập.

CHƯƠNG 2. Kiểu dữ liệu tuần tự

2.1 Tổng quan về kiểu dữ liệu phức hợp trong Python

Trong các chương trước, người học đã làm quen với những kiểu dữ liệu cơ bản như số nguyên, số thực, kiểu logic và chuỗi ký tự. Tuy nhiên, trong thực tế lập trình, đặc biệt là trong các bài toán kỹ thuật và khoa học máy tính, dữ liệu hiếm khi chỉ tồn tại ở dạng một giá trị đơn lẻ. Thông thường, chúng xuất hiện dưới dạng một tập hợp gồm nhiều giá trị có liên quan với nhau, chẳng hạn như danh sách các phép đo, chuỗi tín hiệu theo thời gian, bảng điểm sinh viên hoặc tập hợp các tham số kỹ thuật.

Để biểu diễn và xử lý những tập hợp dữ liệu này, Python cung cấp các kiểu dữ liệu phức hợp. Trong số đó, nhóm kiểu dữ liệu có vai trò đặc biệt quan trọng là các **kiểu dữ liệu tuần tự**. Đây là những kiểu dữ liệu mà các phần tử được lưu trữ theo một thứ tự xác định, cho phép truy cập lần lượt từng phần tử thông qua vị trí của chúng trong tập hợp.

Bên cạnh các kiểu dữ liệu tuần tự, Python còn cung cấp kiểu dữ liệu ánh xạ là `dictionary`. Mặc dù `dictionary` không phải là kiểu tuần tự theo nghĩa truyền thống, nhưng nó lại là một trong những cấu trúc dữ liệu được sử dụng phổ biến nhất trong Python, đặc biệt khi cần biểu diễn dữ liệu có cấu trúc dạng thuộc tính.

Chương này sẽ trình bày một cách hệ thống và chi tiết các kiểu dữ liệu nói trên, tập trung vào bản chất, cách sử dụng và các thao tác thường gặp.

2.2 Khái niệm kiểu dữ liệu tuần tự

Kiểu dữ liệu tuần tự là kiểu dữ liệu trong đó các phần tử được sắp xếp theo một trật tự nhất định. Trật tự này được duy trì trong suốt vòng đời của cấu trúc dữ liệu, trừ khi người lập trình chủ động thay đổi nó. Mỗi phần tử trong tập hợp được gắn với một vị trí cụ thể, gọi là **chỉ mục**.

Đặc điểm quan trọng nhất của kiểu dữ liệu tuần tự là khả năng:

- Truy cập phần tử theo vị trí
- Duyệt lần lượt các phần tử theo thứ tự
- Trích xuất một phần của tập hợp thông qua thao tác cắt lát

Trong Python, ba kiểu dữ liệu tuần tự cơ bản nhất là chuỗi (`str`), danh sách (`list`) và bộ dữ liệu (`tuple`). Mỗi kiểu có những đặc điểm riêng về khả năng thay đổi, hiệu năng và mục đích sử dụng.

2.3 Kiểu dữ liệu String

2.3.1 Bản chất của chuỗi trong Python

Chuỗi ký tự trong Python, được biểu diễn bởi kiểu `str`, là một dãy các ký tự Unicode được lưu trữ theo thứ tự. Mỗi ký tự trong chuỗi là một đơn vị dữ liệu riêng biệt, và toàn bộ chuỗi có thể được xem như một tập hợp tuần tự các ký tự.

```
s = "Python"
```

Trong ví dụ trên, chuỗi "Python" gồm sáu ký tự, được lưu trữ theo thứ tự từ trái sang phải. Python tự động gán cho mỗi ký tự một chỉ mục, bắt đầu từ 0.

2.3.2 Tính bất biến của chuỗi

Một trong những đặc điểm quan trọng nhất của kiểu `str` là tính bất biến. Sau khi chuỗi được tạo ra, nội dung của nó không thể bị thay đổi trực tiếp. Điều này có nghĩa là người lập trình không thể gán một ký tự mới vào một vị trí cụ thể trong chuỗi.

```
s[0] = 'p' # Gây lỗi
```

Bất kỳ thao tác nào làm thay đổi nội dung chuỗi thực chất đều dẫn đến việc tạo ra một đối tượng chuỗi mới trong bộ nhớ. Đặc điểm này giúp chuỗi trở nên an toàn hơn khi sử dụng trong các chương trình lớn, đồng thời tránh được nhiều lỗi khó phát hiện liên quan đến việc thay đổi dữ liệu không mong muốn.

2.3.3 Các thao tác thường gặp với chuỗi

Chuỗi hỗ trợ nhiều phép toán và phương thức phục vụ cho xử lý văn bản. Người lập trình có thể nối chuỗi, lặp chuỗi hoặc kiểm tra sự tồn tại của ký tự trong chuỗi.

```
a = "Lap trinh"  
b = "Python"  
c = a + " " + b
```

Ngoài ra, Python cung cấp nhiều phương thức mạnh để xử lý chuỗi như chuyển đổi chữ hoa, chữ thường, tách chuỗi hoặc loại bỏ khoảng trắng thừa. Những thao tác này rất thường xuyên được sử dụng trong các bài toán xử lý dữ liệu đầu vào.

2.4 Kiểu dữ liệu List

2.4.1 Khái niệm và đặc điểm của List

Danh sách trong Python, được biểu diễn bởi kiểu `list`, là một cấu trúc dữ liệu tuần tự cho phép lưu trữ nhiều phần tử trong cùng một biến. Không giống như chuỗi, danh sách là kiểu dữ liệu **có thể thay đổi**. Điều này có nghĩa là người lập trình có thể thêm, xóa hoặc thay đổi phần tử trong danh sách sau khi danh sách đã được tạo.

```
numbers = [1, 2, 3, 4, 5]
```

Danh sách có thể chứa các phần tử thuộc nhiều kiểu dữ liệu khác nhau, điều này mang lại sự linh hoạt rất lớn trong lập trình Python.

2.4.2 Truy cập và chỉnh sửa phần tử trong List

Các phần tử trong danh sách được truy cập thông qua chỉ mục. Người lập trình có thể thay đổi trực tiếp giá trị của một phần tử bằng cách gán giá trị mới cho vị trí tương ứng.

```
numbers[0] = 10
numbers[-1] = 50
```

Khả năng chỉnh sửa trực tiếp này là điểm khác biệt quan trọng giữa danh sách và các kiểu dữ liệu bất biến như chuỗi và tuple.

2.4.3 Thêm và xóa phần tử trong List

Python cung cấp nhiều phương thức để thao tác với danh sách. Phương thức `append()` cho phép thêm một phần tử mới vào cuối danh sách, trong khi `insert()` cho phép chèn phần tử vào một vị trí bất kỳ.

```
numbers.append(6)
numbers.insert(2, 99)
```

Để xóa phần tử, người lập trình có thể sử dụng các phương thức như `remove()`, `pop()` hoặc câu lệnh `del`. Mỗi phương thức có cách hoạt động và mục đích sử dụng khác nhau.

2.5 Kiểu dữ liệu Tuple

2.5.1 Khái niệm và bản chất của Tuple

Tuple là một cấu trúc dữ liệu dùng để lưu trữ một tập hợp các phần tử theo thứ tự xác định, tương tự như danh sách. Tuy nhiên, điểm khác biệt quan trọng nhất giữa tuple và list là tuple có tính **bất biến** (immutable). Điều này có nghĩa là sau khi tuple được tạo ra, các phần tử của nó không thể bị thay đổi, thêm mới hoặc xóa bỏ.

```
point = (10, 20)
```

Trong ví dụ trên, `point` là một tuple gồm hai phần tử biểu diễn tọa độ trong mặt phẳng. Sau khi được tạo, giá trị của `point` sẽ không thể thay đổi trực tiếp.

2.5.2 Truy cập phần tử trong Tuple

Tuple là kiểu dữ liệu tuần tự, do đó các phần tử của tuple được truy cập thông qua chỉ mục, hoàn toàn tương tự như danh sách và chuỗi.

```
x = point[0]
y = point[1]
```

Python cũng hỗ trợ chỉ mục âm khi làm việc với tuple, cho phép truy cập phần tử từ

cuối về đầu.

2.5.3 Tính bất biến và ý nghĩa thực tiễn

Tính bất biến của tuple mang lại nhiều lợi ích trong lập trình:

- Dữ liệu an toàn hơn, tránh bị thay đổi ngoài ý muốn
- Có thể sử dụng tuple làm khóa trong dictionary
- Hiệu năng tốt hơn so với list trong một số trường hợp

Tuple thường được sử dụng để:

- Lưu trữ các nhóm dữ liệu có cấu trúc cố định (tọa độ, thông số)
- Trả về nhiều giá trị từ một hàm
- Biểu diễn các bản ghi dữ liệu không thay đổi

2.5.4 Kiểu dữ liệu Dictionary

2.5.5 Bản chất của Dictionary

Dictionary (từ điển) là kiểu dữ liệu dùng để lưu trữ dữ liệu dưới dạng **ánh xạ** giữa khóa và giá trị. Mỗi phần tử trong dictionary bao gồm một cặp key:value, trong đó khóa là duy nhất và được dùng để truy cập nhanh đến giá trị tương ứng.

```
student = {  
    "name": "An",  
    "age": 20,  
    "score": 8.5  
}
```

Khác với các kiểu dữ liệu tuần tự, dictionary không truy cập dữ liệu thông qua chỉ mục mà thông qua khóa.

2.5.6 Truy cập và thay đổi dữ liệu trong Dictionary

Để truy cập giá trị trong dictionary, người lập trình sử dụng khóa tương ứng.

```
print(student["name"])
```

Dictionary là kiểu dữ liệu **có thể thay đổi**. Do đó, người lập trình có thể thêm mới, thay đổi hoặc xóa các cặp khóa–giá trị.

```
student["age"] = 21  
student["major"] = "Computer Science"
```

2.5.7 Duyệt Dictionary

Python cho phép duyệt dictionary theo nhiều cách khác nhau, chẳng hạn như duyệt qua các khóa, giá trị hoặc từng cặp khóa–giá trị.

```
for key, value in student.items():
    print(key, value)
```

Dictionary đặc biệt phù hợp để biểu diễn các đối tượng trong thế giới thực, nơi mỗi đối tượng có nhiều thuộc tính khác nhau.

2.5.8 Ứng dụng thực tế của Dictionary

Trong các bài toán kỹ thuật và khoa học máy tính, dictionary thường được sử dụng để:

- Lưu trữ thông tin đối tượng (sinh viên, thiết bị, cảm biến)
- Biểu diễn cấu hình hệ thống
- Tăng tốc tìm kiếm và tra cứu dữ liệu

2.6 Kiểu dữ liệu Set

2.6.1 Khái niệm và đặc điểm của Set

Set là kiểu dữ liệu dùng để lưu trữ một tập hợp các phần tử **không trùng lặp** và **không có thứ tự**. Điều này có nghĩa là set không duy trì thứ tự các phần tử và không cho phép tồn tại hai phần tử giống nhau.

```
numbers = {1, 2, 3, 4}
```

Nếu thêm một phần tử đã tồn tại, set sẽ không thay đổi.

2.6.2 Thao tác cơ bản với Set

Set hỗ trợ các thao tác thêm và xóa phần tử.

```
numbers.add(5)
numbers.remove(3)
```

Do không có thứ tự, set không hỗ trợ truy cập phần tử thông qua chỉ mục.

2.6.3 Các phép toán tập hợp

Một ưu điểm rất lớn của set là hỗ trợ trực tiếp các phép toán tập hợp như hợp, giao và hiệu.

```

A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

union_set = A | B
intersection_set = A & B
difference_set = A - B

```

Những phép toán này rất hữu ích trong các bài toán xử lý dữ liệu, phân tích tập mẫu và loại bỏ trùng lặp.

2.6.4 Ứng dụng của Set

Set thường được sử dụng khi:

- Cần loại bỏ các giá trị trùng lặp
- Kiểm tra nhanh sự tồn tại của phần tử
- Thực hiện các phép toán tập hợp

Trong nhiều bài toán kỹ thuật, set giúp chương trình trở nên ngắn gọn và hiệu quả hơn so với việc sử dụng danh sách.

2.7 Chỉ mục trong kiểu dữ liệu tuần tự

2.7.1 Vai trò của chỉ mục

Trong các kiểu dữ liệu tuần tự như chuỗi (str), danh sách (list) và bộ dữ liệu (tuple), các phần tử không chỉ được lưu trữ một cách ngẫu nhiên mà được sắp xếp theo một thứ tự xác định. Để truy cập từng phần tử cụ thể trong tập hợp đó, Python sử dụng khái niệm **chỉ mục** (index).

Chỉ mục có thể được hiểu là vị trí của phần tử trong cấu trúc dữ liệu tuần tự. Thông qua chỉ mục, người lập trình có thể:

- Truy cập một phần tử cụ thể
- Thay đổi giá trị phần tử (đối với kiểu dữ liệu có thể thay đổi)
- Thực hiện các thao tác cắt lát và duyệt dữ liệu

Việc hiểu đúng và sử dụng chính xác chỉ mục là nền tảng để làm việc hiệu quả với các kiểu dữ liệu tuần tự trong Python.

2.7.2 Chỉ mục dương trong Python

Python sử dụng hệ thống chỉ mục bắt đầu từ 0. Điều này có nghĩa là phần tử đầu tiên trong cấu trúc dữ liệu có chỉ mục là 0, phần tử thứ hai có chỉ mục là 1, và cứ tiếp tục như vậy cho đến phần tử cuối cùng.

Xét ví dụ danh sách sau:

```
data = [10, 20, 30, 40, 50]
```

Cách Python gán chỉ mục dương cho các phần tử có thể được minh họa như sau:

Chỉ mục	0	1	2	3	4
Giá trị	10	20	30	40	50

Khi sử dụng chỉ mục dương, Python sẽ truy cập phần tử tính từ đầu tập hợp.

```
print(data[0])    # 10
print(data[2])    # 30
print(data[4])    # 50
```

Cách đánh chỉ mục từ 0 này giúp việc tính toán độ dài và cắt lát dữ liệu trở nên thuận tiện và nhất quán.

2.7.3 Chỉ mục âm trong Python

Bên cạnh chỉ mục dương, Python còn hỗ trợ **chỉ mục âm**. Chỉ mục âm cho phép truy cập các phần tử từ cuối cấu trúc dữ liệu về đầu. Phần tử cuối cùng có chỉ mục là -1 , phần tử đứng trước nó có chỉ mục là -2 , và cứ tiếp tục như vậy.

Với danh sách `data` ở trên, cách Python gán chỉ mục âm có thể được minh họa như sau:

Chỉ mục âm	-5	-4	-3	-2	-1
Giá trị	10	20	30	40	50

Ví dụ truy cập bằng chỉ mục âm:

```
print(data[-1])    # 50
print(data[-2])    # 40
print(data[-5])    # 10
```

Chỉ mục âm đặc biệt hữu ích khi người lập trình không biết trước độ dài của tập hợp hoặc khi cần thao tác với các phần tử gần cuối, chẳng hạn như phần tử cuối cùng hoặc hai phần tử cuối.

2.7.4 So sánh chỉ mục dương và chỉ mục âm

Mặc dù chỉ mục dương và chỉ mục âm có cách đánh số khác nhau, nhưng chúng đều tham chiếu đến cùng một phần tử trong cấu trúc dữ liệu. Ví dụ, với danh sách `data`:

```
data[0] == data[-5]
data[1] == data[-4]
data[4] == data[-1]
```

Việc hiểu rõ mối quan hệ này giúp người lập trình linh hoạt hơn trong việc truy cập dữ

liệu và tránh nhầm lẫn khi viết chương trình.

2.7.5 Chỉ mục và độ dài dữ liệu

Độ dài của một cấu trúc dữ liệu tuần tự trong Python có thể được xác định bằng hàm `len()`. Nếu một cấu trúc dữ liệu có độ dài là n , thì:

- Chỉ mục dương hợp lệ nằm trong khoảng từ 0 đến $n - 1$
- Chỉ mục âm hợp lệ nằm trong khoảng từ $-n$ đến -1

```
n = len(data)
print(n) # 5
```

Việc truy cập chỉ mục nằm ngoài các khoảng này sẽ gây ra lỗi.

2.7.6 Lỗi truy cập ngoài phạm vi chỉ mục

Một trong những lỗi phổ biến nhất khi làm việc với kiểu dữ liệu tuần tự là lỗi truy cập ngoài phạm vi chỉ mục (`IndexError`).

```
print(data[5])      # Lỗi
print(data[-6])     # Lỗi
```

Lỗi này xảy ra khi chỉ mục được sử dụng không tồn tại trong cấu trúc dữ liệu. Để tránh lỗi, người lập trình cần:

- Kiểm tra độ dài dữ liệu trước khi truy cập
- Sử dụng vòng lặp thay vì truy cập trực tiếp khi không chắc chắn về phạm vi chỉ mục

2.7.7 Chỉ mục trong các kiểu dữ liệu tuần tự khác nhau

Cơ chế chỉ mục trong Python hoạt động thống nhất cho tất cả các kiểu dữ liệu tuần tự.

Với chuỗi:

```
text = "Python"
print(text[0])    # P
print(text[-1])   # n
```

Với tuple:

```
point = (10, 20, 30)
print(point[1])    # 20
print(point[-1])   # 30
```

Điều này giúp người học dễ dàng chuyển đổi tư duy giữa các kiểu dữ liệu khác nhau mà không phải học lại cách sử dụng chỉ mục.

2.8 Duyệt các kiểu dữ liệu tuần tự

Duyệt là thao tác truy cập lần lượt từng phần tử trong một cấu trúc dữ liệu tuần tự. Đây là thao tác nền tảng trong xử lý dữ liệu và xây dựng thuật toán.

```
for value in data:
    print(value)
```

Vòng lặp `for` trong Python được thiết kế đặc biệt phù hợp cho việc duyệt các cấu trúc dữ liệu tuần tự, giúp mã nguồn ngắn gọn và dễ đọc.

2.9 Cắt lát dữ liệu (Slicing) trong các kiểu dữ liệu tuần tự

Trong quá trình làm việc với các kiểu dữ liệu tuần tự như chuỗi, danh sách và tuple, người lập trình thường không chỉ quan tâm đến việc truy cập một phần tử đơn lẻ mà còn cần trích xuất một **đoạn liên tiếp các phần tử**. Thao tác này được gọi là **cắt lát dữ liệu** (slicing) và là một trong những cơ chế mạnh mẽ, linh hoạt nhất của Python.

Cắt lát cho phép tạo ra một cấu trúc dữ liệu mới từ một phần của cấu trúc dữ liệu ban đầu, mà không làm thay đổi dữ liệu gốc. Cơ chế này đặc biệt hữu ích trong các bài toán xử lý chuỗi, phân tích dữ liệu, xử lý tín hiệu và các bài toán kỹ thuật liên quan đến dãy số theo thời gian.

2.9.1 Cú pháp tổng quát của cắt lát

Cú pháp cắt lát trong Python có dạng tổng quát như sau:

```
sequence[start : stop : step]
```

Trong đó:

- `start` là chỉ mục bắt đầu (bao gồm phần tử tại vị trí này)
- `stop` là chỉ mục kết thúc (không bao gồm phần tử tại vị trí này)
- `step` là bước nhảy giữa các phần tử

Ba thành phần này đều là **tùy chọn**. Nếu không được chỉ định, Python sẽ tự động sử dụng các giá trị mặc định phù hợp.

2.9.2 Cắt lát cơ bản với chỉ số bắt đầu và kết thúc

Trường hợp đơn giản nhất của cắt lát là chỉ định vị trí bắt đầu và kết thúc. Python sẽ trích xuất tất cả các phần tử nằm trong khoảng từ `start` đến `stop - 1`.

```
data = [10, 20, 30, 40, 50, 60]
sub = data[1:4]
print(sub)
```

Trong ví dụ trên, kết quả thu được là danh sách `[20, 30, 40]`. Phần tử ở vị trí 4

không được lấy vì stop không bao gồm điểm kết thúc.

Cách đánh chỉ mục này giúp tránh chồng lấn khi ghép các đoạn dữ liệu liên tiếp, một kỹ thuật thường được sử dụng trong xử lý mảng và thuật toán.

2.9.3 Cắt lát khi bỏ qua chỉ mục bắt đầu hoặc kết thúc

Nếu bỏ qua chỉ mục bắt đầu, Python sẽ mặc định lấy từ đầu cấu trúc dữ liệu. Nếu bỏ qua chỉ mục kết thúc, Python sẽ lấy đến phần tử cuối cùng.

```
data = [10, 20, 30, 40, 50]

first_part = data[:3]
last_part = data[2:]
```

Trong ví dụ này, first_part chứa ba phần tử đầu tiên, trong khi last_part chứa các phần tử từ vị trí 2 đến hết. Cách sử dụng này rất phổ biến khi cần tách dữ liệu thành các phần đầu và cuối.

2.9.4 Cắt lát với bước nhảy (step)

Thành phần step cho phép xác định khoảng cách giữa các phần tử được lấy. Nếu step = 1, Python sẽ lấy các phần tử liên tiếp. Nếu step > 1, Python sẽ bỏ qua một số phần tử theo bước nhảy đã chỉ định.

```
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
even_index = data[::-2]
```

Kết quả của ví dụ trên là danh sách các phần tử ở vị trí chẵn. Cơ chế bước nhảy này rất hữu ích khi cần lấy mẫu dữ liệu (sampling) trong các bài toán kỹ thuật và khoa học.

2.9.5 Cắt lát với chỉ số âm

Python cho phép sử dụng chỉ số âm để đếm từ cuối cấu trúc dữ liệu về đầu. Cơ chế này cũng áp dụng đầy đủ cho cắt lát.

```
data = [10, 20, 30, 40, 50]
sub = data[-4:-1]
```

Trong trường hợp này, Python sẽ lấy các phần tử từ vị trí thứ tư tính từ cuối cho đến trước phần tử cuối cùng. Việc sử dụng chỉ số âm giúp thao tác thuận tiện hơn khi làm việc với các đoạn dữ liệu gần cuối.

2.9.6 Đảo ngược dữ liệu bằng cắt lát

Một ứng dụng rất đặc biệt và thường gặp của cắt lát là đảo ngược thứ tự của cấu trúc dữ liệu bằng cách sử dụng bước nhảy âm.

```
data = [1, 2, 3, 4, 5]
reversed_data = data[::-1]
```

Chỉ với một dòng lệnh, toàn bộ danh sách đã được đảo ngược. Đây là một ví dụ điển hình cho thấy sức mạnh và tính ngắn gọn của cú pháp cắt lát trong Python.

2.9.7 Cắt lát với chuỗi

Cắt lát không chỉ áp dụng cho danh sách mà còn hoạt động hoàn toàn tương tự đối với chuỗi.

```
text = "LapTrinhPython"
sub_text = text[3:8]
```

Do chuỗi là kiểu dữ liệu bất biến, thao tác cắt lát sẽ luôn tạo ra một chuỗi mới, giữ nguyên chuỗi ban đầu. Điều này đặc biệt hữu ích trong xử lý văn bản và phân tích dữ liệu đầu vào.

2.9.8 Cắt lát với Tuple

Tuple cũng hỗ trợ cắt lát giống như danh sách và chuỗi.

```
point = (10, 20, 30, 40)
sub_point = point[1:3]
```

Kết quả của phép cắt lát là một tuple mới. Mặc dù tuple là bất biến, nhưng việc trích xuất một phần dữ liệu từ tuple vẫn hoàn toàn hợp lệ.

2.9.9 Những lưu ý quan trọng khi sử dụng cắt lát

Mặc dù cắt lát rất tiện lợi nhưng chúng ta cần lưu ý rằng:

- Cắt lát luôn tạo ra một bản sao dữ liệu mới
- Với các tập dữ liệu lớn, việc cắt lát nhiều lần có thể làm tăng tiêu thụ bộ nhớ
- Cần cẩn thận với chỉ số để tránh nhầm lẫn về phạm vi dữ liệu được trích xuất

Việc hiểu rõ cơ chế cắt lát không chỉ giúp viết mã ngắn gọn hơn mà còn giúp tối ưu hóa chương trình và tránh các lỗi logic khó phát hiện.

2.9.10 Ý nghĩa của cắt lát trong lập trình kỹ thuật

Trong các bài toán kỹ thuật, cắt lát thường được sử dụng để xử lý các dãy số đo, phân đoạn tín hiệu, chia dữ liệu thành các khung thời gian hoặc trích xuất các vùng quan tâm trong dữ liệu. Do đó, việc nắm vững thao tác cắt lát là kỹ năng quan trọng đối với sinh viên và kỹ sư khi sử dụng Python như một công cụ tính toán và mô phỏng.

Chương này đã trình bày chi tiết các kiểu dữ liệu tuần tự trong Python, bao gồm chuỗi, danh sách và tuple, cùng với các thao tác quan trọng như truy cập, duyệt và cắt lát. Đồng

thời, kiểu dữ liệu dictionary cũng được giới thiệu như một cấu trúc dữ liệu ánh xạ quan trọng. Việc nắm vững các kiểu dữ liệu này là nền tảng thiết yếu để tiếp cận các nội dung nâng cao hơn như thuật toán, cấu trúc dữ liệu nâng cao và xử lý dữ liệu trong các ứng dụng kỹ thuật.

2.10 Bài tập luyện tập

Phần bài tập này nhằm giúp sinh viên củng cố và vận dụng các kiến thức đã học về kiểu dữ liệu tuple, dictionary, set cũng như cách truy cập, duyệt và thao tác trên các cấu trúc dữ liệu này trong Python. Các bài tập được sắp xếp theo mức độ từ cơ bản đến nâng cao, phù hợp cho việc tự học và rèn luyện kỹ năng lập trình.

2.10.1 Câu hỏi trắc nghiệm

1. Phát biểu nào sau đây là đúng về kiểu dữ liệu tuple?
 - (a) Tuple có thể thay đổi kích thước sau khi khởi tạo
 - (b) Tuple là kiểu dữ liệu bất biến
 - (c) Tuple không thể chứa các kiểu dữ liệu khác nhau
 - (d) Tuple không hỗ trợ chỉ mục
2. Chỉ mục hợp lệ để truy cập phần tử cuối cùng của một danh sách có độ dài n là:
 - (a) n
 - (b) $n - 1$
 - (c) $-n$
 - (d) 0
3. Kiểu dữ liệu nào sau đây không cho phép các phần tử trùng lặp?
 - (a) list
 - (b) tuple
 - (c) dictionary
 - (d) set
4. Trong dictionary, khóa (key) phải là:
 - (a) Kiểu dữ liệu có thể thay đổi
 - (b) Kiểu dữ liệu bất biến
 - (c) Luôn là chuỗi
 - (d) Có thể trùng lặp
5. Đoạn mã nào sau đây sẽ gây lỗi?
 - (a) $t = (1, 2, 3)$

- (b) `t[0] = 10`
- (c) `d = {"a": 1, "b": 2}`
- (d) `s = {1, 2, 3}`

6. Kết quả của đoạn mã sau là gì?

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A & B)
```

7. Phương thức nào dùng để duyệt cả khóa và giá trị trong dictionary?

- (a) `keys()`
- (b) `values()`
- (c) `items()`
- (d) `get()`

2.11 Bài tập thực hành

Bài tập 1: Làm quen với Tuple

Viết chương trình tạo một tuple lưu trữ thông tin của một điểm trong mặt phẳng tọa độ gồm hoành độ và tung độ. Thực hiện các yêu cầu sau:

- In ra hoành độ và tung độ của điểm
- Tính khoảng cách từ điểm đó đến gốc tọa độ (0, 0)

Bài tập 2: Truy cập và duyệt Tuple

Cho tuple sau:

```
data = (5, 10, 15, 20, 25)
```

1. In ra phần tử đầu tiên và phần tử cuối cùng
2. Duyệt tuple bằng vòng lặp `for` và in ra từng phần tử
3. Tính tổng các phần tử trong tuple

Bài tập 3: Dictionary cơ bản

Viết chương trình quản lý thông tin sinh viên bằng dictionary, trong đó mỗi sinh viên có các thuộc tính: mã sinh viên, họ tên, tuổi và điểm trung bình.

Yêu cầu:

- Khởi tạo dictionary lưu trữ thông tin một sinh viên
- In ra toàn bộ thông tin sinh viên

- Cho phép cập nhật điểm trung bình

Bài tập 4: Duyệt và xử lý Dictionary

Cho dictionary lưu trữ điểm các môn học:

```
scores = {  
    "Toan": 8.0,  
    "Ly": 7.5,  
    "Hoa": 8.5,  
    "Tin": 9.0  
}
```

1. Tính điểm trung bình
2. Tìm môn học có điểm cao nhất
3. In ra danh sách các môn có điểm lớn hơn hoặc bằng 8

Bài tập 5: Làm việc với Set

Viết chương trình nhập vào hai tập hợp số nguyên từ bàn phím. Thực hiện:

- Tính tập hợp các phần tử chung
- Tính tập hợp các phần tử chỉ xuất hiện ở tập thứ nhất
- Kiểm tra một số cho trước có thuộc tập hợp hay không

Bài tập 6: Loại bỏ trùng lặp

Cho một danh sách các số nguyên có thể chứa nhiều giá trị trùng nhau. Sử dụng kiểu dữ liệu set để:

- Loại bỏ các phần tử trùng lặp
- Chuyển kết quả về danh sách
- Sắp xếp danh sách theo thứ tự tăng dần

Bài tập 7: Ứng dụng tổng hợp

Viết chương trình quản lý danh sách thiết bị trong phòng thí nghiệm, trong đó:

- Mỗi thiết bị được biểu diễn bằng một dictionary gồm: mã thiết bị, tên thiết bị, trạng thái
- Danh sách thiết bị được lưu trong một list

Yêu cầu:

1. In ra danh sách toàn bộ thiết bị
2. Đếm số thiết bị đang hoạt động
3. Tìm thiết bị theo mã

Bài tập 8: Bài tập nâng cao

1. Viết chương trình đọc một đoạn văn bản, tách các từ và sử dụng set để đếm số từ khác nhau.
2. Sử dụng dictionary để thống kê tần suất xuất hiện của từng ký tự trong một chuỗi.
3. Kết hợp tuple, list và dictionary để xây dựng cấu trúc dữ liệu lưu trữ bảng điểm của một lớp học.

CHƯƠNG 3. Nhập xuất dữ liệu trong Python

Trong hầu hết các chương trình máy tính, dữ liệu không chỉ được xử lý nội bộ mà còn cần được nhập từ bên ngoài và xuất ra cho người sử dụng hoặc lưu trữ lâu dài. Đối với sinh viên ngành Khoa học Máy tính, việc nắm vững các kỹ thuật nhập xuất dữ liệu trong Python là nền tảng quan trọng để xây dựng các chương trình hoàn chỉnh, có khả năng tương tác và xử lý dữ liệu thực tế.

Chương này trình bày chi tiết các phương pháp nhập xuất dữ liệu trong Python, bao gồm xuất dữ liệu ra màn hình, nhập dữ liệu từ bàn phím và làm việc với tập tin.

3.1 Xuất dữ liệu ra màn hình

3.1.1 Hàm print()

Hàm print() là công cụ cơ bản nhất để hiển thị dữ liệu ra màn hình. Hàm này có thể in ra các kiểu dữ liệu khác nhau như chuỗi, số nguyên, số thực và cả các cấu trúc dữ liệu.

```
print("Hello Python")
print(2025)
print(3.14)
```

Kết quả hiển thị:

Hello Python
2025
3.14

Python tự động chuyển đổi dữ liệu sang dạng chuỗi trước khi in ra màn hình, giúp người lập trình không cần xử lý thủ công.

3.1.2 In nhiều dữ liệu cùng lúc

Hàm print() cho phép in nhiều giá trị trong cùng một lệnh.

```
name = "An"
age = 20
score = 8.5
print("Ten:", name, "- Tuoi:", age, "- Diem:", score)
```

Kết quả hiển thị:

Ten: An - Tuoi: 20 - Diem: 8.5

3.1.3 Tùy chỉnh định dạng khi in

a, Sử dụng `sep` và `end`

```
print(1, 2, 3, sep=" | ")
print("Xin", end=" ")
print("chao Python")
```

Kết quả hiển thị:

1 | 2 | 3
Xin chao Python

b, Định dạng bằng f-string

F-string cho phép chèn biến trực tiếp vào chuỗi và định dạng dữ liệu một cách linh hoạt.

```
a = 7
b = 3
print(f"{a} + {b} = {a + b}")
```

Kết quả hiển thị:

7 + 3 = 10

Định dạng số thực:

```
pi = 3.14159265
print(f"Gia tri pi lam tron: {pi:.3f}")
```

Kết quả hiển thị:

Gia tri pi lam tron: 3.142

3.1.4 In dữ liệu dạng bảng

Trong các bài toán kỹ thuật, dữ liệu thường được trình bày dưới dạng bảng để dễ so sánh và phân tích.

```
print(f"{'Mon hoc':<10}{{'Diem':>6}}")
print(f"{'-'*16}")
print(f"{'Toan':<10}{8.5:>6}")
print(f"{'Ly':<10}{7.5:>6}")
print(f"{'Tin':<10}{9.0:>6}")
```

Kết quả hiển thị:

Mon hoc Diem

Toan	8.5
Lý	7.5
Tin	9.0

3.2 Nhập dữ liệu từ bàn phím

3.2.1 Hàm input()

Hàm `input()` cho phép chương trình nhận dữ liệu từ người dùng thông qua bàn phím.

```
name = input("Nhập tên: ")  
print("Xin chào", name)
```

Ví dụ nhập:

Nhập tên: Minh

Kết quả hiển thị:

Xin chào Minh

3.2.2 Nhập và chuyển đổi kiểu dữ liệu

Dữ liệu nhập vào từ bàn phím luôn có kiểu chuỗi, do đó cần chuyển đổi sang kiểu phù hợp.

```
age = int(input("Nhập tuổi: "))  
height = float(input("Nhập chiều cao (m): "))  
print(f"Tuổi: {age}, Chiều cao: {height} m")
```

Kết quả hiển thị (ví dụ):

Nhập tuổi: 20

Nhập chiều cao (m): 1.68

Tuổi: 20, Chiều cao: 1.68 m

3.3 Nhập xuất dữ liệu với tập tin

3.3.1 Ví dụ tập tin dữ liệu

Giả sử có tập tin `scores.txt` với nội dung sau:

SV01, Toan, 8.5
SV01, Lý, 7.5
SV01, Tin, 9.0
SV02, Toan, 6.5
SV02, Lý, 7.0
SV02, Tin, 8.0

Tập tin này lưu trữ dữ liệu điểm của sinh viên, mỗi dòng gồm mã sinh viên, tên môn học và điểm.

3.3.2 Đọc tập tin và xử lý dữ liệu

```
total = 0
count = 0

with open("scores.txt", "r") as f:
    for line in f:
        parts = line.strip().split(",")
        score = float(parts[2])
        total += score
        count += 1

average = total / count
print(f"Diem trung binh tat ca cac mon: {average:.2f}")
```

Kết quả hiển thị:

Diem trung binh tat ca cac mon: 7.75

3.3.3 Ghi kết quả ra tập tin

```
with open("result.txt", "w") as f:
    f.write(f"Diem trung binh: {average:.2f}\n")
```

Sau khi chạy chương trình, tập tin result.txt sẽ chứa kết quả tính toán.

3.4 Các phương thức làm việc với tập tin trong Python

Khi mở một tập tin bằng hàm `open()`, Python trả về một đối tượng file. Đối tượng này cung cấp nhiều phương thức cho phép người lập trình thao tác linh hoạt với tập tin như đọc, ghi, di chuyển con trỏ, kiểm tra trạng thái và quản lý bộ nhớ đệm.

Việc hiểu rõ các phương thức này giúp chương trình làm việc với tập tin an toàn, hiệu quả và phù hợp với các ứng dụng thực tế.

3.4.1 `close()`

Phương thức `close()` dùng để đóng một tập tin đang mở. Sau khi tập tin đã được đóng, mọi thao tác đọc hoặc ghi lên tập tin đó đều không còn hợp lệ.

```
f = open("example.txt", "w")
f.write("Hello Python")
f.close()
```

Sau khi gọi `close()`, dữ liệu được ghi hoàn toàn xuống tập tin và tài nguyên hệ thống

được giải phóng.

3.4.2 `fileno()`

Phương thức `fileno()` trả về một số nguyên đại diện cho file descriptor – một khái niệm ở mức hệ điều hành dùng để quản lý các luồng vào ra.

```
f = open("example.txt", "r")
print(f.fileno())
f.close()
```

Kết quả hiển thị (ví dụ):

3

Giá trị trả về có thể khác nhau tùy vào hệ thống và thời điểm chạy chương trình.

3.4.3 `flush()`

Phương thức `flush()` dùng để xóa sạch bộ nhớ đệm của luồng file, đảm bảo dữ liệu được ghi ngay xuống tập tin mà không cần chờ đóng file.

```
f = open("example.txt", "w")
f.write("Du lieu tam thoi")
f.flush()
```

Phương thức này đặc biệt quan trọng trong các chương trình ghi log hoặc hệ thống thời gian thực.

3.4.4 `isatty()`

Phương thức `isatty()` trả về `True` nếu tập tin được kết nối với một thiết bị đầu cuối (terminal), ngược lại trả về `False`.

```
f = open("example.txt", "r")
print(f.isatty())
f.close()
```

Kết quả hiển thị:

`False`

3.4.5 `read(n)`

Phương thức `read(n)` dùng để đọc tối đa `n` ký tự từ tập tin, tính từ vị trí con trỏ hiện tại.

```
with open("example.txt", "r") as f:
    content = f.read(5)
    print(content)
```

Kết quả hiển thị (ví dụ):

Hello

3.4.6 readable()

Phương thức `readable()` kiểm tra xem tập tin có thể đọc được hay không.

```
f = open("example.txt", "r")
print(f.readable())
f.close()
```

Kết quả hiển thị:

True

3.4.7 readline(n=-1)

Phương thức `readline()` đọc và trả về một dòng từ tập tin. Nếu tham số `n` được chỉ định, phương thức sẽ đọc tối đa `n` ký tự.

```
with open("scores.txt", "r") as f:
    line = f.readline()
    print(line)
```

Kết quả hiển thị (ví dụ):

SV01, Toan, 8.5

3.4.8 readlines(n=-1)

Phương thức `readlines()` đọc toàn bộ các dòng trong tập tin và trả về dưới dạng một danh sách chuỗi.

```
with open("scores.txt", "r") as f:
    lines = f.readlines()
    print(lines)
```

Kết quả hiển thị (ví dụ):

['SV01, Toan, 8.5\n', 'SV01, Ly, 7.5\n', 'SV01, Tin, 9.0\n']

3.4.9 seek(offset, from=SEEK_SET)

Phương thức `seek()` dùng để thay đổi vị trí con trỏ đọc/ghi trong tập tin.

```
with open("example.txt", "r") as f:  
    f.seek(6)  
    print(f.read())
```

Phương thức này thường được dùng khi cần truy cập ngẫu nhiên dữ liệu trong tập tin.

3.4.10 seekable()

Phương thức seekable() trả về True nếu luồng file hỗ trợ truy cập ngẫu nhiên.

```
f = open("example.txt", "r")  
print(f.seekable())  
f.close()
```

3.4.11 tell()

Phương thức tell() trả về vị trí hiện tại của con trỏ trong tập tin.

```
with open("example.txt", "r") as f:  
    print(f.tell())  
    f.read(5)  
    print(f.tell())
```

Kết quả hiển thị (ví dụ):

```
0  
5
```

3.4.12 truncate(size=None)

Phương thức truncate() dùng để cắt gọn kích thước tập tin về số byte được chỉ định.

```
with open("example.txt", "r+") as f:  
    f.truncate(5)
```

Sau khi thực hiện, nội dung tập tin chỉ còn 5 ký tự đầu tiên.

3.4.13 writable()

Phương thức writable() kiểm tra xem tập tin có cho phép ghi hay không.

```
f = open("example.txt", "w")  
print(f.writable())  
f.close()
```

Kết quả hiển thị:

True

3.4.14 `write(s)`

Phương thức `write()` ghi chuỗi s vào tập tin và trả về số ký tự đã ghi.

```
with open("example.txt", "w") as f:
    n = f.write("Python File I/O")
    print(n)
```

Kết quả hiển thị:

15

3.4.15 `writelines(lines)`

Phương thức `writelines()` ghi một danh sách các chuỗi vào tập tin.

```
lines = ["Dòng 1\n", "Dòng 2\n", "Dòng 3\n"]
with open("example.txt", "w") as f:
    f.writelines(lines)
```

Sau khi chạy, tập tin sẽ chứa toàn bộ các dòng trong danh sách.

3.5 Nhận xét

Các phương thức làm việc với tập tin trong Python cung cấp khả năng kiểm soát chi tiết quá trình đọc, ghi và quản lý dữ liệu. Việc sử dụng đúng phương thức không chỉ giúp chương trình hoạt động chính xác mà còn nâng cao hiệu suất và độ tin cậy của hệ thống, đặc biệt trong các ứng dụng kỹ thuật và xử lý dữ liệu lớn.

3.6 Ứng dụng thực tế

Trong thực tế kỹ thuật và công nghệ thông tin, nhập xuất dữ liệu đóng vai trò then chốt trong hầu hết các hệ thống phần mềm.

Trong các hệ thống thu thập dữ liệu cảm biến, Python thường được sử dụng để đọc dữ liệu từ tập tin log hoặc từ thiết bị đo, xử lý và ghi lại kết quả phân tích. Trong các bài toán quản lý, dữ liệu sinh viên, thiết bị hoặc tài nguyên thường được nhập từ tập tin, xử lý và xuất ra dưới dạng báo cáo.

Đối với sinh viên ngành Khoa học Máy tính, việc thành thạo nhập xuất dữ liệu giúp hình thành tư duy xây dựng chương trình hoàn chỉnh, có khả năng tương tác với người dùng và làm việc với dữ liệu thực tế, tạo tiền đề cho các học phần nâng cao như xử lý dữ liệu, trí tuệ nhân tạo và hệ thống thông tin.

3.7 Bài tập chương

3.7.1 Câu hỏi trắc nghiệm

1. Phát biểu nào sau đây là đúng về hàm `print()` trong Python?

- (a) Chỉ in được dữ liệu kiểu chuỗi
 - (b) Không thể in nhiều giá trị trong một lệnh
 - (c) Tự động chuyển dữ liệu sang dạng chuỗi khi in
 - (d) Luôn kết thúc bằng ký tự tab
2. Tham số nào của hàm `print()` dùng để thay đổi ký tự phân cách giữa các giá trị?
- (a) `end`
 - (b) `sep`
 - (c) `format`
 - (d) `split`
3. Giá trị trả về của hàm `input()` có kiểu dữ liệu là:
- (a) `int`
 - (b) `float`
 - (c) `str`
 - (d) Phụ thuộc vào dữ liệu nhập
4. Phương thức nào dùng để đọc toàn bộ nội dung của tập tin?
- (a) `readline()`
 - (b) `read()`
 - (c) `readlines()`
 - (d) `write()`
5. Khi mở tập tin ở chế độ "`w`", điều gì sẽ xảy ra nếu tập tin đã tồn tại?
- (a) Nội dung cũ được giữ nguyên
 - (b) Ghi tiếp vào cuối tập tin
 - (c) Nội dung cũ bị xóa
 - (d) Phát sinh lỗi
6. Phương thức nào dùng để di chuyển con trỏ đọc/ghi trong tập tin?
- (a) `tell()`
 - (b) `seek()`
 - (c) `flush()`
 - (d) `truncate()`
7. Câu lệnh nào giúp đảm bảo tập tin được đóng tự động sau khi sử dụng?
- (a) `try...except`

- (b) if...else
- (c) with
- (d) for

3.7.2 Bài tập thực hành

Bài tập 1: Viết chương trình in ra màn hình thông tin cá nhân gồm họ tên, tuổi và điểm trung bình của sinh viên. Thông tin cần được trình bày rõ ràng trên từng dòng.

Bài tập 2: Viết chương trình nhập hai số thực từ bàn phím, tính tổng, hiệu, tích và thương của hai số đó, sau đó in kết quả ra màn hình với định dạng dễ đọc.

Bài tập 3: Viết chương trình in bảng cửu chương của một số nguyên dương nhập từ bàn phím. Bảng kết quả phải được trình bày theo dạng bảng, mỗi dòng là một phép nhân.

Bài tập 4: Viết chương trình nhập họ tên và năm sinh của một người, sau đó in ra màn hình tuổi hiện tại của người đó (giả sử năm hiện tại là 2025).

Bài tập 5: Viết chương trình đọc một tập tin văn bản chứa danh sách các số nguyên, mỗi số nằm trên một dòng. Tính tổng và trung bình cộng của các số trong tập tin, sau đó in kết quả ra màn hình.

Bài tập 6: Cho tập tin scores.txt lưu trữ điểm các môn học của sinh viên theo cấu trúc:

TenMon,Diem

Viết chương trình đọc tập tin, tính điểm trung bình và in ra danh sách các môn có điểm lớn hơn hoặc bằng điểm trung bình.

Bài tập 7: Viết chương trình cho phép người dùng nhập thông tin sinh viên gồm mã sinh viên, họ tên và điểm trung bình. Thông tin này được ghi vào tập tin. Sau đó, đọc lại tập tin và in ra toàn bộ danh sách sinh viên.

Bài tập 8: Viết chương trình đọc một tập tin văn bản bất kỳ và đếm số dòng, số từ và số ký tự trong tập tin đó. Kết quả được in ra màn hình.

Bài tập 9: Viết chương trình ghi kết quả bảng điểm của một lớp học ra tập tin dưới dạng bảng có căn lề rõ ràng. Mỗi dòng chứa tên sinh viên và điểm trung bình tương ứng.

Bài tập 10: Viết chương trình đọc dữ liệu từ một tập tin chứa thông tin nhiều sinh viên (mỗi sinh viên gồm mã, tên, điểm), thực hiện các yêu cầu sau:

- Tính điểm trung bình của toàn bộ sinh viên
- Tìm sinh viên có điểm cao nhất
- Ghi kết quả thống kê ra một tập tin mới

3.7.3 Gợi ý

Sinh viên nên sử dụng cấu trúc `with` khi làm việc với tập tin để đảm bảo an toàn tài nguyên, đồng thời chú ý định dạng dữ liệu khi in ra màn hình nhằm nâng cao tính trực quan và dễ đọc của chương trình.

3.8 Kết luận

Thông qua hệ thống bài tập trên, sinh viên có thể rèn luyện kỹ năng nhập xuất dữ liệu trong Python một cách toàn diện, từ các thao tác cơ bản đến xử lý dữ liệu thực tế với tập tin. Đây là bước chuẩn bị quan trọng để tiếp cận các chương nâng cao hơn trong lập trình Python.

CHƯƠNG 4. Điều khiển luồng và câu lệnh điều kiện trong Python

Trong lập trình, không phải mọi chương trình đều thực hiện các câu lệnh theo một trình tự cố định từ trên xuống dưới. Thực tế, chương trình thường phải đưa ra quyết định dựa trên điều kiện, từ đó lựa chọn nhánh xử lý phù hợp. Cơ chế này được gọi là điều khiển luồng chương trình.

Trong Python, điều khiển luồng được xây dựng dựa trên các biểu thức logic và giá trị Boolean. Việc hiểu rõ bản chất của giá trị Boolean, các toán tử logic và cách chúng kết hợp với nhau là nền tảng quan trọng để sử dụng các câu lệnh điều kiện một cách chính xác và hiệu quả.

4.1 Giá trị Boolean và vai trò trong điều khiển luồng

4.1.1 Khái niệm giá trị Boolean

Boolean là kiểu dữ liệu logic chỉ có hai giá trị:

- True: biểu thị giá trị đúng
- False: biểu thị giá trị sai

Trong Python, kiểu Boolean được sử dụng để biểu diễn kết quả của các phép so sánh và các phép toán logic. Các câu lệnh điều kiện như `if`, `elif`, `else` đều dựa trên giá trị Boolean để quyết định luồng thực thi của chương trình.

```
x = 5  
print(x > 3)
```

Kết quả:

True

4.1.2 Biểu thức logic

Biểu thức logic là biểu thức khi được đánh giá sẽ cho kết quả là một giá trị Boolean. Biểu thức logic thường được tạo thành từ:

- Các toán hạng (số, biến, biểu thức)
- Các toán tử so sánh hoặc toán tử logic

4.2 Các toán tử trả về giá trị Boolean

4.2.1 Toán tử so sánh

Các toán tử so sánh dùng để so sánh hai giá trị và trả về kết quả Boolean.

Toán tử	Ý nghĩa
==	Bằng nhau
!=	Khác nhau
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng

```
a = 10
b = 20
print(a == b)
print(a < b)
```

Kết quả:

False

True

4.2.2 Toán tử kiểm tra thành viên

Python cung cấp các toán tử dùng để kiểm tra sự tồn tại của phần tử trong một tập hợp.

Toán tử	Ý nghĩa
in	Phần tử thuộc tập hợp
not in	Phần tử không thuộc tập hợp

```
numbers = [1, 2, 3, 4]
print(3 in numbers)
print(5 not in numbers)
```

Kết quả:

True

True

4.2.3 Toán tử so sánh đối tượng

Hai toán tử `is` và `is not` dùng để so sánh danh tính của đối tượng trong bộ nhớ, không phải giá trị của chúng.

```
a = [1, 2]
b = [1, 2]
print(a == b)
print(a is b)
```

Kết quả:

True

False

4.3 Các giá trị tương đương Boolean trong Python

4.3.1 Khái niệm Truthy và Falsy

Trong Python, không chỉ True và False mới được dùng trong biểu thức điều kiện. Mọi đối tượng đều có thể được đánh giá là đúng hoặc sai trong ngữ cảnh Boolean.

4.3.2 Các giá trị được coi là False

Các giá trị sau được Python xem là False:

- False
- None
- Số 0 dưới mọi dạng (0, 0 . 0)
- Chuỗi rỗng " "
- Danh sách, tuple, set, dictionary rỗng

```
print(bool(0))
print(bool(""))
print(bool([]))
```

Kết quả:

False

False

False

Mọi giá trị khác đều được coi là True.

4.4 Toán tử logic

4.4.1 Ba toán tử logic cơ bản

Python cung cấp ba toán tử logic cơ bản:

Toán tử	Ý nghĩa
and	Và
or	Hoặc
not	Phủ định

4.4.2 Bảng logic

Toán tử AND

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Toán tử OR

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Toán tử NOT

A	not A
True	False
False	True

4.5 Thứ tự ưu tiên của các phép toán logic

Khi trong một biểu thức có nhiều toán tử logic, Python sẽ đánh giá theo thứ tự ưu tiên sau:

1. not
2. and
3. or

```
result = not False or False and True
print(result)
```

Biểu thức trên tương đương với:

$$(not False) or (False and True)$$

Kết quả:

True

Việc sử dụng dấu ngoặc tròn giúp biểu thức rõ ràng hơn và tránh nhầm lẫn.

4.6 Các chú ý quan trọng khi làm việc với phép toán logic

Khi sử dụng các phép toán logic trong Python, cần lưu ý một số điểm sau:

- Tránh viết các biểu thức logic quá dài và phức tạp
- Nên sử dụng dấu ngoặc để làm rõ thứ tự đánh giá

- Phân biệt rõ giữa == và is
- Hiểu rõ khái niệm truthy và falsy để tránh lỗi logic

4.7 Câu lệnh if - elif - else

Sau khi đã nắm vững các khái niệm về giá trị Boolean và các phép toán logic, bước tiếp theo trong điều khiển luồng chương trình là sử dụng câu lệnh điều kiện. Trong Python, câu lệnh if - elif - else cho phép chương trình lựa chọn và thực thi các khối lệnh khác nhau tùy thuộc vào điều kiện logic.

Câu lệnh điều kiện là một trong những cấu trúc quan trọng nhất trong lập trình, giúp chương trình có khả năng đưa ra quyết định và xử lý linh hoạt các tình huống khác nhau.

4.7.1 Câu lệnh if

a, Cú pháp cơ bản

Câu lệnh if được sử dụng để kiểm tra một điều kiện. Nếu điều kiện đúng (True), khối lệnh bên trong if sẽ được thực thi.

```
if dieu_kien:  
    # khối lệnh được thực thi nếu điều kiện đúng
```

Trong Python, khối lệnh được xác định bằng **thụt đầu dòng** (indentation), thường là 4 dấu cách. Đây là đặc điểm quan trọng và bắt buộc của ngôn ngữ Python.

b, Ví dụ minh họa

```
x = 10  
if x > 5:  
    print("x lon hon 5")
```

Kết quả hiển thị:

x lon hon 5

Nếu điều kiện không đúng, khối lệnh bên trong if sẽ bị bỏ qua và chương trình tiếp tục thực hiện các câu lệnh phía sau.

4.7.2 Câu lệnh if - else

a, Cú pháp

Câu lệnh else được sử dụng để chỉ ra khối lệnh sẽ được thực thi khi điều kiện trong if là sai.

```

if dieu_kien:
    # khôi lệnh nếu điều kiện đúng
else:
    # khôi lệnh nếu điều kiện sai

```

b, Ví dụ minh họa

```

age = 16
if age >= 18:
    print("Du dieu kien bo phieu")
else:
    print("Chua du dieu kien bo phieu")

```

Kết quả hiển thị:

Chua du dieu kien bo phieu

Trong ví dụ trên, chương trình lựa chọn đúng một trong hai nhánh xử lý dựa trên giá trị của điều kiện.

4.7.3 Câu lệnh `if - elif - else`

a, Nhu cầu sử dụng `elif`

Trong nhiều bài toán thực tế, chương trình cần kiểm tra nhiều điều kiện khác nhau chứ không chỉ hai trường hợp đúng hoặc sai. Khi đó, Python cung cấp từ khóa `elif` (viết tắt của `else if`) để kiểm tra các điều kiện bổ sung.

b, Cú pháp tổng quát

```

if dieu_kien_1:
    # khôi lệnh 1
elif dieu_kien_2:
    # khôi lệnh 2
elif dieu_kien_3:
    # khôi lệnh 3
else:
    # khôi lệnh mặc định

```

Python sẽ kiểm tra các điều kiện theo thứ tự từ trên xuống dưới. Khi gặp điều kiện đầu tiên đúng, khôi lệnh tương ứng sẽ được thực thi và các điều kiện phía sau sẽ không được kiểm tra nữa.

c, Ví dụ phân loại kết quả học tập

```

score = 7.8

if score >= 8.5:
    print("Xep loai Gioi")
elif score >= 7.0:
    print("Xep loai Kha")
elif score >= 5.0:
    print("Xep loai Trung binh")
else:
    print("Xep loai Yeu")

```

Kết quả hiển thị:

Xep loai Kha

Ví dụ này minh họa rõ cách Python chỉ chọn một nhánh phù hợp nhất dựa trên thứ tự điều kiện.

4.7.4 Câu lệnh điều kiện lồng nhau**a, Khái niệm**

Câu lệnh điều kiện lồng nhau là trường hợp bên trong một khối `if` hoặc `elif` lại chứa một câu lệnh `if` khác. Cấu trúc này thường được sử dụng khi việc ra quyết định phụ thuộc vào nhiều mức điều kiện.

b, Ví dụ minh họa

```

age = 20
has_id = True

if age >= 18:
    if has_id:
        print("Du dieu kien vao phong thi")
    else:
        print("Can giay to tuy than")
else:
    print("Chua du tuoi")

```

Kết quả hiển thị:

Du dieu kien vao phong thi

Mặc dù điều kiện lồng nhau mạnh mẽ, nhưng nếu sử dụng quá nhiều mức lồng sẽ làm chương trình khó đọc và khó bảo trì.

4.7.5 Một số lưu ý quan trọng khi sử dụng câu lệnh điều kiện

Khi làm việc với câu lệnh `if - elif - else`, sinh viên cần chú ý các điểm sau:

- Điều kiện trong `if` phải là biểu thức trả về giá trị Boolean
- Thứ tự các điều kiện trong `elif` rất quan trọng
- Tránh viết các câu lệnh điều kiện quá dài và phức tạp
- Luôn đảm bảo thực đầu dòng chính xác

4.7.6 Ý nghĩa của câu lệnh điều kiện trong lập trình kỹ thuật

Trong các bài toán kỹ thuật, câu lệnh điều kiện thường được sử dụng để:

- Kiểm tra trạng thái của hệ thống hoặc thiết bị
- Phân loại dữ liệu do đặc
- Ra quyết định điều khiển trong các hệ thống tự động

Việc sử dụng đúng và hiệu quả câu lệnh `if - elif - else` giúp chương trình phản ứng chính xác trước các tình huống khác nhau, từ đó nâng cao tính ổn định và độ tin cậy của hệ thống. Câu lệnh `if - elif - else` là công cụ trung tâm trong điều khiển luồng chương trình Python. Nắm vững cấu trúc này giúp sinh viên xây dựng các chương trình có khả năng ra quyết định linh hoạt, tạo nền tảng cho việc học các cấu trúc điều khiển nâng cao hơn như vòng lặp và xử lý ngoại lệ.

4.7.7 Câu lệnh điều kiện lồng nhau

a, Khái niệm

Câu lệnh điều kiện lồng nhau (nested conditional statements) là cấu trúc trong đó một hoặc nhiều câu lệnh `if - elif - else` được đặt bên trong khối lệnh của một câu lệnh điều kiện khác. Cấu trúc này cho phép chương trình đưa ra quyết định theo nhiều cấp độ, phù hợp với các bài toán có logic phân nhánh phức tạp.

Về bản chất, mỗi khối `if` bên trong chỉ được kiểm tra khi điều kiện của khối bên ngoài đã được thỏa mãn. Do đó, điều kiện lồng nhau thể hiện rõ mối quan hệ phụ thuộc giữa các quyết định trong chương trình.

b, Cú pháp tổng quát

```
if dieu_kien ngoai:
    if dieu_kien trong:
        # khối lệnh
    else:
        # khối lệnh
else:
    # khối lệnh
```

Python không giới hạn số mức lồng nhau, tuy nhiên việc lồng quá nhiều cấp sẽ làm

chương trình khó đọc và dễ gây lỗi logic.

c, Ví dụ kiểm tra quyền truy cập hệ thống

```
username = "admin"
password = "123456"

if username == "admin":
    if password == "123456":
        print("Dang nhap thanh cong")
    else:
        print("Sai mat khau")
else:
    print("Nguoi dung khong ton tai")
```

Kết quả hiển thị:

Dang nhap thanh cong

Trong ví dụ này, việc kiểm tra mật khẩu chỉ được thực hiện khi tên người dùng hợp lệ. Điều này phản ánh đúng logic xử lý trong các hệ thống thực tế.

d, Ví dụ phân loại tín hiệu đo

```
signal = 4.2

if signal >= 0:
    if signal < 1:
        print("Tin hieu rat yeu")
    elif signal < 3:
        print("Tin hieu yeu")
    elif signal < 5:
        print("Tin hieu trung binh")
    else:
        print("Tin hieu manh")
else:
    print("Gia tri tin hieu khong hop le")
```

Kết quả hiển thị:

Tin hieu trung binh

Ví dụ trên minh họa rõ việc kết hợp điều kiện lồng nhau và `elif` để phân loại dữ liệu đo trong kỹ thuật.

e, So sánh điều kiện lồng nhau và `elif`

Trong nhiều trường hợp, câu lệnh `elif` có thể thay thế cho điều kiện lồng nhau để mã nguồn trở nên gọn gàng và dễ hiểu hơn. Tuy nhiên, điều kiện lồng nhau vẫn cần thiết khi

các quyết định có mối quan hệ phân cấp rõ ràng.

f, Lưu ý khi sử dụng điều kiện lồng nhau

Khi sử dụng câu lệnh điều kiện lồng nhau, cần lưu ý:

- Không lồng quá sâu nếu không cần thiết
- Đặt tên biến rõ ràng để tránh nhầm lẫn logic
- Kiểm tra kỹ các trường hợp biên (edge cases)
- Ưu tiên tính dễ đọc hơn độ ngắn của mã nguồn

4.7.8 Ứng dụng câu lệnh điều kiện trong bài toán kỹ thuật

a, Vai trò của điều kiện trong lập trình kỹ thuật

Trong các hệ thống kỹ thuật, chương trình thường phải đưa ra quyết định dựa trên dữ liệu đo, trạng thái hệ thống hoặc các ngưỡng an toàn. Câu lệnh điều kiện giúp chương trình:

- Phản ứng với các tình huống khác nhau
- Đảm bảo an toàn cho hệ thống
- Tự động hóa quá trình ra quyết định

b, Ứng dụng trong giám sát nhiệt độ

```
temperature = 85

if temperature < 70:
    print("Nhiet do an toan")
elif temperature < 90:
    print("Canh bao: Nhiet do cao")
else:
    print("Nguy hiem: Can dung he thong")
```

Kết quả hiển thị:

Canh bao: Nhiet do cao

Đây là mô hình cơ bản của các hệ thống giám sát nhiệt trong công nghiệp.

c, Ứng dụng trong xử lý dữ liệu đo

```
voltage = 220

if voltage < 200:
    status = "Dien ap thap"
elif voltage <= 240:
    status = "Dien ap binh thuong"
else:
    status = "Dien ap cao"

print(status)
```

Kết quả hiển thị:

Dien ap binh thuong

Việc phân loại dữ liệu đo như điện áp, dòng điện hoặc áp suất là bài toán phổ biến trong kỹ thuật điện và điều khiển.

d, Ứng dụng trong điều khiển thiết bị

```
speed = 45

if speed == 0:
    print("Dong co dung")
elif speed < 30:
    print("Dong co chay cham")
elif speed < 60:
    print("Dong co chay binh thuong")
else:
    print("Dong co chay nhanh")
```

Kết quả hiển thị:

Dong co chay binh thuong

Trong các hệ thống điều khiển, các ngưỡng tốc độ, áp suất hoặc lưu lượng thường được kiểm tra bằng câu lệnh điều kiện.

e, Ứng dụng trong đánh giá chất lượng

```
error_rate = 2.5

if error_rate <= 1:
    print("Chat luong tot")
elif error_rate <= 3:
    print("Chat luong chap nhan duoc")
else:
    print("Chat luong khong dat")
```

Kết quả hiển thị:

Chat luong chap nhan duoc

Ví dụ này phản ánh quy trình kiểm tra chất lượng trong sản xuất và kỹ thuật.

Câu lệnh điều kiện đóng vai trò trung tâm trong hầu hết các bài toán kỹ thuật, từ xử lý dữ liệu đo, giám sát hệ thống đến điều khiển tự động. Việc sử dụng linh hoạt các cấu trúc if - elif - else và điều kiện lồng nhau giúp chương trình phản ứng chính xác, an toàn và hiệu quả trước các tình huống thực tế.

4.8 Vòng lặp trong Python

Trong quá trình xây dựng chương trình, rất nhiều bài toán yêu cầu thực hiện lặp đi lặp lại một hoặc nhiều thao tác. Thay vì viết cùng một đoạn mã nhiều lần, Python cung cấp các cấu trúc vòng lặp cho phép tự động hóa quá trình này. Hai cấu trúc vòng lặp cơ bản và quan trọng nhất trong Python là vòng lặp `for` và vòng lặp `while`.

Vòng lặp không chỉ giúp chương trình ngắn gọn hơn mà còn đóng vai trò trung tâm trong việc xử lý dữ liệu dạng tập hợp, phân tích dữ liệu đo, mô phỏng kỹ thuật và xây dựng các thuật toán.

4.8.1 Vòng lặp `for`

Vòng lặp `for` trong Python được thiết kế theo hướng duyệt qua từng phần tử của một tập hợp dữ liệu. Khác với một số ngôn ngữ lập trình truyền thống, `for` trong Python không dựa trực tiếp vào chỉ số mà dựa trên chính các phần tử trong tập hợp, giúp mã nguồn trở nên trực quan và dễ đọc hơn.

a, Cách hoạt động của vòng lặp `for`

Khi sử dụng vòng lặp `for`, Python sẽ lần lượt lấy từng phần tử trong một đối tượng có thể lặp (iterable), gán vào biến vòng lặp và thực hiện khối lệnh tương ứng. Quá trình này kết thúc khi không còn phần tử nào để duyệt.

b, Ví dụ lặp với dãy số

```
for i in range(1, 6):
    print(i)
```

Kết quả hiển thị:

1
2
3
4
5

Hàm `range()` thường được sử dụng để tạo ra một dãy số phục vụ cho vòng lặp, đặc biệt trong các bài toán tính toán số và kỹ thuật.

c, Vòng lặp for với kiểu dữ liệu String

Chuỗi ký tự trong Python là một kiểu dữ liệu tuần tự, do đó có thể được duyệt từng ký tự bằng vòng lặp `for`.

```
text = "Python"
for ch in text:
    print(ch)
```

Kết quả hiển thị:

P
Y
t
h
o
n

Ví dụ này cho thấy mỗi ký tự trong chuỗi được xử lý như một phần tử độc lập.

d, Vòng lặp for với List

Danh sách (list) là cấu trúc dữ liệu được sử dụng rất phổ biến trong Python. Vòng lặp `for` cho phép xử lý từng phần tử trong danh sách một cách tự nhiên.

```
values = [2, 4, 6, 8]

for v in values:
    print(v * v)
```

Kết quả hiển thị:

4

16

36

64

Trong các bài toán kỹ thuật, cách duyệt này thường được sử dụng để tính toán trên các tập dữ liệu đo hoặc các tham số đầu vào.

e, Vòng lặp for với Tuple

Tuple có cách sử dụng tương tự list nhưng không thể thay đổi giá trị. Vòng lặp for vẫn hoạt động hoàn toàn giống nhau.

```
coordinates = (3, 5, 7)

for c in coordinates:
    print(c)
```

Kết quả hiển thị:

3

5

7

f, Vòng lặp for với Set

Set là kiểu dữ liệu tập hợp không có thứ tự. Khi duyệt bằng vòng lặp for, thứ tự các phần tử có thể khác nhau giữa các lần chạy chương trình.

```
sensors = {"Temp", "Pressure", "Humidity"}

for s in sensors:
    print(s)
```

Kết quả hiển thị (một khả năng):

Pressure

Temp

Humidity

Điều này cho thấy set không đảm bảo thứ tự, một điểm rất quan trọng khi xử lý dữ liệu.

g, Vòng lặp for với Dictionary

Khi duyệt dictionary, vòng lặp for mặc định duyệt qua các khóa (key).

```
scores = {"An": 8, "Binh": 7, "Cuong": 9}

for name in scores:
    print(name, scores[name])
```

Kết quả hiển thị:

An 8
Binh 7
Cuong 9

4.8.2 Vòng lặp while

Khác với vòng lặp for, vòng lặp while hoạt động dựa trên một điều kiện logic. Chương trình sẽ tiếp tục lặp khi điều kiện còn đúng và dừng lại khi điều kiện trở thành sai.

a, Cách hoạt động của vòng lặp while

Mỗi lần lặp, Python kiểm tra điều kiện. Nếu điều kiện là True, khối lệnh bên trong sẽ được thực thi. Do đó, vòng lặp while thường được sử dụng khi số lần lặp không biết trước.

b, Ví dụ đếm số

```
count = 1

while count <= 5:
    print(count)
    count += 1
```

Kết quả hiển thị:

1
2
3
4
5

Vòng lặp while yêu cầu người lập trình phải đảm bảo điều kiện kết thúc, nếu không chương trình sẽ rơi vào vòng lặp vô hạn.

c, Ví dụ kiểm tra dữ liệu đeo

```
value = 0

while value < 10:
    value += 3
    print("Gia tri hien tai:", value)
```

Kết quả hiển thị:

Gia tri hien tai: 3
 Gia tri hien tai: 6
 Gia tri hien tai: 9
 Gia tri hien tai: 12

4.8.3 So sánh vòng lặp for và while

Vòng lặp for phù hợp khi làm việc với các tập hợp dữ liệu hoặc khi biết trước số lần lặp. Ngược lại, vòng lặp while phù hợp hơn khi điều kiện kết thúc phụ thuộc vào trạng thái hoặc kết quả tính toán trong quá trình chạy chương trình.

Trong thực tế, việc lựa chọn đúng loại vòng lặp giúp chương trình trở nên rõ ràng, an toàn và hiệu quả hơn. Vòng lặp for và while là nền tảng của lập trình Python. Việc hiểu rõ cách hoạt động và ứng dụng của từng loại vòng lặp giúp sinh viên xử lý hiệu quả các bài toán liên quan đến dữ liệu, mô phỏng và điều khiển trong kỹ thuật, đồng thời tạo tiền đề cho việc học các cấu trúc nâng cao hơn như vòng lặp lồng nhau và xử lý ngoại lệ.

4.8.4 Vòng lặp lồng nhau

a, Khái niệm

Vòng lặp lồng nhau là cấu trúc trong đó một vòng lặp được đặt bên trong một vòng lặp khác. Trong Python, cả vòng lặp for và while đều có thể được lồng nhau. Cấu trúc này thường được sử dụng khi xử lý dữ liệu nhiều chiều, chẳng hạn như ma trận, bảng số liệu, hoặc khi cần kết hợp nhiều điều kiện lặp trong cùng một bài toán.

Về mặt thực thi, với mỗi lần lặp của vòng lặp bên ngoài, toàn bộ vòng lặp bên trong sẽ được thực hiện trọn vẹn.

b, Ví dụ vòng lặp for lồng nhau

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, "* ", j, " = ", i * j)
```

Kết quả hiển thị:

1 * 1 = 1
 1 * 2 = 2

```
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
```

Ví dụ trên minh họa rõ cách vòng lặp lồng nhau được sử dụng để xây dựng bảng nhân, một dạng bài toán cơ bản nhưng rất quan trọng trong tư duy thuật toán.

c, Vòng lặp lồng nhau với dữ liệu dạng bảng

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for row in matrix:
    for value in row:
        print(value, end=" ")
    print()
```

Kết quả hiển thị:

```
1 2 3
4 5 6
7 8 9
```

Trong các bài toán kỹ thuật và khoa học, dữ liệu thường được lưu trữ dưới dạng ma trận hoặc bảng hai chiều. Vòng lặp lồng nhau là công cụ cơ bản để duyệt và xử lý dạng dữ liệu này.

d, Vòng lặp while lồng nhau

```
i = 1
while i <= 3:
    j = 1
    while j <= 3:
        print(i, j)
        j += 1
    i += 1
```

Kết quả hiển thị:

```
1 1
```

```
1 2  
1 3  
2 1  
2 2  
2 3  
3 1  
3 2  
3 3
```

Vòng lặp `while` lồng nhau cho phép kiểm soát chi tiết điều kiện lặp, tuy nhiên đòi hỏi người lập trình phải rất cẩn thận để tránh vòng lặp vô hạn.

4.8.5 Lệnh `break`, `continue` và `pass`

a, Lệnh `break`

Lệnh `break` được sử dụng để kết thúc ngay lập tức vòng lặp đang thực thi, bất kể điều kiện lặp còn đúng hay không. Khi gặp `break`, chương trình sẽ thoát khỏi vòng lặp và tiếp tục thực hiện các câu lệnh phía sau vòng lặp đó.

```
for i in range(1, 10):  
    if i == 5:  
        break  
    print(i)
```

Kết quả hiển thị:

```
1  
2  
3  
4
```

Lệnh `break` thường được sử dụng trong các bài toán tìm kiếm hoặc khi cần dừng sớm quá trình lặp.

b, Lệnh `continue`

Lệnh `continue` cho phép bỏ qua phần còn lại của vòng lặp hiện tại và chuyển ngay sang lần lặp tiếp theo.

```
for i in range(1, 6):  
    if i == 3:  
        continue  
    print(i)
```

Kết quả hiển thị:

```
1
```

2
4
5

Trong các bài toán xử lý dữ liệu, `continue` rất hữu ích để bỏ qua các giá trị không hợp lệ hoặc không cần thiết.

c, Lệnh `pass`

Lệnh `pass` không thực hiện hành động gì, nó chỉ đóng vai trò là một câu lệnh rỗng. `pass` thường được sử dụng khi cú pháp yêu cầu phải có một khối lệnh nhưng người lập trình chưa muốn hoặc chưa cần cài đặt nội dung.

```
for i in range(3):
    pass
print("Vong lap ket thuc")
```

Kết quả hiển thị:

Vong lap ket thuc

4.8.6 Ứng dụng vòng lặp trong bài toán kỹ thuật

a, Tính giá trị trung bình của dữ liệu đo

```
measurements = [10.2, 10.5, 9.8, 10.1]

total = 0
for m in measurements:
    total += m

average = total / len(measurements)
print("Gia tri trung binh:", average)
```

Kết quả hiển thị:

Gia tri trung binh: 10.15

Đây là ví dụ điển hình cho việc sử dụng vòng lặp để xử lý dữ liệu đo trong kỹ thuật.

b, Phát hiện giá trị vượt ngưỡng an toàn

```
values = [45, 50, 72, 48, 90]

for v in values:
    if v > 80:
        print("Canh bao! Gia tri vuot nguong:", v)
        break
```

Kết quả hiển thị:

Canh bao! Gia tri vuot nguong: 90

Ứng dụng này phản ánh cách vòng lặp kết hợp với điều kiện để giám sát an toàn trong hệ thống kỹ thuật.

c, Xử lý dữ liệu nhiều chiều

```
data = [
    [2, 4, 6],
    [1, 3, 5]
]

for row in data:
    for value in row:
        print(value * 2, end=" ")
    print()
```

Kết quả hiển thị:

4 8 12
2 6 10

4.8.7 Xử lý ngoại lệ trong Python (try – except)**a, Khái niệm về ngoại lệ**

Trong quá trình thực thi chương trình, không phải lúc nào mọi thao tác cũng diễn ra đúng như mong muốn. Các lỗi như chia cho số 0, nhập sai kiểu dữ liệu, truy cập phần tử không tồn tại hoặc lỗi đọc tập tin có thể xảy ra bất kỳ lúc nào. Những lỗi này được Python gọi là *ngoại lệ* (exception).

Nếu không được xử lý, ngoại lệ sẽ làm chương trình dừng đột ngột, gây gián đoạn quá trình tính toán hoặc điều khiển. Trong các hệ thống kỹ thuật và khoa học, điều này là không thể chấp nhận được. Do đó, Python cung cấp cơ chế xử lý ngoại lệ thông qua cấu trúc `try – except` nhằm giúp chương trình tiếp tục hoạt động một cách an toàn.

b, Cú pháp cơ bản của try – except

Cấu trúc xử lý ngoại lệ cơ bản trong Python có dạng:

```
try:
    # đoạn mã có thể phát sinh lỗi
except:
    # đoạn mã xử lý khi lỗi xảy ra
```

Python sẽ thử thực thi các câu lệnh trong khối `try`. Nếu xảy ra ngoại lệ, chương trình lập tức chuyển sang khối `except` thay vì dừng lại.

c, Ví dụ chia cho số 0

```
a = 10  
b = 0  
  
try:  
    result = a / b  
    print(result)  
except:  
    print("Khong the chia cho 0")
```

Kết quả hiển thị:

Khong the chia cho 0

Ví dụ trên cho thấy chương trình không bị dừng mà xử lý lỗi một cách mềm dẻo.

d, Xử lý ngoại lệ với kiểu dữ liệu nhập từ bàn phím

Trong thực tế, dữ liệu nhập từ người dùng thường không đảm bảo đúng kiểu mong muốn.

```
try:  
    x = int(input("Nhập một số nguyên: "))  
    print("Giá trị vừa nhập:", x)  
except:  
    print("Đữ liệu nhập vào không hợp lệ")
```

Ví dụ người dùng nhập:

abc

Kết quả hiển thị:

Đữ liệu nhập vào không hợp lệ

e, Bắt ngoại lệ cụ thể

Python cho phép bắt từng loại ngoại lệ cụ thể thay vì bắt tất cả các lỗi chung chung. Điều này giúp chương trình xử lý chính xác hơn.

```
try:  
    x = int("abc")  
except ValueError:  
    print("Loi chuyen doi kieu du lieu")
```

Kết quả hiển thị:

Loi chuyen doi kieu du lieu

Một số ngoại lệ thường gặp bao gồm: ValueError, ZeroDivisionError, TypeError, IndexError, KeyError.

f, Câu trúc try - except - else

Khối else được thực thi khi không có ngoại lệ nào xảy ra trong try.

```
try:  
    a = 10  
    b = 2  
    result = a / b  
except ZeroDivisionError:  
    print("Chia cho 0")  
else:  
    print("Kết quả:", result)
```

Kết quả hiển thị:

Kết quả: 5.0

g, Câu trúc try - except - finally

Khối finally luôn được thực thi, dù có xảy ra ngoại lệ hay không. Câu trúc này đặc biệt quan trọng khi làm việc với tài nguyên như tập tin hoặc kết nối thiết bị.

```
try:  
    f = open("data.txt", "r")  
    content = f.read()  
    print(content)  
except:  
    print("Không mở được tập tin")  
finally:  
    print("Kết thúc xử lý tập tin")
```

h, Kết hợp vòng lặp và xử lý ngoại lệ

Trong nhiều bài toán kỹ thuật, chương trình cần liên tục yêu cầu dữ liệu cho đến khi hợp lệ.

```
while True:  
    try:  
        value = float(input("Nhập giá trị dương: "))  
        if value <= 0:  
            raise ValueError  
            break  
    except ValueError:  
        print("Giá trị không hợp lệ, vui lòng nhập lại")  
  
print("Giá trị hợp lệ:", value)
```

i, Ứng dụng xử lý ngoại lệ trong bài toán kỹ thuật

Xử lý ngoại lệ được sử dụng rộng rãi trong:

- Kiểm tra dữ liệu đo từ cảm biến
- Đảm bảo an toàn khi điều khiển thiết bị
- Tránh dừng đột ngột hệ thống trong môi trường thực
- Tăng độ ổn định và tin cậy của phần mềm kỹ thuật

j, Ví dụ kiểm tra dữ liệu cảm biến

```
sensor_values = [10.5, 9.8, None, 11.2]  
  
for v in sensor_values:  
    try:  
        print("Giá trị:", float(v))  
    except:  
        print("Dữ liệu cảm biến không hợp lệ")
```

Kết quả hiển thị:

Gia tri: 10.5
Gia tri: 9.8
Du lieu cam bien khong hop le
Gia tri: 11.2

k, Kết luận

Xử lý ngoại lệ là một kỹ năng bắt buộc đối với lập trình viên Python, đặc biệt trong các lĩnh vực kỹ thuật và khoa học. Việc sử dụng hợp lý cấu trúc try – except giúp chương trình trở nên an toàn, ổn định và sẵn sàng ứng phó với các tình huống không mong muốn trong môi trường thực tế.

4.9 Bài tập chương: Câu lệnh điều kiện, vòng lặp và xử lý ngoại lệ

Phần bài tập này nhằm giúp sinh viên củng cố và vận dụng các kiến thức đã học về câu lệnh điều kiện, vòng lặp và xử lý ngoại lệ trong Python. Các bài tập được thiết kế theo hướng từ cơ bản đến nâng cao, gắn với tư duy thuật toán và các tình huống thực tế trong kỹ thuật.

4.9.1 Câu hỏi trắc nghiệm

Câu 1. Giá trị Boolean của biểu thức sau là gì?

```
3 < 5 and 10 > 7
```

- A. True
- B. False
- C. None
- D. Lỗi cú pháp

Câu 2. Đoạn mã sau in ra kết quả gì?

```
x = 5
if x > 5:
    print("A")
else:
    print("B")
```

- A. A
- B. B
- C. A B
- D. Không in gì

Câu 3. Câu lệnh nào được thực thi khi tất cả các điều kiện trong `if` và `elif` đều sai?

- A. `if`
- B. `elif`
- C. `else`
- D. Không có câu lệnh nào

Câu 4. Đoạn mã sau in ra bao nhiêu dòng?

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

- A. 3
- B. 4
- C. 5
- D. 6

Câu 5. Kết quả của đoạn mã sau là gì?

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

- A. 3 dòng
- B. 4 dòng
- C. 5 dòng
- D. 6 dòng

Câu 6. Khi nào vòng lặp while kết thúc?

- A. Khi điều kiện sai
- B. Khi gặp continue
- C. Khi gặp pass
- D. Khi chương trình kết thúc

Câu 7. Lệnh nào dùng để thoát khỏi vòng lặp?

- A. exit
- B. stop
- C. break
- D. return

Câu 8. Lệnh continue có tác dụng gì?

- A. Kết thúc vòng lặp
- B. Bỏ qua vòng lặp hiện tại và chuyển sang vòng lặp tiếp theo
- C. Không làm gì
- D. Kết thúc chương trình

Câu 9. Đoạn mã sau xử lý loại ngoại lệ nào?

```
try:
    x = int("abc")
except ValueError:
    print("Loi")
```

- A. TypeError
- B. ZeroDivisionError
- C. ValueError
- D. IndexError

Câu 10. Khối lệnh nào luôn được thực thi trong cấu trúc xử lý ngoại lệ?

- A. try
- B. except
- C. else
- D. finally

4.9.2 Bài tập thực hành

Bài 1. Viết chương trình nhập vào một số nguyên. Kiểm tra và in ra số đó là số dương, số âm hay bằng 0. Chương trình phải xử lý trường hợp người dùng nhập sai kiểu dữ liệu.

Bài 2. Viết chương trình sử dụng vòng lặp `for` để tính tổng các số chẵn từ 1 đến n , với n được nhập từ bàn phím. Nếu n không hợp lệ, chương trình yêu cầu nhập lại.

Bài 3. Cho một danh sách các giá trị đo:

```
data = [10.5, 12.3, -1, 9.8, None, 11.0]
```

Hãy sử dụng vòng lặp và xử lý ngoại lệ để:

- BỎ qua các giá trị không hợp lệ
- Tính giá trị trung bình của các giá trị hợp lệ

Bài 4. Viết chương trình mô phỏng hệ thống kiểm soát nhiệt độ:

- Nếu nhiệt độ nhỏ hơn 70: in “An toàn”
- Từ 70 đến 90: in “Cảnh báo”
- Lớn hơn 90: in “Nguy hiểm”

Chương trình lặp cho đến khi người dùng nhập giá trị âm thì dừng.

Bài 5. Viết chương trình duyệt một dictionary chứa tên thiết bị và trạng thái hoạt động

(True/False). In ra danh sách các thiết bị đang hoạt động. Sử dụng vòng lặp và câu lệnh điều kiện.

4.9.3 Bài tập tổng hợp

Bài tập lớn. Xây dựng chương trình quản lý dữ liệu đo từ một hệ thống kỹ thuật đơn giản với các yêu cầu sau:

- Cho phép người dùng nhập liên tiếp các giá trị đo (số thực)
- Sử dụng vòng lặp để tiếp tục nhập cho đến khi người dùng nhập “q”
- Sử dụng xử lý ngoại lệ để loại bỏ các dữ liệu không hợp lệ
- Sau khi kết thúc nhập, chương trình tính và in ra:
 - Số lượng giá trị hợp lệ
 - Giá trị lớn nhất, nhỏ nhất
 - Giá trị trung bình
- Nếu không có dữ liệu hợp lệ, in ra thông báo phù hợp

4.10 Hàm và Module trong Python

Trong các chương trước, chương trình Python chủ yếu được xây dựng dưới dạng các câu lệnh viết liên tiếp. Cách viết này phù hợp với các bài toán nhỏ, tuy nhiên khi chương trình trở nên dài và phức tạp hơn, việc quản lý mã nguồn sẽ gặp nhiều khó khăn. Để giải quyết vấn đề này, Python cung cấp khái niệm *hàm* và *module*, cho phép chia nhỏ chương trình thành các khối chức năng độc lập, dễ hiểu và dễ tái sử dụng.

4.10.1 Khái niệm về hàm

Hàm (function) là một khối mã được thiết kế để thực hiện một nhiệm vụ cụ thể. Một khi đã được định nghĩa, hàm có thể được gọi nhiều lần ở các vị trí khác nhau trong chương trình. Việc sử dụng hàm giúp chương trình:

- Tránh lặp lại mã nguồn
- Dễ bảo trì và mở rộng
- Tăng tính rõ ràng về mặt logic

Trong lập trình kỹ thuật, hàm thường được dùng để mô tả các phép tính, mô hình toán học, xử lý dữ liệu đo hoặc các thao tác điều khiển.

4.10.2 Định nghĩa và gọi hàm

a, Cú pháp định nghĩa hàm

Trong Python, hàm được định nghĩa bằng từ khóa `def`.

```
def say_hello():
    print("Xin chào Python")
```

Hàm trên chưa được thực thi cho đến khi nó được gọi.

b, Gọi hàm

```
say_hello()
```

Kết quả hiển thị:

Xin chao Python

4.10.3 Hàm có tham số

Hàm có thể nhận dữ liệu đầu vào thông qua các tham số. Điều này giúp hàm trở nên linh hoạt và có thể áp dụng cho nhiều trường hợp khác nhau.

```
def tinh_binh_phuong(x):
    print("Binh phuong la:", x * x)

tinh_binh_phuong(5)
```

Kết quả hiển thị:

Binh phuong la: 25

Trong ví dụ trên, biến `x` chỉ tồn tại bên trong hàm và không ảnh hưởng đến các biến bên ngoài.

4.10.4 Tham số của hàm trong Python

Trong Python, tham số hàm đóng vai trò là cầu nối giữa dữ liệu bên ngoài và logic xử lý bên trong hàm. Việc hiểu rõ cách truyền tham số giúp người lập trình xây dựng các hàm linh hoạt, dễ tái sử dụng và phù hợp với nhiều tình huống khác nhau trong thực tế.

Python hỗ trợ nhiều cơ chế truyền tham số khác nhau, từ tham số cơ bản, tham số mặc định cho đến các dạng tham số mở rộng như `*args` và `**kwargs`.

a, Tham số vị trí

Tham số vị trí là dạng tham số cơ bản nhất. Giá trị truyền vào hàm sẽ được gán theo đúng thứ tự khai báo.

```
def tinh_dien_tich_hcn(chieu_dai, chieu_rong):
    return chieu_dai * chieu_rong

dt = tinh_dien_tich_hcn(5, 3)
print("Dien tich:", dt)
```

Kết quả hiển thị:

Dien tich: 15

Trong ví dụ này, giá trị 5 được gán cho `chieu_dai` và 3 được gán cho `chieu_rong` dựa trên vị trí.

b, Tham số mặc định

Tham số mặc định cho phép hàm có giá trị mặc định nếu người dùng không truyền tham số tương ứng. Điều này giúp hàm trở nên linh hoạt hơn và giảm số lượng tham số bắt buộc.

```
def tinh_lai_suat(von, lai_suat=0.05):  
    return von * lai_suat  
  
print(tinh_lai_suat(1000000))  
print(tinh_lai_suat(1000000, 0.1))
```

Kết quả hiển thị:

50000.0
100000.0

Trong các bài toán kỹ thuật và kinh tế, tham số mặc định thường được dùng cho các hệ số, ngưỡng hoặc giá trị chuẩn.

c, Lưu ý khi sử dụng tham số mặc định

Tham số mặc định chỉ được đánh giá một lần tại thời điểm hàm được định nghĩa. Do đó, không nên sử dụng các kiểu dữ liệu mutable (như list hoặc dictionary) làm giá trị mặc định.

```
def them_gia_tri(x, ds=[]):  
    ds.append(x)  
    return ds  
  
print(them_gia_tri(1))  
print(them_gia_tri(2))
```

Kết quả hiển thị:

[1]
[1, 2]

Cách viết trên dễ gây lỗi logic. Cách đúng là:

```

def them_gia_tri(x, ds=None):
    if ds is None:
        ds = []
    ds.append(x)
    return ds

print(them_gia_tri(1))
print(them_gia_tri(2))

```

d, Tham số từ khóa

Python cho phép truyền tham số theo tên, không phụ thuộc vào thứ tự.

```

def thong_tin_sv(ten, tuoi, diem):
    print(ten, tuoi, diem)

thong_tin_sv(tuoi=20, ten="An", diem=8.5)

```

Cách truyền này giúp mã nguồn dễ đọc và hạn chế nhầm lẫn khi hàm có nhiều tham số.

e, Tham số *args

*args cho phép hàm nhận một số lượng tham số vị trí không xác định. Các giá trị truyền vào sẽ được gom lại thành một tuple.

```

def tinh_tong(*args):
    tong = 0
    for x in args:
        tong += x
    return tong

print(tinh_tong(1, 2, 3))
print(tinh_tong(5, 10, 15, 20))

```

Kết quả hiển thị:

6

50

Trong các bài toán kỹ thuật, *args thường được dùng khi số lượng dữ liệu đầu vào không cố định, ví dụ như tập các giá trị đo.

f, Tham số **kwargs

**kwargs cho phép hàm nhận một số lượng tham số từ khóa không xác định. Các tham số này được lưu trữ dưới dạng dictionary.

```
def thong_tin_thiet_bi(**kwargs):
    for key, value in kwargs.items():
        print(key, ":", value)

thong_tin_thiet_bi(ten="Sensor A", do_chinh_xac=0.01,
→ trang_thai=True)
```

Kết quả hiển thị:

ten : Sensor A
do_chinh_xac : 0.01
trang_thai : True

g, Kết hợp tham số thường, *args và **kwargs

Python cho phép kết hợp nhiều loại tham số trong cùng một hàm, theo thứ tự: tham số thường → *args → **kwargs.

```
def xu_ly_du_lieu(loai, *values, **options):
    print("Loai:", loai)
    print("Gia tri:", values)
    print("Tuy chon:", options)

xu_ly_du_lieu(
    "nhiet_do",
    20.5, 21.0, 22.3,
    don_vi="C",
    canh_bao=30
)
```

4.10.5 Hàm trả về giá trị

Nhiều hàm không chỉ thực hiện thao tác mà còn cần trả về kết quả để sử dụng tiếp trong chương trình. Python sử dụng từ khóa `return` cho mục đích này.

```
def tinh_trung_binh(a, b):
    return (a + b) / 2

kq = tinh_trung_binh(6, 8)
print("Gia tri trung binh:", kq)
```

Kết quả hiển thị:

Gia tri trung binh: 7.0

Trong các bài toán kỹ thuật, việc trả về kết quả từ hàm là rất phổ biến, đặc biệt trong

các phép tính lặp hoặc mô phỏng.

a, Hàm trả về nhiều giá trị

Trong Python, một hàm có thể trả về nhiều giá trị cùng lúc. Thực chất, các giá trị này được đóng gói trong một tuple.

```
def thong_ke(ds):
    tong = sum(ds)
    trung_binh = tong / len(ds)
    return tong, trung_binh

kq = thong_ke([2, 4, 6, 8])
print(kq)
```

Kết quả hiển thị:

(20, 5.0)

Có thể tách các giá trị trả về:

```
tong, tb = thong_ke([2, 4, 6, 8])
print("Tong:", tong)
print("Trung binh:", tb)
```

b, Ý nghĩa trong lập trình kỹ thuật

Việc sử dụng linh hoạt các loại tham số giúp:

- Thiết kế hàm tổng quát, tái sử dụng cao
- Giảm số lượng hàm cần viết
- Dễ mở rộng chương trình khi yêu cầu thay đổi

Trong các hệ thống kỹ thuật lớn, các hàm thường nhận nhiều loại dữ liệu đầu vào khác nhau. Việc hiểu rõ tham số mặc định, `*args`, `**kwargs` và cách trả về nhiều giá trị là kỹ năng bắt buộc đối với lập trình viên Python.

4.10.6 Phạm vi biến trong hàm

Biến được khai báo bên trong hàm gọi là *biến cục bộ*. Biến này chỉ tồn tại trong phạm vi của hàm.

```
x = 10

def test():
    x = 5
    print("x trong ham:", x)

test()
print("x ben ngoai:", x)
```

Kết quả hiển thị:

```
x trong ham: 5
x ben ngoai: 10
```

4.10.7 Hàm vô danh (Lambda function)

a, Khái niệm

Hàm vô danh, hay còn gọi là *lambda function*, là hàm không có tên, thường được sử dụng cho các phép toán đơn giản trong thời gian ngắn. Hàm lambda giúp mã nguồn ngắn gọn hơn, nhưng không nên lạm dụng trong các logic phức tạp.

b, Cú pháp hàm lambda

```
square = lambda x: x * x
print(square(4))
```

Kết quả hiển thị:

```
16
```

c, Ứng dụng lambda trong xử lý tập hợp

Hàm lambda thường được kết hợp với các hàm xử lý tập hợp như `map()`, `filter()`, và `reduce()` để xử lý dữ liệu một cách ngắn gọn và hiệu quả.

```
values = [1, 2, 3, 4]
result = list(map(lambda x: x * 2, values))
print(result)
```

Kết quả hiển thị:

```
[2, 4, 6, 8]
```

4.10.8 Các hàm xử lý tập hợp trong Python

Python cung cấp một số hàm built-in mạnh mẽ để xử lý tập hợp dữ liệu. Các hàm này thường được sử dụng kết hợp với hàm lambda hoặc các hàm thông thường để biến đổi, lọc và tổng hợp dữ liệu một cách hiệu quả.

a, Hàm map ()

Hàm map () áp dụng một hàm lên từng phần tử của một hoặc nhiều iterable (như list, tuple) và trả về một iterator chứa kết quả. Đây là công cụ mạnh mẽ để biến đổi dữ liệu mà không cần viết vòng lặp tách rời.

Cú pháp: map(function, iterable, ...)

Ví dụ 1: Áp dụng hàm lên một list

```
# Sử dụng với hàm thông thường
def binh_phuong(x):
    return x ** 2

numbers = [1, 2, 3, 4, 5]
result = list(map(binh_phuong, numbers))
print("Binh phuong:", result)

# Sử dụng với lambda
result2 = list(map(lambda x: x ** 3, numbers))
print("Lap phuong:", result2)
```

Kết quả hiển thị:

Binh phuong: [1, 4, 9, 16, 25]
Lap phuong: [1, 8, 27, 64, 125]

Ví dụ 2: Chuyển đổi kiểu dữ liệu

```
# Chuyển chuỗi thành số
str_numbers = ["1", "2", "3", "4", "5"]
int_numbers = list(map(int, str_numbers))
print("Chuyen sang int:", int_numbers)

# Chuyển số thành chuỗi
numbers = [10, 20, 30]
str_nums = list(map(str, numbers))
print("Chuyen sang str:", str_nums)
```

Kết quả hiển thị:

Chuyen sang int: [1, 2, 3, 4, 5]
Chuyen sang str: ['10', '20', '30']

Ví dụ 3: Map với nhiều iterable

Khi map () nhận nhiều iterable, hàm được áp dụng sẽ nhận nhiều đối số tương ứng.

```
# Tính tổng từng cặp phần tử
a = [1, 2, 3, 4]
b = [10, 20, 30, 40]
tong = list(map(lambda x, y: x + y, a, b))
print("Tong cac cap:", tong)

# Tính trung bình hai list
tb = list(map(lambda x, y: (x + y) / 2, a, b))
print("Trung binh:", tb)
```

Kết quả hiển thị:

Tong cac cap: [11, 22, 33, 44]
Trung binh: [5.5, 11.0, 16.5, 22.0]

Ứng dụng trong kỹ thuật:

Trong xử lý dữ liệu đo từ cảm biến, map () thường được dùng để chuyển đổi đơn vị hoặc chuẩn hóa dữ liệu.

```
# Chuyển đổi nhiệt độ từ Celsius sang Fahrenheit
celsius = [0, 10, 20, 30, 100]
fahrenheit = list(map(lambda c: c * 9/5 + 32, celsius))
print("Fahrenheit:", fahrenheit)

# Chuẩn hóa dữ liệu điện áp (chia cho 1000 để đổi mV sang V)
voltage_mv = [1500, 3300, 5000, 12000]
voltage_v = list(map(lambda mv: mv / 1000, voltage_mv))
print("Dien ap (V):", voltage_v)
```

Kết quả hiển thị:

Fahrenheit: [32.0, 50.0, 68.0, 86.0, 212.0]
Dien ap (V): [1.5, 3.3, 5.0, 12.0]

b, Hàm filter()

Hàm filter() lọc các phần tử trong iterable dựa trên một điều kiện. Chỉ những phần tử làm cho hàm điều kiện trả về True mới được giữ lại.

Cú pháp: filter(function, iterable)

Ví dụ 1: Lọc số chẵn

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Lọc số chẵn
```

```

so_chan = list(filter(lambda x: x % 2 == 0, numbers))
print("So chan:", so_chan)

# Lọc số lớn hơn 5
lon_hon_5 = list(filter(lambda x: x > 5, numbers))
print("Lon hon 5:", lon_hon_5)

```

Kết quả hiển thị:

So chan: [2, 4, 6, 8, 10]
 Lon hon 5: [6, 7, 8, 9, 10]

Ví dụ 2: Lọc chuỗi

```

names = ["An", "Binh", "C", "Dung", "E", "Phong"]

# Lọc tên có độ dài lớn hơn 1
ten_dai = list(filter(lambda name: len(name) > 1, names))
print("Ten dai hon 1 ky tu:", ten_dai)

# Lọc tên bắt đầu bằng chữ cái có mã ASCII > 68
ten_loc = list(filter(lambda name: ord(name[0]) > 68,
                      names))
print("Ten bat dau tu 'E' tro di:", ten_loc)

```

Kết quả hiển thị:

Ten dai hon 1 ky tu: ['An', 'Binh', 'Dung', 'Phong']
 Ten bat dau tu 'E' tro di: ['E', 'Phong']

Ứng dụng trong kỹ thuật:

Lọc dữ liệu không hợp lệ hoặc vượt ngưỡng từ tập dữ liệu đo.

```

# Dữ liệu nhiệt độ từ cảm biến
nhiet_do = [22.5, 23.0, 150.0, 22.8, -50.0, 23.2, 24.0]

# Lọc dữ liệu hợp lệ (từ 0 đến 100 độ C)
nhiet_do_hop_le = list(filter(lambda t: 0 <= t <= 100,
                               nhiet_do))
print("Nhiet do hop le:", nhiet_do_hop_le)

# Cảnh báo nhiệt độ cao
canh_bao = list(filter(lambda t: t > 30, nhiet_do_hop_le))
print("Canh bao nhiet do cao:", canh_bao)

```

Kết quả hiển thị:

Nhiet do hop le: [22.5, 23.0, 22.8, 23.2, 24.0]
 Canh bao nhiet do cao: []

c, Hàm reduce ()

Hàm `reduce()` (trong module `functools`) áp dụng một hàm hai đối số lên các phần tử của iterable theo cách tích lũy, từ trái sang phải, để rút gọn iterable thành một giá trị duy nhất.

Cú pháp: `reduce(function, iterable[, initializer])`

Ví dụ 1: Tính tổng và tích

```
from functools import reduce

numbers = [1, 2, 3, 4, 5]

# Tính tổng
tong = reduce(lambda x, y: x + y, numbers)
print("Tong:", tong)

# Tính tích
tich = reduce(lambda x, y: x * y, numbers)
print("Tich:", tich)
```

Kết quả hiển thị:

Tong: 15
 Tich: 120

Quá trình hoạt động của `reduce()` với tổng:

- Bước 1: $1 + 2 = 3$
- Bước 2: $3 + 3 = 6$
- Bước 3: $6 + 4 = 10$
- Bước 4: $10 + 5 = 15$

Ví dụ 2: Tìm giá trị lớn nhất và nhỏ nhất

```
from functools import reduce

numbers = [15, 7, 23, 9, 31, 12]

# Tìm max
max_val = reduce(lambda x, y: x if x > y else y, numbers)
```

```
print("Gia tri lon nhat:", max_val)

# Tìm min
min_val = reduce(lambda x, y: x if x < y else y, numbers)
print("Gia tri nho nhat:", min_val)
```

Kết quả hiển thị:

Gia tri lon nhat: 31
Gia tri nho nhat: 7

Ví dụ 3: Ghép chuỗi

```
from functools import reduce

words = ["Python", "la", "ngon", "ngu", "manh", "me"]
cau = reduce(lambda x, y: x + " " + y, words)
print(cau)
```

Kết quả hiển thị:

Python la ngon ngu manh me

Ứng dụng trong kỹ thuật:

Tính toán tích lũy trên chuỗi dữ liệu.

```
from functools import reduce

# Tính tổng công suất tiêu thụ của các thiết bị
cong_suat = [100, 250, 75, 300, 150] # Watt
tong_cong_suat = reduce(lambda x, y: x + y, cong_suat)
print("Tong cong suat:", tong_cong_suat, "W")

# Tính điện trở tương đương mạch nối tiếp
dien_tro = [10, 20, 30, 15] # Ohm
R_tuong_duong = reduce(lambda x, y: x + y, dien_tro)
print("Dien tro tuong duong:", R_tuong_duong, "Ohm")
```

Kết quả hiển thị:

Tong cong suat: 875 W
Dien tro tuong duong: 75 Ohm

4.10.9 List Comprehension

List comprehension là một cú pháp ngắn gọn và pythonic để tạo list mới từ một iterable có sẵn. Đây là một trong những tính năng được yêu thích nhất của Python vì tính rõ ràng và hiệu suất cao.

a, Cú pháp cơ bản

Cú pháp tổng quát: [expression for item in iterable if condition]

Ví dụ 1: Tạo list đơn giản

```
# Cách thông thường
binh_phuong = []
for x in range(1, 6):
    binh_phuong.append(x ** 2)
print("Cach thuong:", binh_phuong)

# Dùng list comprehension
binh_phuong_2 = [x ** 2 for x in range(1, 6)]
print("List comprehension:", binh_phuong_2)
```

Kết quả hiển thị:

Cach thuong: [1, 4, 9, 16, 25]
List comprehension: [1, 4, 9, 16, 25]

b, List comprehension với điều kiện

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Lọc số chẵn
so_chan = [x for x in numbers if x % 2 == 0]
print("So chan:", so_chan)

# Bình phương các số lẻ
binh_phuong_le = [x ** 2 for x in numbers if x % 2 != 0]
print("Binh phuong so le:", binh_phuong_le)

# Điều kiện if-else trong expression
phan_loai = ["chan" if x % 2 == 0 else "le" for x in
             numbers]
print("Phan loai:", phan_loai)
```

Kết quả hiển thị:

So chan: [2, 4, 6, 8, 10]
Binh phuong so le: [1, 9, 25, 49, 81]

c, List comprehension lồng nhau

```
# Tạo ma trận
matrix = [[i * j for j in range(1, 4)] for i in range(1, 4)]
for row in matrix:
    print(row)

# Làm phẳng ma trận
flat = [num for row in matrix for num in row]
print("Ma tran phang:", flat)
```

Kết quả hiển thị:

```
[1, 2, 3]
[2, 4, 6]
[3, 6, 9]
Ma tran phang: [1, 2, 3, 2, 4, 6, 3, 6, 9]
```

d, Ứng dụng trong xử lý chuỗi

```
# Chuyển chữ thường thành chữ hoa
words = ["python", "programming", "language"]
upper_words = [w.upper() for w in words]
print("Chu hoa:", upper_words)

# Lọc từ có độ dài > 5
long_words = [w for w in words if len(w) > 5]
print("Tu dai:", long_words)

# Lấy ký tự đầu của mỗi từ
first_chars = [w[0] for w in words]
print("Ky tu dau:", first_chars)
```

Kết quả hiển thi:

```
Chu hoa: ['PYTHON', 'PROGRAMMING', 'LANGUAGE']
Tu dai: ['python', 'programming', 'language']
Ky tu dau: ['p', 'p', 'l']
```

e. Ứng dụng trong kỹ thuật

Ví dụ 1: Xử lý dữ liệu cảm biến

```
# Dữ liệu thô từ cảm biến nhiệt độ
raw_data = [22.5, 23.1, 22.8, 150.0, 23.4, -10.0, 24.1,
→ 22.9]

# Lọc và làm sạch dữ liệu (0 <= T <= 50)
clean_data = [t for t in raw_data if 0 <= t <= 50]
print("Du lieu sach:", clean_data)

# Chuyển đổi sang Fahrenheit
fahrenheit = [t * 9/5 + 32 for t in clean_data]
print("Fahrenheit:", fahrenheit)

# Đánh dấu nhiệt độ cao (> 23.5)
warning = ["CANH BAO" if t > 23.5 else "BINH THUONG"
           for t in clean_data]
print("Trang thai:", warning)
```

Kết quả hiển thị:

Du lieu sach: [22.5, 23.1, 22.8, 23.4, 24.1, 22.9]
 Fahrenheit: [72.5, 73.58, 73.04, 74.12, 75.38, 73.22]
 Trang thai: ['BINH THUONG', 'BINH THUONG', 'BINH THUONG', 'BINH THUONG',

Ví dụ 2: Tính toán kỹ thuật

```
# Tính công suất từ điện áp và dòng điện (P = V * I)
voltage = [5, 12, 24, 48] # Volt
current = [0.5, 1.0, 2.0, 1.5] # Ampere

power = [v * i for v, i in zip(voltage, current)]
print("Cong suat (W):", power)

# Tính điện trở (R = V / I)
resistance = [v / i for v, i in zip(voltage, current)]
print("Dien tro (Ohm):", resistance)
```

Kết quả hiển thị:

Cong suat (W): [2.5, 12.0, 48.0, 72.0]
 Dien tro (Ohm): [10.0, 12.0, 12.0, 32.0]

f, Dictionary và Set Comprehension

Python cũng hỗ trợ comprehension cho dictionary và set.

```

# Dictionary comprehension
numbers = [1, 2, 3, 4, 5]
squares_dict = {x: x**2 for x in numbers}
print("Dict bình phương:", squares_dict)

# Set comprehension (loại bỏ trùng lặp)
numbers_dup = [1, 2, 2, 3, 3, 3, 4, 5]
unique_squares = {x**2 for x in numbers_dup}
print("Set bình phương:", unique_squares)

```

Kết quả hiển thị:

Dict bình phương: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
Set bình phương: {1, 4, 9, 16, 25}

g, So sánh hiệu suất

List comprehension thường nhanh hơn so với vòng lặp thông thường hoặc map(), đồng thời code ngắn gọn và dễ đọc hơn.

```

# Ba cách tạo list
numbers = range(100)

# Cách 1: Vòng lặp for
result1 = []
for x in numbers:
    result1.append(x ** 2)

# Cách 2: map()
result2 = list(map(lambda x: x ** 2, numbers))

# Cách 3: List comprehension (nhanh nhất)
result3 = [x ** 2 for x in numbers]

print("Các cách cho kết quả giống nhau:",
      result1[:5] == result2[:5] == result3[:5])

```

Kết quả hiển thị:

Các cách cho kết quả giống nhau: True

h, Lưu ý khi sử dụng

Mặc dù list comprehension rất mạnh mẽ, cần tránh lạm dụng với các biểu thức quá phức tạp vì sẽ làm giảm tính dễ đọc của code. Với logic phức tạp, nên sử dụng vòng lặp thông thường hoặc tách thành nhiều bước.

4.10.10 So sánh các phương pháp xử lý tập hợp

Phương pháp	Đặc điểm
map()	Biến đổi từng phần tử, trả về iterator
filter()	Lọc phần tử theo điều kiện, trả về iterator
reduce()	Rút gọn thành một giá trị duy nhất
List comprehension	Tạo list mới, cú pháp ngắn gọn, hiệu suất cao
Vòng lặp for	Linh hoạt nhất, phù hợp với logic phức tạp

Trong thực tế, lựa chọn phương pháp phụ thuộc vào:

- Độ phức tạp của logic
- Yêu cầu về hiệu suất
- Tính dễ đọc của code
- Quy ước của team/dự án

4.10.11 Khái niệm về Module

Module là một tập tin Python chứa các hàm, biến và lớp. Việc sử dụng module cho phép tổ chức chương trình thành nhiều tệp nhỏ, giúp quản lý mã nguồn hiệu quả hơn.

4.10.12 Sử dụng module có sẵn

Python cung cấp rất nhiều module chuẩn hỗ trợ tính toán và xử lý dữ liệu.

```
import math

print(math.sqrt(16))
print(math.pi)
```

Kết quả hiển thị:

4.0
3.141592653589793

4.10.13 Tự xây dựng module

Giả sử tạo tập tin my_math.py với nội dung:

```
def cong(a, b):
    return a + b

def tru(a, b):
    return a - b
```

Sử dụng module này trong chương trình khác:

```
import my_math

print(my_math.cong(3, 4))
print(my_math.tru(10, 5))
```

Kết quả hiển thị:

7

5

4.10.14 Ý nghĩa của hàm và module trong lập trình kỹ thuật

Trong các hệ thống kỹ thuật, hàm và module giúp:

- Chuẩn hóa các phép tính và mô hình
- Tái sử dụng mã nguồn giữa các dự án
- Giảm lỗi và tăng độ tin cậy của hệ thống

4.10.15 Kết luận

Hàm và module là nền tảng của lập trình Python hiện đại. Việc nắm vững cách xây dựng và sử dụng hàm, hàm vô danh và module giúp sinh viên chuyển từ lập trình đơn giản sang xây dựng các chương trình có cấu trúc, sẵn sàng cho các nội dung nâng cao như lập trình hướng đối tượng và phân tích dữ liệu.

4.11 Bài tập

4.11.1 Bài tập 1: Hàm tính toán cơ bản

Viết các hàm tính toán sau:

1. Hàm `giasi_thua(n)`: Tính giai thừa của số nguyên dương n bằng đệ quy
2. Hàm `to_hop(n, k)`: Tính tổ hợp $C_n^k = \frac{n!}{k!(n-k)!}$
3. Hàm `fibonacci(n)`: Tính số Fibonacci thứ n
4. Hàm `fibonacci_iterative(n)`: Tính Fibonacci bằng phương pháp lặp (so sánh hiệu suất)

Yêu cầu: Kiểm tra tính hợp lệ của tham số đầu vào.

4.11.2 Bài tập 2: Hàm xử lý chuỗi

Viết các hàm xử lý chuỗi:

1. `dem_tu(chuoi)`: Đếm số từ trong chuỗi
2. `dao_nguoc_chuoi(chuoi)`: Đảo ngược toàn bộ chuỗi
3. `dao_nguoc_tu(chuoi)`: Đảo ngược từng từ trong chuỗi
4. `loai_bo_ky_tu_dac_biet(chuoi)`: Chỉ giữ lại chữ cái, chữ số và khoảng trắng

5. chuan_hoa_chuoi (chuoi): Loại bỏ khoảng trắng thừa

4.11.3 Bài tập 3: Hàm làm việc với List

Viết các hàm xử lý danh sách số:

1. thong_ke_list (ds): Trả về dictionary chứa min, max, sum, mean, count
2. loc_chan (ds), loc_le (ds), loc_duong (ds): Lọc các phần tử thỏa điều kiện
3. sap_xep_tang (ds), sap_xep_giam (ds): Sắp xếp danh sách

Kiểm tra với danh sách: [5, -3, 8, 12, -7, 0, 15, 4, -1, 9]

4.11.4 Bài tập 4: Lambda và Higher-order Functions

Cho danh sách sinh viên:

```
sinh_vien = [  
    {'ten': 'An', 'diem': 8.5},  
    {'ten': 'Binh', 'diem': 6.0},  
    {'ten': 'Cuong', 'diem': 7.5},  
    {'ten': 'Dung', 'diem': 9.0},  
    {'ten': 'Em', 'diem': 5.5}  
]
```

Sử dụng map, filter, reduce và lambda để:

1. Lấy danh sách tên sinh viên
2. Tăng điểm mỗi sinh viên thêm 0.5 (tối đa 10)
3. Lọc sinh viên đạt (điểm ≥ 7)
4. Tính điểm trung bình của lớp
5. Tìm sinh viên có điểm cao nhất

4.11.5 Bài tập 5: Module và Package

Tạo module `math_utils.py` chứa:

1. Hàm la_so_nguyen_to (n): Kiểm tra số nguyên tố
2. Hàm gcd (a, b): Tính ước chung lớn nhất
3. Hàm lcm (a, b): Tính bội chung nhỏ nhất
4. Hàm giai_phuong_trinh_bac_2 (a, b, c): Giải phương trình $ax^2 + bx + c = 0$
5. Các hằng số: PI, E

Tạo file `main.py` để import và test các hàm trong module.

Ghi chú: Xem lời giải chi tiết tại Phụ lục - Lời giải bài tập.

CHƯƠNG 5. Lập trình hướng đối tượng trong Python

Trong các chương trước, chúng ta đã làm quen với lập trình theo hướng thủ tục, nơi chương trình được tổ chức thành các hàm và dữ liệu được xử lý tuần tự. Tuy nhiên, khi chương trình trở nên phức tạp hơn, việc quản lý và bảo trì mã nguồn theo cách này có thể gặp nhiều khó khăn. Lập trình hướng đối tượng (Object-Oriented Programming - OOP) ra đời như một giải pháp mạnh mẽ giúp tổ chức mã nguồn một cách logic, dễ bảo trì và tái sử dụng hơn.

Python là một ngôn ngữ hỗ trợ đầy đủ các tính năng của lập trình hướng đối tượng. Hiểu và vận dụng tốt OOP không chỉ giúp sinh viên viết mã tốt hơn mà còn là nền tảng để học các framework và thư viện hiện đại trong Python.

5.1 Giới thiệu về lập trình hướng đối tượng

5.1.1 Khái niệm lập trình hướng đối tượng

Lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm về các “đối tượng”. Đối tượng là một thực thể có thể chứa cả dữ liệu (gọi là thuộc tính) và các hành động liên quan đến dữ liệu đó (gọi là phương thức). Để hiểu rõ hơn về OOP, chúng ta cần nhìn nhận sự khác biệt cơ bản giữa lập trình thủ tục và lập trình hướng đối tượng.

Trong lập trình thủ tục truyền thống, chương trình được tổ chức thành các hàm độc lập, mỗi hàm thực hiện một nhiệm vụ cụ thể. Dữ liệu và các hàm xử lý dữ liệu đó thường được tách biệt nhau. Khi chương trình trở nên phức tạp với nhiều loại dữ liệu và nhiều hàm xử lý khác nhau, việc quản lý mối quan hệ giữa dữ liệu và hàm trở nên khó khăn. Một thay đổi nhỏ trong cấu trúc dữ liệu có thể yêu cầu sửa đổi nhiều hàm khác nhau, dẫn đến nguy cơ phát sinh lỗi và khó bảo trì.

Lập trình hướng đối tượng tiếp cận vấn đề theo một cách khác. Thay vì tách biệt dữ liệu và hành vi, OOP nhóm chúng lại với nhau thành các đối tượng. Mỗi đối tượng đại diện cho một thực thể cụ thể trong bài toán, chứa đựng cả thông tin (thuộc tính) và hành vi (phương thức) liên quan đến thực thể đó. Cách tiếp cận này phản ánh trực quan hơn cách chúng ta nhìn nhận thế giới thực, nơi mỗi sự vật đều có đặc điểm và hành động riêng.

Ví dụ, khi mô hình hóa một chiếc ô tô trong chương trình, lập trình thủ tục có thể tạo các biến riêng lẻ để lưu màu sắc, tốc độ, vị trí, và các hàm riêng để tăng tốc, phanh, rẽ. Trong khi đó, OOP sẽ tạo ra một đối tượng ô tô, bên trong nó chứa tất cả các thuộc tính (màu sắc, tốc độ, vị trí) và các phương thức (tăng tốc, phanh, rẽ) liên quan. Điều này không chỉ làm cho code dễ hiểu hơn mà còn dễ dàng mở rộng và bảo trì.

5.1.2 Lợi ích của lập trình hướng đối tượng

Lập trình hướng đối tượng mang lại nhiều lợi ích quan trọng trong phát triển phần mềm, đặc biệt đối với các dự án có quy mô lớn và phức tạp. Một trong những lợi ích nổi bật nhất là tính mô-đun hóa. Khi mã nguồn được tổ chức thành các đối tượng độc lập, mỗi đối tượng đảm nhận một trách nhiệm cụ thể, việc phát triển và kiểm thử trở nên dễ dàng

hơn nhiều. Các thành viên trong nhóm có thể làm việc song song trên các lớp khác nhau mà không lo xung đột code.

Tính tái sử dụng là một lợi thế quan trọng khác của OOP. Một khi đã xây dựng được một lớp hoạt động tốt, lớp đó có thể được sử dụng lại trong nhiều phần khác nhau của chương trình, thậm chí trong các dự án khác. Điều này không chỉ tiết kiệm thời gian mà còn giảm thiểu lỗi, vì code đã được kiểm thử kỹ lưỡng sẽ ít có khả năng gây ra vấn đề khi được tái sử dụng.

Khả năng bảo trì là một yếu tố then chốt quyết định tuổi thọ của phần mềm. Trong OOP, việc thay đổi hoặc mở rộng chức năng thường chỉ cần can thiệp vào một hoặc vài lớp cụ thể, mà không ảnh hưởng đến toàn bộ hệ thống. Điều này đặc biệt quan trọng khi phần mềm cần được cập nhật thường xuyên để đáp ứng yêu cầu thay đổi của người dùng.

Tính trừu tượng cho phép người lập trình ẩn đi các chi tiết triển khai phức tạp bên trong đối tượng, chỉ để lộ ra bên ngoài những giao diện cần thiết. Người sử dụng đối tượng không cần biết nó hoạt động như thế nào bên trong, chỉ cần biết cách sử dụng các phương thức công khai. Điều này giúp giảm độ phức tạp nhận thức khi làm việc với hệ thống lớn.

Cuối cùng, OOP cung cấp một cách tiếp cận tự nhiên để mô hình hóa các vấn đề thực tế. Trong thế giới thực, chúng ta đã quen với việc tư duy theo các đối tượng cụ thể: một con chó, một chiếc xe, một tài khoản ngân hàng. OOP cho phép chúng ta chuyển đổi trực tiếp các thực thể này thành các đối tượng trong chương trình, làm cho quá trình thiết kế và triển khai trở nên trực quan và dễ hiểu hơn.

5.1.3 Các khái niệm cơ bản trong OOP

Trước khi đi sâu vào chi tiết kỹ thuật, việc hiểu rõ các thuật ngữ cơ bản trong OOP là vô cùng quan trọng. Bốn khái niệm then chốt cần nắm vững là lớp, đối tượng, thuộc tính và phương thức.

Lớp (Class) có thể được hình dung như một bản thiết kế hoặc khuôn mẫu để tạo ra các đối tượng. Giống như bản vẽ kiến trúc của một ngôi nhà, lớp định nghĩa cấu trúc và hành vi mà các đối tượng được tạo ra từ nó sẽ có. Lớp quy định những thuộc tính nào mỗi đối tượng sẽ chứa và những phương thức nào đối tượng đó có thể thực hiện. Tuy nhiên, bản thân lớp không phải là một thực thể cụ thể, nó chỉ là định nghĩa trừu tượng.

Đối tượng (Object), còn được gọi là thể hiện (instance), là một thực thể cụ thể được tạo ra từ lớp. Nếu lớp là bản vẽ của ngôi nhà, thì đối tượng là ngôi nhà thực sự được xây dựng theo bản vẽ đó. Từ một lớp, chúng ta có thể tạo ra nhiều đối tượng khác nhau, mỗi đối tượng tồn tại độc lập với trạng thái riêng của nó. Mặc dù tất cả các đối tượng cùng được tạo từ một lớp sẽ có cùng cấu trúc và khả năng, nhưng giá trị dữ liệu cụ thể mà mỗi đối tượng lưu trữ có thể hoàn toàn khác nhau.

Thuộc tính (Attribute) là các biến được gắn liền với một đối tượng cụ thể, dùng để lưu trữ trạng thái hoặc thông tin của đối tượng đó. Ví dụ, nếu chúng ta có một đối tượng đại

điện cho sinh viên, các thuộc tính có thể là tên, tuổi, mã số sinh viên, điểm trung bình. Mỗi đối tượng sinh viên sẽ có giá trị riêng cho các thuộc tính này, phản ánh thông tin cá nhân của từng sinh viên.

Phương thức (Method) là các hàm được định nghĩa bên trong lớp, thể hiện các hành động mà đối tượng có thể thực hiện. Phương thức thường hoạt động trên dữ liệu của chính đối tượng đó (tức là trên các thuộc tính của nó). Tiếp tục với ví dụ về sinh viên, các phương thức có thể là cập nhật điểm, tính điểm trung bình, hiển thị thông tin. Phương thức cho phép đối tượng không chỉ lưu trữ dữ liệu mà còn có khả năng xử lý và thao tác trên dữ liệu đó.

5.2 Lớp và đối tượng trong Python

5.2.1 Định nghĩa lớp

Việc định nghĩa lớp trong Python là bước đầu tiên để áp dụng tư duy hướng đối tượng vào chương trình. Lớp được định nghĩa bằng từ khóa `class`, sau là tên lớp và dấu hai chấm. Tên lớp theo quy ước thường được viết theo kiểu PascalCase, nghĩa là mỗi từ trong tên đều viết hoa chữ cái đầu và không có khoảng trắng hoặc ký tự đặc biệt ở giữa. Quy ước này giúp phân biệt rõ ràng giữa tên lớp và tên biến hoặc hàm, vốn thường được viết theo kiểu `snake_case`.

Khi định nghĩa một lớp, chúng ta đang tạo ra một kiểu dữ liệu mới trong Python. Kiểu dữ liệu này không chỉ chứa dữ liệu mà còn chứa cả các hành vi liên quan đến dữ liệu đó. Điều này khác biệt hoàn toàn với các kiểu dữ liệu nguyên thủy như `int`, `str` hay `list`, vì chúng ta có toàn quyền kiểm soát cách thức hoạt động của kiểu dữ liệu mới này.

a, Cú pháp định nghĩa lớp

```
class TenLop:  
    # Các thuộc tính và phương thức  
    pass
```

b, Ví dụ định nghĩa lớp đơn giản

Hãy bắt đầu với một ví dụ đơn giản về lớp `Student` đại diện cho sinh viên:

```
class Student:  
    pass
```

Lớp `Student` này chưa có bất kỳ thuộc tính hay phương thức nào, nhưng đã là một lớp hợp lệ trong Python. Từ khóa `pass` được sử dụng khi chúng ta muốn định nghĩa một cấu trúc rỗng mà chưa cần triển khai chi tiết ngay lập tức.

5.2.2 Tạo đối tượng từ lớp

Sau khi đã có định nghĩa lớp, bước tiếp theo là tạo ra các đối tượng cụ thể từ lớp đó. Quá trình này được gọi là khởi tạo đối tượng (instantiation) hoặc tạo thể hiện (instance

creation). Mỗi đối tượng được tạo ra từ lớp sẽ có cấu trúc giống nhau nhưng tồn tại độc lập trong bộ nhớ với trạng thái riêng biệt.

Khi gọi tên lớp với cặp dấu ngoặc tròn, Python thực hiện một chuỗi các thao tác phức tạp bên dưới. Đầu tiên, Python cấp phát một vùng nhớ mới để chứa đối tượng. Sau đó, nó khởi tạo các thuộc tính mặc định nếu có. Cuối cùng, nó gọi phương thức khởi tạo `__init__()` nếu phương thức này được định nghĩa trong lớp. Kết quả trả về là một tham chiếu đến đối tượng mới được tạo, và tham chiếu này được gán cho biến mà chúng ta chỉ định.

```
class Student:
    pass

# Tạo đối tượng sinh viên
sv1 = Student()
sv2 = Student()

print(type(sv1))
print(type(sv2))
```

Kết quả hiển thị:

```
<class '__main__.Student'>
<class '__main__.Student'>
```

Trong ví dụ trên, sv1 và sv2 là hai đối tượng khác nhau được tạo từ cùng lớp Student. Mặc dù cùng được tạo từ một lớp, mỗi đối tượng chiếm một vùng nhớ riêng biệt và độc lập hoàn toàn với nhau. Điều này có nghĩa là bất kỳ thay đổi nào trên sv1 sẽ không ảnh hưởng đến sv2 và ngược lại.

5.2.3 Thuộc tính của đối tượng

Thuộc tính đóng vai trò như các biến thành viên của đối tượng, lưu trữ trạng thái và thông tin đặc trưng của đối tượng đó. Mỗi đối tượng có bộ giá trị thuộc tính riêng, phản ánh trạng thái hiện tại của nó. Đây chính là cơ chế cho phép các đối tượng được tạo từ cùng một lớp có thể khác biệt nhau về mặt dữ liệu.

Trong Python, thuộc tính của đối tượng được truy cập thông qua cú pháp chấm (dot notation). Khi viết `object.attribute`, chúng ta đang yêu cầu Python lấy giá trị của thuộc tính `attribute` thuộc về đối tượng `object`. Cơ chế này rất trực quan và phản ánh đúng mối quan hệ sở hữu giữa đối tượng và thuộc tính của nó.

a, Thêm thuộc tính động

Trong Python, chúng ta có thể thêm thuộc tính cho đối tượng bất kỳ lúc nào sau khi đối tượng được tạo:

```

class Student:
    pass

sv1 = Student()
sv1.name = "Nguyen Van An"
sv1.age = 20
sv1.major = "Computer Science"

print(f"Ten: {sv1.name}")
print(f"Tuoi: {sv1.age}")
print(f"Chuyen nganh: {sv1.major}")

```

Kết quả hiển thị:

Ten: Nguyen Van An
 Tuoi: 20
 Chuyen nganh: Computer Science

Tuy nhiên, cách thêm thuộc tính động như trên không được khuyến khích trong thực tế vì không có tính nhất quán. Thay vào đó, chúng ta nên sử dụng phương thức khởi tạo.

5.3 Phương thức khởi tạo - Constructor

5.3.1 Phương thức `__init__()`

Phương thức `__init__()` là một trong những phương thức đặc biệt quan trọng nhất trong lập trình hướng đối tượng với Python. Đây là phương thức được Python tự động gọi ngay sau khi một đối tượng mới được tạo ra từ lớp. Vai trò chính của phương thức này là thiết lập trạng thái ban đầu cho đối tượng bằng cách khởi tạo các thuộc tính với những giá trị phù hợp.

Mặc dù `__init__()` thường được gọi là constructor (hàm khởi tạo), về mặt kỹ thuật nó không hoàn toàn giống với constructor trong các ngôn ngữ như C++ hay Java. Trong Python, việc tạo đối tượng thực sự được thực hiện bởi phương thức `__new__()`, còn `__init__()` chỉ đảm nhiệm việc khởi tạo đối tượng đã được tạo. Tuy nhiên, trong hầu hết các trường hợp, chúng ta chỉ cần làm việc với `__init__()` mà không cần quan tâm đến `__new__()`.

Một đặc điểm quan trọng của `__init__()` là nó không trả về giá trị. Nếu cố gắng trả về một giá trị khác `None`, Python sẽ báo lỗi. Điều này hợp lý vì mục đích của phương thức này là thiết lập đối tượng chứ không phải tạo ra một giá trị mới để trả về.

a, Cú pháp

```
class TenLop:
    def __init__(self, tham_so_1, tham_so_2):
        self.thuoc_tinh_1 = tham_so_1
        self.thuoc_tinh_2 = tham_so_2
```

Tham số `self` xuất hiện như tham số đầu tiên trong mọi phương thức của lớp và đóng vai trò vô cùng quan trọng trong OOP với Python. Tham số này là tham chiếu đến chính đối tượng mà phương thức đang được gọi trên đó. Khi chúng ta viết `obj.method()`, Python tự động chuyển đổi thành `Class.method(obj)`, do đó `self` sẽ nhận được giá trị là `obj`.

Tên `self` chỉ là một quy ước được cộng đồng Python tuân theo rộng rãi, không phải là từ khóa bắt buộc. Về mặt lý thuyết, chúng ta có thể đặt tên khác cho tham số này, nhưng việc sử dụng `self` giúp code trở nên dễ đọc và dễ hiểu hơn cho người khác. Thông qua `self`, các phương thức có thể truy cập và thay đổi các thuộc tính của đối tượng, cũng như gọi các phương thức khác của cùng đối tượng đó.

Một điểm quan trọng cần lưu ý là mặc dù `self` xuất hiện trong định nghĩa phương thức, chúng ta không truyền giá trị cho nó khi gọi phương thức. Python tự động xử lý việc này, điều này giúp cú pháp gọi phương thức trở nên gọn gàng và tự nhiên hơn.

b, Ví dụ sử dụng `__init__()`

```
class Student:
    def __init__(self, name, age, major):
        self.name = name
        self.age = age
        self.major = major

    # Tạo đối tượng với các giá trị ban đầu
    sv1 = Student("Nguyen Van An", 20, "Computer Science")
    sv2 = Student("Tran Thi Binh", 21, "Data Science")

    print(f"Sinh vien 1: {sv1.name}, {sv1.age} tuoi")
    print(f"Sinh vien 2: {sv2.name}, {sv2.age} tuoi")
```

Kết quả hiển thị:

Sinh vien 1: Nguyen Van An, 20 tuoi
 Sinh vien 2: Tran Thi Binh, 21 tuoi

Với cách này, mỗi khi tạo một đối tượng `Student`, các thuộc tính `name`, `age` và `major` sẽ được khởi tạo ngay lập tức với các giá trị được truyền vào.

5.3.2 Giá trị mặc định cho tham số

Các tham số trong `__init__()` có thể có giá trị mặc định, cho phép tạo đối tượng mà không cần truyền đầy đủ các tham số:

```
class Student:
    def __init__(self, name, age=18, major="Undeclared"):
        self.name = name
        self.age = age
        self.major = major

sv1 = Student("Nguyen Van An")
sv2 = Student("Tran Thi Binh", 21)
sv3 = Student("Le Van Cuong", 20, "AI")

print(f"{sv1.name}: {sv1.age} tuoi, {sv1.major}")
print(f"{sv2.name}: {sv2.age} tuoi, {sv2.major}")
print(f"{sv3.name}: {sv3.age} tuoi, {sv3.major}")
```

Kết quả hiển thị:

```
Nguyen Van An: 18 tuoi, Undeclared
Tran Thi Binh: 21 tuoi, Undeclared
Le Van Cuong: 20 tuoi, AI
```

5.4 Phương thức của lớp

5.4.1 Định nghĩa phương thức

Phương thức là các hàm được định nghĩa bên trong lớp. Chúng thể hiện các hành vi mà đối tượng có thể thực hiện. Mọi phương thức đều có tham số đầu tiên là `self`.

a, Ví dụ định nghĩa phương thức

```
class Student:
    def __init__(self, name, age, major):
        self.name = name
        self.age = age
        self.major = major
        self.gpa = 0.0

    def display_info(self):
        print(f"Ten: {self.name}")
        print(f"Tuoi: {self.age}")
        print(f"Chuyen nganh: {self.major}")
        print(f"GPA: {self.gpa}")
```

```

def update_gpa(self, new_gpa):
    if 0.0 <= new_gpa <= 4.0:
        self.gpa = new_gpa
        print(f"Cap nhat GPA thanh cong: {self.gpa}")
    else:
        print("GPA khong hop le. Phai nam trong khoang
              → 0.0 - 4.0")

# Sử dụng các phương thức
sv1 = Student("Nguyen Van An", 20, "Computer Science")
sv1.display_info()
print()
sv1.update_gpa(3.5)
print()
sv1.display_info()

```

Kết quả hiển thị:

Ten: Nguyen Van An
Tuoi: 20
Chuyen nganh: Computer Science
GPA: 0.0

Cap nhat GPA thanh cong: 3.5

Ten: Nguyen Van An
Tuoi: 20
Chuyen nganh: Computer Science
GPA: 3.5

5.4.2 Phương thức với giá trị trả về

Phương thức có thể trả về giá trị như các hàm thông thường:

```

class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height

    def calculate_perimeter(self):
        return 2 * (self.width + self.height)

```

```

def is_square(self):
    return self.width == self.height

# Sử dụng
rect = Rectangle(5, 10)
print(f"Dien tich: {rect.calculate_area()}")
print(f"Chu vi: {rect.calculate_perimeter()}")
print(f"La hinh vuong: {rect.is_square()}")

square = Rectangle(5, 5)
print(f"\nLa hinh vuong: {square.is_square()}")

```

Kết quả hiển thị:

Dien tich: 50
 Chu vi: 30
 La hinh vuong: False

La hinh vuong: True

5.5 Tính đóng gói (Encapsulation)**5.5.1 Khái niệm đóng gói**

Đóng gói là một trong bốn tính chất cơ bản của OOP. Nó đề cập đến việc ẩn giấu các chi tiết nội bộ của đối tượng và chỉ cung cấp các giao diện công khai để tương tác với đối tượng đó.

Đóng gói giúp:

- Bảo vệ dữ liệu khỏi việc truy cập và sửa đổi trực tiếp không mong muốn
- Tăng tính bảo mật của chương trình
- Dễ dàng thay đổi cài đặt nội bộ mà không ảnh hưởng đến code bên ngoài

5.5.2 Thuộc tính và phương thức riêng tư

Trong Python, quy ước đặt tên để chỉ ra mức độ truy cập của thuộc tính và phương thức:

- **Public:** Không có dấu gạch dưới ở đầu, có thể truy cập từ bất kỳ đâu
- **Protected:** Một dấu gạch dưới ở đầu (_), quy ước không nên truy cập từ bên ngoài lớp
- **Private:** Hai dấu gạch dưới ở đầu (__), khó truy cập hơn từ bên ngoài lớp

a, Ví dụ về đóng gói

```

class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number # Public
        self.__balance = balance             # Private

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Nop {amount}. So du moi:
                   {self.__balance}")
        else:
            print("So tien nopol phai lon hon 0")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Rut {amount}. So du con lai:
                  {self.__balance}")
        else:
            print("So tien khong hop le hoac khong du")

    def get_balance(self):
        return self.__balance

# Sử dụng
account = BankAccount("123456", 1000)
account.deposit(500)
account.withdraw(200)
print(f"So du hien tai: {account.get_balance()}")

# Không thể truy cập trực tiếp __balance từ bên ngoài
# print(account.__balance) # Sẽ gây lỗi

```

Kết quả hiển thị:

Nop 500. So du moi: 1500
 Rut 200. So du con lai: 1300
 So du hien tai: 1300

Trong ví dụ trên, thuộc tính `__balance` được bảo vệ, chỉ có thể thay đổi thông qua các phương thức `deposit()` và `withdraw()`, đảm bảo tính nhất quán của dữ liệu.

5.5.3 Getter và Setter

Getter và setter là các phương thức được sử dụng để truy cập và sửa đổi giá trị của thuộc tính private một cách có kiểm soát:

```

class Student:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    # Getter cho name
    def get_name(self):
        return self.__name

    # Setter cho name
    def set_name(self, name):
        if isinstance(name, str) and len(name) > 0:
            self.__name = name
        else:
            print("Ten khong hop le")

    # Getter cho age
    def get_age(self):
        return self.__age

    # Setter cho age
    def set_age(self, age):
        if isinstance(age, int) and 0 < age < 150:
            self.__age = age
        else:
            print("Tuoi khong hop le")

    # Sử dụng
sv = Student("Nguyen Van An", 20)
print(f"Ten: {sv.get_name()}, Tuoi: {sv.get_age()}")

sv.set_age(21)
print(f"Tuoi moi: {sv.get_age()}")

sv.set_age(-5)  # Sẽ bị từ chối

```

Kết quả hiển thị:

Ten: Nguyen Van An, Tuoi: 20

Tuoi moi: 21

Tuoi khong hop le

5.5.4 Property decorator

Python cung cấp decorator `@property` để tạo getter và setter một cách pythonic hơn:

```
class Temperature:
    def __init__(self, celsius):
        self.__celsius = celsius

    @property
    def celsius(self):
        return self.__celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            print("Nhiet do khong the thap hon -273.15°C")
        else:
            self.__celsius = value

    @property
    def fahrenheit(self):
        return self.__celsius * 9/5 + 32

# Sử dụng
temp = Temperature(25)
print(f"Nhiet do: {temp.celsius}°C = {temp.fahrenheit}°F")

temp.celsius = 30
print(f"Nhiet do moi: {temp.celsius}°C =
    {temp.fahrenheit}°F")

temp.celsius = -300 # Sẽ bị từ chối
```

Kết quả hiển thị:

```
Nhiet do: 25°C = 77.0°F
Nhiet do moi: 30°C = 86.0°F
Nhiet do khong the thap hon -273.15°C
```

5.6 Tính kế thừa (Inheritance)

5.6.1 Khái niệm kế thừa

Kế thừa là cơ chế cho phép một lớp (lớp con) kế thừa các thuộc tính và phương thức từ một lớp khác (lớp cha). Điều này giúp tái sử dụng mã nguồn và tạo ra mối quan hệ phân cấp giữa các lớp.

Lợi ích của kế thừa:

- Tái sử dụng code: Không cần viết lại các thuộc tính và phương thức đã có trong lớp cha
- Mở rộng chức năng: Lớp con có thể thêm các thuộc tính và phương thức mới
- Ghi đè (override): Lớp con có thể thay đổi cách thức hoạt động của phương thức từ lớp cha

5.6.2 Cú pháp kế thừa

```
class LopCha:
    # Định nghĩa lớp cha
    pass

class LopCon(LopCha):
    # Định nghĩa lớp con
    pass
```

5.6.3 Ví dụ kế thừa cơ bản

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Ten: {self.name}")
        print(f"Tuoi: {self.age}")

class Student(Person):
    def __init__(self, name, age, student_id, major):
        # Gọi constructor của lớp cha
        super().__init__(name, age)
        self.student_id = student_id
        self.major = major

    def display_info(self):
        # Gọi phương thức của lớp cha
        super().display_info()
        print(f"Ma sinh vien: {self.student_id}")
        print(f"Chuyen nganh: {self.major}")

    # Sử dụng
person = Person("Nguyen Van A", 30)
print("Thong tin person:")
person.display_info()
```

```

print("\nThong tin student:")
student = Student("Tran Thi B", 20, "SV001", "Computer
→ Science")
student.display_info()

```

Kết quả hiển thị:

Thong tin person:

Ten: Nguyen Van A

Tuoi: 30

Thong tin student:

Ten: Tran Thi B

Tuoi: 20

Ma sinh vien: SV001

Chuyen nganh: Computer Science

5.6.4 Phương thức super()

Hàm `super()` được sử dụng để gọi phương thức của lớp cha từ lớp con. Điều này đặc biệt hữu ích khi muốn mở rộng chức năng của lớp cha mà không viết lại toàn bộ code.

5.6.5 Kế thừa nhiều cấp

Python hỗ trợ kế thừa nhiều cấp, cho phép tạo ra chuỗi kế thừa dài:

```

class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} phat ra tieng keu")

class Mammal(Animal):
    def __init__(self, name, fur_color):
        super().__init__(name)
        self.fur_color = fur_color

    def feed_milk(self):
        print(f"{self.name} cho con bu")

class Dog(Mammal):
    def __init__(self, name, fur_color, breed):
        super().__init__(name, fur_color)

```

```

        self.breed = breed

    def speak(self):
        print(f"{self.name} sua: Gau gau!")

    def fetch(self):
        print(f"{self.name} chay theo qua bong")

# Sử dụng
dog = Dog("Lucky", "Nau", "Golden Retriever")
dog.speak()
dog.feed_milk()
dog.fetch()
print(f"Mau long: {dog.fur_color}")
print(f"Giong: {dog.breed}")

```

Kết quả hiển thị:

Lucky sua: Gau gau!
 Lucky cho con bu
 Lucky chay theo qua bong
 Mau long: Nau
 Giong: Golden Retriever

5.6.6 Kiểm tra quan hệ kế thừa

Python cung cấp các hàm để kiểm tra quan hệ kế thừa giữa các lớp và đối tượng:

```

class Animal:
    pass

class Dog(Animal):
    pass

dog = Dog()

# Kiểm tra kiểu của đối tượng
print(isinstance(dog, Dog))      # True
print(isinstance(dog, Animal))    # True
print(isinstance(dog, object))    # True (mọi lớp đều kế thừa
                                ← từ object)

# Kiểm tra quan hệ giữa các lớp
print(issubclass(Dog, Animal))   # True
print(issubclass(Animal, Dog))    # False

```

Kết quả hiển thị:

```
True
True
True
True
False
```

5.7 Tính đa hình (Polymorphism)**5.7.1 Khái niệm đa hình**

Đa hình là khả năng của các đối tượng thuộc các lớp khác nhau có thể phản ứng khác nhau đối với cùng một thông điệp (gọi phương thức cùng tên). Đa hình cho phép viết code linh hoạt hơn, có thể làm việc với nhiều loại đối tượng khác nhau mà không cần biết chính xác kiểu của chúng.

5.7.2 Đa hình thông qua ghi đè phương thức

Lớp con có thể ghi đè (override) phương thức của lớp cha để cung cấp cài đặt riêng:

```
class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        return 0

    def describe(self):
        print(f"Hinh {self.name} co dien tich:
              {self.area()}")

class Circle(Shape):
    def __init__(self, radius):
        super().__init__("Tron")
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

class Rectangle(Shape):
    def __init__(self, width, height):
        super().__init__("Chu nhat")
        self.width = width
        self.height = height

    def area(self):
```

```

        return self.width * self.height

class Triangle(Shape):
    def __init__(self, base, height):
        super().__init__("Tam giac")
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

# Sử dụng đa hình
shapes = [
    Circle(5),
    Rectangle(4, 6),
    Triangle(3, 8)
]

for shape in shapes:
    shape.describe()

```

Kết quả hiển thị:

Hinh Tron co dien tich: 78.53975

Hinh Chu nhat co dien tich: 24

Hinh Tam giac co dien tich: 12.0

Trong ví dụ trên, mặc dù các đối tượng có kiểu khác nhau, chúng ta vẫn có thể gọi phương thức `describe()` và `area()` một cách thống nhất. Đây là sức mạnh của đa hình.

5.7.3 Đa hình với các toán tử

Python cho phép định nghĩa lại các toán tử cho các lớp tự định nghĩa thông qua các phương thức đặc biệt:

```

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):

```

```

        return Vector2D(self.x - other.x, self.y - other.y)

    def __str__(self):
        return f"Vector2D({self.x}, {self.y})"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

# Sử dụng
v1 = Vector2D(3, 4)
v2 = Vector2D(1, 2)

v3 = v1 + v2
print(f"v1 + v2 = {v3}")

v4 = v1 - v2
print(f"v1 - v2 = {v4}")

print(f"v1 == v2: {v1 == v2}")
print(f"v1 == v1: {v1 == v1}")

```

Kết quả hiển thị:

```

v1 + v2 = Vector2D(4, 6)
v1 - v2 = Vector2D(2, 2)
v1 == v2: False
v1 == v1: True

```

5.8 Các phương thức đặc biệt (Magic Methods)**5.8.1 Giới thiệu về magic methods**

Magic methods (còn gọi là dunder methods - double underscore methods) là các phương thức đặc biệt trong Python có tên bắt đầu và kết thúc bằng hai dấu gạch dưới. Chúng cho phép các lớp tự định nghĩa tương tác với các tính năng built-in của Python.

5.8.2 Một số magic methods thường dùng

Phương thức	Mục đích
<code>__init__()</code>	Khởi tạo đối tượng
<code>__str__()</code>	Biểu diễn chuỗi của đối tượng (cho print)
<code>__repr__()</code>	Biểu diễn chính thức của đối tượng
<code>__len__()</code>	Trả về độ dài khi gọi len()
<code>__eq__()</code>	So sánh bằng (==)
<code>__lt__()</code>	So sánh nhỏ hơn (<)
<code>__gt__()</code>	So sánh lớn hơn (>)
<code>__add__()</code>	Phép cộng (+)
<code>__sub__()</code>	Phép trừ (-)
<code>__mul__()</code>	Phép nhân (*)

5.8.3 Ví dụ sử dụng magic methods

```

class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f'{self.title} by {self.author}'

    def __repr__(self):
        return f"Book('{self.title}', '{self.author}', {self.pages})"

    def __len__(self):
        return self.pages

    def __eq__(self, other):
        return (self.title == other.title and
                self.author == other.author)

    def __lt__(self, other):
        return self.pages < other.pages

# Sử dụng
book1 = Book("Python Programming", "John Smith", 300)
book2 = Book("Data Science", "Jane Doe", 250)
book3 = Book("Python Programming", "John Smith", 300)

```

```

print(book1)                                # Gọi __str__
print(repr(book2))                         # Gọi __repr__
print(f"So trang: {len(book1)}") # Gọi __len__
print(f"book1 == book3: {book1 == book3}") # Gọi __eq__
print(f"book2 < book1: {book2 < book1}")   # Gọi __lt__

```

Kết quả hiển thị:

```

'Python Programming' cua John Smith
Book('Data Science', 'Jane Doe', 250)
So trang: 300
book1 == book3: True
book2 < book1: True

```

5.9 Thuộc tính và phương thức lớp (Class Attributes and Methods)

5.9.1 Thuộc tính lớp

Thuộc tính lớp là thuộc tính được chia sẻ bởi tất cả các đối tượng của lớp đó. Chúng được định nghĩa bên trong lớp nhưng bên ngoài các phương thức.

```

class Student:
    # Thuộc tính lớp
    school_name = "Dai hoc Bach Khoa Ha Noi"
    student_count = 0

    def __init__(self, name, student_id):
        # Thuộc tính đối tượng
        self.name = name
        self.student_id = student_id
        # Tăng số lượng sinh viên
        Student.student_count += 1

    def display_info(self):
        print(f"Sinh vien: {self.name}")
        print(f"Ma SV: {self.student_id}")
        print(f"Truong: {Student.school_name}")

    # Sử dụng
    print(f"So sinh vien ban dau: {Student.student_count}")

sv1 = Student("Nguyen Van An", "SV001")
sv2 = Student("Tran Thi Binh", "SV002")

print(f"So sinh vien hien tai: {Student.student_count}")

```

```
sv1.display_info()
```

Kết quả hiển thị:

```
So sinh vien ban dau: 0
So sinh vien hien tai: 2
Sinh vien: Nguyen Van An
Ma SV: SV001
Truong: Dai hoc Bach Khoa Ha Noi
```

5.9.2 Phương thức lớp

Phương thức lớp được định nghĩa bằng decorator `@classmethod` và nhận tham số đầu tiên là `cls` (tham chiếu đến lớp, không phải đối tượng):

```
class Employee:
    company_name = "Tech Corp"
    raise_amount = 1.05

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def apply_raise(self):
        self.salary = int(self.salary * self.raise_amount)

    @classmethod
    def set_raise_amount(cls, amount):
        cls.raise_amount = amount

    @classmethod
    def from_string(cls, emp_str):
        name, salary = emp_str.split('-')
        return cls(name, int(salary))

    def display_info(self):
        print(f"Nhan vien: {self.name}, Luong:
              {self.salary}")

    # Sử dụng phương thức lớp
Employee.set_raise_amount(1.10)

emp1 = Employee("An", 50000)
```

```
emp2 = Employee.from_string("Binh-60000")

emp1.display_info()
emp1.apply_raise()
emp1.display_info()

emp2.display_info()
```

Kết quả hiển thị:

Nhan vien: An, Luong: 50000
 Nhan vien: An, Luong: 55000
 Nhan vien: Binh, Luong: 60000

5.9.3 Phương thức tĩnh

Phương thức tĩnh được định nghĩa bằng decorator `@staticmethod` và không nhận `self` hay `cls`. Chúng giống như các hàm thông thường nhưng thuộc về namespace của lớp:

```
class MathUtils:

    @staticmethod
    def is_even(num):
        return num % 2 == 0

    @staticmethod
    def is_prime(num):
        if num < 2:
            return False
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                return False
        return True

    @staticmethod
    def factorial(n):
        if n <= 1:
            return 1
        return n * MathUtils.factorial(n - 1)

    # Sử dụng phương thức tĩnh
    print(f"10 la so chan: {MathUtils.is_even(10)}")
    print(f"7 la so nguyen to: {MathUtils.is_prime(7)}")
    print(f"5! = {MathUtils.factorial(5)}")
```

Kết quả hiển thị:

```
10 la so chan: True
7 la so nguyen to: True
5! = 120
```

5.10 Ứng dụng OOP trong bài toán thực tế**5.10.1 Hệ thống quản lý thư viện**

Dưới đây là một ví dụ tổng hợp áp dụng các khái niệm OOP vào bài toán quản lý thư viện:

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_borrowed = False

    def borrow(self):
        if not self.is_borrowed:
            self.is_borrowed = True
            return True
        return False

    def return_book(self):
        self.is_borrowed = False

    def __str__(self):
        status = "Đã muon" if self.is_borrowed else "Chưa muon"
        return f"{self.title} - {self.author} ({status})"

class Member:
    def __init__(self, name, member_id):
        self.name = name
        self.member_id = member_id
        self.borrowed_books = []

    def borrow_book(self, book):
        if book.borrow():
            self.borrowed_books.append(book)
            print(f"{self.name} da muon: {book.title}")
            return True
        else:
```

```

        print(f"Sach '{book.title}' khong co san")
        return False

    def return_book(self, book):
        if book in self.borrowed_books:
            book.return_book()
            self.borrowed_books.remove(book)
            print(f"{self.name} da tra: {book.title}")
        else:
            print(f"{self.name} khong muon sach nay")

    class Library:
        def __init__(self, name):
            self.name = name
            self.books = []
            self.members = []

        def add_book(self, book):
            self.books.append(book)
            print(f"Da them sach: {book.title}")

        def register_member(self, member):
            self.members.append(member)
            print(f"Da dang ky thanh vien: {member.name}")

        def display_available_books(self):
            print(f"\nSach con trong tai {self.name}:")
            for book in self.books:
                if not book.is_borrowed:
                    print(f" - {book}")

    # Sử dụng hệ thống
library = Library("Thu vien HUCE")

    # Thêm sách
book1 = Book("Python Programming", "John Smith", "ISBN001")
book2 = Book("Data Structures", "Jane Doe", "ISBN002")
book3 = Book("Algorithms", "Bob Johnson", "ISBN003")

library.add_book(book1)
library.add_book(book2)
library.add_book(book3)

    # Đăng ký thành viên

```

```

member1 = Member("Nguyen Van An", "M001")
member2 = Member("Tran Thi Binh", "M002")

library.register_member(member1)
library.register_member(member2)

# Muốn sách
print("\n--- Muon sach ---")
member1.borrow_book(book1)
member1.borrow_book(book2)
member2.borrow_book(book1) # Sách đã được mượn

# Hiển thị sách còn trong
library.display_available_books()

# Trả sách
print("\n--- Tra sach ---")
member1.return_book(book1)

# Hiển thị lại
library.display_available_books()

```

Kết quả hiển thị:

Da them sach: Python Programming

Da them sach: Data Structures

Da them sach: Algorithms

Da dang ky thanh vien: Nguyen Van An

Da dang ky thanh vien: Tran Thi Binh

--- Muon sach ---

Nguyen Van An da muon: Python Programming

Nguyen Van An da muon: Data Structures

Sach 'Python Programming' khong co san

Sach con trong tai Thu vien HUCE:

- Algorithms - Bob Johnson (Con trong)

--- Tra sach ---

Nguyen Van An da tra: Python Programming

Sach con trong tai Thu vien HUCE:

- Python Programming - John Smith (Con trong)

- Algorithms - Bob Johnson (Con trong)

5.11 Bài tập thực hành

5.11.1 Bài tập cơ bản

Bài 1. Tạo lớp BankAccount với các yêu cầu:

- Thuộc tính: số tài khoản, tên chủ tài khoản, số dư
- Phương thức: nộp tiền, rút tiền, kiểm tra số dư
- Không cho phép rút quá số dư hiện có

Bài 2. Tạo lớp Circle với:

- Thuộc tính: bán kính
- Phương thức: tính diện tích, tính chu vi
- Sử dụng property để đảm bảo bán kính luôn dương

Bài 3. Tạo lớp Product đại diện cho sản phẩm trong cửa hàng với:

- Thuộc tính: tên, giá, số lượng tồn kho
- Phương thức: bán hàng (giảm tồn kho), nhập hàng (tăng tồn kho)
- Không cho phép bán khi hết hàng

5.11.2 Bài tập nâng cao

Bài 4. Xây dựng hệ thống quản lý phương tiện giao thông:

- Tạo lớp cha Vehicle với thuộc tính: tên, tốc độ tối đa
- Tạo các lớp con: Car, Motorcycle, Truck
- Mỗi lớp con có thuộc tính riêng (ví dụ: Car có số ghế ngồi, Truck có tải trọng)
- Tất cả đều có phương thức display_info() nhưng hiển thị thông tin khác nhau

Bài 5. Xây dựng hệ thống quản lý sinh viên:

- Lớp Student với thuộc tính: họ tên, mã sinh viên, danh sách điểm các môn
- Phương thức: thêm điểm, tính điểm trung bình, xếp loại học lực
- Sử dụng property để bảo vệ các thuộc tính quan trọng
- Sử dụng magic methods để so sánh sinh viên theo điểm trung bình

Bài 6. Tạo lớp Matrix đại diện cho ma trận với:

- Khởi tạo từ list 2 chiều
- Các phép toán: cộng, trừ, nhân ma trận (sử dụng magic methods)
- Phương thức: chuyển vị, tính định thức (nếu là ma trận vuông)

5.12 Tổng kết chương

Lập trình hướng đối tượng là một trong những kỹ năng quan trọng nhất mà sinh viên ngành Khoa học Máy tính cần nắm vững. Trong chương này, chúng ta đã học về:

- Khái niệm cơ bản: lớp, đối tượng, thuộc tính, phương thức
- Phương thức khởi tạo (`__init__`) và vai trò của `self`
- Tính đóng gói: cách bảo vệ dữ liệu thông qua thuộc tính private và getter/setter
- Tính kế thừa: tái sử dụng code và mở rộng chức năng
- Tính đa hình: cùng một giao diện, nhiều cách thức thực hiện khác nhau
- Magic methods: tùy biến cách đối tượng tương tác với các tính năng built-in
- Thuộc tính và phương thức lớp: dữ liệu và hành vi được chia sẻ

OOP không chỉ là một kỹ thuật lập trình mà còn là một cách tư duy về cách tổ chức và thiết kế chương trình. Việc nắm vững OOP sẽ giúp sinh viên:

- Viết code dễ đọc, dễ bảo trì và dễ mở rộng
- Làm việc hiệu quả với các framework và thư viện hiện đại
- Phát triển các hệ thống phần mềm quy mô lớn
- Chuẩn bị tốt cho việc học các chủ đề nâng cao hơn

Để thành thạo OOP, sinh viên cần luyện tập thường xuyên thông qua các bài tập và dự án thực tế. Hãy bắt đầu từ những ví dụ đơn giản, sau đó dần dần xây dựng các hệ thống phức tạp hơn.

5.13 Bài tập

5.13.1 Bài tập 1: Class Sinh viên

Tạo class `SinhVien` để quản lý thông tin sinh viên với các yêu cầu:

Thuộc tính:

- `ma_sv`: Mã sinh viên
- `ho_ten`: Họ tên sinh viên
- `nam_sinh`: Năm sinh
- `diem_list`: Danh sách các điểm (mặc định là list rỗng)

Phương thức:

- `them_diem(diem)`: Thêm điểm mới (kiểm tra $0 \leq \text{điểm} \leq 10$)
- `tinh_diem_tb()`: Tính điểm trung bình
- `xep_loai()`: Xếp loại học lực (Xuất sắc ≥ 9 , Giỏi ≥ 8 , Khá ≥ 7 , Trung bình ≥ 5 , Yếu < 5)

- `__str__()`: Hiển thị thông tin sinh viên

5.13.2 Bài tập 2: Class Hình học

Tạo hệ thống phân cấp class cho các hình học:

Class cơ sở Hình:

- Phương thức `dien_tich()`: Tính diện tích (abstract)
- Phương thức `chu_vi()`: Tính chu vi (abstract)
- Phương thức `__str__()`: Hiển thị thông tin hình

Các class con:

1. `HinhTron(ban_kinh)`: Kế thừa từ `Hinh`
2. `HinhChuNhat(chieu_dai, chieu_rong)`: Kế thừa từ `Hinh`
3. `TamGiac(a, b, c)`: Kế thừa từ `Hinh` (sử dụng công thức Heron)

Tạo danh sách các hình và tính tổng diện tích.

5.13.3 Bài tập 3: Class TaiKhoan Ngân hàng

Tạo class `TaiKhoan` để quản lý tài khoản ngân hàng:

Thuộc tính:

- `so_tai_khoan`: Số tài khoản
- `chu_tai_khoan`: Chủ tài khoản
- `__so_du`: Số dư (private attribute)
- `lich_su`: Lịch sử giao dịch
- `lai_suat_nam`: Lãi suất (class variable = 0.05)

Phương thức:

- `nap_tien(so_tien)`: Nạp tiền vào tài khoản
- `rut_tien(so_tien)`: Rút tiền (kiểm tra số dư đủ)
- `chuyen_tien(tai_khoan_nhan, so_tien)`: Chuyển tiền đến tài khoản khác
- `xem_so_du()`: Xem số dư hiện tại
- `_ghi_lich_su(loai, so_tien)`: Ghi lại giao dịch (protected method)
- `in_lich_su()`: In lịch sử các giao dịch

Yêu cầu: Sử dụng encapsulation để bảo vệ thuộc tính số dư, ghi lại thời gian và loại giao dịch trong lịch sử.

Ghi chú: Xem lời giải chi tiết tại Phụ lục - Lời giải bài tập.

CHƯƠNG 6. Thư viện NumPy trong Kỹ thuật

6.1 Giới thiệu về NumPy

NumPy (viết tắt của Numerical Python) là thư viện cốt lõi cho tính toán khoa học và xử lý dữ liệu số trong Python. Được phát triển từ năm 2005, NumPy cung cấp cấu trúc dữ liệu mảng đa chiều (`ndarray`) với hiệu suất cao và một tập hợp phong phú các hàm toán học để thao tác trên các mảng này.

Trong lĩnh vực kỹ thuật, NumPy đóng vai trò nền tảng cho hầu hết các công việc liên quan đến tính toán số, xử lý ma trận, phân tích dữ liệu đo lường, mô phỏng hệ thống và xử lý tín hiệu. Sự hiệu quả của NumPy đến từ việc các phép toán được thực thi bằng mã C tối ưu, nhanh hơn nhiều lần so với các vòng lặp Python thông thường.

6.1.1 Tại sao NumPy quan trọng trong kỹ thuật

Trong công việc kỹ thuật, chúng ta thường xuyên làm việc với các tập dữ liệu lớn từ cảm biến, mô phỏng số, hoặc đo lường thực nghiệm. NumPy giúp:

- Xử lý ma trận và vector một cách hiệu quả, phục vụ tính toán đại số tuyến tính
- Thực hiện các phép biến đổi toán học phức tạp trên tập dữ liệu lớn
- Tiết kiệm bộ nhớ nhờ cấu trúc dữ liệu tối ưu
- Tích hợp dễ dàng với các thư viện khác như SciPy, Pandas, Matplotlib, TensorFlow
- Hỗ trợ broadcasting - cơ chế tự động mở rộng mảng khi thực hiện phép toán

6.1.2 Cài đặt và import NumPy

Cài đặt NumPy:

```
# Cài đặt qua pip  
pip install numpy  
  
# Hoặc qua conda  
conda install numpy
```

Import thư viện:

```
import numpy as np  
  
# Kiểm tra phiên bản  
print(np.__version__)
```

Quy ước sử dụng alias `np` là chuẩn trong cộng đồng Python và được khuyến nghị sử dụng để code ngắn gọn hơn.

6.2 Mảng NumPy - ndarray

6.2.1 Khái niệm mảng đa chiều

Mảng NumPy (ndarray) là cấu trúc dữ liệu đồng nhất chứa các phần tử cùng kiểu. Khác với list Python có thể chứa nhiều kiểu dữ liệu khác nhau, mảng NumPy yêu cầu tất cả phần tử có cùng kiểu, điều này giúp tối ưu hóa bộ nhớ và tốc độ tính toán.

Mảng có thể có nhiều chiều:

- Mảng 1 chiều (1D): vector - biểu diễn chuỗi dữ liệu đo, tín hiệu
- Mảng 2 chiều (2D): ma trận - biểu diễn dữ liệu dạng bảng, ảnh grayscale
- Mảng 3 chiều (3D) trở lên: tensor - biểu diễn ảnh màu, dữ liệu không gian-thời gian

6.2.2 Tạo mảng NumPy

a, Tạo từ list Python

```
import numpy as np

# Mảng 1 chiều từ list
arr1d = np.array([1, 2, 3, 4, 5])
print("Mảng 1D:", arr1d)
print("Kiểu:", type(arr1d))

# Mảng 2 chiều từ list lồng nhau
arr2d = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
print("\nMảng 2D:")
print(arr2d)

# Mảng 3 chiều
arr3d = np.array([[[1, 2], [3, 4]],
                  [[5, 6], [7, 8]]])
print("\nMảng 3D:")
print(arr3d)
```

Kết quả:

Mảng 1D: [1 2 3 4 5]
 Kiểu: <class 'numpy.ndarray'>

Mảng 2D:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Mảng 3D:

```
[[ [1 2]
  [3 4]]]
```

```
[ [5 6]
  [7 8]]]
```

b, Các hàm tạo mảng đặc biệt

NumPy cung cấp nhiều hàm để tạo mảng với giá trị đặc biệt, rất hữu ích trong các tính toán kỹ thuật.

```
# Ma trận không (zeros)
zeros = np.zeros((3, 4))
print("Ma tran khong 3x4:")
print(zeros)

# Ma trận một (ones)
ones = np.ones((2, 3))
print("\nMa tran mot 2x3:")
print(ones)

# Ma trận đơn vị (identity matrix)
identity = np.eye(4)
print("\nMa tran don vi 4x4:")
print(identity)

# Ma trận với giá trị cụ thể
full = np.full((3, 3), 7)
print("\nMa tran toan gia tri 7:")
print(full)

# Ma trận đường chéo
diag = np.diag([1, 2, 3, 4])
print("\nMa tran duong cheo:")
print(diag)
```

Kết quả:

Ma tran khong 3x4:

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Mảng một chiều:

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Mảng đơn vị 4x4:

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

c, Tạo mảng với dãy số

```
# Tạo dãy từ 0 đến 9
arange1 = np.arange(10)
print("arange(10):", arange1)

# Tạo dãy từ 5 đến 15 (không bao gồm 15)
arange2 = np.arange(5, 15)
print("arange(5, 15):", arange2)

# Tạo dãy với bước nhảy
arange3 = np.arange(0, 1, 0.1)
print("arange(0, 1, 0.1):", arange3)

# Tạo dãy chia đều (linspace)
linspace1 = np.linspace(0, 10, 5)
print("linspace(0, 10, 5):", linspace1)

# Dãy logarit
logspace1 = np.logspace(0, 3, 4)
print("logspace(0, 3, 4):", logspace1)
```

Kết quả:

```
arange(10): [0 1 2 3 4 5 6 7 8 9]
arange(5, 15): [ 5  6  7  8  9 10 11 12 13 14]
arange(0, 1, 0.1): [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
linspace(0, 10, 5): [ 0.    2.5   5.    7.5  10. ]
logspace(0, 3, 4): [ 1.    10.   100.  1000.]
```

Ứng dụng trong kỹ thuật: Hàm `linspace` thường được dùng để tạo trực thời gian hoặc trực tần số trong phân tích tín hiệu.

d, Tạo mảng ngẫu nhiên

```
# Số ngẫu nhiên phân phối đều [0, 1)
rand1 = np.random.rand(3, 3)
print("random.rand(3, 3):")
print(rand1)

# Số ngẫu nhiên phân phối chuẩn (mean=0, std=1)
randn = np.random.randn(3, 3)
print("\nrandom.randn(3, 3):")
print(randn)

# Số nguyên ngẫu nhiên
randint = np.random.randint(0, 100, (3, 4))
print("\nrandom.randint(0, 100, (3, 4)):")
print(randint)

# Chọn ngẫu nhiên từ mảng cho trước
choice = np.random.choice([10, 20, 30, 40], size=(2, 3))
print("\nrandom.choice:")
print(choice)
```

6.2.3 Thuộc tính của mảng NumPy

Mảng NumPy có nhiều thuộc tính quan trọng giúp hiểu rõ cấu trúc dữ liệu.

```
arr = np.array([[1, 2, 3, 4],
               [5, 6, 7, 8],
               [9, 10, 11, 12]])

print("Mang goc:")
print(arr)
print("\nshape (kich thuoc):", arr.shape)      # (3, 4)
print("ndim (so chieu):", arr.ndim)            # 2
print("size (tong phan tu):", arr.size)         # 12
print("dtype (kieu du lieu):", arr.dtype)        # int64
print("itemsize (kich thuoc 1 phan tu):", arr.itemsize) # 8
→ bytes
print(" nbytes (tong kich thuoc):", arr.nbytes) # 96 bytes
```

Kết quả:

Mang goc:

```
[[ 1  2  3  4]
 [ 5  6  7  8]]
```

```
[ 9 10 11 12]]
```

```
shape (kich thuoc): (3, 4)
ndim (so chieu): 2
size (tong phan tu): 12
dtype (kieu du lieu): int64
itemsize (kich thuoc 1 phan tu): 8
nbytes (tong kich thuoc): 96
```

6.2.4 Kiểu dữ liệu trong NumPy

NumPy hỗ trợ nhiều kiểu dữ liệu số với độ chính xác khác nhau, cho phép tối ưu bộ nhớ và tốc độ tính toán.

```
# Chỉ định kiểu dữ liệu khi tạo mảng
int32_arr = np.array([1, 2, 3], dtype=np.int32)
float64_arr = np.array([1.5, 2.5, 3.5], dtype=np.float64)
bool_arr = np.array([True, False, True], dtype=np.bool_)

print("int32:", int32_arr.dtype)
print("float64:", float64_arr.dtype)
print("bool:", bool_arr.dtype)

# Chuyển đổi kiểu dữ liệu
arr_float = np.array([1, 2, 3])
arr_int = arr_float.astype(np.float32)
print("\nChuyen doi kieu:", arr_int.dtype)
```

Các kiểu dữ liệu phổ biến:

- int8, int16, int32, int64: Số nguyên có dấu
- uint8, uint16, uint32, uint64: Số nguyên không dấu
- float16, float32, float64: Số thực dấu phẩy động
- complex64, complex128: Số phức
- bool_: Boolean

6.3 Chỉ mục và cắt lát mảng NumPy

Việc truy cập và thao tác với các phần tử hoặc các nhóm phần tử trong mảng là kỹ năng cơ bản nhưng cực kỳ quan trọng khi làm việc với NumPy. NumPy cung cấp nhiều cách linh hoạt để lấy dữ liệu từ mảng, từ chỉ mục đơn giản đến các kỹ thuật nâng cao như fancy indexing và boolean indexing.

6.3.1 Chỉ mục cơ bản

a, Chỉ mục mảng 1 chiều

Tương tự như list Python, mảng 1 chiều có thể truy cập bằng chỉ số từ 0.

```
arr = np.array([10, 20, 30, 40, 50])

print("Phan tu dau tien:", arr[0])      # 10
print("Phan tu cuoi cung:", arr[-1])    # 50
print("Phan tu thu 3:", arr[2])          # 30

# Thay đổi giá trị
arr[1] = 999
print("Sau khi thay doi:", arr)
```

Kết quả:

```
Phan tu dau tien: 10
Phan tu cuoi cung: 50
Phan tu thu 3: 30
Sau khi thay doi: [ 10 999 30 40 50]
```

b, Chỉ mục mảng 2 chiều

Với mảng 2 chiều, ta sử dụng cú pháp `arr[row, col]` để truy cập phần tử. Lưu ý rằng chỉ số hàng (row) đứng trước, chỉ số cột (col) đứng sau.

```
matrix = np.array([[10, 20, 30],
                  [40, 50, 60],
                  [70, 80, 90]])

print("Phan tu o hang 0, cot 1:", matrix[0, 1])      # 20
print("Phan tu o hang 2, cot 2:", matrix[2, 2])      # 90
print("Phan tu o hang 1, cot 0:", matrix[1, 0])      # 40

# Có thể dùng chỉ số âm
print("Hang cuoi, cot cuoi:", matrix[-1, -1])      # 90

# Thay đổi giá trị
matrix[1, 1] = 555
print("\nMa tran sau khi thay doi:")
print(matrix)
```

Kết quả:

```
Phan tu o hang 0, cot 1: 20
```

```
Phan tu o hang 2, cot 2: 90
Phan tu o hang 1, cot 0: 40
Hang cuoi, cot cuoi: 90
```

Ma tran sau khi thay doi:

```
[[ 10  20  30]
 [ 40 555  60]
 [ 70  80  90]]
```

6.3.2 Cắt lát (Slicing)

Slicing cho phép trích xuất một phần của mảng theo cú pháp `start:stop:step`. Đây là công cụ mạnh mẽ để lấy các dòng, cột hoặc vùng dữ liệu cụ thể.

a, Cắt lát mảng 1 chiều

```
arr = np.array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

print("Phan tu tu 2 den 5:", arr[2:6])          # [20 30 40 50]
print("Tu dau den chi so 4:", arr[:5])         # [0 10 20 30
→   40]
print("Tu chi so 5 den cuoi:", arr[5:])        # [50 60 70 80
→   90]
print("Tat ca phan tu:", arr[:])                # Toan bo mang
print("Phan tu chan (buoc 2):", arr[::2])       # [0 20 40 60
→   80]
print("Dao nguoc mang:", arr[::-1])             # [90 80 70 ...]
→   0]
```

Kết quả:

```
Phan tu tu 2 den 5: [20 30 40 50]
Tu dau den chi so 4: [ 0 10 20 30 40]
Tu chi so 5 den cuoi: [50 60 70 80 90]
Tat ca phan tu: [ 0 10 20 30 40 50 60 70 80 90]
Phan tu chan (buoc 2): [ 0 20 40 60 80]
Dao nguoc mang: [90 80 70 60 50 40 30 20 10  0]
```

b, Cắt lát mảng 2 chiều

Với mảng 2 chiều, ta có thể cắt lát theo từng chiều một cách độc lập.

```
A = np.array([[10, 20, 30, 40],
              [50, 60, 70, 80],
              [90, 100, 110, 120],
              [130, 140, 150, 160]])
```

```

print("Ma tran goc:")
print(A)

# Lấy toàn bộ dòng đầu tiên
print("\nDong dau tien:", A[0, :])

# Lấy toàn bộ cột thứ hai
print("Cot thu hai:", A[:, 1])

# Lấy ma trận con 2x2 từ góc trên trái
print("\nMa tran con 2x2 (trai tren):")
print(A[:2, :2])

# Lấy ma trận con 2x3 ở giữa
print("\nMa tran con 2x3 (giua):")
print(A[1:3, 1:4])

# Lấy từng dòng cách nhau
print("\nCac dong chan:")
print(A[::-2, :])

# Lấy từng cột cách nhau
print("\nCac cot le:")
print(A[:, 1::2])

```

Kết quả:

Ma tran goc:

```

[[ 10   20   30   40]
 [ 50   60   70   80]
 [ 90  100  110  120]
 [130  140  150  160]]

```

Dong dau tien: [10 20 30 40]
Cot thu hai: [20 60 100 140]

Ma tran con 2x2 (trai tren):

```

[[10 20]
 [50 60]]

```

Ma tran con 2x3 (giua):

```

[[ 60   70   80]
 [ 90  100  110]]

```

```
[100 110 120] ]
```

c, Lưu ý về view và copy

Một điểm quan trọng: slicing trong NumPy tạo ra *view* (tham chiếu) chứ không phải *copy* (bản sao). Thay đổi trên view sẽ ảnh hưởng đến mảng gốc.

```
arr = np.array([1, 2, 3, 4, 5])

# Tạo view
sub_arr = arr[1:4]
print("Mang con:", sub_arr)

# Thay đổi view
sub_arr[0] = 999
print("Mang goc sau khi thay doi view:", arr)

# Tạo copy thật sự
arr2 = np.array([1, 2, 3, 4, 5])
sub_arr2 = arr2[1:4].copy()
sub_arr2[0] = 888
print("Mang goc khong thay doi:", arr2)
```

Kết quả:

```
Mang con: [2 3 4]
Mang goc sau khi thay doi view: [ 1 999 3 4 5]
Mang goc khong thay doi: [1 2 3 4 5]
```

6.3.3 Chỉ mục nâng cao

a, Fancy Indexing - Chỉ mục bằng mảng

NumPy cho phép sử dụng mảng số nguyên làm chỉ số để truy cập nhiều phần tử không liên tiếp.

```
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90])

# Lấy các phần tử ở vị trí 1, 3, 5
indices = [1, 3, 5]
print("Cac phan tu tai [1,3,5]:", arr[indices])

# Lấy theo thứ tự bất kỳ
indices2 = [7, 2, 5, 1]
print("Lay theo thu tu:", arr[indices2])

# Áp dụng với mảng 2 chiều
```

```

matrix = np.array([[10, 20, 30],
                  [40, 50, 60],
                  [70, 80, 90]])

# Lấy phần tử ở (0,1), (1,2), (2,0)
rows = np.array([0, 1, 2])
cols = np.array([1, 2, 0])
print("Cac phan tu:", matrix[rows, cols]) # [20, 60, 70]

```

Kết quả:

Cac phan tu tai [1,3,5]: [20 40 60]
Lay theo thu tu: [80 30 60 20]
Cac phan tu: [20 60 70]

b, Boolean Indexing - Chỉ mục điều kiện

Đây là kỹ thuật mạnh mẽ cho phép lọc dữ liệu dựa trên điều kiện logic. Rất hữu ích trong xử lý và làm sạch dữ liệu.

```

arr = np.array([15, 22, 8, 31, 19, 7, 42, 11])

# Tạo mặt nạ boolean
mask = arr > 20
print("Mat na boolean:", mask)
print("Cac phan tu > 20:", arr[mask])

# Viết gọn trong một dòng
print("Cac phan tu < 15:", arr[arr < 15])

# Kết hợp nhiều điều kiện
print("Phan tu trong khoang [10, 25]:",
      arr[(arr >= 10) & (arr <= 25)])

# Lọc số chẵn
print("So chan:", arr[arr % 2 == 0])

# Lọc số lẻ
print("So le:", arr[arr % 2 != 0])

```

Kết quả:

Mat na boolean: [False True False True False False True False]
Cac phan tu > 20: [22 31 42]
Cac phan tu < 15: [8 7 11]
Phan tu trong khoang [10, 25]: [15 22 19 11]

```
So chan: [22 8 42]
```

```
So le: [15 31 19 7 11]
```

Ứng dụng trong kỹ thuật: Lọc dữ liệu đo từ cảm biến, loại bỏ các giá trị ngoài ngưỡng hoặc lỗi.

```
# Dữ liệu nhiệt độ từ cảm biến
temperatures = np.array([22.5, 23.1, 150.0, 22.8, -10.0,
                         23.4, 24.1, 22.9, 200.0])

# Lọc dữ liệu hợp lệ (0°C đến 50°C)
valid_temps = temperatures[(temperatures >= 0) &
                             (temperatures <= 50)]
print("Nhiet do hop le:", valid_temps)

# Thay thế dữ liệu lỗi bằng giá trị trung bình
mean_temp = np.mean(valid_temps)
temperatures[(temperatures < 0) | (temperatures > 50)] =
    mean_temp
print("Sau khi xu ly:", temperatures)
```

c, Chỉ mục với np.where

Hàm `np.where()` trả về chỉ số của các phần tử thỏa điều kiện, hoặc cho phép thay thế giá trị theo điều kiện.

```
arr = np.array([10, 25, 30, 15, 40, 8])

# Tìm chỉ số của các phần tử > 20
indices = np.where(arr > 20)
print("Chi so cac phan tu > 20:", indices[0])

# Thay thế theo điều kiện: nếu > 20 thì giữ nguyên,
# ngược lại thay bằng 0
result = np.where(arr > 20, arr, 0)
print("Ket qua:", result)

# Phân loại: > 25 -> 'cao', <= 25 -> 'thap'
labels = np.where(arr > 25, 'cao', 'thap')
print("Phan loai:", labels)
```

Kết quả:

```
Chi so cac phan tu > 20: [1 2 4]
```

```
Ket qua: [ 0 25 30  0 40  0]
```

```
Phan loai: ['thap' 'thap' 'cao' 'thap' 'cao' 'thap']
```

6.4 Các phép toán trên ma trận

Một trong những sức mạnh lớn nhất của NumPy là khả năng thực hiện các phép toán trên toàn bộ mảng mà không cần vòng lặp tường minh. Các phép toán này được thực thi rất nhanh nhờ tối ưu hóa ở tầng thấp.

6.4.1 Phép toán từng phần tử (Element-wise Operations)

Các phép toán cơ bản được áp dụng trên từng phần tử tương ứng của hai mảng có cùng kích thước.

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])

b = np.array([[10, 20, 30],
              [40, 50, 60]])

# Cộng
print("a + b:")
print(a + b)

# Trừ
print("\na - b:")
print(a - b)

# Nhân từng phần tử
print("\na * b (element-wise):")
print(a * b)

# Chia
print("\nb / a:")
print(b / a)

# Lũy thừa
print("\na ** 2:")
print(a ** 2)

# Chia lấy phần nguyên
print("\nb // a:")
print(b // a)

# Chia lấy phần dư
print("\nb % 3:")
print(b % 3)
```

Kết quả:

```
a + b:
[[11 22 33]
 [44 55 66]]
```

```
a - b:
[[-9 -18 -27]
 [-36 -45 -54]]
```

```
a * b (element-wise):
[[ 10 40 90]
 [160 250 360]]
```

```
b / a:
[[10. 10. 10.]
 [10. 10. 10.]]
```

```
a ** 2:
[[ 1 4 9]
 [16 25 36]]
```

6.4.2 Phép toán với scalar

NumPy tự động áp dụng phép toán với scalar lên tất cả phần tử của mảng.

```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])

print("arr * 10:")
print(arr * 10)

print("\narr + 5:")
print(arr + 5)

print("\n2 ** arr:")
print(2 ** arr)

print("\narr / 2:")
print(arr / 2)
```

Kết quả:

```
arr * 10:
[[10 20 30]
 [40 50 60]]
```

```
arr + 5:
[[ 6  7  8]
 [ 9 10 11]]
```

```
2 ** arr:
[[ 2  4  8]
 [16 32 64]]
```

6.4.3 Nhân ma trận (Matrix Multiplication)

Trong đại số tuyến tính, nhân ma trận là phép toán quan trọng, khác hoàn toàn với nhân từng phần tử.

```
# Ma trận A (2x3)
A = np.array([[1, 2, 3],
              [4, 5, 6]])

# Ma trận B (3x2)
B = np.array([[7, 8],
              [9, 10],
              [11, 12]])

# Nhân ma trận - cách 1: dùng np.dot()
C1 = np.dot(A, B)
print("np.dot(A, B):")
print(C1)

# Nhân ma trận - cách 2: dùng toán tử @
C2 = A @ B
print("\nA @ B:")
print(C2)

# Nhân ma trận - cách 3: dùng matmul
C3 = np.matmul(A, B)
print("\nnp.matmul(A, B):")
print(C3)

# Kiểm tra kích thước
print("\nKích thước A:", A.shape)
print("Kích thước B:", B.shape)
print("Kích thước C:", C1.shape)
```

Kết quả:

```
np.dot(A, B):
```

```
[ [ 58  64]
 [139 154] ]
```

A @ B:

```
[ [ 58  64]
 [139 154] ]
```

Kích thước A: (2, 3)

Kích thước B: (3, 2)

Kích thước C: (2, 2)

Lưu ý: Để nhân được hai ma trận, số cột của ma trận thứ nhất phải bằng số hàng của ma trận thứ hai.

6.4.4 Chuyển vị ma trận (Transpose)

Chuyển vị là phép hoán đổi hàng và cột của ma trận.

```
A = np.array([[1, 2, 3],
              [4, 5, 6]])

print("Ma tran goc:")
print(A)
print("Kich thuoc:", A.shape)

# Chuyển vị - cách 1
A_T1 = np.transpose(A)
print("\nChuyen vi (transpose):")
print(A_T1)

# Chuyển vị - cách 2
A_T2 = A.T
print("\nChuyen vi (thuoc tinh .T):")
print(A_T2)
print("Kich thuoc:", A_T2.shape)
```

Kết quả:

Ma tran goc:

```
[ [1 2 3]
 [4 5 6] ]
```

Kích thước: (2, 3)

Chuyen vi (transpose):

```
[ [1 4]
```

```
[2 5]
[3 6]
```

Kích thước: (3, 2)

6.4.5 Các phép toán ma trận nâng cao

NumPy cung cấp module `linalg` chứa nhiều hàm đại số tuyến tính nâng cao.

a, Ma trận nghịch đảo

```
# Ma trận vuông
A = np.array([[4, 7],
              [2, 6]])

print("Ma tran A:")
print(A)

# Tính ma trận nghịch đảo
A_inv = np.linalg.inv(A)
print("\nMa tran nghich dao A^(-1):")
print(A_inv)

# Kiểm chứng: A * A^(-1) = I (ma trận đơn vị)
I = A @ A_inv
print("\nA @ A^(-1):")
print(I)
```

Kết quả:

Ma tran A:

```
[[4 7]
 [2 6]]
```

Ma tran nghich dao A^(-1):

```
[[ 0.6 -0.7]
 [-0.2  0.4]]
```

A @ A^(-1):

```
[[1. 0.]
 [0. 1.]]
```

b, Định thức

```
A = np.array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 10]])

det_A = np.linalg.det(A)
print("Dinh thuc cua A:", det_A)

# Ma trận suy biến (định thức = 0)
B = np.array([[1, 2],
              [2, 4]])
det_B = np.linalg.det(B)
print("Dinh thuc cua B:", det_B)
```

Kết quả:

Dinh thuc cua A: -3.0

Dinh thuc cua B: 0.0

c, Hạng của ma trận

```
A = np.array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

rank = np.linalg.matrix_rank(A)
print("Hang cua ma tran:", rank)
```

d, Giá trị riêng và vector riêng

```
A = np.array([[4, -2],
             [1, 1]])

# Tính giá trị riêng và vector riêng
eigenvalues, eigenvectors = np.linalg.eig(A)

print("Gia tri rieng:")
print(eigenvalues)

print("\nVector rieng:")
print(eigenvectors)

# Kiểm chung: A * v = lambda * v
v1 = eigenvectors[:, 0]
```

```

lambda1 = eigenvalues[0]
print("\nKiểm chung:")
print("A @ v1:", A @ v1)
print("lambda1 * v1:", lambda1 * v1)

```

e, Vết của ma trận (Trace)

```

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Vết = tổng các phần tử trên đường chéo chính
trace = np.trace(A)
print("Vết cua ma tran:", trace) # 1 + 5 + 9 = 15

```

f, Giải hệ phương trình tuyến tính

NumPy có thể giải hệ phương trình tuyến tính dạng $Ax = b$.

```

# Hệ phương trình:
# 3x + 2y = 7
# x + 4y = 11

A = np.array([[3, 2],
              [1, 4]])

b = np.array([7, 11])

# Giải hệ
x = np.linalg.solve(A, b)
print("Nghiệm của hệ:", x)

# Kiểm chứng
print("Kiểm chung A @ x:", A @ x)
print("b:", b)

```

Kết quả:

```

Nghiệm của hệ: [1. 2.]
Kiểm chung A @ x: [ 7. 11.]
b: [ 7 11]

```

Ứng dụng trong kỹ thuật: Giải hệ phương trình mô tả mạch điện, cân bằng lực trong kết cấu, hoặc phân tích hệ thống điều khiển.

6.4.6 Broadcasting - Cơ chế mở rộng tự động

Broadcasting là một cơ chế mạnh mẽ cho phép NumPy thực hiện các phép toán trên các mảng có kích thước khác nhau.

```
# Ví dụ 1: Cộng vector vào ma trận
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

vector = np.array([10, 20, 30])

# Vector được broadcast thành ma trận 3x3
result = matrix + vector
print("Ma tran + vector:")
print(result)

# Ví dụ 2: Chuẩn hóa từng cột
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])

# Trung bình từng cột
col_means = np.mean(data, axis=0)
print("\nTrung binh cot:", col_means)

# Chuẩn hóa
normalized = data - col_means
print("Chuan hoa:")
print(normalized)
```

Kết quả:

Ma tran + vector:

```
[[11 22 33]
 [14 25 36]
 [17 28 39]]
```

Trung binh cot: [4. 5. 6.]

Chuan hoa:

```
[[ -3. -3. -3.]
 [ 0.  0.  0.]
 [ 3.  3.  3.]]
```

Quy tắc broadcasting:

1. Nếu hai mảng có số chiều khác nhau, mảng có ít chiều hơn sẽ được thêm chiều mới ở phía trước
2. Nếu kích thước ở một chiều không khớp, chiều có kích thước 1 sẽ được kéo dãn để khớp với chiều kia
3. Nếu kích thước không khớp và không có chiều nào bằng 1, sẽ báo lỗi

```
# Ví dụ broadcasting phức tạp hơn
A = np.ones((3, 4))          # shape (3, 4)
B = np.array([1, 2, 3, 4])    # shape (4,)
C = np.array([[10], [20], [30]]) # shape (3, 1)

print("A + B (broadcast hang):")
print(A + B)

print("\nA + C (broadcast cot):")
print(A + C)

print("\nB + C (broadcast 2 chieu):")
print(B + C)
```

6.5 Các phép xử lý và thống kê trên ma trận

NumPy cung cấp một bộ công cụ phong phú để xử lý và phân tích thống kê dữ liệu. Các hàm này có thể áp dụng trên toàn bộ mảng hoặc theo từng trục (axis) cụ thể.

6.5.1 Các hàm thống kê cơ bản

```
data = np.array([[10, 20, 30, 40],
                [50, 60, 70, 80],
                [90, 100, 110, 120]])

print("Ma tran du lieu:")
print(data)

# Tổng
print("\nTong toan bo:", np.sum(data))
print("Tong theo cot (axis=0):", np.sum(data, axis=0))
print("Tong theo dong (axis=1):", np.sum(data, axis=1))

# Trung bình
print("\nTrung binh toan bo:", np.mean(data))
print("Trung binh theo cot:", np.mean(data, axis=0))
print("Trung binh theo dong:", np.mean(data, axis=1))
```

```
# Min, Max
print("\nGia tri nho nhat:", np.min(data))
print("Gia tri lon nhat:", np.max(data))
print("Min theo cot:", np.min(data, axis=0))
print("Max theo dong:", np.max(data, axis=1))
```

Kết quả:

Mảng dữ liệu:

```
[[ 10   20   30   40]
 [ 50   60   70   80]
 [ 90  100  110  120]]
```

Tổng toàn bộ: 780

Tổng theo cột (axis=0): [150 180 210 240]

Tổng theo dòng (axis=1): [100 260 420]

Trung bình toàn bộ: 65.0

Trung bình theo cột: [50. 60. 70. 80.]

Trung bình theo dòng: [25. 65. 105.]

Gia trị nhỏ nhất: 10

Gia trị lớn nhất: 120

6.5.2 Chỉ số của giá trị min/max

```
arr = np.array([[15, 32, 8],
                [41, 25, 19],
                [7, 50, 12]])

# Chỉ số của giá trị nhỏ nhất (theo dạng phẳng)
print("Chi so min (flat):", np.argmin(arr))

# Chỉ số của giá trị lớn nhất
print("Chi so max (flat):", np.argmax(arr))

# Chỉ số min theo từng cột
print("Chi so min theo cot:", np.argmin(arr, axis=0))

# Chỉ số max theo từng dòng
print("Chi so max theo dong:", np.argmax(arr, axis=1))

# Tìm vị trí (hàng, cột) của giá trị max
```

```
max_pos = np.unravel_index(np.argmax(arr), arr.shape)
print("Vi tri max (hang, cot):", max_pos)
```

Kết quả:

```
Chi so min (flat): 6
Chi so max (flat): 7
Chi so min theo cot: [2 1 0]
Chi so max theo dong: [1 0 1]
Vi tri max (hang, cot): (2, 1)
```

6.5.3 Phương sai và độ lệch chuẩn

```
data = np.array([[20, 25, 30],
                [22, 28, 35],
                [18, 24, 29]])

# Phương sai
variance = np.var(data)
print("Phuong sai:", variance)

# Độ lệch chuẩn
std_dev = np.std(data)
print("Do lech chuan:", std_dev)

# Theo từng cột
print("\nDo lech chuan theo cot:", np.std(data, axis=0))

# Theo từng dòng
print("Do lech chuan theo dong:", np.std(data, axis=1))
```

6.5.4 Tích lũy và tích lũy theo chiều

```
arr = np.array([1, 2, 3, 4, 5])

# Tổng tích lũy
cumsum = np.cumsum(arr)
print("Tong tich luy:", cumsum)

# Tích tích lũy
cumprod = np.cumprod(arr)
print("Tich tich luy:", cumprod)

# Với ma trận
```

```

matrix = np.array([[1, 2, 3],
                  [4, 5, 6]])

print("\nTong tich luy theo cot:")
print(np.cumsum(matrix, axis=0))

print("\nTong tich luy theo dong:")
print(np.cumsum(matrix, axis=1))

```

Kết quả:

Tong tich luy: [1 3 6 10 15]
 Tich tich luy: [1 2 6 24 120]

Tong tich luy theo cot:

```

[[1 2 3]
 [5 7 9]]

```

Tong tich luy theo dong:

```

[[ 1 3 6]
 [ 4 9 15]]

```

6.5.5 Sắp xếp mảng

```

arr = np.array([3, 1, 4, 1, 5, 9, 2, 6])

# Sắp xếp (trả về bản sao đã sắp xếp)
sorted_arr = np.sort(arr)
print("Sap xep:", sorted_arr)

# Chỉ số sắp xếp
indices = np.argsort(arr)
print("Chi so sap xep:", indices)

# Sắp xếp ma trận
matrix = np.array([[3, 1, 4],
                  [1, 5, 9],
                  [2, 6, 5]])

print("\nSap xep theo hang:")
print(np.sort(matrix, axis=1))

print("\nSap xep theo cot:")
print(np.sort(matrix, axis=0))

```

Kết quả:

Sap xep: [1 1 2 3 4 5 6 9]
Chi so sap xep: [1 3 6 0 2 4 7 5]

Sap xep theo hang:

```
[[1 3 4]
 [1 5 9]
 [2 5 6]]
```

6.5.6 Loại bỏ trùng lặp và tìm giá trị duy nhất

```
arr = np.array([1, 2, 2, 3, 3, 3, 4, 5, 5])

# Giá trị duy nhất
unique = np.unique(arr)
print("Gia tri duy nhat:", unique)

# Đếm số lần xuất hiện
unique, counts = np.unique(arr, return_counts=True)
print("\nGia tri:", unique)
print("So lan xuat hien:", counts)

# Tìm giá trị duy nhất trong ma trận
matrix = np.array([[1, 2, 3],
                   [2, 3, 4],
                   [3, 4, 5]])
print("\nGia tri duy nhat trong ma tran:",
      np.unique(matrix))
```

Kết quả:

Gia tri duy nhat: [1 2 3 4 5]

Gia tri: [1 2 3 4 5]
So lan xuat hien: [1 2 3 1 2]

Gia tri duy nhat trong ma tran: [1 2 3 4 5]

6.5.7 Các hàm toán học phổ biến

NumPy cung cấp các hàm toán học áp dụng trên từng phần tử của mảng.

```
arr = np.array([1, 4, 9, 16, 25])
```

```

# Căn bậc hai
print("Căn bậc hai:", np.sqrt(arr))

# Lũy thừa
print("Lũy thừa 2:", np.power(arr, 2))

# Hàm mũ
print("e^x:", np.exp([1, 2, 3]))

# Logarit
print("log(arr):", np.log(arr))
print("log10(arr):", np.log10(arr))

# Làm tròn
values = np.array([1.234, 2.567, 3.891])
print("\nLam tron 2 chu so:", np.round(values, 2))
print("Lam tron xuong:", np.floor(values))
print("Lam tron len:", np.ceil(values))

# Giá trị tuyệt đối
print("\nGia tri tuyet doi:", np.abs([-1, -2, 3, -4]))

```

Kết quả:

```

Căn bậc hai: [1. 2. 3. 4. 5.]
Lũy thừa 2: [ 1 16 81 256 625]
e^x: [ 2.71828183 7.3890561 20.08553692]
log(arr): [0.           1.38629436 2.19722458 2.77258872 3.21887582]

Lam tron 2 chu so: [1.23 2.57 3.89]
Lam tron xuong: [1. 2. 3.]
Lam tron len: [2. 3. 4.]

Gia tri tuyet doi: [1 2 3 4]

```

6.5.8 Hàm lượng giác

```

angles_deg = np.array([0, 30, 45, 60, 90])

# Chuyển độ sang radian
angles_rad = np.radians(angles_deg)

print("Sin:", np.sin(angles_rad))
print("Cos:", np.cos(angles_rad))

```

```

print("Tan:", np.tan(angles_rad))

# Hàm ngược
print("\narcsin(0.5):", np.arcsin(0.5))
print("arccos(0.5):", np.arccos(0.5))
print("arctan(1):", np.arctan(1))

# Chuyển radian sang độ
print("\nChuyen sang do:", np.degrees(np.pi))

```

6.5.9 Chuẩn hóa dữ liệu

Chuẩn hóa là kỹ thuật quan trọng trong xử lý dữ liệu và machine learning.

```

# Dữ liệu mẫu
data = np.array([[100, 200, 300],
                 [150, 250, 350],
                 [200, 300, 400]])

print("Du lieu goc:")
print(data)

# Chuẩn hóa Min-Max về [0, 1]
data_min = np.min(data)
data_max = np.max(data)
normalized = (data - data_min) / (data_max - data_min)
print("\nChuan hoa Min-Max:")
print(normalized)

# Chuẩn hóa Z-score (mean=0, std=1)
mean = np.mean(data)
std = np.std(data)
z_normalized = (data - mean) / std
print("\nChuan hoa Z-score:")
print(z_normalized)

# Chuẩn hóa theo từng cột
col_means = np.mean(data, axis=0)
col_stds = np.std(data, axis=0)
col_normalized = (data - col_means) / col_stds
print("\nChuan hoa theo cot:")
print(col_normalized)

```

Ứng dụng trong kỹ thuật: Chuẩn hóa dữ liệu từ nhiều cảm biến có đơn vị và tỷ lệ khác nhau để so sánh và phân tích thống nhất.

6.5.10 Reshape và thay đổi cấu trúc mảng

```

# Mảng ban đầu
arr = np.arange(12)
print("Mang ban dau:", arr)

# Reshape thành ma trận 3x4
reshaped = arr.reshape(3, 4)
print("\nReshape 3x4:")
print(reshaped)

# Reshape thành 2x2x3
reshaped_3d = arr.reshape(2, 2, 3)
print("\nReshape 2x2x3:")
print(reshaped_3d)

# Làm phẳng (flatten)
flattened = reshaped.flatten()
print("\nLam phang:", flattened)

# Ravel (tương tự flatten nhưng trả về view)
raveled = reshaped.ravel()
print("Ravel:", raveled)

# Reshape tự động (-1)
auto_reshape = arr.reshape(3, -1)    # Tự tính chiều thứ 2
print("\nReshape tu dong 3x?:")
print(auto_reshape)

```

Kết quả:

Mang ban dau: [0 1 2 3 4 5 6 7 8 9 10 11]

Reshape 3x4:

```

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

```

Lam phang: [0 1 2 3 4 5 6 7 8 9 10 11]

Reshape tu dong 3x?:

```

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

```

6.5.11 Nối và chia mảng

```

a = np.array([[1, 2],
              [3, 4]])

b = np.array([[5, 6],
              [7, 8]])

# Nối theo chiều dọc (vertical stack)
v_stack = np.vstack((a, b))
print("Vertical stack:")
print(v_stack)

# Nối theo chiều ngang (horizontal stack)
h_stack = np.hstack((a, b))
print("\nHorizontal stack:")
print(h_stack)

# Concatenate với axis
concat_0 = np.concatenate((a, b), axis=0)
print("\nConcatenate axis=0:")
print(concat_0)

concat_1 = np.concatenate((a, b), axis=1)
print("\nConcatenate axis=1:")
print(concat_1)

# Chia mảng
arr = np.arange(12).reshape(3, 4)
print("\nMảng ban đầu:")
print(arr)

# Chia thành 3 phần theo chiều ngang
splits = np.split(arr, 3, axis=0)
print("\nChia thành 3 phần theo hàng:")
for i, part in enumerate(splits):
    print(f"Phần {i+1}:")
    print(part)

```

Kết quả:

Vertical stack:

```

[[1 2]
 [3 4]
 [5 6]

```

```
[7 8]]
```

Horizontal stack:

```
[[1 2 5 6]
 [3 4 7 8]]
```

6.6 Ứng dụng NumPy trong kỹ thuật

Sau khi nắm vững các khái niệm cơ bản và nâng cao của NumPy, phần này sẽ trình bày các ứng dụng thực tế trong lĩnh vực kỹ thuật, từ xử lý tín hiệu, phân tích dữ liệu cảm biến đến tính toán kỹ thuật.

6.6.1 Xử lý tín hiệu

a, Tạo tín hiệu sin

```
# Tham số tín hiệu
frequency = 5    # Hz
sampling_rate = 100  # Hz
duration = 2    # giây

# Tạo trục thời gian
t = np.linspace(0, duration, sampling_rate * duration)

# Tạo tín hiệu sin
amplitude = 1.0
signal = amplitude * np.sin(2 * np.pi * frequency * t)

print("Thoi gian (10 diem dau):", t[:10])
print("Tin hieu (10 diem dau):", signal[:10])

# Thêm nhiễu
noise = 0.1 * np.random.randn(len(signal))
noisy_signal = signal + noise

print("\nTin hieu co nhieu (10 diem dau):",
→ noisy_signal[:10])
```

b, Tích chập tín hiệu (Convolution)

Tích chập được sử dụng trong lọc tín hiệu, xử lý ảnh và nhiều ứng dụng khác.

```
# Tín hiệu đầu vào
signal = np.array([1, 2, 3, 4, 5, 4, 3, 2, 1])

# Kernel lọc trung bình trượt (moving average)
kernel = np.ones(3) / 3
```

```

# Tích chập
filtered = np.convolve(signal, kernel, mode='same')
print("Tin hieu goc:", signal)
print("Tin hieu sau loc:", filtered)

# Kernel phát hiện cạnh
edge_kernel = np.array([1, 0, -1])
edges = np.convolve(signal, edge_kernel, mode='same')
print("Phat hien canh:", edges)

```

Kết quả:

Tin hieu goc: [1 2 3 4 5 4 3 2 1]
Tin hieu sau loc: [1. 2. 3. 4. 4.33 3.67 3. 2. 1.
Phat hien canh: [1. 2. 2. 1. 0. -1. -2. -2. -1.]

c, Biến đổi FFT (Fast Fourier Transform)

FFT là công cụ mạnh mẽ để phân tích tín hiệu trong miền tần số.

```

# Tạo tín hiệu tổng hợp (2 tần số)
sampling_rate = 1000 # Hz
t = np.linspace(0, 1, sampling_rate)

# Tín hiệu = 50Hz + 120Hz
signal = np.sin(2 * np.pi * 50 * t) + 0.5 * np.sin(2 * np.pi
→ * 120 * t)

# Biến đổi FFT
fft_result = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(len(signal), 1/sampling_rate)

# Lấy biên độ (magnitude)
magnitude = np.abs(fft_result)

# Chỉ lấy nửa đầu (tần số dương)
half = len(signal) // 2
print("Tan so (10 diem dau):", fft_freq[:10])
print("Bien do (10 diem dau):", magnitude[:10])

# Tìm tần số có biên độ lớn nhất
dominant_freq_idx = np.argmax(magnitude[:half])
dominant_freq = fft_freq[dominant_freq_idx]
print(f"\nTan so chinh: {dominant_freq} Hz")

```

6.6.2 Xử lý dữ liệu cảm biến

a, Làm sạch dữ liệu nhiệt độ

```
# Dữ liệu nhiệt độ từ cảm biến (có lỗi)
temperatures = np.array([
    22.5, 23.1, 22.8, 150.0, 23.4, 22.9, -10.0,
    23.2, 24.1, 22.7, 200.0, 23.5, 22.8
])

print("Dữ liệu gốc:", temperatures)

# Phát hiện outliers (ngoài khoảng hợp lệ)
valid_range = (0, 50)
outliers = (temperatures < valid_range[0]) | (temperatures >
    valid_range[1])
print("Outliers:", temperatures[outliers])

# Thay thế outliers bằng median của dữ liệu hợp lệ
valid_data = temperatures[~outliers]
median_temp = np.median(valid_data)
temperatures_clean = temperatures.copy()
temperatures_clean[outliers] = median_temp

print("Dữ liệu sau khi làm sạch:", temperatures_clean)
print("Trung bình:", np.mean(temperatures_clean))
print("Độ lệch chuẩn:", np.std(temperatures_clean))
```

Kết quả:

Dữ liệu gốc: [22.5 23.1 22.8 150. ... 200. 23.5 22.8]
 Outliers: [150. -10. 200.]
 Dữ liệu sau khi làm sạch: [22.5 23.1 22.8 23. 23.4 22.9 23. 23.2 24.1 23.5]

b, Tính trung bình trượt (Moving Average)

```
# Dữ liệu đo dao động
measurements = np.array([
    100, 102, 98, 105, 99, 101, 103, 97, 104, 100,
    102, 98, 106, 99, 101, 103, 98, 104, 101, 102
])

# Trung bình trượt của số 5
window = 5
smoothed = np.convolve(measurements,
    np.ones(window)/window,
```

```

        mode='valid')

print("Do dai du lieu goc:", len(measurements))
print("Do dai sau khi lam muot:", len(smoothed))
print("Du lieu goc (10 diem):", measurements[:10])
print("Du lieu lam muot (10 diem):", smoothed[:10])

```

6.6.3 Tính toán ma trận cho kết cấu

a, Phân tích khung phẳng

```

# Ma trận độ cứng đơn giản cho hệ lò xo
# k1 = 100 N/m, k2 = 200 N/m, k3 = 150 N/m
# Hệ 3 lò xo nối tiếp-song song

K = np.array([[300, -200, 0],
              [-200, 350, -150],
              [0, -150, 150]])

print("Ma tran do cung K:")
print(K)

# Vector lực
F = np.array([100, 0, 0]) # N

# Giải hệ: K * u = F để tìm chuyển vị u
u = np.linalg.solve(K, F)
print("\nChuyen vi (mm):", u)

# Tính lực phản lực
reactions = K @ u
print("Luc phan luc:", reactions)

```

b, Tính ứng suất trong vật liệu

```

# Ma trận biến dạng (strain tensor)
strain = np.array([[0.001, 0.0002, 0],
                   [0.0002, 0.0015, 0],
                   [0, 0, 0.0008]])

# Hệ số đàn hồi (đơn giản hóa - vật liệu đẳng hướng)
E = 200e9 # Pa (thép)
nu = 0.3 # Hệ số Poisson

```

```
# Ma trận độ cứng (simplified)
D = E / (1 - nu**2) * np.array([
    [1, nu, 0],
    [nu, 1, 0],
    [0, 0, (1-nu)/2]
])

# Tính ứng suất (chỉ tính trên mặt phẳng)
stress_2d = D @ strain[:2, :2].flatten()[:3]
print("Ung suat (Pa):", stress_2d)
print("Ung suat (MPa):", stress_2d / 1e6)
```

6.6.4 Mô phỏng và tính toán số

a, Mô phỏng Monte Carlo

```
# Uớc lượng giá trị pi bằng phương pháp Monte Carlo
n_points = 100000

# Tạo điểm ngẫu nhiên trong hình vuông [-1, 1] x [-1, 1]
x = np.random.uniform(-1, 1, n_points)
y = np.random.uniform(-1, 1, n_points)

# Tính khoảng cách đến tâm
distances = np.sqrt(x**2 + y**2)

# Đếm điểm trong hình tròn bán kính 1
inside_circle = np.sum(distances <= 1)

# Uớc lượng pi
pi_estimate = 4 * inside_circle / n_points
print(f"Uoc luong pi: {pi_estimate}")
print(f"Sai so: {abs(pi_estimate - np.pi)}")
```

b, Tính tích phân bằng phương pháp hình thang

```
# Tính tích phân của f(x) = x^2 từ 0 đến 1
# Kết quả chính xác: 1/3 ~ 0.333...
def f(x):
    return x**2

# Tạo lưới điểm
n = 1000
```

```

x = np.linspace(0, 1, n)
y = f(x)

# Phương pháp hình thang
integral = np.trapz(y, x)
print(f"Tích phân: {integral}")
print(f"Sai số: {abs(integral - 1/3)}")

# So sánh với số điểm khác nhau
for n in [10, 100, 1000, 10000]:
    x = np.linspace(0, 1, n)
    y = f(x)
    result = np.trapz(y, x)
    error = abs(result - 1/3)
    print(f"n={n:5d}: {result:.10f}, sai số: {error:.2e}")

```

6.6.5 Xử lý ma trận hình ảnh

Mặc dù xử lý ảnh thường dùng OpenCV hoặc PIL, NumPy vẫn là nền tảng cho các thao tác cơ bản.

```

# Tạo ảnh grayscale mô phỏng (8x8)
image = np.random.randint(0, 256, (8, 8), dtype=np.uint8)
print("Anh goc:")
print(image)

# Lật ảnh theo chiều dọc
flipped_v = np.flipud(image)
print("\nLat doc:")
print(flipped_v)

# Lật ảnh theo chiều ngang
flipped_h = np.fliplr(image)
print("\nLat ngang:")
print(flipped_h)

# Xoay ảnh 90 độ
rotated = np.rot90(image)
print("\nXoay 90 do:")
print(rotated)

# Nguộing hóa (thresholding)
threshold = 128
binary = (image > threshold).astype(np.uint8) * 255

```

```
print("\nAnh nhì phân:")
print(binary)
```

6.6.6 Phân tích thống kê dữ liệu thực nghiệm

```
# Dữ liệu đo cường độ bê tông (MPa)
concrete_strength = np.array([
    28.5, 29.1, 27.8, 28.9, 29.5, 28.2, 29.8,
    28.0, 29.3, 28.7, 29.0, 28.4, 29.6, 28.8
])

print("Số mẫu:", len(concrete_strength))
print("Trung bình:", np.mean(concrete_strength))
print("Trung vị:", np.median(concrete_strength))
print("Độ lệch chuẩn:", np.std(concrete_strength))
print("Phuông sai:", np.var(concrete_strength))
print("Min:", np.min(concrete_strength))
print("Max:", np.max(concrete_strength))

# Tính khoảng tin cậy 95% (giả định phân phối chuẩn)
from scipy import stats
confidence = 0.95
n = len(concrete_strength)
mean = np.mean(concrete_strength)
std_err = stats.sem(concrete_strength)
ci = stats.t.interval(confidence, n-1, loc=mean,
→ scale=std_err)

print(f"\nKhoảng tin cay 95%: [{ci[0]:.2f}, {ci[1]:.2f}]")

# Phân vị (quartiles)
q1, q2, q3 = np.percentile(concrete_strength, [25, 50, 75])
print(f"\nQ1: {q1}, Q2: {q2}, Q3: {q3}")
print(f"IQR (Interquartile Range): {q3 - q1}")
```

6.6.7 Hồi quy tuyến tính

```
# Dữ liệu: Quan hệ giữa nhiệt độ và điện trở
temperature = np.array([20, 30, 40, 50, 60, 70, 80]) # °C
resistance = np.array([100.2, 103.5, 107.1, 110.8,
    114.2, 117.9, 121.5]) # Ohm

# Tạo ma trận cho hồi quy: [1, T]
```

```

X = np.column_stack([np.ones(len(temperature)),
                     temperature])
y = resistance

# Giải phương trình chuẩn: (X^T X) beta = X^T y
beta = np.linalg.inv(X.T @ X) @ X.T @ y

print(f"He so hoi quy: R = {beta[0]:.2f} + {beta[1]:.4f} *"
      " T")

# Dự đoán
T_new = 55
R_pred = beta[0] + beta[1] * T_new
print(f"\nDu doan dien tro tai {T_new}°C: {R_pred:.2f} Ohm")

# Tính R²
y_pred = X @ beta
ss_res = np.sum((y - y_pred)**2)
ss_tot = np.sum((y - np.mean(y))**2)
r_squared = 1 - (ss_res / ss_tot)
print(f"R²: {r_squared:.4f}")

```

Kết quả mẫu:

He so hoi quy: R = 96.54 + 0.3127 * T

Du doan dien tro tai 55°C: 113.74 Ohm

R²: 0.9998

6.7 Bài tập thực hành

Phần bài tập này tập trung vào các kỹ năng NumPy cơ bản và một bài nâng cao, giúp củng cố kiến thức đã học.

Bài 1: Tạo và thao tác mảng cơ bản

Viết chương trình thực hiện các yêu cầu sau:

1. Tạo mảng 1 chiều chứa các số từ 10 đến 50 (bước nhảy 5)
2. Tạo ma trận 4x5 với các giá trị ngẫu nhiên từ 0 đến 100
3. Tạo ma trận đơn vị 5x5
4. Tạo ma trận 3x3 với giá trị đường chéo là [1, 2, 3]

Gợi ý: Sử dụng `np.arange()`, `np.random.randint()`, `np.eye()`, `np.diag()`

Bài 2: Indexing và Slicing

Cho ma trận:

```
A = np.array([[10, 20, 30, 40],
             [50, 60, 70, 80],
             [90, 100, 110, 120],
             [130, 140, 150, 160]])
```

Thực hiện các yêu cầu:

1. Lấy phần tử ở hàng 2, cột 3
2. Lấy toàn bộ hàng thứ 2
3. Lấy toàn bộ cột thứ 3
4. Lấy ma trận con 2×2 từ góc dưới phải
5. Lấy các phần tử có giá trị lớn hơn 80
6. Thay thế tất cả giá trị chẵn bằng 0

Gợi ý: Sử dụng indexing, slicing, boolean indexing

Bài 3: Các phép toán ma trận cơ bản

Cho hai ma trận:

```
A = np.array([[1, 2], [3, 4], [5, 6]]) # 3x2
B = np.array([[7, 8, 9], [10, 11, 12]]) # 2x3
```

Thực hiện:

1. Nhân ma trận A và B (matrix multiplication)
2. Tìm ma trận chuyển vị của A
3. Tính tổng tất cả phần tử của ma trận A
4. Tính trung bình từng cột của ma trận A
5. Tìm giá trị lớn nhất và nhỏ nhất trong mỗi hàng của B

Gợi ý: Sử dụng @ hoặc np.dot(), .T, np.sum(), np.mean(), np.max(), np.min()

Bài 4: Thống kê và xử lý dữ liệu

Cho dữ liệu nhiệt độ đo được trong 14 ngày ($^{\circ}\text{C}$):

```
temps = np.array([22.5, 23.1, 22.8, 23.5, 150.0, 22.9, 23.7,
                  23.2, 24.1, 22.7, -20.0, 23.5, 23.0,
                  ↳ 22.8])
```

Yêu cầu:

1. Tìm nhiệt độ trung bình, độ lệch chuẩn, min, max
2. Phát hiện và in ra các giá trị ngoại lệ (outliers) nằm ngoài khoảng [0°C, 50°C]
3. Thay thế các outliers bằng giá trị trung vị (median) của dữ liệu hợp lệ
4. Tính lại nhiệt độ trung bình sau khi làm sạch dữ liệu

Gợi ý: Sử dụng `np.mean()`, `np.std()`, boolean indexing, `np.median()`

Bài 5: Reshape và Broadcasting

1. Tạo mảng 1D chứa các số từ 1 đến 24
2. Reshape thành ma trận 4x6
3. Reshape thành ma trận 3D có kích thước 2x3x4
4. Cộng một vector [10, 20, 30, 40, 50, 60] vào từng hàng của ma trận 4x6 (sử dụng broadcasting)
5. Tính tổng theo từng chiều của ma trận 3D

Gợi ý: Sử dụng `np.arange()`, `reshape()`, broadcasting, `np.sum(axis=...)`

Bài 6: Sắp xếp và tìm kiếm

Cho mảng điểm của sinh viên:

```
scores = np.array([7.5, 8.2, 6.8, 9.1, 5.5, 8.7, 7.9, 6.2,  
                  8.5, 7.3])
```

Yêu cầu:

1. Sắp xếp điểm theo thứ tự tăng dần
2. Tìm chỉ số của sinh viên có điểm cao nhất
3. Đếm số sinh viên có điểm ≥ 8.0
4. Tính điểm trung bình của 3 sinh viên có điểm cao nhất
5. Phân loại: ≥ 8.5 : "Giỏi", ≥ 7.0 : "Khá", ≥ 5.5 : "Trung bình", còn lại: "Yếu"

Gợi ý: Sử dụng `np.sort()`, `np.argmax()`, boolean indexing, `np.where()`

Bài 7: Ma trận và hệ phương trình (Nâng cao)

Trong một mạch điện có 3 node, áp dụng định luật Kirchhoff ta có hệ phương trình:

$$\begin{aligned}3I_1 - I_2 &= 12 \\-I_1 + 4I_2 - I_3 &= 0 \\-I_2 + 3I_3 &= 6\end{aligned}$$

Trong đó I_1, I_2, I_3 là dòng điện (Ampere) tại các node.

Yêu cầu:

1. Thiết lập ma trận hệ số A và vector hằng số b
2. Giải hệ phương trình để tìm I_1, I_2, I_3 bằng np.linalg.solve()
3. Kiểm chứng nghiệm bằng cách tính $A \times I$ và so sánh với b
4. Tính định thức của ma trận A
5. Tìm ma trận nghịch đảo của A và giải hệ bằng cách nhân $A^{-1} \times b$
6. So sánh kết quả từ hai phương pháp

Mở rộng:

- Tính công suất tiêu thụ tại mỗi node nếu điện trở tương ứng là $R_1 = 2\Omega, R_2 = 3\Omega, R_3 = 4\Omega$ (sử dụng công thức $P = I^2 \times R$)
- Tính tổng công suất tiêu thụ của toàn mạch

Gợi ý:

```
import numpy as np

# Ma trận hệ số
A = np.array([[3, -1, 0],
              [-1, 4, -1],
              [0, -1, 3]])

# Vector hằng số
b = np.array([12, 0, 6])

# Giải hệ
I = np.linalg.solve(A, b)
print("Dòng điện:", I)

# Kiểm chứng
check = A @ I
print("Kiem chung:", check)
print("Vector b:", b)
print("Sai so:", np.allclose(check, b))
```

Đáp án tham khảo cho Bài 7:

- $I_1 \approx 5.09 \text{ A}, I_2 \approx 3.27 \text{ A}, I_3 \approx 3.09 \text{ A}$
- Định thức $\det(A) = 33$
- $P_1 \approx 51.8 \text{ W}, P_2 \approx 32.0 \text{ W}, P_3 \approx 38.2 \text{ W}$

- Tổng công suất: ≈ 122.0 W

Lưu ý: Bài 7 là bài nâng cao kết hợp kiến thức NumPy với ứng dụng thực tế trong kỹ thuật điện. Sinh viên cần nắm vững đại số tuyến tính và cách sử dụng module `np.linalg`.

CHƯƠNG 7. Trực quan hóa dữ liệu - Visualization

7.1 Giới thiệu về Visualization trong Data Science

Trực quan hóa dữ liệu (Data Visualization) là quá trình chuyển đổi dữ liệu số thành các biểu đồ, đồ thị trực quan, giúp con người dễ dàng khám phá mẫu hình (patterns), phát hiện bất thường (anomalies), kiểm tra giả thuyết và truyền đạt thông tin hiệu quả. Trong bối cảnh Machine Learning và Deep Learning, visualization đóng vai trò then chốt ở mọi giai đoạn của quy trình phân tích dữ liệu.

7.1.1 Vai trò của Visualization trong Machine Learning

Trong quy trình Machine Learning, visualization xuất hiện ở nhiều giai đoạn:

1. Exploratory Data Analysis (EDA) - Phân tích khám phá dữ liệu:

- Hiểu cấu trúc và phân phối dữ liệu
- Phát hiện giá trị ngoại lệ (outliers) và dữ liệu thiếu
- Xác định mối quan hệ giữa các biến
- Đánh giá sự cân bằng của tập dữ liệu (class imbalance)

2. Feature Engineering - Thiết kế đặc trưng:

- Trực quan hóa phân phối của features trước và sau transform
- So sánh tầm quan trọng của features
- Kiểm tra tương quan để loại bỏ multicollinearity

3. Model Training - Huấn luyện mô hình:

- Vẽ learning curves để phát hiện overfitting/underfitting
- Theo dõi loss và metrics theo epochs
- Visualization decision boundaries

4. Model Evaluation - Đánh giá mô hình:

- Confusion matrix, ROC curve, Precision-Recall curve
- Phân tích lỗi dự đoán
- So sánh performance của nhiều models

7.1.2 Thư viện Visualization trong Python

Python cung cấp hệ sinh thái thư viện visualization phong phú:

- **Matplotlib**: Thư viện nền tảng, linh hoạt, kiểm soát chi tiết cao
- **Seaborn**: Xây dựng trên Matplotlib, tập trung vào statistical visualization
- **Plotly**: Biểu đồ tương tác (interactive), phù hợp với web dashboards

- **Pandas plotting:** Tích hợp trực tiếp với DataFrame

Chương này tập trung vào Matplotlib và Seaborn - hai thư viện cốt lõi trong Data Science và Machine Learning.

7.2 Cài đặt và cấu hình

7.2.1 Cài đặt thư viện

```
pip install matplotlib seaborn pandas scikit-learn
```

7.2.2 Import và cấu hình

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# Cấu hình style
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")

# Cấu hình matplotlib
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['font.size'] = 12

# Hiển thị inline trong Jupyter
%matplotlib inline
```

7.2.3 Lưu hình ảnh

```
# Lưu hình với resolution cao
plt.savefig('Hinh_ve/figure_name.png', dpi=300,
            bbox_inches='tight')

# Lưu nhiều formats
plt.savefig('Hinh_ve/figure_name.pdf', format='pdf',
            bbox_inches='tight')
```

7.3 Matplotlib: Nền tảng Visualization

Matplotlib là thư viện cốt lõi cho visualization trong Python, cung cấp kiểm soát chi tiết cao và tính linh hoạt tuyệt vời.

7.3.1 Cấu trúc cơ bản: Figure và Axes

Matplotlib sử dụng kiến trúc phân tầng với hai khái niệm chính:

- **Figure:** Toàn bộ cửa sổ hoặc trang vẽ

- **Axes:** Vùng vẽ cụ thể (subplot) trong Figure

```
# Tạo figure và axes
fig, ax = plt.subplots(figsize=(8, 6))

# Vẽ dữ liệu
x = np.linspace(0, 10, 100)
ax.plot(x, np.sin(x), label='sin(x)')
ax.plot(x, np.cos(x), label='cos(x)')

# Tùy chỉnh
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Hàm lượng giác')
ax.legend()
ax.grid(True, alpha=0.3)

plt.savefig('Hinh_ve/ch8_basic_plot.png', dpi=300,
           bbox_inches='tight')
plt.show()
```



Hình 7.1: Đồ thị hàm lượng giác cơ bản

7.3.2 Line Plot - Đồ thị đường

Line plot phù hợp cho dữ liệu chuỗi thời gian, theo dõi xu hướng.

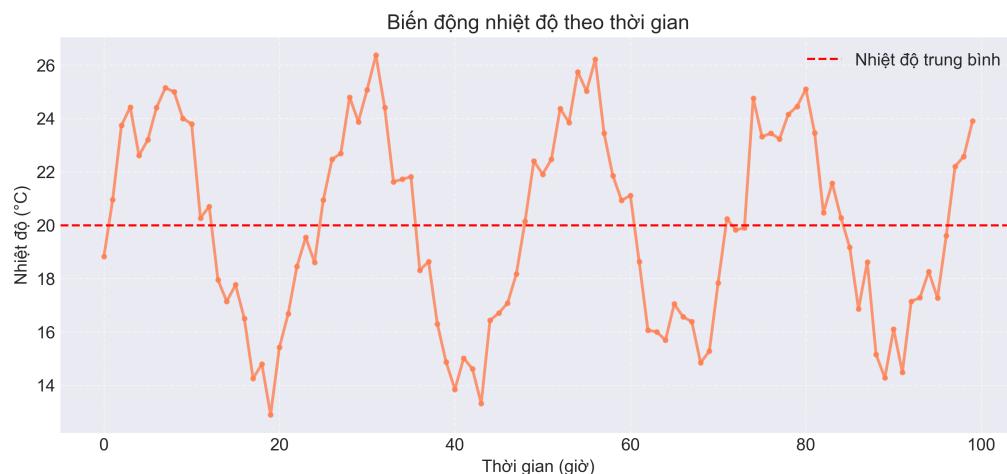
```

# Tạo dữ liệu thời gian
time = np.arange(0, 100)
temperature = 20 + 5 * np.sin(2 * np.pi * time / 24) + \
              np.random.normal(0, 1, 100)

fig, ax = plt.subplots(figsize=(12, 5))
ax.plot(time, temperature, linewidth=2, color='coral',
         marker='o', markersize=3, alpha=0.8)
ax.set_xlabel('Thời gian (giờ)')
ax.set_ylabel('Nhiệt độ (°C)')
ax.set_title('Biến động nhiệt độ theo thời gian')
ax.grid(True, linestyle='--', alpha=0.5)
ax.axhline(y=20, color='red', linestyle='--',
            label='Nhiệt độ trung bình')
ax.legend()

plt.savefig('Hinh_ve/ch8_line_plot.png', dpi=300,
            bbox_inches='tight')
plt.show()

```



Hình 7.2: Line plot theo dõi nhiệt độ theo thời gian

7.3.3 Scatter Plot - Biểu đồ phân tán

Scatter plot hiển thị mối quan hệ giữa hai biến liên tục, rất hữu ích trong EDA và phân tích correlation.

```

# Tạo dữ liệu với correlation
np.random.seed(42)
n = 200
x = np.random.randn(n)
y = 2 * x + np.random.randn(n) * 0.5

```

```

colors = x + y  # Màu theo tổng

fig, ax = plt.subplots(figsize=(8, 8))
scatter = ax.scatter(x, y, c=colors, cmap='viridis',
                     s=50, alpha=0.6, edgecolors='black')
ax.set_xlabel('Feature X', fontsize=12)
ax.set_ylabel('Feature Y', fontsize=12)
ax.set_title('Scatter Plot: Correlation giữa X và Y',
             fontsize=14)
plt.colorbar(scatter, ax=ax, label='X + Y')

# Vẽ đường hồi quy
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
ax.plot(x, p(x), "r--", linewidth=2,
        label=f'y={z[0]:.2f}x+{z[1]:.2f}')
ax.legend()

plt.savefig('Hinh_ve/ch8_scatter_plot.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

7.3.4 Histogram và Distribution Plot

Histogram hiển thị phân phối của biến liên tục, quan trọng để kiểm tra normality và phát hiện skewness.

```

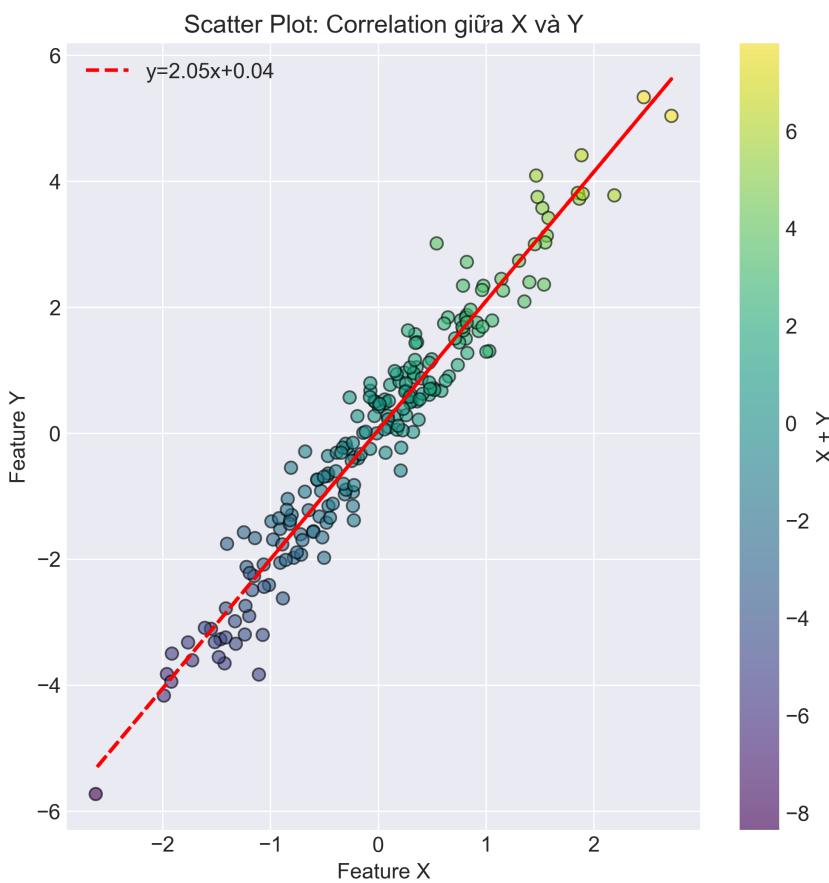
# Tạo dữ liệu từ các phân phối khác nhau
normal_data = np.random.normal(100, 15, 1000)
uniform_data = np.random.uniform(50, 150, 1000)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Histogram 1: Normal distribution
axes[0].hist(normal_data, bins=30, color='skyblue',
              edgecolor='black', alpha=0.7)
axes[0].axvline(normal_data.mean(), color='red',
                 linestyle='--', linewidth=2, label='Mean')
axes[0].set_xlabel('Giá trị')
axes[0].set_ylabel('Tần suất')
axes[0].set_title('Phân phối chuẩn (Normal)')
axes[0].legend()

# Histogram 2: Uniform distribution

```

**Hình 7.3:** Scatter plot với regression line

```

axes[1].hist(uniform_data, bins=30, color='lightcoral',
             edgecolor='black', alpha=0.7)
axes[1].axvline(uniform_data.mean(), color='red',
                 linestyle='--', linewidth=2, label='Mean')
axes[1].set_xlabel('Giá trị')
axes[1].set_ylabel('Tần suất')
axes[1].set_title('Phân phối đều (Uniform)')
axes[1].legend()

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_histogram.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

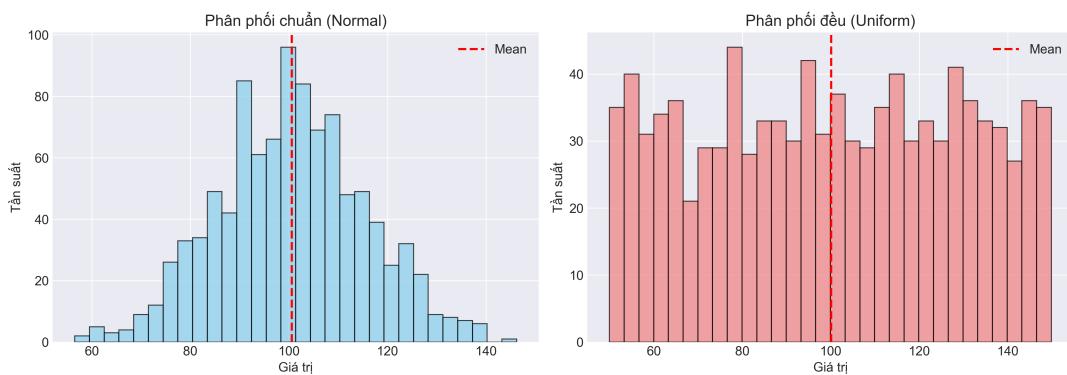
7.3.5 Bar Chart - Biểu đồ cột

Bar chart phù hợp cho dữ liệu categorical, so sánh giữa các nhóm.

```

# Dữ liệu accuracy của các models
models = ['Linear\nRegression', 'Decision\nTree',

```

**Hình 7.4:** So sánh histogram của phân phối chuẩn và phân phối đều

```
'Random\nForest', 'XGBoost', 'Neural\nNetwork']
accuracy = [0.78, 0.82, 0.88, 0.91, 0.89]
colors_list = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#FFA07A',
    ↪ '#98D8C8']

fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(models, accuracy, color=colors_list,
    edgecolor='black', linewidth=1.5)

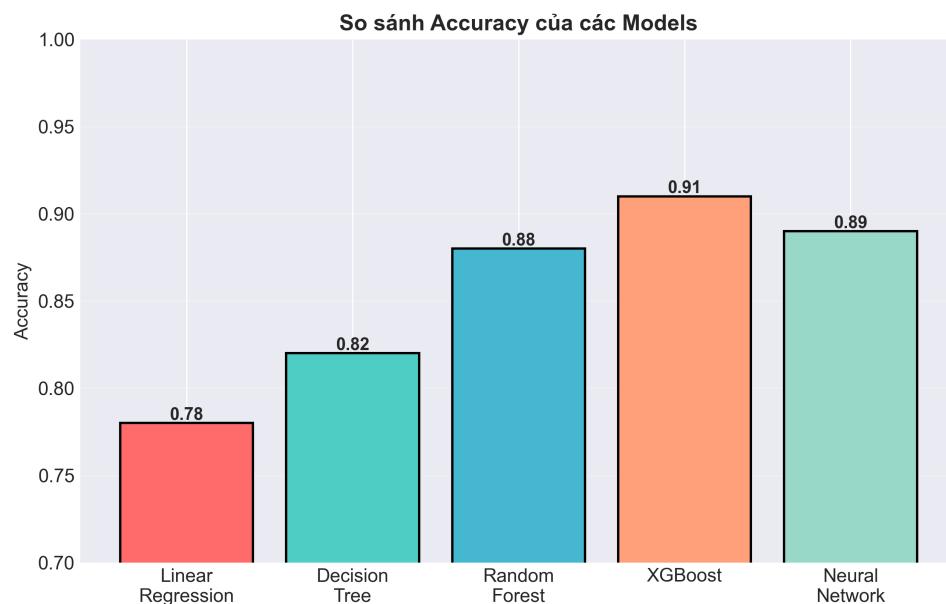
# Thêm giá trị trên mỗi cột
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height,
        f'{height:.2f}',
        ha='center', va='bottom', fontsize=11,
        ↪ fontweight='bold')

ax.set_ylabel('Accuracy', fontsize=12)
ax.set_title('So sánh Accuracy của các Models', fontsize=14,
    ↪ fontweight='bold')
ax.set_ylim(0.7, 1.0)
ax.grid(axis='y', alpha=0.3)

plt.savefig('Hinh_ve/ch8_bar_chart.png', dpi=300,
    ↪ bbox_inches='tight')
plt.show()
```

7.3.6 Subplots - Vẽ nhiều biểu đồ

Subplots cho phép so sánh nhiều khía cạnh dữ liệu cùng lúc.

**Hình 7.5:** Bar chart so sánh accuracy của các models

```

# Tạo dữ liệu training history
epochs = np.arange(1, 51)
train_loss = 2.5 * np.exp(-epochs/10) + 0.1
val_loss = 2.5 * np.exp(-epochs/12) + 0.15 +
    np.random.rand(50) * 0.1
train_acc = 1 - train_loss / 3
val_acc = 1 - val_loss / 3

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Loss curves
axes[0, 0].plot(epochs, train_loss, label='Train Loss',
    linewidth=2)
axes[0, 0].plot(epochs, val_loss, label='Val Loss',
    linewidth=2)
axes[0, 0].set_xlabel('Epochs')
axes[0, 0].set_ylabel('Loss')
axes[0, 0].set_title('Training vs Validation Loss')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Accuracy curves
axes[0, 1].plot(epochs, train_acc, label='Train Acc',
    linewidth=2)
axes[0, 1].plot(epochs, val_acc, label='Val Acc',
    linewidth=2)

```

```

axes[0, 1].set_xlabel('Epochs')
axes[0, 1].set_ylabel('Accuracy')
axes[0, 1].set_title('Training vs Validation Accuracy')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Learning rate schedule
lr = 0.01 * np.exp(-epochs/20)
axes[1, 0].plot(epochs, lr, color='green', linewidth=2)
axes[1, 0].set_xlabel('Epochs')
axes[1, 0].set_ylabel('Learning Rate')
axes[1, 0].set_title('Learning Rate Schedule')
axes[1, 0].grid(True, alpha=0.3)

# Overfitting gap
gap = val_loss - train_loss
axes[1, 1].fill_between(epochs, 0, gap, alpha=0.5,
    color='red')
axes[1, 1].plot(epochs, gap, color='darkred', linewidth=2)
axes[1, 1].set_xlabel('Epochs')
axes[1, 1].set_ylabel('Loss Gap')
axes[1, 1].set_title('Overfitting Gap (Val - Train)')
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_subplots.png', dpi=300,
    bbox_inches='tight')
plt.show()

```

7.4 Seaborn: Statistical Visualization

Seaborn được xây dựng trên Matplotlib, tập trung vào statistical graphics với API đơn giản và aesthetic mặc định đẹp hơn.

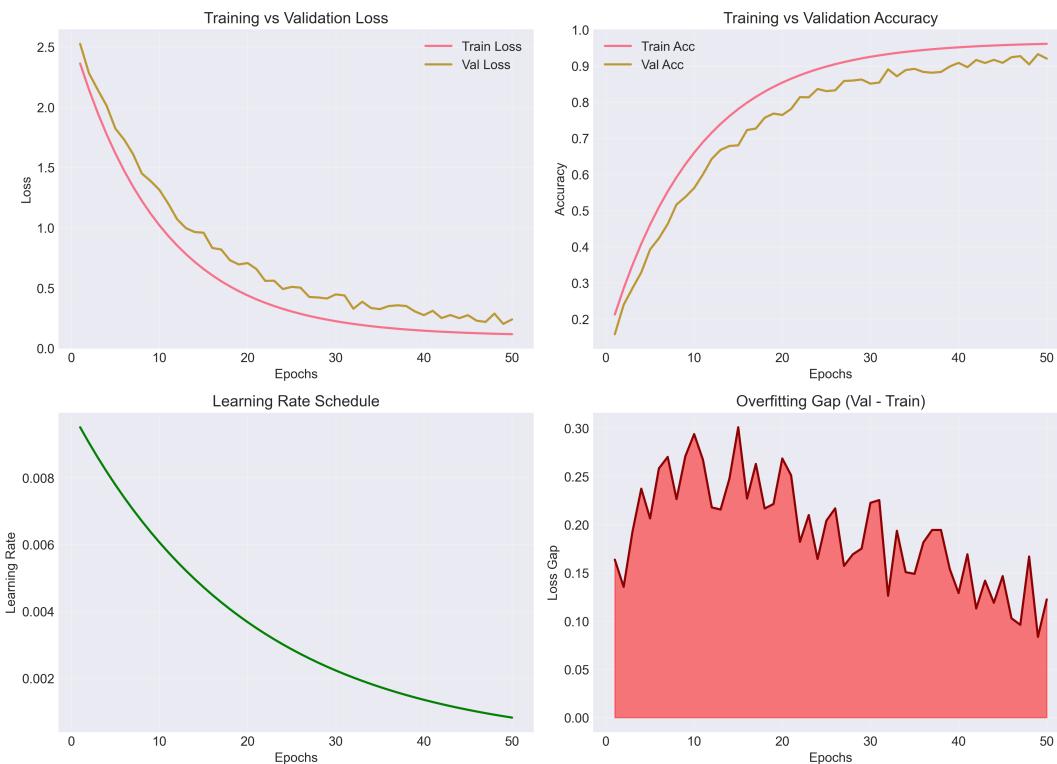
7.4.1 Boxplot - Phát hiện Outliers

Boxplot hiển thị five-number summary (min, Q1, median, Q3, max) và outliers, rất hữu ích trong EDA.

```

# Tạo dữ liệu với outliers
np.random.seed(42)
model_a = np.append(np.random.normal(0.85, 0.05, 100), [0.5,
    0.95, 0.98])
model_b = np.append(np.random.normal(0.88, 0.04, 100), [0.6,
    0.97])

```



Hình 7.6: Subplots hiển thị training history của neural network

```

model_c = np.random.normal(0.90, 0.06, 100)

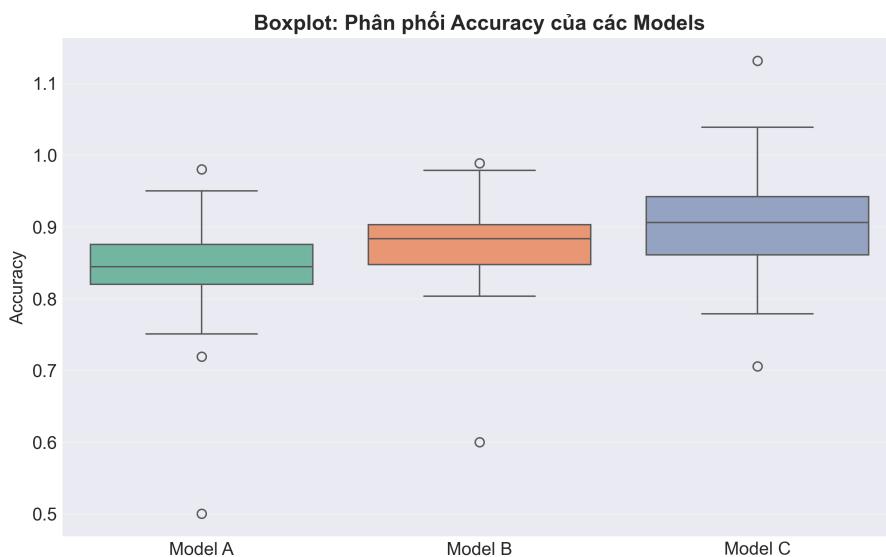
# Pad arrays to same length
max_len = max(len(model_a), len(model_b), len(model_c))
model_b = np.pad(model_b, (0, max_len - len(model_b)),
                  constant_values=np.nan)
model_c = np.pad(model_c, (0, max_len - len(model_c)),
                  constant_values=np.nan)

df = pd.DataFrame({ 'Model A': model_a, 'Model B': model_b,
                     'Model C': model_c})

fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data=df, palette='Set2', ax=ax)
ax.set_ylabel('Accuracy', fontsize=12)
ax.set_title('Boxplot: Phân phối Accuracy của các Models',
             fontsize=14, fontweight='bold')
ax.grid(axis='y', alpha=0.3)

plt.savefig('Hinh_ve/ch8_boxplot.png', dpi=300,
            bbox_inches='tight')
plt.show()

```



Hình 7.7: Boxplot phát hiện outliers trong performance của models

7.4.2 Violin Plot - Phân phối chi tiết

Violin plot kết hợp boxplot với kernel density estimation, cho thấy shape của distribution rõ ràng hơn.

```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

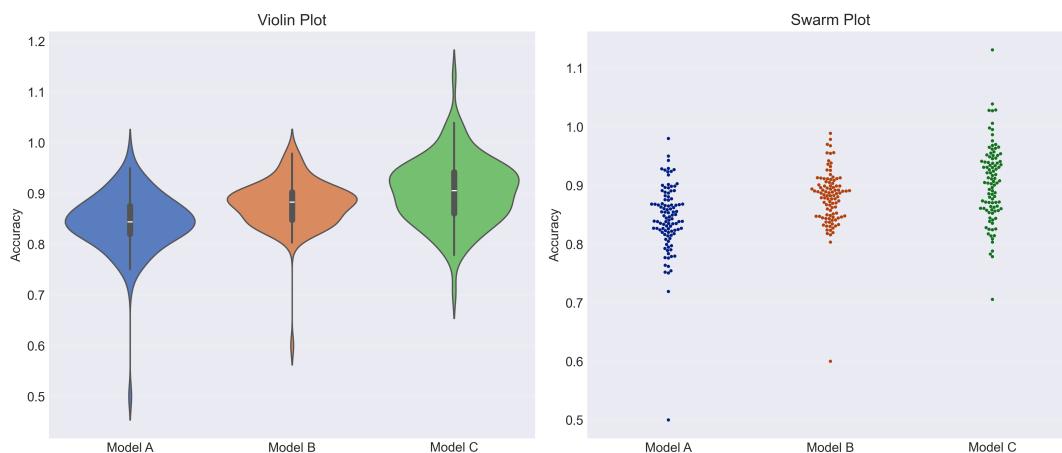
# Violin plot
sns.violinplot(data=df, palette='muted', ax=axes[0])
axes[0].set_ylabel('Accuracy')
axes[0].set_title('Violin Plot')
axes[0].grid(axis='y', alpha=0.3)

# Swarm plot (hiển thị từng điểm)
sns.swarmplot(data=df, palette='dark', size=3, ax=axes[1])
axes[1].set_ylabel('Accuracy')
axes[1].set_title('Swarm Plot')
axes[1].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_violin_swarm.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

7.4.3 Heatmap - Correlation Matrix

Heatmap hiển thị correlation matrix, giúp phát hiện multicollinearity và chọn features.

**Hình 7.8:** Violin plot và Swarm plot cho phân phối chi tiết

```
# Tạo dữ liệu features
np.random.seed(42)
n_samples = 500
feature_data = {
    'age': np.random.randint(18, 65, n_samples),
    'income': np.random.normal(50000, 15000, n_samples),
    'credit_score': np.random.randint(300, 850, n_samples),
    'debt': np.random.normal(10000, 5000, n_samples),
    'loan_amount': np.random.normal(30000, 10000, n_samples)
}

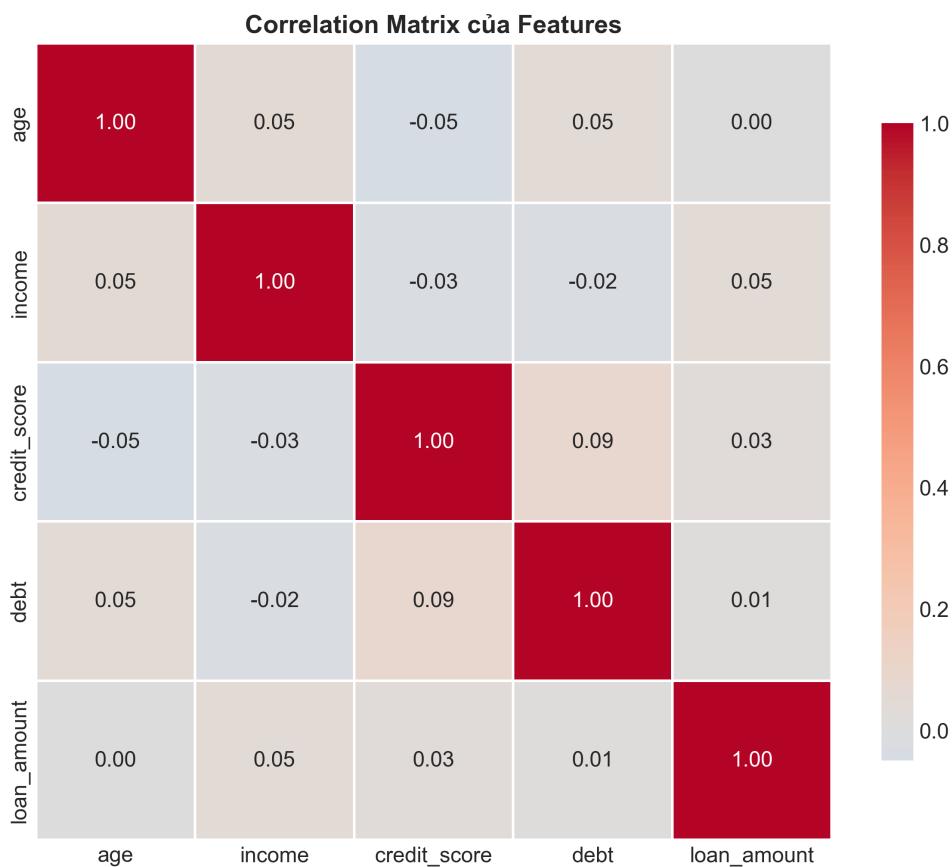
df_features = pd.DataFrame(feature_data)
correlation_matrix = df_features.corr()

fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt='%.2f',
            cmap='coolwarm', center=0, square=True,
            linewidths=1, cbar_kws={"shrink": 0.8}, ax=ax)
ax.set_title('Correlation Matrix của Features',
             fontsize=14, fontweight='bold')

plt.savefig('Hinh_ve/ch8_heatmap.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

7.4.4 Pairplot - Quan hệ giữa nhiều biến

Pairplot tạo grid of scatter plots cho tất cả các cặp biến, rất hữu ích trong EDA.

**Hình 7.9:** Heatmap hiển thị correlation matrix

```

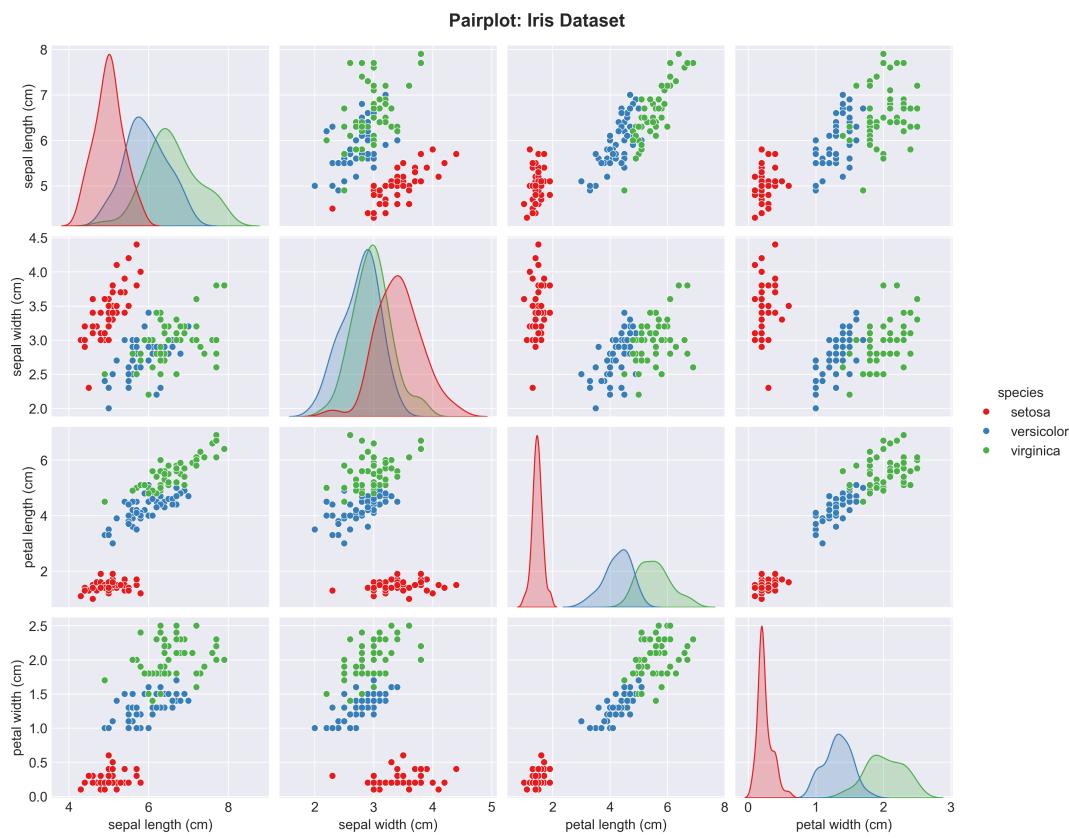
# Load Iris dataset
from sklearn.datasets import load_iris
iris_data = load_iris()
iris_df = pd.DataFrame(data=iris_data.data,
                        columns=iris_data.feature_names)
iris_df['species'] = iris_data.target
iris_df['species'] = iris_df['species'].map(
    {0: 'setosa', 1: 'versicolor', 2: 'virginica'})

# Pairplot
pairplot = sns.pairplot(iris_df, hue='species',
    palette='Set1',
    diag_kind='kde', height=2.5,
    aspect=1.2)
pairplot.fig.suptitle('Pairplot: Iris Dataset',
    y=1.02, fontsize=16,
    fontweight='bold')

plt.savefig('Hinh_ve/ch8_pairplot.png', dpi=300,
    bbox_inches='tight')

```

```
plt.show()
```



Hình 7.10: Pairplot khám phá quan hệ giữa các features trong Iris dataset

7.4.5 Distribution Plot với KDE

Kết hợp histogram với Kernel Density Estimation (KDE) để smooth distribution curve.

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Normal distribution
normal = np.random.normal(100, 15, 1000)
sns.histplot(normal, kde=True, color='blue', ax=axes[0, 0])
axes[0, 0].set_title('Normal Distribution')
axes[0, 0].axvline(normal.mean(), color='red',
                    linestyle='--',
                    label=f'Mean: {normal.mean():.2f}')
axes[0, 0].legend()

# Bimodal distribution
bimodal = np.concatenate([np.random.normal(80, 10, 500),
                           np.random.normal(120, 10, 500)])
sns.histplot(bimodal, kde=True, color='green', ax=axes[0,
                                                       1])
```

```

axes[0, 1].set_title('Bimodal Distribution')

# Skewed distribution
skewed = np.random.exponential(2, 1000)
sns.histplot(skewed, kde=True, color='orange', ax=axes[1,
→ 0])
axes[1, 0].set_title('Right-Skewed Distribution')

# Multiple distributions
for i, (mu, sigma, label) in enumerate([(0, 1, 'Group 1'),
                                         (3, 1.5, 'Group
→ 2'),
                                         (-2, 0.8, 'Group
→ 3')]):
    data = np.random.normal(mu, sigma, 500)
    sns.kdeplot(data, ax=axes[1, 1], label=label,
                 linewidth=2)
axes[1, 1].set_title('Comparison of Multiple Distributions')
axes[1, 1].legend()

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_distributions.png', dpi=300,
→ bbox_inches='tight')
plt.show()

```

7.5 Visualization cho Machine Learning

7.5.1 Confusion Matrix

Confusion matrix hiển thị performance của classification model, cho biết true/false positives/negatives.

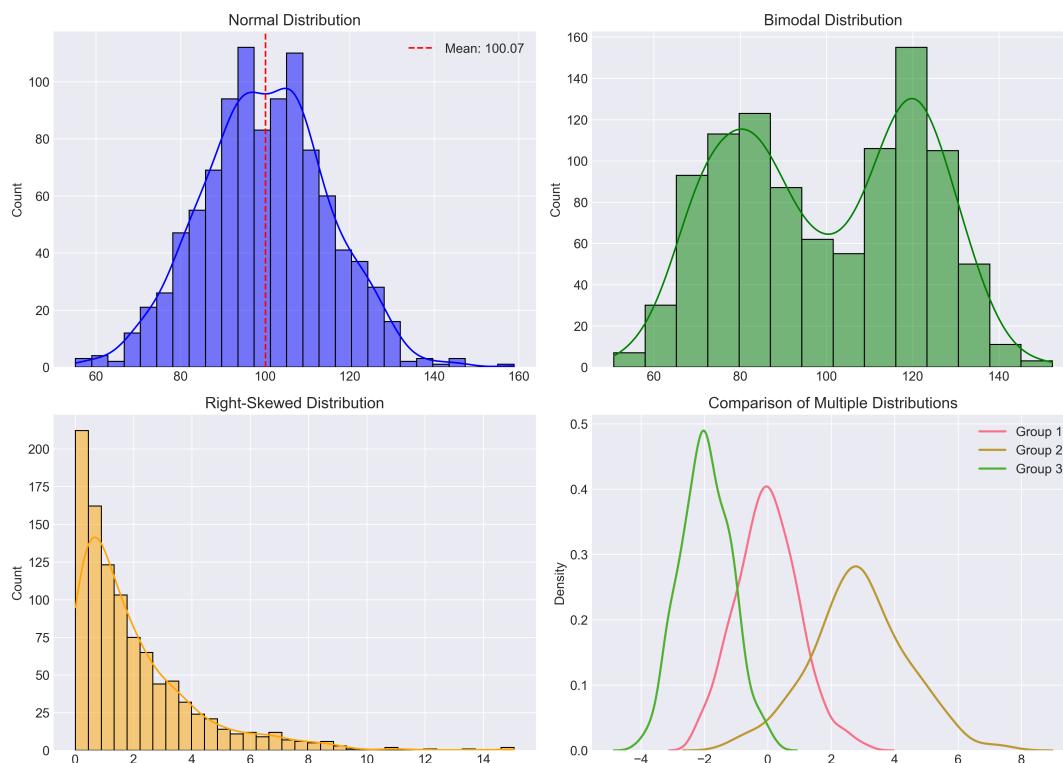
```

from sklearn.metrics import confusion_matrix
import seaborn as sns

# Giả lập dữ liệu predictions
np.random.seed(42)
y_true = np.random.randint(0, 3, 300) # 3 classes
y_pred = y_true.copy()
# Thêm một số predictions sai
errors = np.random.choice(300, 50, replace=False)
y_pred[errors] = np.random.randint(0, 3, 50)

# Tính confusion matrix
cm = confusion_matrix(y_true, y_pred)

```



Hình 7.11: Các loại phân phối dữ liệu với histogram và KDE

```
# Vẽ confusion matrix
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0', 'Class 1', 'Class 2'],
            yticklabels=['Class 0', 'Class 1', 'Class 2'],
            cbar_kws={'label': 'Count'}, ax=ax)
ax.set_xlabel('Predicted Label', fontsize=12)
ax.set_ylabel('True Label', fontsize=12)
ax.set_title('Confusion Matrix', fontsize=14,
             fontweight='bold')

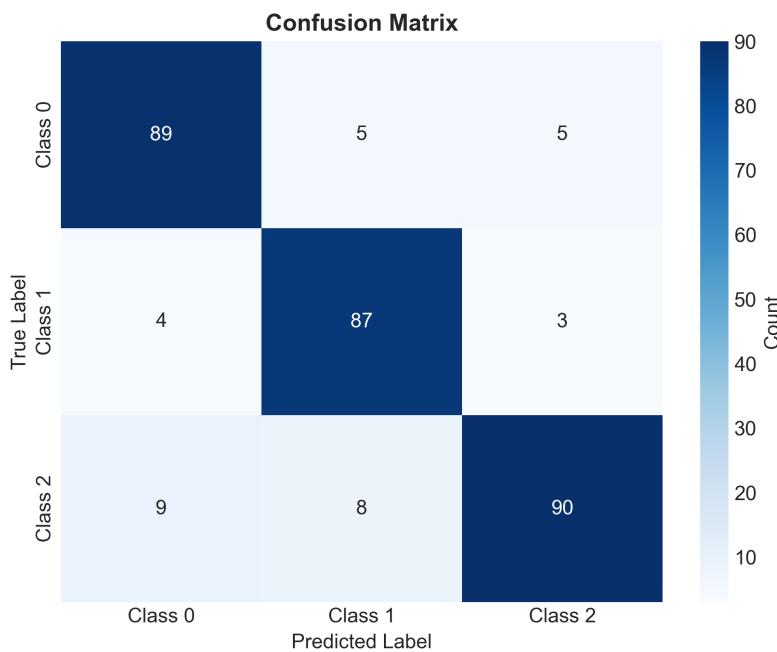
plt.savefig('Hinh_ve/ch8_confusion_matrix.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

7.5.2 ROC Curve và AUC

ROC curve hiển thị trade-off giữa True Positive Rate và False Positive Rate.

```
from sklearn.metrics import roc_curve, auc

# Giả lập probability predictions cho 3 models
```

**Hình 7.12:** Confusion Matrix cho multi-class classification

```

np.random.seed(42)
y_true_binary = np.random.randint(0, 2, 500)

# Model 1: Good
y_scores_1 = y_true_binary + np.random.normal(0, 0.3, 500)
# Model 2: Medium
y_scores_2 = y_true_binary + np.random.normal(0, 0.5, 500)
# Model 3: Poor
y_scores_3 = np.random.rand(500)

fig, ax = plt.subplots(figsize=(8, 8))

# Vẽ ROC cho từng model
for scores, label, color in [(y_scores_1, 'Model 1',
                                'blue'),
                                (y_scores_2, 'Model 2',
                                'green'),
                                (y_scores_3, 'Model 3',
                                'red')]:
    fpr, tpr, _ = roc_curve(y_true_binary, scores)
    roc_auc = auc(fpr, tpr)
    ax.plot(fpr, tpr, color=color, linewidth=2,
            label=f'{label} (AUC = {roc_auc:.2f})')

# Đường random baseline

```

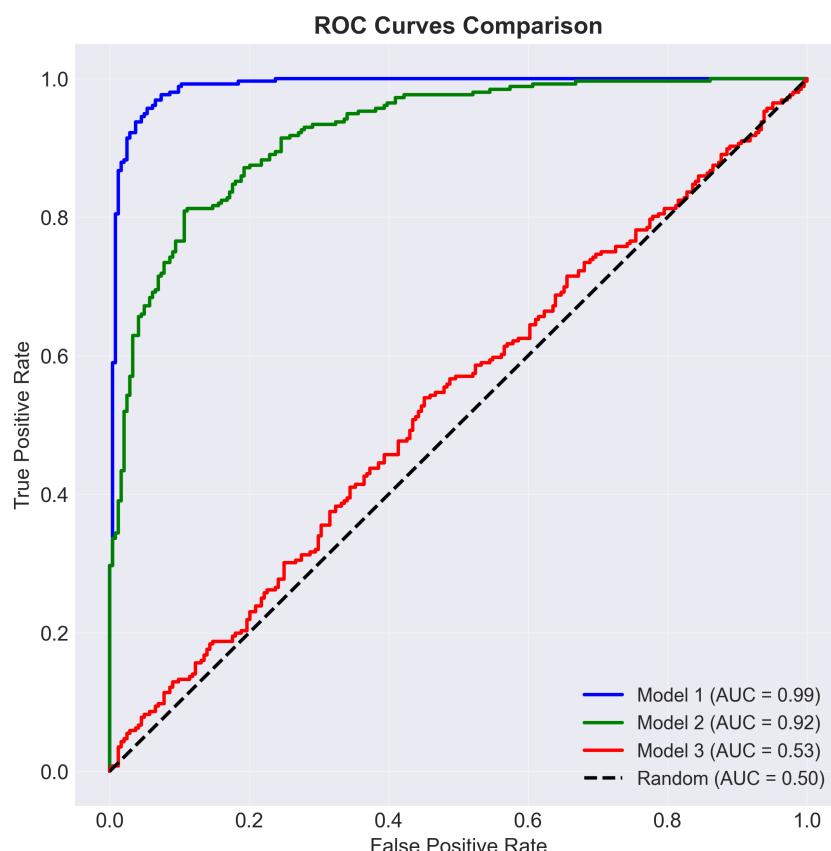
```

ax.plot([0, 1], [0, 1], 'k--', linewidth=2, label='Random
        ↵ (AUC = 0.50)')

ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.set_title('ROC Curves Comparison', fontsize=14,
             ↵ fontweight='bold')
ax.legend(loc='lower right', fontsize=11)
ax.grid(True, alpha=0.3)

plt.savefig('Hinh_ve/ch8_roc_curve.png', dpi=300,
            ↵ bbox_inches='tight')
plt.show()

```



Hình 7.13: ROC curves so sánh performance của nhiều models

7.5.3 Learning Curves - Phát hiện Overfitting

Learning curves giúp chẩn đoán overfitting, underfitting và đánh giá training progress.

```

# Tạo learning curves cho 3 scenarios
epochs = np.arange(1, 101)

```

```

# Scenario 1: Good fit
train_loss_good = 2 * np.exp(-epochs/15) + 0.05
val_loss_good = 2 * np.exp(-epochs/17) + 0.08
train_acc_good = 1 - 0.6 * np.exp(-epochs/12)
val_acc_good = 1 - 0.65 * np.exp(-epochs/14)

# Scenario 2: Overfitting
train_loss_over = 2 * np.exp(-epochs/10) + 0.02
val_loss_over = 2 * np.exp(-epochs/25) + 0.15 + epochs *
    ↵ 0.002
train_acc_over = 1 - 0.5 * np.exp(-epochs/8)
val_acc_over = 1 - 0.8 * np.exp(-epochs/20) - epochs * 0.001

# Scenario 3: Underfitting
train_loss_under = 1.5 + np.random.rand(100) * 0.2
val_loss_under = 1.6 + np.random.rand(100) * 0.2
train_acc_under = 0.5 + np.random.rand(100) * 0.05
val_acc_under = 0.48 + np.random.rand(100) * 0.05

fig, axes = plt.subplots(2, 3, figsize=(18, 10))

# Good Fit
axes[0, 0].plot(epochs, train_loss_good, label='Train',
    ↵ linewidth=2)
axes[0, 0].plot(epochs, val_loss_good, label='Validation',
    ↵ linewidth=2)
axes[0, 0].set_title('Good Fit: Loss', fontsize=12,
    ↵ fontweight='bold')
axes[0, 0].set_xlabel('Epochs')
axes[0, 0].set_ylabel('Loss')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

axes[1, 0].plot(epochs, train_acc_good, label='Train',
    ↵ linewidth=2)
axes[1, 0].plot(epochs, val_acc_good, label='Validation',
    ↵ linewidth=2)
axes[1, 0].set_title('Good Fit: Accuracy', fontsize=12,
    ↵ fontweight='bold')
axes[1, 0].set_xlabel('Epochs')
axes[1, 0].set_ylabel('Accuracy')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

```

```

# Overfitting
axes[0, 1].plot(epochs, train_loss_over, label='Train',
                 linewidth=2)
axes[0, 1].plot(epochs, val_loss_over, label='Validation',
                 linewidth=2)
axes[0, 1].set_title('Overfitting: Loss', fontsize=12,
                     fontweight='bold')
axes[0, 1].set_xlabel('Epochs')
axes[0, 1].set_ylabel('Loss')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

axes[1, 1].plot(epochs, train_acc_over, label='Train',
                 linewidth=2)
axes[1, 1].plot(epochs, val_acc_over, label='Validation',
                 linewidth=2)
axes[1, 1].set_title('Overfitting: Accuracy', fontsize=12,
                     fontweight='bold')
axes[1, 1].set_xlabel('Epochs')
axes[1, 1].set_ylabel('Accuracy')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

# Underfitting
axes[0, 2].plot(epochs, train_loss_under, label='Train',
                 linewidth=2, alpha=0.7)
axes[0, 2].plot(epochs, val_loss_under, label='Validation',
                 linewidth=2, alpha=0.7)
axes[0, 2].set_title('Underfitting: Loss', fontsize=12,
                     fontweight='bold')
axes[0, 2].set_xlabel('Epochs')
axes[0, 2].set_ylabel('Loss')
axes[0, 2].legend()
axes[0, 2].grid(True, alpha=0.3)

axes[1, 2].plot(epochs, train_acc_under, label='Train',
                 linewidth=2, alpha=0.7)
axes[1, 2].plot(epochs, val_acc_under, label='Validation',
                 linewidth=2, alpha=0.7)
axes[1, 2].set_title('Underfitting: Accuracy', fontsize=12,
                     fontweight='bold')
axes[1, 2].set_xlabel('Epochs')
axes[1, 2].set_ylabel('Accuracy')
axes[1, 2].legend()

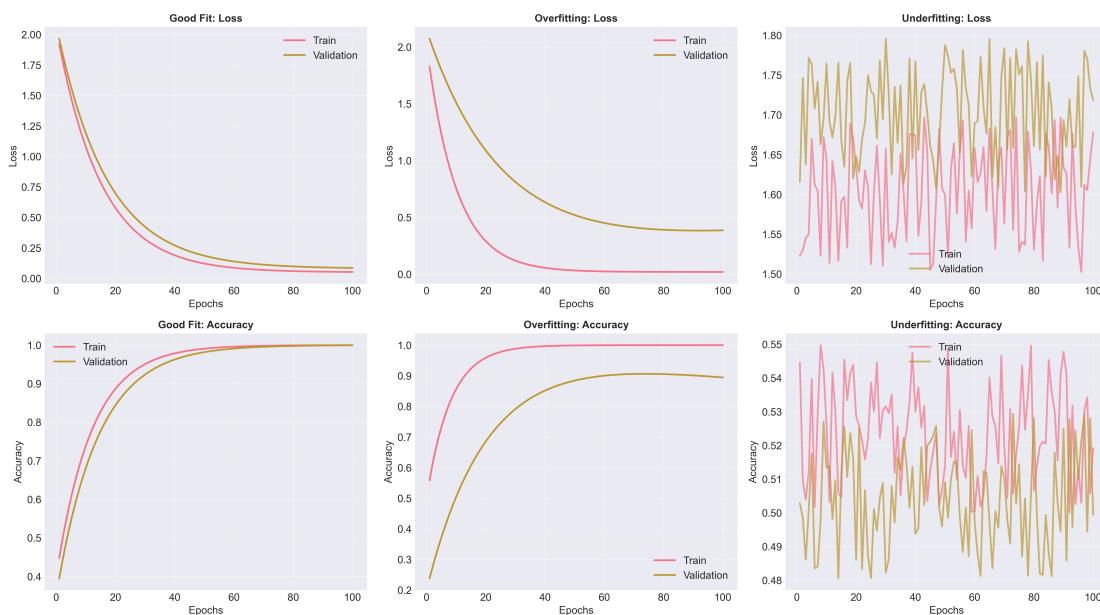
```

```

axes[1, 2].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_learning_curves.png', dpi=300,
            bbox_inches='tight')
plt.show()

```



Hình 7.14: Learning curves cho 3 scenarios: Good fit, Overfitting, Underfitting

7.5.4 Feature Importance

Hiển thị mức độ quan trọng của features trong model.

```

# Feature importance từ Random Forest (giả lập)
features = ['Age', 'Income', 'Credit Score', 'Employment
            Years',
            'Debt Ratio', 'Loan Amount', 'Previous Loans',
            'Property Value', 'Education Level', 'City
            Size']

importances = np.array([0.15, 0.22, 0.18, 0.09, 0.14,
                      0.08, 0.06, 0.04, 0.03, 0.01])

# Sort theo importance
indices = np.argsort(importances)[::-1]
sorted_features = [features[i] for i in indices]
sorted_importances = importances[indices]

fig, ax = plt.subplots(figsize=(10, 6))

```

```

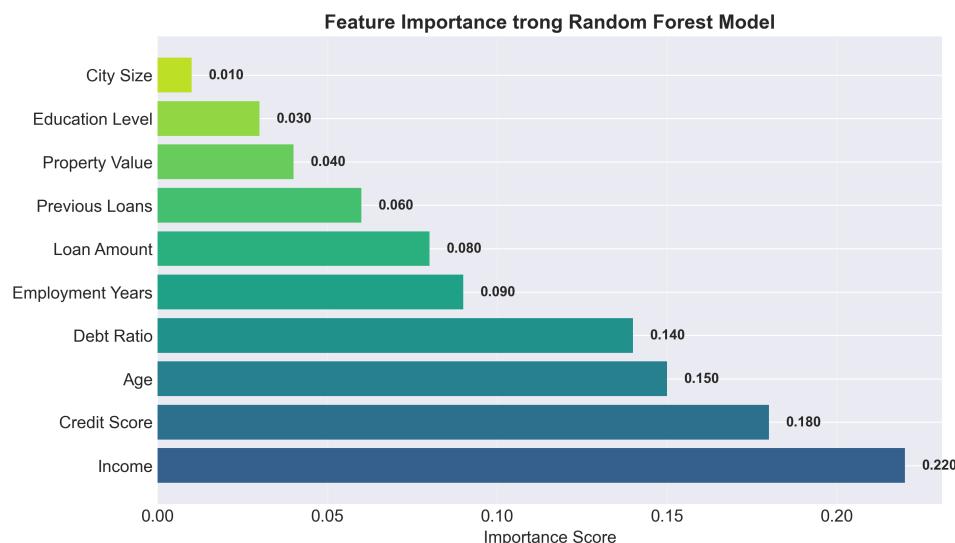
colors = plt.cm.viridis(np.linspace(0.3, 0.9,
                                   len(sorted_features)))
bars = ax.barh(sorted_features, sorted_importances,
               color=colors)

# Thêm giá trị
for i, (bar, imp) in enumerate(zip(bars,
                                    sorted_importances)):
    ax.text(imp + 0.005, i, f'{imp:.3f}',
            va='center', fontsize=10, fontweight='bold')

ax.set_xlabel('Importance Score', fontsize=12)
ax.set_title('Feature Importance trong Random Forest Model',
             fontsize=14, fontweight='bold')
ax.grid(axis='x', alpha=0.3)

plt.savefig('Hinh_ve/ch8_feature_importance.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

**Hình 7.15:** Feature importance từ Random Forest model

7.5.5 Precision-Recall Curve

Precision-Recall curve phù hợp với imbalanced datasets.

```

from sklearn.metrics import precision_recall_curve,
                           average_precision_score

# Giả lập dữ liệu imbalanced

```

```

np.random.seed(42)
y_true_imb = np.concatenate([np.ones(100), np.zeros(900)])
y_scores_good = y_true_imb + np.random.normal(0, 0.3, 1000)
y_scores_poor = y_true_imb + np.random.normal(0, 0.7, 1000)

fig, ax = plt.subplots(figsize=(8, 8))

for scores, label, color in [(y_scores_good, 'Good Model',
→ 'blue'),
                               (y_scores_poor, 'Poor Model',
→ 'red')]:
    precision, recall, _ =
        precision_recall_curve(y_true_imb, scores)
    ap = average_precision_score(y_true_imb, scores)
    ax.plot(recall, precision, color=color, linewidth=2,
            label=f'{label} (AP = {ap:.2f})')

# Baseline
ax.axhline(y=0.1, color='k', linestyle='--', linewidth=2,
            label='Baseline (Random)')

ax.set_xlabel('Recall', fontsize=12)
ax.set_ylabel('Precision', fontsize=12)
ax.set_title('Precision-Recall Curve (Imbalanced Data)',
            fontsize=14, fontweight='bold')
ax.legend(loc='upper right', fontsize=11)
ax.grid(True, alpha=0.3)

plt.savefig('Hinh_ve/ch8_pr_curve.png', dpi=300,
→ bbox_inches='tight')
plt.show()

```

7.6 Visualization cho Deep Learning

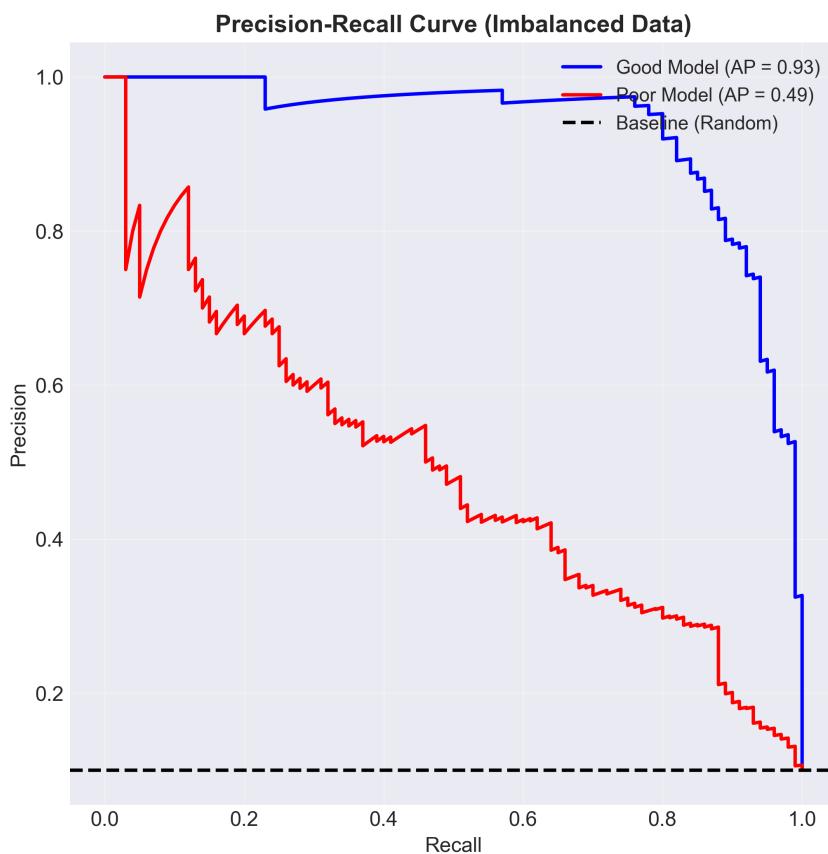
7.6.1 Training History Visualization

Theo dõi metrics trong quá trình training neural network.

```

# Giả lập training history từ Keras/TensorFlow
epochs = np.arange(1, 51)
history = {
    'loss': 1.5 * np.exp(-epochs/10) + 0.1 +
        np.random.rand(50) * 0.05,
    'val_loss': 1.5 * np.exp(-epochs/12) + 0.15 +
        np.random.rand(50) * 0.08,
}

```

**Hình 7.16:** Precision-Recall curve cho imbalanced dataset

```

'accuracy': 1 - 0.8 * np.exp(-epochs/8),
'val_accuracy': 1 - 0.85 * np.exp(-epochs/10),
'lr': 0.001 * np.exp(-epochs/20)
}

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Loss
axes[0, 0].plot(epochs, history['loss'], 'b-',
                 label='Training Loss', linewidth=2)
axes[0, 0].plot(epochs, history['val_loss'], 'r-',
                 label='Validation Loss', linewidth=2)
axes[0, 0].set_xlabel('Epoch', fontsize=11)
axes[0, 0].set_ylabel('Loss', fontsize=11)
axes[0, 0].set_title('Model Loss', fontsize=13,
                     fontweight='bold')
axes[0, 0].legend(fontsize=10)
axes[0, 0].grid(True, alpha=0.3)

# Accuracy

```

```

axes[0, 1].plot(epochs, history['accuracy'], 'b-',
                 label='Training Accuracy', linewidth=2)
axes[0, 1].plot(epochs, history['val_accuracy'], 'r-',
                 label='Validation Accuracy', linewidth=2)
axes[0, 1].set_xlabel('Epoch', fontsize=11)
axes[0, 1].set_ylabel('Accuracy', fontsize=11)
axes[0, 1].set_title('Model Accuracy', fontsize=13,
                     fontweight='bold')
axes[0, 1].legend(fontsize=10)
axes[0, 1].grid(True, alpha=0.3)

# Learning Rate
axes[1, 0].plot(epochs, history['lr'], 'g-', linewidth=2)
axes[1, 0].set_xlabel('Epoch', fontsize=11)
axes[1, 0].set_ylabel('Learning Rate', fontsize=11)
axes[1, 0].set_title('Learning Rate Schedule', fontsize=13,
                     fontweight='bold')
axes[1, 0].set_yscale('log')
axes[1, 0].grid(True, alpha=0.3)

# Loss Gap (Overfitting indicator)
gap = history['val_loss'] - history['loss']
axes[1, 1].fill_between(epochs, 0, gap, alpha=0.3,
                       color='red')
axes[1, 1].plot(epochs, gap, 'r-', linewidth=2)
axes[1, 1].axhline(y=0, color='k', linestyle='--',
                    alpha=0.5)
axes[1, 1].set_xlabel('Epoch', fontsize=11)
axes[1, 1].set_ylabel('Val Loss - Train Loss', fontsize=11)
axes[1, 1].set_title('Overfitting Gap', fontsize=13,
                     fontweight='bold')
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_dl_training.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

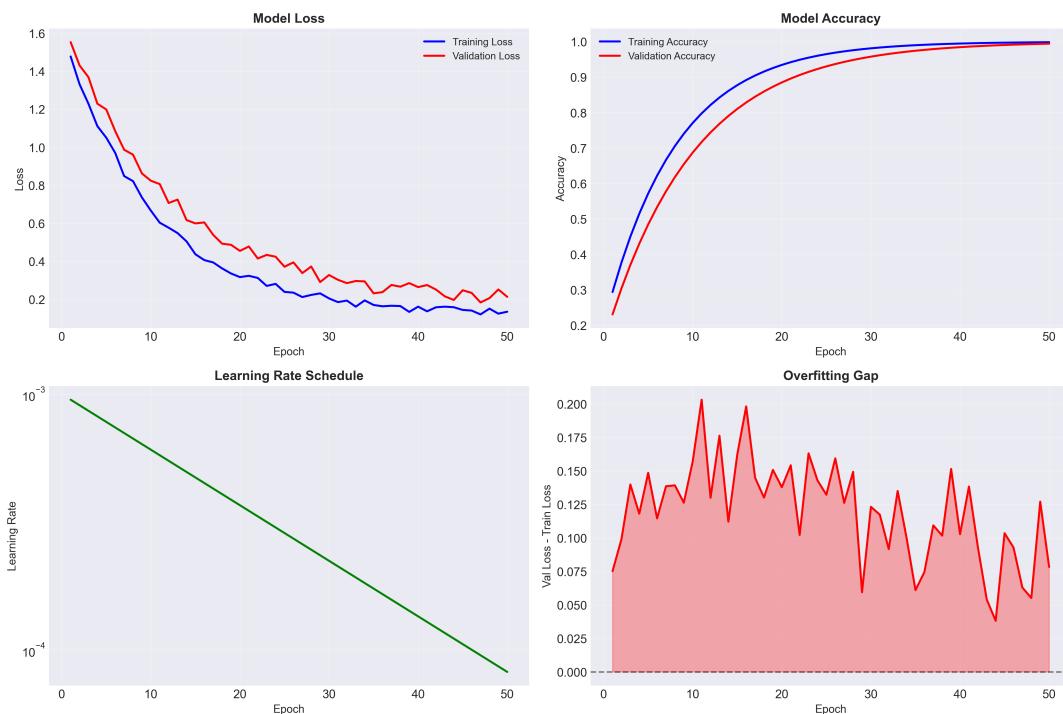
7.6.2 Visualizing Neural Network Architecture

Mô tả kiến trúc neural network một cách trực quan.

```

# Visualization đơn giản cho NN architecture
fig, ax = plt.subplots(figsize=(12, 8))

```



Hình 7.17: Training history visualization cho deep learning model

```

layers = [4, 6, 6, 3] # Input, Hidden1, Hidden2, Output
layer_names = ['Input\nLayer', 'Hidden\nLayer 1',
               'Hidden\nLayer 2', 'Output\nLayer']

# Vẽ neurons
for i, (n_neurons, name) in enumerate(zip(layers,
                                             layer_names)):
    x = i * 3
    for j in range(n_neurons):
        y = j - n_neurons/2
        circle = plt.Circle((x, y), 0.3, color='lightblue',
                             ec='black', linewidth=2)
        ax.add_patch(circle)
    ax.text(x, -n_neurons/2 - 1, name, ha='center',
            fontsize=12, fontweight='bold')

# Vẽ connections (chỉ vẽ một số để tránh rối)
for i in range(len(layers) - 1):
    for j in range(min(layers[i], 3)):
        for k in range(min(layers[i+1], 3)):
            x1, y1 = i * 3, j - layers[i]/2
            x2, y2 = (i+1) * 3, k - layers[i+1]/2
            ax.plot([x1+0.3, x2-0.3], [y1, y2], 'gray',

```

```

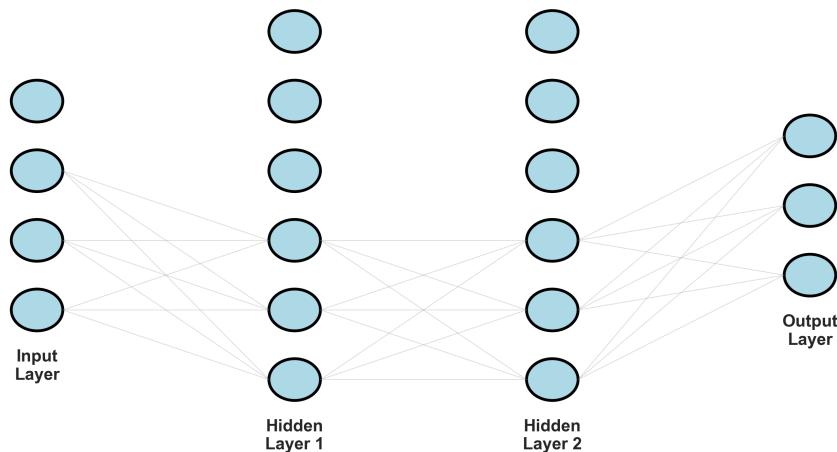
alpha=0.3, linewidth=0.5)

ax.set_xlim(-1, 10)
ax.set_ylim(-5, 4)
ax.axis('off')
ax.set_title('Neural Network Architecture (4-6-6-3)', fontsize=14, fontweight='bold')

plt.savefig('Hinh_ve/ch8_nn_architecture.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

Neural Network Architecture (4-6-6-3)

**Hình 7.18:** Visualization của neural network architecture

7.6.3 Activation Maps Visualization

Trực quan hóa activation outputs của các layers trong CNN.

```

# Giả lập activation maps từ CNN layers
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
fig.suptitle('CNN Activation Maps', fontsize=16,
             fontweight='bold')

for idx, ax in enumerate(axes.flat):
    # Tạo activation map giả lập
    activation = np.random.rand(28, 28) * (idx + 1) / 8
    activation = activation * (activation > 0.3)  #
    #ReLU-like

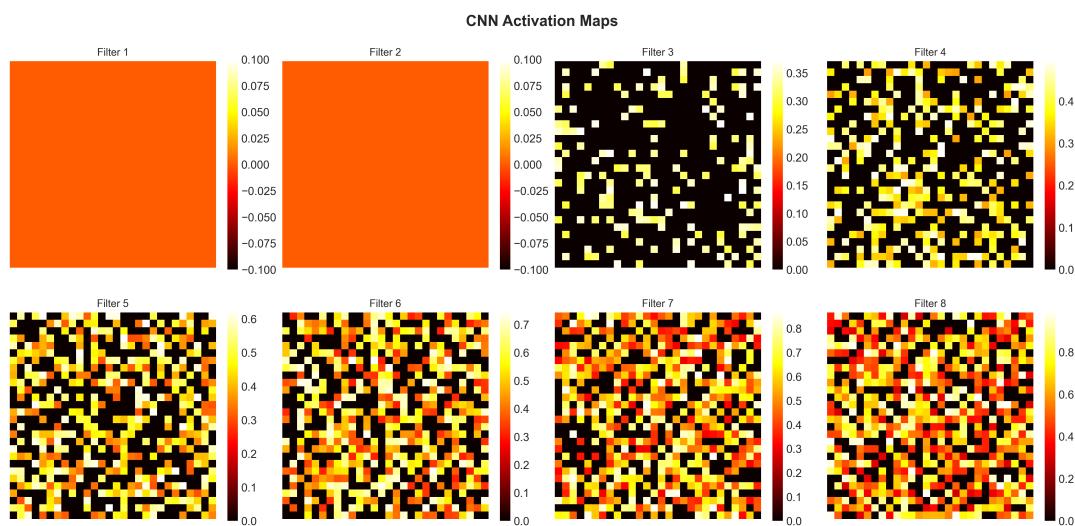
```

```

im = ax.imshow(activation, cmap='hot',
    ↪ interpolation='nearest')
ax.set_title(f'Filter {idx + 1}', fontsize=11)
ax.axis('off')
plt.colorbar(im, ax=ax, fraction=0.046)

plt.tight_layout()
plt.savefig('Hinh_ve/ch8_activation_maps.png', dpi=300,
    ↪ bbox_inches='tight')
plt.show()

```



Hình 7.19: Activation maps từ CNN convolutional layers

7.7 Best Practices trong Data Visualization

7.7.1 Nguyên tắc thiết kế biểu đồ hiệu quả

1. Chọn loại biểu đồ phù hợp:

- Line plot: Dữ liệu chuỗi thời gian, xu hướng
- Bar chart: So sánh giữa các categories
- Scatter plot: Quan hệ giữa hai biến liên tục
- Histogram: Phân phối của một biến
- Box plot: So sánh distributions, phát hiện outliers
- Heatmap: Ma trận correlation, confusion matrix

2. Sử dụng màu sắc hiệu quả:

- Chọn palette phù hợp: sequential, diverging, qualitative
- Tránh quá nhiều màu khác nhau

- Đảm bảo contrast tốt cho dễ nhìn
- Cân nhắc color blindness

3. Labels và Annotations:

- Title rõ ràng, descriptive
- Axis labels với units
- Legend khi cần
- Annotations cho điểm quan trọng

4. Simplicity:

- Loại bỏ chart junk không cần thiết
- Focus vào message chính
- Sử dụng whitespace hợp lý

7.7.2 Code Template tái sử dụng

```
def plot_learning_curves(history, figsize=(12, 5)):
    """
    Vẽ learning curves từ training history

    Parameters:
    -----
    history : dict
        Dictionary chứa 'loss', 'val_loss', 'accuracy',
        ↳ 'val_accuracy'
    figsize : tuple
        Kích thước figure
    """
    fig, axes = plt.subplots(1, 2, figsize=figsize)

    # Loss
    axes[0].plot(history['loss'], label='Train')
    axes[0].plot(history['val_loss'], label='Validation')
    axes[0].set_xlabel('Epoch')
    axes[0].set_ylabel('Loss')
    axes[0].set_title('Model Loss')
    axes[0].legend()
    axes[0].grid(True, alpha=0.3)

    # Accuracy
    axes[1].plot(history['accuracy'], label='Train')
```

```

    axes[1].plot(history['val_accuracy'],
                  label='Validation')
    axes[1].set_xlabel('Epoch')
    axes[1].set_ylabel('Accuracy')
    axes[1].set_title('Model Accuracy')
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)

    plt.tight_layout()
    return fig

# Sử dụng
# fig = plot_learning_curves(history)
# plt.savefig('learning_curves.png', dpi=300,
#             bbox_inches='tight')

```

7.8 Bài tập thực hành

Bài tập 1: Biểu đồ cơ bản Vẽ biểu đồ đường thể hiện nhiệt độ trong 7 ngày: [25, 27, 26, 28, 30, 29, 27] độ C. Thêm grid, labels, title và lưu hình với DPI cao.

Bài tập 2: Histogram và Distribution Tạo 1000 số ngẫu nhiên từ phân phối chuẩn (mean=100, std=15). Vẽ histogram với 30 bins, thêm đường KDE và đánh dấu mean, median trên biểu đồ.

Bài tập 3: Scatter plot với màu sắc Tạo 100 điểm dữ liệu 2D. Vẽ scatter plot với màu sắc thể hiện giá trị của biến thứ 3. Thêm colorbar và tùy chỉnh colormap.

Bài tập 4: Bar chart so sánh Cho điểm thi của 5 sinh viên ở 3 môn (Toán, Lý, Hóa). Vẽ grouped bar chart so sánh điểm của từng sinh viên. Thêm giá trị trên mỗi cột.

Bài tập 5: EDA với Iris Dataset Sử dụng dataset Iris từ scikit-learn. Thực hiện EDA đầy đủ:

- Vẽ pairplot để xem correlation giữa các features
- Vẽ heatmap của correlation matrix
- Vẽ boxplot để phát hiện outliers cho mỗi feature theo từng class
- Vẽ violin plot để so sánh distribution

Bài tập 6: Confusion Matrix Tạo confusion matrix giả định cho bài toán phân loại 4 lớp với 200 samples. Vẽ confusion matrix dạng heatmap, thêm annotations. Tính và hiển thị:

- Precision, Recall, F1-score cho mỗi class
- Overall accuracy

- Class-wise performance bar chart

Bài tập 7: Learning Curves Analysis Mô phỏng training history của một mô hình:

- Epochs: 50
- Train loss giảm từ 2.0 xuống 0.1
- Val loss giảm từ 2.0 xuống 0.3 rồi tăng lên 0.5 (overfitting)
- Train accuracy tăng từ 0.3 lên 0.98
- Val accuracy tăng từ 0.3 lên 0.92 rồi giảm xuống 0.88

Vẽ learning curves và phân tích:

- Mô hình có overfitting không?
- Từ epoch nào bắt đầu overfitting?
- Đề xuất early stopping point

Bài tập 8: ROC và PR Curves (Nâng cao) Tạo dữ liệu cho bài toán binary classification:

- 500 samples (250 positive, 250 negative)
- So sánh 3 models với predicted probabilities khác nhau
- Vẽ ROC curves của cả 3 models trên cùng một plot
- Vẽ Precision-Recall curves của cả 3 models
- Tính AUC cho mỗi model
- Phân tích model nào tốt nhất và tại sao

Bài tập 9: Feature Importance Visualization Tạo random forest model cho regression problem. Trích xuất feature importance và visualization:

- Bar chart horizontal cho top 15 features
- Sắp xếp theo importance
- Color coding theo mức độ quan trọng
- Thêm cumulative importance line

Bài tập 10: Dashboard tổng hợp (Nâng cao) Tạo một dashboard đầy đủ cho một bài toán classification:

- Dataset: Sử dụng wine quality dataset hoặc tương tự
- Chia layout: 2 rows, 3 columns (6 subplots)
- Subplot 1: Distribution của target variable
- Subplot 2: Correlation heatmap
- Subplot 3: Top 5 feature importance

- Subplot 4: Confusion matrix
- Subplot 5: ROC curve với AUC score
- Subplot 6: Learning curves (loss và accuracy)

Yêu cầu: Sử dụng consistent color scheme, proper spacing, informative titles, và lưu với resolution cao.

7.9 Kết luận

Chương này đã giới thiệu toàn diện về visualization trong Python, với trọng tâm vào các ứng dụng trong Data Science, Machine Learning và Deep Learning. Visualization không chỉ là công cụ để "vẽ đẹp" mà là kỹ năng thiết yếu trong toàn bộ quy trình Data Science. Từ việc hiểu dữ liệu ban đầu (EDA), đến theo dõi quá trình training, đánh giá model, và truyền đạt insights cho stakeholders - visualization đều đóng vai trò then chốt.

Trong các chương tiếp theo, chúng ta sẽ áp dụng các kỹ thuật visualization này vào các bài toán thực tế về xử lý dữ liệu, machine learning và deep learning. Các bài tập thực hành trong chương này sẽ giúp củng cố kỹ năng và chuẩn bị tốt cho các ứng dụng phức tạp hơn.

Lưu ý quan trọng: Mọi biểu đồ trong chương này đều có thể tái tạo bằng cách chạy code examples. Hình ảnh được lưu trong thư mục `Hinh_ve/` với prefix `ch8_`. Sinh viên nên thực hành chạy từng đoạn code để hiểu sâu hơn về cách hoạt động của từng loại visualization và tùy chỉnh cho bài toán cụ thể của mình.

CHƯƠNG 8. Tính toán Khoa học với Thư viện SciPy

8.1 Giới thiệu về SciPy

Trong lĩnh vực tính toán khoa học và kỹ thuật, việc giải quyết các bài toán phức tạp như tích phân, giải phương trình vi phân, tối ưu hóa hay nội suy dữ liệu là những yêu cầu thường xuyên. Thư viện SciPy (Scientific Python) được phát triển nhằm cung cấp một hệ sinh thái đầy đủ các công cụ tính toán số cho Python, mở rộng và bổ sung cho khả năng xử lý mảng của NumPy.

SciPy không chỉ đơn thuần là một tập hợp các hàm toán học, mà còn là một framework tổng thể được thiết kế theo triết lý khoa học: mỗi module được phát triển bởi các chuyên gia trong lĩnh vực tương ứng, đảm bảo tính chính xác, hiệu suất cao và tuân thủ các chuẩn mực toán học quốc tế. Điều này khiến SciPy trở thành công cụ không thể thiếu trong nghiên cứu khoa học, phát triển sản phẩm công nghệ và giảng dạy đại học.

8.1.1 Cài đặt và Import

Trước khi sử dụng SciPy, chúng ta cần cài đặt thư viện thông qua pip. Vì SciPy phụ thuộc vào NumPy, hệ thống sẽ tự động cài đặt NumPy nếu chưa có sẵn.

```
pip install scipy numpy matplotlib
```

Sau khi cài đặt, việc import các module của SciPy tuân theo quy ước: thay vì import toàn bộ thư viện, ta chỉ import các submodule cần thiết để tránh xung đột tên và tối ưu bộ nhớ.

```
import numpy as np
from scipy import integrate, optimize, interpolate
import matplotlib.pyplot as plt
```

8.1.2 Các Hàm Cơ bản và Hằng số Toán học

SciPy cung cấp một bộ sưu tập phong phú các hàm đặc biệt (special functions) và hằng số toán học chính xác cao, được sử dụng rộng rãi trong các bài toán khoa học và kỹ thuật.

a, Hằng số Toán học

Module `scipy.constants` chứa các hằng số vật lý và toán học được định nghĩa theo chuẩn quốc tế CODATA. Việc sử dụng các hằng số này đảm bảo tính chính xác và nhất quán trong tính toán khoa học.

```
from scipy import constants
# Hằng số toán học
```

```

print(f"Số pi: {constants.pi}")
print(f"Số e: {np.e}")
print(f"Tỉ số vàng: {constants.golden}")

# Hằng số vật lý
print(f"Tốc độ ánh sáng: {constants.c} m/s")
print(f"Hằng số Planck: {constants.h} J·s")
print(f"Hằng số Avogadro: {constants.Avogadro} mol^-1")
print(f"Hằng số khí lý tưởng: {constants.R} J/(mol·K)")

```

b, Hàm Đặc biệt

Module `scipy.special` cung cấp hàng trăm hàm đặc biệt trong toán học, bao gồm các hàm gamma, hàm Bessel, hàm elliptic, và nhiều hàm khác. Những hàm này xuất hiện thường xuyên trong lý thuyết xác suất, cơ học lượng tử, xử lý tín hiệu và nhiều lĩnh vực khác.

```

from scipy import special

# Hàm Gamma và log-Gamma
x = 5
print(f"Gamma({x}) = {special.gamma(x)}") # Gamma(5) = 4! =
    ↳ 24
print(f"ln[Gamma({x})] = {special.loggamma(x)}")

# Hàm Bessel bậc nhất loại thứ nhất (quan trọng trong sóng
    ↳ điện từ)
x_values = np.linspace(0, 10, 100)
j0 = special.j0(x_values) # Bessel function of first kind,
    ↳ order 0

# Hàm lỗi (Error function) - quan trọng trong xác suất và
    ↳ thống kê
z = 1.5
print(f"erf({z}) = {special.erf(z)}")
print(f"erfc({z}) = {special.erfc(z)}") # Complementary
    ↳ error function

# Tổ hợp và gaii thừa
n, k = 10, 3
print(f"C({n}, {k}) = {special.comb(n, k)}")
print(f"{n}! = {special.factorial(n)}")

```

Ví dụ, hàm Gamma $\Gamma(n) = (n - 1)!$ là mở rộng liên tục của gaii thừa sang số thực

và phức. Hàm lỗi erf(x) xuất hiện trong phân phối chuẩn và nhiều bài toán nhiệt động lực học. Hiểu rõ các hàm này giúp chúng ta giải quyết các bài toán phức tạp một cách chính xác và hiệu quả.

8.2 Tổng quan Các Gói của SciPy

SciPy được tổ chức thành các module chuyên biệt, mỗi module tập trung vào một lĩnh vực cụ thể của tính toán khoa học. Kiến trúc modular này không chỉ giúp dễ dàng bảo trì và phát triển, mà còn cho phép người dùng chỉ import những phần cần thiết, tối ưu hóa hiệu suất chương trình.

8.2.1 scipy.integrate - Tích phân và Phương trình Vi phân

Module này cung cấp các phương pháp tích phân số chính xác cao và các công cụ giải phương trình vi phân thường (ODE) cũng như phương trình vi phân đại số (DAE). Trong kỹ thuật, việc tính toán diện tích dưới đường cong, tổng năng lượng, hoặc mô phỏng động học của hệ thống đều dựa vào các công cụ này.

8.2.2 scipy.optimize - Tối ưu hóa và Tìm nghiệm

Tối ưu hóa là bài toán tìm giá trị cực đại hoặc cực tiểu của một hàm số dưới các ràng buộc nhất định. Module `scipy.optimize` cung cấp đầy đủ các thuật toán từ cơ bản (gradient descent, Newton) đến nâng cao (simulated annealing, differential evolution). Bên cạnh đó, module này còn chứa các công cụ tìm nghiệm của phương trình phi tuyến, từ phương pháp chia đôi (bisection) đến các phương pháp lặp Newton-Raphson.

8.2.3 scipy.interpolate - Nội suy và Xấp xỉ

Trong thực tế, dữ liệu thường được thu thập tại các điểm rời rạc. Nội suy (interpolation) là kỹ thuật ước lượng giá trị tại các điểm trung gian dựa trên dữ liệu đã biết. Module này cung cấp nhiều phương pháp nội suy khác nhau: tuyến tính, đa thức, spline, và radial basis functions, mỗi phương pháp phù hợp với từng loại dữ liệu và yêu cầu độ mịn khác nhau.

8.2.4 scipy.linalg - Đại số Tuyến tính

Mở rộng từ `numpy.linalg`, module này cung cấp các thuật toán đại số tuyến tính chuyên sâu hơn, bao gồm phân tích QR, SVD, phân tách Cholesky, và giải các hệ phương trình tuyến tính với ma trận đặc biệt (tam giác, băng, thưa). Đây là nền tảng cho machine learning, xử lý ảnh, và mô phỏng hệ thống vật lý.

8.2.5 scipy.signal - Xử lý Tín hiệu

Module này chuyên về xử lý tín hiệu số và analog: thiết kế bộ lọc, phân tích phổ, convolution, correlation, và biến đổi Fourier. Ứng dụng rộng rãi trong viễn thông, âm thanh, hình ảnh và điều khiển tự động.

8.2.6 scipy.stats - Thống kê và Xác suất

Cung cấp hơn 100 phân phối xác suất liên tục và rời rạc, các kiểm định thống kê (t-test, chi-square, ANOVA), và công cụ phân tích dữ liệu thống kê. Đây là công cụ thiết yếu cho khoa học dữ liệu và nghiên cứu thực nghiệm.

8.2.7 `scipy.sparse` - Ma trận Thưa

Trong nhiều bài toán thực tế (mạng xã hội, phương trình đạo hàm riêng, đồ thị), ma trận có kích thước lớn nhưng phần lớn phần tử bằng 0. Module này cung cấp các cấu trúc dữ liệu và thuật toán tối ưu cho ma trận thưa, tiết kiệm bộ nhớ và tăng tốc độ tính toán.

8.2.8 `scipy.fft` - Biến đổi Fourier Nhanh

Biến đổi Fourier là công cụ chuyển đổi tín hiệu từ miền thời gian sang miền tần số. Module này cung cấp các thuật toán FFT hiệu quả cao, hỗ trợ nhiều loại biến đổi (DFT, DCT, DST) và tối ưu cho các kiến trúc phần cứng khác nhau.

8.3 Tích phân với SciPy

Tích phân là một trong những phép toán cơ bản nhất trong giải tích, có ứng dụng rộng rãi từ tính diện tích, thể tích, đến tính công, năng lượng, xác suất và nhiều đại lượng vật lý khác. Trong khi tích phân giải tích đòi hỏi kỹ năng toán học cao và không phải lúc nào cũng tìm được nghiệm dạng đóng, tích phân số (numerical integration) cung cấp phương pháp xấp xỉ giá trị tích phân với độ chính xác kiểm soát được.

8.3.1 Cơ sở Lý thuyết Tích phân Số

Ý tưởng cơ bản của tích phân số là chia khoảng tích phân thành nhiều đoạn nhỏ, xấp xỉ diện tích dưới đường cong trên mỗi đoạn bằng các hình đơn giản (hình chữ nhật, hình thang, parabol), rồi cộng tổng các diện tích này lại. Độ chính xác phụ thuộc vào số lượng đoạn chia và phương pháp xấp xỉ được sử dụng.

Các phương pháp tích phân số phổ biến bao gồm:

- **Phương pháp hình chữ nhật:** Xấp xỉ đơn giản nhất, sử dụng hình chữ nhật có chiều cao bằng giá trị hàm tại một điểm trong đoạn.
- **Phương pháp hình thang** (Trapezoidal rule): Nối các điểm liên tiếp bằng đoạn thẳng, diện tích là tổng các hình thang.
- **Phương pháp Simpson:** Sử dụng parabol bậc hai để xấp xỉ hàm trên mỗi cặp đoạn, cho độ chính xác cao hơn.
- **Phương pháp Gauss:** Lựa chọn thông minh các điểm lấy mẫu và trọng số để đạt độ chính xác cao nhất với số điểm cho trước.

SciPy tích hợp các thuật toán adaptive (thích nghi), tự động điều chỉnh bước chia để đạt độ chính xác mong muốn, đặc biệt hiệu quả với các hàm có dao động mạnh hoặc điểm kỳ dị.

8.3.2 Tích phân Một biến với quad

Hàm `integrate.quad` là công cụ chính để tính tích phân xác định một biến. Tên "quad" xuất phát từ "quadrature", thuật ngữ cổ điển chỉ phép tính tích phân số.

```

from scipy import integrate
import numpy as np

# Ví dụ 1: Tích phân cơ bản
# Tính integral từ 0 đến pi của sin(x) dx = 2
def f1(x):
    return np.sin(x)

result, error = integrate.quad(f1, 0, np.pi)
print(f"Integral[0,pi] sin(x) dx = {result:.10f}")
print(f"Sai số ước lượng: {error:.2e}")

# Ví dụ 2: Hàm với tham số
# Tính integral từ 0 đến vô cùng của e^(-ax) dx = 1/a với a
# = 2
def f2(x, a):
    return np.exp(-a * x)

result, error = integrate.quad(f2, 0, np.inf, args=(2,))
print(f"Integral[0,inf] e^(-2x) dx = {result:.6f}")
print(f"Lý thuyết: 1/2 = {0.5}")

# Ví dụ 3: Hàm phức tạp (phân phôi chuẩn)
# Tích phân của hàm mật độ xác suất chuẩn
def gaussian(x, mu, sigma):
    return (1 / (sigma * np.sqrt(2 * np.pi))) * \
        np.exp(-0.5 * ((x - mu) / sigma)**2)

# Xác suất P(mu-sigma <= X <= mu+sigma) ~= 0.6827
mu, sigma = 0, 1
prob, err = integrate.quad(gaussian, mu - sigma, mu + sigma,
                            args=(mu, sigma))
print(f"P(mu-sigma <= X <= mu+sigma) = {prob:.4f} (~=
    68.27%)")

```

Hàm quad trả về hai giá trị: kết quả tích phân và ước lượng sai số tuyệt đối. Việc có sai số ước lượng giúp đánh giá độ tin cậy của kết quả, đặc biệt quan trọng trong các tính toán khoa học nghiêm ngặt.

8.3.3 Tích phân Bộ

Trong nhiều bài toán thực tế, ta cần tính tích phân trên miền hai chiều hoặc ba chiều. SciPy cung cấp các hàm dblquad (tích phân kép) và tplquad (tích phân bộ ba) cho mục đích này.

```

# Ví dụ 4: Tích phân kép - Tính thể tích
# V = double integral f(x,y) dA trên miền D
# Tính thể tích dưới mặt  $z = x^2 + y^2$  trên hình chữ nhật
→ [0,1]×[0,1]

def f_surface(y, x):
    # Lưu ý: thứ tự tham số (y, x) do thứ tự tích phân
    return x**2 + y**2

# Cận tích phân: x từ 0 đến 1, y từ 0 đến 1
volume, error = integrate.dblquad(f_surface, 0, 1, # cận
→ của x
                                lambda x: 0,           # cận
                                → dưới của y
                                lambda x: 1)          # cận
                                → trên của y
print(f"Thể tích dưới mặt  $z = x^2 + y^2$ : {volume:.6f}")

# Ví dụ 5: Tích phân với cận biến thiên
# Tính diện tích hình tròn đơn vị bằng tích phân kép
# A = double integral 1 dA trên  $x^2 + y^2 \leq 1$ 

def integrand(y, x):
    return 1

# Với mỗi x, y chạy từ  $-\sqrt{1-x^2}$  đến  $\sqrt{1-x^2}$ 
area, err = integrate.dblquad(
    integrand,
    -1, 1, # x từ -1 đến 1
    lambda x: -np.sqrt(1 - x**2), # cận dưới y
    lambda x: np.sqrt(1 - x**2)   # cận trên y
)
print(f"Diện tích hình tròn đơn vị: {area:.6f}")
print(f"So với pi = {np.pi:.6f}")

```

Cần lưu ý rằng trong dblquad, tích phân ngoài cùng (theo biến đầu tiên trong danh sách tham số) được thực hiện sau, và các cận của tích phân trong có thể phụ thuộc vào biến của tích phân ngoài. Điều này cho phép mô tả các miền tích phân phức tạp.

8.3.4 Ứng dụng: Tính Năng lượng và Công

Trong vật lý và kỹ thuật, tích phân thường được sử dụng để tính các đại lượng như công, năng lượng, moment, và khối lượng.

```

# Ví dụ 6: Tính công của lực biến thiên
# Một lò xo có độ cứng k = 100 N/m được nén từ vị trí x=0
→ đến x=0.2m
# Lực: F(x) = kx, Công: W = integral F(x) dx

k = 100 # N/m
x_max = 0.2 # m

def spring_force(x):
    return k * x

work, err = integrate.quad(spring_force, 0, x_max)
print(f"Công nén lò xo: {work:.2f} J")
print(f"Lý thuyết: ½kx² = {0.5 * k * x_max**2:.2f} J")

# Ví dụ 7: Khối lượng của vật thể có mật độ biến thiên
# Thanh dài L = 2m có mật độ tuyến tính rho(x) = rho0*(1 +
→ x^2) kg/m
# Khối lượng: M = integral rho(x) dx

rho_0 = 5 # kg/m
L = 2 # m

def density(x):
    return rho_0 * (1 + x**2)

mass, err = integrate.quad(density, 0, L)
print(f"Khối lượng thanh: {mass:.3f} kg")

```

8.4 Giải Phương trình Vi phân với SciPy

Phương trình vi phân (Differential Equations) là công cụ toán học mô tả sự biến thiên của các đại lượng theo thời gian hoặc không gian. Hầu hết các định luật vật lý, từ cơ học Newton, phương trình Maxwell, đến phương trình Schrödinger, đều được biểu diễn dưới dạng phương trình vi phân. Trong kỹ thuật, phương trình vi phân được sử dụng để mô hình hóa mạch điện, hệ thống điều khiển, động lực học chất lỏng, và vô số ứng dụng khác.

8.4.1 Phương trình Vi phân Thường (ODE)

Phương trình vi phân thường (Ordinary Differential Equation - ODE) là phương trình chứa hàm số một biến và các đạo hàm của nó. Dạng tổng quát của ODE bậc nhất là:

$$\frac{dy}{dt} = f(t, y)$$

với điều kiện ban đầu $y(t_0) = y_0$. Bài toán tìm hàm $y(t)$ thỏa mãn phương trình và điều kiện ban đầu được gọi là bài toán giá trị ban đầu (Initial Value Problem - IVP).

8.4.2 Phương pháp Số Giải ODE

Không giống như tích phân, phần lớn các phương trình vi phân không có nghiệm giải tích dạng đóng. Phương pháp số là cách tiếp cận chính để giải quyết các phương trình này. Các phương pháp phổ biến bao gồm:

- **Phương pháp Euler:** Đơn giản nhất, sử dụng xấp xỉ bằng đạo hàm hữu hạn, nhưng độ chính xác thấp.
- **Phương pháp Runge-Kutta:** Gia đình các phương pháp chính xác hơn, trong đó RK45 (Runge-Kutta bậc 4-5) là phổ biến nhất, cân bằng giữa độ chính xác và hiệu suất.
- **Phương pháp Predictor-Corrector:** Kết hợp dự đoán và hiệu chỉnh để cải thiện độ ổn định.
- **Phương pháp Implicit:** Sử dụng cho các hệ "stiff" (cứng), khi có sự chênh lệch lớn về tốc độ biến thiên.

8.4.3 Sử dụng solve_ivp

Hàm `integrate.solve_ivp` (solve initial value problem) là công cụ hiện đại và linh hoạt nhất trong SciPy để giải ODE. Nó tự động lựa chọn thuật toán phù hợp và điều chỉnh bước thời gian để đảm bảo độ chính xác.

```
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# Ví dụ 8: Phân rã phóng xạ
# dN/dt = -lambda*N, với N(0) = N0
# Nghiệm giải tích: N(t) = N0 * e^(-lambda*t)

lambda_decay = 0.5 # hằng số phân rã
N0 = 100 # số nguyên tử ban đầu

def decay_rate(t, N):
    return -lambda_decay * N

# Giải trên khoảng thời gian [0, 10]
t_span = (0, 10)
t_eval = np.linspace(0, 10, 100) # Các điểm muôn tính
→ nghiệm

sol = solve_ivp(decay_rate, t_span, [N0], t_eval=t_eval)
```

```

# So sánh với nghiệm giải tích
N_exact = N0 * np.exp(-lambda_decay * sol.t)

plt.figure(figsize=(10, 6))
plt.plot(sol.t, sol.y[0], 'b-', label='Numerical solution')
plt.plot(sol.t, N_exact, 'r--', label='Exact solution')
plt.xlabel('Time (s)')
plt.ylabel('N(t)')
plt.title('Radioactive Decay: dN/dt = -lambda*N')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('../Hinh_ve/ch9_decay.png', dpi=300,
           bbox_inches='tight')
plt.show()

print(f"Sai số tối đa: {np.max(np.abs(sol.y[0] -
                                      N_exact)):.6e}")

```

8.4.4 Hệ Phương trình Vi phân

Nhiều hệ thống thực tế được mô tả bởi hệ nhiều phương trình vi phân tương tác với nhau. Ví dụ, chuyển động của con lắc kép, động lực học quần thể sinh học, hoặc mạch điện RLC đều yêu cầu giải hệ ODE.

```

# Ví dụ 9: Con lắc đơn
# Phương trình: theta'' + (g/L)sin(theta) = 0
# Chuyển về hệ bậc nhất: theta' = omega, omega' =
#   - (g/L)sin(theta)

g = 9.81    # m/s2
L = 1.0      # m

def pendulum(t, y):
    theta, omega = y
    dtheta_dt = omega
    domega_dt = -(g / L) * np.sin(theta)
    return [dtheta_dt, domega_dt]

# Điều kiện ban đầu: theta(0) = 30 độ, omega(0) = 0
theta0 = np.radians(30)
omega0 = 0
y0 = [theta0, omega0]

t_span = (0, 10)

```

```
t_eval = np.linspace(0, 10, 500)

sol = solve_ivp(pendulum, t_span, y0, t_eval=t_eval,
                 method='RK45')

# Vẽ biểu đồ
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Góc theo thời gian
ax1.plot(sol.t, np.degrees(sol.y[0]), 'b-', linewidth=2)
ax1.set_ylabel('Góc theta (độ)')
ax1.set_title('Đạo động Con lắc Đơn')
ax1.grid(True, alpha=0.3)

# Không gian pha (theta, omega)
ax2.plot(np.degrees(sol.y[0]), sol.y[1], 'r-', linewidth=2)
ax2.set_xlabel('Goc theta (do)')
ax2.set_ylabel('Van toc goc omega (rad/s)')
ax2.set_title('Khong gian Pha')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('../Hinh_ve/ch9_pendulum.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

8.4.5 Ứng dụng: Mô hình Lotka-Volterra (Săn mồi - Con mồi)

Mô hình Lotka-Volterra mô tả động lực học của hai quần thể tương tác: con mồi (prey) và kẻ săn mồi (predator). Đây là ví dụ kinh điển của hệ phi tuyến cho thấy hành vi phức tạp từ các quy tắc đơn giản.

```
# Ví dụ 10: Mô hình Lotka-Volterra
# dx/dt = alpha*x - beta*x*y (con mồi)
# dy/dt = delta*x*y - gamma*y (săn mồi)

def lotka_volterra(t, z, alpha, beta, delta, gamma):
    x, y = z # x: con mồi, y: săn mồi
    dx_dt = alpha * x - beta * x * y
    dy_dt = delta * x * y - gamma * y
    return [dx_dt, dy_dt]

# Tham số
alpha = 1.0 # Tốc độ sinh con mồi
```

```

beta = 0.1      # Tốc độ săn mồi tiêu diệt con mồi
delta = 0.075 # Hiệu suất chuyển đổi con mồi → săn mồi
gamma = 1.5     # Tốc độ chết tự nhiên của săn mồi

# Điều kiện ban đầu
x0, y0 = 10, 5 # Số lượng ban đầu
y0_state = [x0, y0]

t_span = (0, 50)
t_eval = np.linspace(0, 50, 1000)

sol = solve_ivp(lotka_volterra, t_span, y0_state,
                 t_eval=t_eval,
                 args=(alpha, beta, delta, gamma),
                 method='RK45')

# Visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Biến thiên theo thời gian
ax1.plot(sol.t, sol.y[0], 'b-', label='Con mồi (x)',
          linewidth=2)
ax1.plot(sol.t, sol.y[1], 'r-', label='Săn mồi (y)',
          linewidth=2)
ax1.set_xlabel('Thời gian')
ax1.set_ylabel('Số lượng')
ax1.set_title('Động lực học Quần thể')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Không gian pha
ax2.plot(sol.y[0], sol.y[1], 'g-', linewidth=2)
ax2.plot(x0, y0, 'ko', markersize=10, label='Điểm ban đầu')
ax2.set_xlabel('Con mồi (x)')
ax2.set_ylabel('Săn mồi (y)')
ax2.set_title('Không gian Pha')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('../Hinh_ve/ch9_lotka_volterra.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

Biểu đồ không gian pha cho thấy quỹ đạo chu kỳ, phản ánh sự dao động định kỳ của hai quần thể: khi con mồi nhiều, săn mồi phát triển, dẫn đến giảm con mồi, rồi săn mồi cũng giảm theo, cho phép con mồi hồi phục. Chu trình này lặp lại mãi mãi trong mô hình lý tưởng.

8.5 Tìm Nghiệm Phương trình với SciPy

Tìm nghiệm của phương trình là một trong những bài toán cơ bản và quan trọng nhất trong toán học ứng dụng. Khác với phương trình đại số đơn giản có công thức nghiệm tổng quát, phần lớn các phương trình phi tuyến trong thực tế không thể giải bằng công thức đóng. Các phương pháp số cung cấp cách tiếp cận hiệu quả để tìm nghiệm xấp xỉ với độ chính xác mong muốn.

8.5.1 Cơ sở Lý thuyết

Bài toán tìm nghiệm phương trình $f(x) = 0$ được gọi là bài toán tìm điểm không (root-finding problem). Điểm x^* được gọi là nghiệm (hoặc điểm không, root) của phương trình nếu $f(x^*) = 0$. Về mặt hình học, nghiệm là giao điểm giữa đồ thị hàm số $y = f(x)$ và trục hoành.

Các phương pháp số tìm nghiệm có thể phân thành hai loại chính:

- **Phương pháp ngoặc (Bracketing methods):** Yêu cầu hai điểm ban đầu a và b sao cho $f(a) \cdot f(b) < 0$ (hàm đổi dấu), đảm bảo tồn tại ít nhất một nghiệm trong khoảng $[a, b]$. Phương pháp chia đôi (bisection) là đại diện tiêu biểu.
- **Phương pháp lặp mở (Open methods):** Chỉ cần một hoặc vài điểm xuất phát, sử dụng thông tin về đạo hàm để hội tụ nhanh hơn. Phương pháp Newton-Raphson là ví dụ điển hình.

8.5.2 Phương pháp Chia đôi (Bisection Method)

Phương pháp chia đôi là thuật toán đơn giản và tin cậy nhất để tìm nghiệm. Ý tưởng dựa trên định lý giá trị trung gian: nếu hàm liên tục $f(x)$ đổi dấu trên $[a, b]$, tức $f(a) \cdot f(b) < 0$, thì tồn tại ít nhất một nghiệm trong khoảng đó.

Thuật toán hoạt động như sau:

1. Tính điểm giữa $c = \frac{a+b}{2}$
2. Nếu $f(c) = 0$ hoặc $|b - a|$ đủ nhỏ: dừng, trả về c
3. Nếu $f(a) \cdot f(c) < 0$: nghiệm nằm trong $[a, c]$, đặt $b = c$
4. Ngược lại: nghiệm nằm trong $[c, b]$, đặt $a = c$
5. Quay lại bước 1

Mỗi bước lặp làm giảm khoảng chứa nghiệm đi một nửa, nên sau n bước, sai số tối đa là $\frac{b-a}{2^n}$. Điều này đảm bảo hội tụ chắc chắn, nhưng tốc độ hội tụ tuyến tính (linear convergence) tương đối chậm.

```

from scipy.optimize import bisect
import numpy as np
import matplotlib.pyplot as plt

# Ví dụ 11: Tìm nghiệm bằng phương pháp chia đôi
# Giải phương trình:  $x^3 - 2x - 5 = 0$ 

def f(x):
    return x**3 - 2*x - 5

# Vẽ đồ thị để xác định khoảng chứa nghiệm
x = np.linspace(0, 3, 1000)
y = f(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'b-', linewidth=2, label='f(x) =  $x^3 - 2x - 5$ ')
plt.axhline(y=0, color='k', linestyle='--', linewidth=0.5)
plt.grid(True, alpha=0.3)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Đồ thị Hàm số để Xác định Khoảng Chủ Nghiem')
plt.legend()
plt.savefig('../Hinh_ve/ch9_bisection_plot.png', dpi=300,
           bbox_inches='tight')
plt.show()

# Quan sát đồ thị:  $f(2) = 8 - 4 - 5 = -1 < 0$ ,  $f(3) = 27 - 6 - 5 = 16 > 0$ 
# Nghiệm nằm trong khoảng [2, 3]

root = bisect(f, 2, 3, xtol=1e-10)
print(f"Nghiệm tìm được: x = {root:.10f}")
print(f"Kiểm tra: f({root:.10f}) = {f(root):.2e}")

# So sánh với nghiệm chính xác
# Nghiệm chính xác: x ~ 2.0945514815
exact_root = 2.0945514815
print(f"Sai số: {abs(root - exact_root):.2e}")

```

Phương pháp chia đôi có ưu điểm là đơn giản, tin cậy và luôn hội tụ nếu điều kiện ban đầu thỏa mãn. Tuy nhiên, tốc độ hội tụ chậm là nhược điểm lớn khi cần độ chính xác cao hoặc giải nhiều phương trình.

8.5.3 Phương pháp Newton-Raphson và `fsolve`

Phương pháp Newton-Raphson (còn gọi là phương pháp tiếp tuyến) là một trong những thuật toán mạnh mẽ nhất để tìm nghiệm phương trình phi tuyến. Khác với bisection chỉ sử dụng giá trị hàm, Newton-Raphson khai thác thông tin đạo hàm để hội tụ nhanh hơn nhiều.

Xuất phát từ điểm dự đoán x_n , phương pháp xấp xỉ hàm bằng tiếp tuyến tại điểm đó:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n)$$

Nghiệm của tiếp tuyến (cho $f(x) = 0$) cho điểm dự đoán mới:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Phương pháp này có tốc độ hội tụ bậc hai (quadratic convergence): nếu x_n gần nghiệm, sai số ở bước tiếp theo tỷ lệ với bình phương sai số hiện tại: $|x_{n+1} - x^*| \propto |x_n - x^*|^2$. Điều này có nghĩa số chữ số chính xác tăng gấp đôi sau mỗi bước lặp khi gần nghiệm.

Tuy nhiên, phương pháp Newton có nhược điểm:

- Cần tính đạo hàm $f'(x)$, không phải lúc nào cũng dễ dàng hoặc khả thi
- Có thể không hội tụ nếu điểm xuất phát xa nghiệm
- Có thể gặp vấn đề khi $f'(x) \approx 0$ (đạo hàm gần không)

Hàm `optimize.fsolve` của SciPy sử dụng thuật toán hybr (một biến thể cải tiến của Powell's hybrid method), kết hợp ưu điểm của Newton-Raphson với các kỹ thuật tránh phân kỳ. Nếu không cung cấp đạo hàm, `fsolve` tự động tính xấp xỉ bằng sai phân hữu hạn.

```
from scipy.optimize import fsolve, newton

# Ví dụ 12: Sử dụng fsolve để tìm nghiệm
# Giải phương trình: cos(x) = x

def equation(x):
    return np.cos(x) - x

# Phương pháp 1: fsolve (không cần đạo hàm)
root_fsolve = fsolve(equation, 0.5) # 0.5 là điểm xuất phát
print(f"Nghiệm (fsolve): x = {root_fsolve[0]:.10f}")

# Phương pháp 2: newton (cần đạo hàm, hội tụ nhanh hơn)
def derivative(x):
```

```

    return -np.sin(x) - 1

root_newton = newton(equation, 0.5, fprime=derivative)
print(f"Nghiệm (Newton): x = {root_newton:.10f}")

# Visualization
x = np.linspace(-1, 2, 1000)
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x, np.cos(x), 'b-', linewidth=2, label='y = cos(x)')
ax.plot(x, x, 'r-', linewidth=2, label='y = x')
ax.plot(root_fsolve[0], np.cos(root_fsolve[0]), 'go',
         markersize=12, label=f'Nghiệm: x ~=
                           {root_fsolve[0]:.4f}')
ax.axhline(y=0, color='k', linestyle='--', linewidth=0.5,
            alpha=0.3)
ax.axvline(x=0, color='k', linestyle='--', linewidth=0.5,
            alpha=0.3)
ax.grid(True, alpha=0.3)
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_title('Giao điểm cos(x) = x', fontsize=14,
             fontweight='bold')
ax.legend(fontsize=11)
plt.savefig('../Hinh_ve/ch9_fsolve.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

8.5.4 Giải Hệ Phương trình Phi tuyến

Trong thực tế, ta thường gặp hệ nhiều phương trình phi tuyến đồng thời. Ví dụ, trong kỹ thuật điện, phân tích mạch phi tuyến yêu cầu giải hệ phương trình Kirchhoff phi tuyến. Trong hóa học, tính toán cân bằng phản ứng cũng dẫn đến hệ phương trình phi tuyến.

Hàm `fsolve` có thể xử lý hệ phương trình bằng cách mở rộng phương pháp Newton lên không gian nhiều chiều. Thay vì đạo hàm, ta sử dụng ma trận Jacobian chứa các đạo hàm riêng.

```

# Ví dụ 13: Giải hệ phương trình phi tuyến
# x^2 + y^2 = 4
# x - y^2 = 0

def system(vars):
    x, y = vars
    eq1 = x**2 + y**2 - 4 # x^2 + y^2 - 4 = 0

```

```

eq2 = x - y**2           # x - y2 = 0
return [eq1, eq2]

# Giải với nhiều điểm xuất phát khác nhau để tìm các nghiệm
# khác nhau
initial_guesses = [(1, 1), (-1, 1), (2, -1)]

print("Các nghiệm của hệ phương trình:")
solutions = []
for guess in initial_guesses:
    sol = fsolve(system, guess)
    # Kiểm tra tính hợp lệ
    residual = system(sol)
    if np.allclose(residual, 0, atol=1e-6):
        # Kiểm tra trùng lặp
        is_duplicate = False
        for existing_sol in solutions:
            if np.allclose(sol, existing_sol, atol=1e-6):
                is_duplicate = True
                break
        if not is_duplicate:
            solutions.append(sol)
            print(f"  x = {sol[0]:.6f}, y = {sol[1]:.6f}")
            print(f"    Kiểm tra: x2 + y2 = {sol[0]**2 +
                  sol[1]**2:.6f}")
            print(f"    x - y2 = {sol[0] -
                  sol[1]**2:.6f}")

# Visualization
theta = np.linspace(0, 2*np.pi, 1000)
circle_x = 2 * np.cos(theta)
circle_y = 2 * np.sin(theta)

y_parabola = np.linspace(-2.5, 2.5, 1000)
x_parabola = y_parabola**2

fig, ax = plt.subplots(figsize=(10, 10))
ax.plot(circle_x, circle_y, 'b-', linewidth=2, label='x2 +
         y2 = 4')
ax.plot(x_parabola, y_parabola, 'r-', linewidth=2, label='x =
         y2')

for sol in solutions:
    ax.plot(sol[0], sol[1], 'go', markersize=15)

```

```

ax.annotate(f' ({sol[0]:.2f}, {sol[1]:.2f})',
            xy=(sol[0], sol[1]), xytext=(sol[0]+0.3,
            ↳ sol[1]+0.3),
            fontsize=11, fontweight='bold')

ax.set_xlim(-3, 5)
ax.set_ylim(-3, 3)
ax.set_aspect('equal')
ax.grid(True, alpha=0.3)
ax.axhline(y=0, color='k', linewidth=0.5)
ax.axvline(x=0, color='k', linewidth=0.5)
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_title('Nghiệm của Hệ Phương trình Phi tuyến',
            fontsize=14, fontweight='bold')
ax.legend(fontsize=12)
plt.savefig('../Hinh_ve/ch9_system.png', dpi=300,
            ↳ bbox_inches='tight')
plt.show()

```

8.5.5 Ứng dụng: Tính Điểm Cân bằng trong Phản ứng Hóa học

Trong hóa học, việc tính nồng độ các chất ở trạng thái cân bằng thường dẫn đến hệ phương trình phi tuyến từ hằng số cân bằng và định luật bảo toàn khối lượng.

```

# Ví dụ 14: Cân bằng phản ứng: N2 + 3H2 <=> 2NH3
# Cho N2 = 1 mol, H2 = 3 mol ban đầu
# Hằng số cân bằng K_c = 0.5 tại nhiệt độ nhất định

def equilibrium(x):
    """
    x: số mol NH3 tạo thành
    Tại cân bằng:
    - N2: 1 - x/2
    - H2: 3 - 3x/2
    - NH3: x
    K_c = [NH3]^2 / ([N2][H2]^3)
    """
    K_c = 0.5
    n_N2 = 1 - x / 2
    n_H2 = 3 - 3 * x / 2
    n_NH3 = x

    # Điều kiện vật lý: tất cả nồng độ phải dương

```

```

if n_N2 <= 0 or n_H2 <= 0 or n_NH3 <= 0:
    return 1e10 # Trả về giá trị lớn nếu không hợp lý

# Phương trình cân bằng (giả sử thể tích V = 1 L)
return (n_NH3**2) / (n_N2 * n_H2**3) - K_c

# Tìm nghiệm
x_equilibrium = fsolve(equilibrium, 0.5)[0] # Bắt đầu từ
→ 0.5 mol

n_N2_eq = 1 - x_equilibrium / 2
n_H2_eq = 3 - 3 * x_equilibrium / 2
n_NH3_eq = x_equilibrium

print("Trạng thái cân bằng:")
print(f" N2: {n_N2_eq:.4f} mol")
print(f" H2: {n_H2_eq:.4f} mol")
print(f" NH3: {n_NH3_eq:.4f} mol")
print(f"Kiểm tra K_c: {(n_NH3_eq**2) / (n_N2_eq *
→ n_H2_eq**3):.4f}")

```

8.6 Nội suy Dữ liệu với SciPy

Nội suy (interpolation) là kỹ thuật ước lượng giá trị của hàm số tại các điểm trung gian dựa trên tập hợp các điểm dữ liệu đã biết. Trong thực tế khoa học và kỹ thuật, dữ liệu thường được đo đạc hoặc tính toán tại các điểm rời rạc, nhưng ta cần giá trị tại các điểm khác. Ví dụ: nhiệt độ được đo mỗi giờ nhưng cần biết nhiệt độ vào lúc 10:30, hoặc tính toán số liệu chỉ có tại 10 điểm nhưng cần vẽ đường cong mượt.

8.6.1 Cơ sở Lý thuyết Nội suy

Cho tập điểm dữ liệu (x_i, y_i) với $i = 0, 1, \dots, n$, ta muốn tìm hàm $f(x)$ sao cho $f(x_i) = y_i$ với mọi i , và có thể tính $f(x)$ tại bất kỳ x nào trong khoảng $[x_0, x_n]$. Hàm $f(x)$ được gọi là hàm nội suy.

Các yêu cầu đối với hàm nội suy:

- **Chính xác:** Phải đi qua tất cả các điểm dữ liệu đã cho
- **Mịn:** Đường cong nên liên tục và có đạo hàm liên tục (tránh gãy khúc)
- **Ổn định:** Không dao động mạnh giữa các điểm dữ liệu
- **Hiệu quả:** Tính toán nhanh, đặc biệt khi có nhiều điểm cần nội suy

8.6.2 Nội suy Tuyến tính

Phương pháp đơn giản nhất là nối các điểm liên tiếp bằng đoạn thẳng. Trong mỗi đoạn $[x_i, x_{i+1}]$, hàm nội suy là:

$$f(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$

Nội suy tuyến tính có ưu điểm đơn giản, nhanh, và ổn định, nhưng hạn chế là đường cong không trơn (có góc tại các điểm dữ liệu), không phù hợp khi cần đạo hàm.

```
from scipy.interpolate import interp1d
import numpy as np
import matplotlib.pyplot as plt

# Ví dụ 15: Nội suy tuyến tính
# Dữ liệu nhiệt độ đo được trong ngày
time_hours = np.array([0, 6, 12, 18, 24]) # giờ
temperature = np.array([15, 12, 25, 22, 16]) # °C

# Tạo hàm nội suy tuyến tính
f_linear = interp1d(time_hours, temperature, kind='linear')

# Tính nhiệt độ tại các thời điểm chi tiết
time_detailed = np.linspace(0, 24, 1000)
temp_interpolated = f_linear(time_detailed)

# Visualization
plt.figure(figsize=(12, 6))
plt.plot(time_hours, temperature, 'ro', markersize=10,
         label='Dữ liệu đo đặc', zorder=3)
plt.plot(time_detailed, temp_interpolated, 'b-',
         linewidth=2,
         label='Nội suy tuyến tính')
plt.grid(True, alpha=0.3)
plt.xlabel('Thời gian (giờ)', fontsize=12)
plt.ylabel('Nhiệt độ (°C)', fontsize=12)
plt.title('Nội suy Tuyến tính Dữ liệu Nhiệt độ',
          fontsize=14, fontweight='bold')
plt.legend(fontsize=11)
plt.xlim(0, 24)
plt.savefig('../Hinh_ve/ch9_linear_interp.png', dpi=300,
           bbox_inches='tight')
plt.show()
```

```
# Tính nhiệt độ tại thời điểm cụ thể
time_query = 10.5
temp_at_query = f_linear(time_query)
print(f"Nhiệt độ ước lượng lúc {time_query}h:
      {temp_at_query:.2f} °C")
```

8.6.3 Nội suy Đa thức và Spline

Nội suy đa thức sử dụng một đa thức bậc n đi qua $n + 1$ điểm. Tuy nhiên, khi số điểm nhiều, đa thức bậc cao dễ gây ra hiện tượng Runge (dao động lớn ở biên), làm hàm nội suy không ổn định.

Giải pháp tốt hơn là sử dụng **spline** (đường cong trơn từng khúc). Spline là đường cong ghép nối từ nhiều đa thức bậc thấp (thường bậc 3 - cubic spline), mỗi đa thức áp dụng cho một đoạn, với điều kiện liên tục của hàm và các đạo hàm tại điểm nối. Cubic spline đảm bảo hàm, đạo hàm bậc nhất và bậc hai liên tục, cho đường cong mượt mà và tự nhiên.

```
from scipy.interpolate import interp1d, CubicSpline

# Ví dụ 16: So sánh các phương pháp nội suy
# Hàm Runge:  $f(x) = 1/(1 + 25x^2)$  - ví dụ kinh điển về vấn đề
# của nội suy đa thức

x_data = np.linspace(-1, 1, 9) # 9 điểm dữ liệu
y_data = 1 / (1 + 25 * x_data**2)

# Tạo các hàm nội suy
f_linear = interp1d(x_data, y_data, kind='linear')
f_quadratic = interp1d(x_data, y_data, kind='quadratic')
f_cubic = interp1d(x_data, y_data, kind='cubic')
f_spline = CubicSpline(x_data, y_data)

# Điểm chi tiết để vẽ
x_plot = np.linspace(-1, 1, 1000)
y_exact = 1 / (1 + 25 * x_plot**2)

# So sánh
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

methods = [
    ('Linear', f_linear, axes[0, 0]),
    ('Quadratic', f_quadratic, axes[0, 1]),
    ('Cubic', f_cubic, axes[1, 0]),
    ('Cubic Spline', f_spline, axes[1, 1])]
```

```

]

for name, f_interp, ax in methods:
    y_interp = f_interp(x_plot)

    ax.plot(x_plot, y_exact, 'g--', linewidth=2, label='Hàm
        ↪ gốc', alpha=0.7)
    ax.plot(x_plot, y_interp, 'b-', linewidth=2, label=f'Nội
        ↪ suy {name}')
    ax.plot(x_data, y_data, 'ro', markersize=8, label='Điểm
        ↪ dữ liệu', zorder=3)
    ax.set_title(f'{name} Interpolation', fontsize=13,
        ↪ fontweight='bold')
    ax.set_xlabel('x', fontsize=11)
    ax.set_ylabel('y', fontsize=11)
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.3)

    # Tính sai số trung bình
    error = np.mean(np.abs(y_interp - y_exact))
    ax.text(0.05, 0.95, f'Sai số TB: {error:.4f}',
            transform=ax.transAxes, verticalalignment='top',
            bbox=dict(boxstyle='round', facecolor='wheat',
            ↪ alpha=0.5),
            fontsize=10)

plt.tight_layout()
plt.savefig('../Hinh_ve/ch9_interp_comparison.png', dpi=300,
    ↪ bbox_inches='tight')
plt.show()

```

Từ ví dụ trên, ta thấy cubic spline cho kết quả tốt nhất: đường cong mịn và sai số thấp. Đây là phương pháp nội suy phổ biến nhất trong thực tế kỹ thuật và khoa học.

8.6.4 Nội suy Hai chiều

Khi dữ liệu phụ thuộc hai biến độc lập, chẳng hạn nhiệt độ theo tọa độ địa lý (kinh độ, vĩ độ) hoặc độ cao địa hình trên bản đồ, ta cần nội suy hai chiều.

```

from scipy.interpolate import interp2d, RectBivariateSpline

# Ví dụ 17: Nội suy 2D – Bản đồ nhiệt độ
# Dữ liệu nhiệt độ tại các điểm lưới

```

```

x_grid = np.array([0, 1, 2, 3, 4]) # Kinh độ
y_grid = np.array([0, 1, 2, 3]) # Vĩ độ

# Ma trận nhiệt độ (độ C)
temperature_data = np.array([
    [20, 21, 22, 23, 24],
    [21, 22, 24, 25, 26],
    [22, 24, 26, 27, 28],
    [21, 23, 25, 26, 27]
])

# Tạo hàm nội suy 2D
f_2d = RectBivariateSpline(y_grid, x_grid, temperature_data)

# Lưới chi tiết để vẽ contour map
x_fine = np.linspace(0, 4, 100)
y_fine = np.linspace(0, 3, 100)
X_fine, Y_fine = np.meshgrid(x_fine, y_fine)
z_fine = f_2d(y_fine, x_fine)

# Visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Dữ liệu gốc
X_grid, Y_grid = np.meshgrid(x_grid, y_grid)
c1 = ax1.contourf(X_grid, Y_grid, temperature_data,
                   levels=15, cmap='RdYlBu_r')
ax1.plot(X_grid, Y_grid, 'ko', markersize=8)
ax1.set_title('Dữ liệu Nhiệt độ Gốc (5x4 điểm)',
              fontsize=13, fontweight='bold')
ax1.set_xlabel('Kinh độ', fontsize=11)
ax1.set_ylabel('Vĩ độ', fontsize=11)
plt.colorbar(c1, ax=ax1, label='Nhiệt độ (°C)')

# Sau nội suy
c2 = ax2.contourf(X_fine, Y_fine, z_fine, levels=50,
                   cmap='RdYlBu_r')
ax2.plot(X_grid, Y_grid, 'ko', markersize=6, label='Điểm')
ax2.set_title('Bản đồ Nhiệt độ sau Nội suy (100x100 điểm)',
              fontsize=13, fontweight='bold')
ax2.set_xlabel('Kinh độ', fontsize=11)
ax2.set_ylabel('Vĩ độ', fontsize=11)
ax2.legend(fontsize=10)

```

```

plt.colorbar(c2, ax=ax2, label='Nhiệt độ (°C)')

plt.tight_layout()
plt.savefig('../Hinh_ve/ch9_2d_interp.png', dpi=300,
           bbox_inches='tight')
plt.show()

# Tính nhiệt độ tại điểm cụ thể
lon, lat = 2.5, 1.5
temp_at_point = f_2d(lat, lon)[0, 0]
print(f"Nhiệt độ ước lượng tại ({lon}, {lat}):
      {temp_at_point:.2f} °C")

```

8.6.5 Úng dụng: Xử lý Dữ liệu Thiếu

Trong phân tích dữ liệu thực tế, thường gặp tình huống dữ liệu bị thiếu do lỗi cảm biến, mất kết nối, hoặc chi phí đo đạc. Nội suy là phương pháp phổ biến để điền (impute) các giá trị thiếu.

```

# Ví dụ 18: Xử lý dữ liệu cảm biến bị thiếu
# Dữ liệu áp suất theo thời gian, một số điểm bị mất

np.random.seed(42)
time_full = np.arange(0, 24, 0.5) # Đo mỗi 30 phút
pressure_full = 1013 + 5 * np.sin(2 * np.pi * time_full /
                                  24) + \
                np.random.normal(0, 0.5, len(time_full))

# Giả lập mất dữ liệu (30% điểm)
missing_indices = np.random.choice(len(time_full),
                                      size=int(0.3 *
                                               len(time_full)),
                                      replace=False)
time_measured = np.delete(time_full, missing_indices)
pressure_measured = np.delete(pressure_full,
                               missing_indices)

# Nội suy để khôi phục
f_recover = CubicSpline(time_measured, pressure_measured)
pressure_recovered = f_recover(time_full)

# Visualization
plt.figure(figsize=(14, 6))
plt.plot(time_full, pressure_full, 'g-', linewidth=2,

```

```

label='Dữ liệu đầy đủ (ground truth)', alpha=0.5)
plt.plot(time_measured, pressure_measured, 'ro',
         markersize=6,
         label=f'Dữ liệu đo được ({len(time_measured)} điểm)', zorder=3)
plt.plot(time_full, pressure_recovered, 'b--', linewidth=2,
         label='Dữ liệu sau khôi phục (nội suy)')
plt.xlabel('Thời gian (giờ)', fontsize=12)
plt.ylabel('Áp suất (hPa)', fontsize=12)
plt.title('Khôi phục Dữ liệu Thiếu bằng Nội suy',
          fontsize=14, fontweight='bold')
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.savefig('../Hinh_ve/ch9_missing_data.png', dpi=300,
            bbox_inches='tight')
plt.show()

# Đánh giá chất lượng khôi phục
mse = np.mean((pressure_recovered - pressure_full)**2)
print(f"Mean Squared Error: {mse:.4f} hPa²")
print(f"Root Mean Squared Error: {np.sqrt(mse):.4f} hPa")

```

Kết quả cho thấy cubic spline khôi phục dữ liệu với độ chính xác cao, sai số trung bình chỉ khoảng 0.5-1 hPa so với dữ liệu thực. Phương pháp này đặc biệt hữu ích trong xử lý dữ liệu chuỗi thời gian từ các cảm biến IoT, trạm khí tượng, hoặc thiết bị y tế.

8.7 Bài tập Thực hành

Bài tập 1: Tích phân Ứng dụng Tính diện tích hình phẳng giới hạn bởi các đường:

- $y = x^2$
- $y = 2x + 3$

Yêu cầu:

1. Tìm tọa độ giao điểm bằng fsolve
2. Tính diện tích bằng tích phân $\int_{x_1}^{x_2} (f_{\text{trên}}(x) - f_{\text{dưới}}(x))dx$
3. Vẽ đồ thị minh họa và tô màu vùng diện tích

Bài tập 2: Giải ODE - Mạch RC Mạch điện RC được mô tả bởi phương trình:

$$RC \frac{dV_C}{dt} + V_C = V_{\text{in}}(t)$$

Với $R = 1000 \Omega$, $C = 10^{-6} F$, và điện áp đầu vào là sóng vuông có biên độ 5V, chu kỳ 10ms.

1. Viết hàm mô tả ODE
2. Giải phương trình với $V_C(0) = 0$ trên khoảng 50ms
3. Vẽ đồ thị $V_{in}(t)$ và $V_C(t)$ trên cùng một figure
4. Nhận xét về quá trình nạp và xả của tụ điện

Bài tập 3: Tìm Nghiệm - Phương trình Kepler Phương trình Kepler trong cơ học thiên thể:

$$M = E - e \sin(E)$$

trong đó M là góc lệch tâm trung bình (mean anomaly), E là góc lệch tâm (eccentric anomaly), và e là độ lệch tâm quỹ đạo.

Cho $M = \pi/4$ và $e = 0.3$:

1. Giải phương trình để tìm E bằng phương pháp chia đôi trong khoảng $[0, 2\pi]$
2. Giải bằng `fsoolve` với điểm xuất phát $E_0 = M$
3. So sánh tốc độ hội tụ và số bước lặp của hai phương pháp

Bài tập 4: Nội suy - Dữ liệu Nhiệt độ Cho dữ liệu nhiệt độ trung bình tháng tại Hà Nội ($^{\circ}\text{C}$):

Tháng	1	2	3	4	5	6	7	8	9	10	11	12
Nhiệt độ	17	18	21	25	29	30	30	29	28	25	22	18

1. Nội suy tuyến tính, cubic, và cubic spline
2. Tính nhiệt độ ước lượng cho ngày 15 của mỗi tháng (tháng $i =$ ngày thứ $30i + 15$ trong năm)
3. Vẽ và so sánh ba phương pháp trên cùng một đồ thị
4. Tính đạo hàm của nhiệt độ (tốc độ thay đổi nhiệt độ) bằng cubic spline

Bài tập 5: Hệ ODE - Con lắc Kép Con lắc kép được mô tả bởi hệ phương trình (giả sử $m_1 = m_2 = m$, $l_1 = l_2 = l$):

$$\ddot{\theta}_1 = \frac{-g(2m_1 + m_2) \sin \theta_1 - m_2 g \sin(\theta_1 - 2\theta_2)}{l_1(2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))} - \frac{2 \sin(\theta_1 - \theta_2) m_2 (\dot{\theta}_2^2 l_2 + \dot{\theta}_1^2 l_1 \cos(\theta_1 - \theta_2))}{l_1(2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}$$

$$\ddot{\theta}_2 = \text{(phương trình tương tự)}$$

1. Chuyển về hệ bậc nhất với 4 biến: $[\theta_1, \omega_1, \theta_2, \omega_2]$
2. Giải với điều kiện ban đầu: $\theta_1 = 90, \theta_2 = -90, \omega_1 = \omega_2 = 0$
3. Vẽ quỹ đạo của hai khối nặng trong không gian 2D
4. Vẽ biểu đồ năng lượng theo thời gian để kiểm tra bảo toàn năng lượng

Bài tập 6: Tích phân Kép - Khối lượng Vật thể Một tảng kim loại hình tròn bán kính $R = 0.1$ m có mật độ khối lượng phụ thuộc vào khoảng cách đến tâm:

$$\rho(r) = \rho_0(1 + \alpha r^2)$$

với $\rho_0 = 7850$ kg/m³ (thép), $\alpha = 10$ m⁻², độ dày đồng đều $h = 0.01$ m.

1. Thiết lập tích phân kép trong tọa độ cực: $M = \iint \rho(r)h dA$
2. Tính khối lượng bằng dblquad
3. Tính vị trí trọng tâm (x_c, y_c)
4. So sánh với trường hợp mật độ đều

Bài tập 7: Giải Hệ Phương trình - Thiết kế Cầu Trong phân tích kết cấu dàn đơn giản, ta có hệ phương trình cân bằng lực. Cho dàn 3 thanh với các góc $\theta_1 = 30^\circ$, $\theta_2 = 45^\circ$, $\theta_3 = 60^\circ$, chịu tải trọng đứng $P = 1000$ N:

$$\begin{aligned} T_1 \cos \theta_1 + T_2 \cos \theta_2 + T_3 \cos \theta_3 &= 0 \\ T_1 \sin \theta_1 + T_2 \sin \theta_2 + T_3 \sin \theta_3 &= P \\ l_1 T_1 + l_2 T_2 - l_3 T_3 &= 0 \end{aligned}$$

với $l_1 = l_2 = l_3 = 1$ m.

1. Giải hệ phương trình để tìm lực trong các thanh (T_1, T_2, T_3)
2. Xác định thanh nào chịu kéo (dương), thanh nào chịu nén (âm)
3. Khảo sát sự thay đổi lực khi P biến thiên từ 0 đến 2000 N

Bài tập 8: Nội suy 2D - Bản đồ Địa hình Cho độ cao (m) tại 9 điểm trên lưới (x, y) :

y \ x	0	50	100
0	100	120	140
50	110	150	170
100	105	130	145

1. Nội suy 2D để tạo lưới chi tiết 100×100 điểm
2. Vẽ bản đồ contour và bản đồ 3D
3. Tính gradient (độ dốc) tại mỗi điểm
4. Xác định điểm có độ dốc lớn nhất (nguy hiểm cho xây dựng)

Bài tập 9: Mô phỏng Động lực học - Hệ Lò xo - Khối lượng Hệ hai lò xo nối tiếp với hai khối lượng:

$$m_1 \ddot{x}_1 = -k_1 x_1 + k_2(x_2 - x_1) - c_1 \dot{x}_1$$

$$m_2 \ddot{x}_2 = -k_2(x_2 - x_1) - c_2 \dot{x}_2 + F(t)$$

Với $m_1 = m_2 = 1$ kg, $k_1 = k_2 = 100$ N/m, $c_1 = c_2 = 1$ N·s/m, và lực kích thích $F(t) = 10 \sin(5t)$ N.

1. Chuyển về hệ bậc nhất 4 phương trình
2. Giải với điều kiện ban đầu $x_1 = x_2 = \dot{x}_1 = \dot{x}_2 = 0$
3. Vẽ biến thiên $x_1(t)$ và $x_2(t)$
4. Phân tích phổ tần số (FFT) của dao động
5. Xác định tần số cộng hưởng

Bài tập 10: Tổng hợp (Nâng cao) Bài toán tối ưu hóa quỹ đạo: Một vật được phóng lên với vận tốc ban đầu v_0 tại góc θ so với phương ngang. Vật chịu lực cản không khí tỷ lệ với vận tốc: $F_d = -bv$.

Phương trình chuyển động:

$$\begin{aligned}\ddot{x} &= -b\dot{x} \\ \ddot{y} &= -g - b\dot{y}\end{aligned}$$

Cho $v_0 = 50$ m/s, $b = 0.1$ s $^{-1}$, $g = 9.81$ m/s 2 :

1. Giải hệ ODE để tìm quỹ đạo với góc phỏng θ bất kỳ
2. Sử dụng `fsoolve` để tìm góc θ sao cho tầm xa là 200 m
3. Vẽ quỹ đạo tối ưu
4. So sánh với trường hợp không có lực cản
5. Tính sai số nếu bỏ qua lực cản

8.8 Kết luận

Trong chương này, chúng ta đã khám phá những khả năng mạnh mẽ của thư viện SciPy trong việc giải quyết các bài toán tính toán khoa học. Từ việc tính tích phân số, giải phương trình vi phân, tìm nghiệm của phương trình phi tuyến, đến nội suy dữ liệu, SciPy cung cấp các công cụ chuyên nghiệp và đáng tin cậy cho mọi nhu cầu tính toán.

Những kiến thức cốt lõi đã được trình bày:

Tích phân Số: Chúng ta đã tìm hiểu cách sử dụng `integrate.quad` và `dblquad` để tính tích phân một biến và tích phân bội, áp dụng vào các bài toán vật lý như tính năng lượng, công, và khối lượng. Phương pháp adaptive quadrature của SciPy tự động điều chỉnh để đạt độ chính xác cao với hiệu suất tối ưu.

Phương trình Vi phân: Hàm `solve_ivp` cho phép giải các ODE phức tạp với nhiều lựa chọn thuật toán (RK45, RK23, BDF, LSODA). Chúng ta đã thấy ứng dụng trong mô hình hóa các hệ thống động học từ phân rã phóng xạ, con lắc đơn, đến mô hình sinh thái Lotka-Volterra. Việc hiểu cách chuyển ODE bậc cao về hệ bậc nhất là kỹ năng quan trọng.

Tìm Nghiệm: Phương pháp bisection cung cấp giải pháp tin cậy nhưng chậm, trong khi `fsoolve` dựa trên Newton-Raphson hội tụ nhanh hơn nhiều. Chúng ta đã học cách áp dụng cho cả phương trình đơn và hệ phương trình phi tuyến, với các ví dụ từ toán học thuần túy đến bài toán cân bằng hóa học.

Nội suy: Từ nội suy tuyến tính đơn giản đến cubic spline phức tạp, mỗi phương pháp có ưu nhược điểm riêng. Cubic spline nổi bật với đường cong mượt mà và ổn định, phù hợp cho đa số ứng dụng. Nội suy 2D mở rộng khả năng xử lý dữ liệu không gian như bản đồ địa hình và phân bố nhiệt độ.

Thông qua các ví dụ thực tế và bài tập đa dạng, chương này không chỉ dạy cách sử dụng các hàm SciPy, mà còn giúp hiểu bản chất toán học đằng sau, khả năng áp dụng vào các bài toán thực tế, và cách đánh giá, so sánh các phương pháp khác nhau.

SciPy là công cụ không thể thiếu trong hành trang của mọi kỹ sư, nhà khoa học dữ liệu, và nhà nghiên cứu. Sự kết hợp giữa NumPy (xử lý mảng), SciPy (tính toán khoa học), và Matplotlib (visualization) tạo nên một hệ sinh thái mạnh mẽ, miễn phí và mã nguồn mở, cạnh tranh với các phần mềm thương mại đắt tiền như MATLAB.

CHƯƠNG 9. Lời Giải Bài Tập

Phần này cung cấp lời giải chi tiết cho các bài tập trong từng chương của giáo trình. Sinh viên nên tự giải quyết bài tập trước khi tham khảo lời giải.

9.1 Chương 1: Biến, Toán tử, Kiểu dữ liệu

9.1.1 Lời giải Bài tập 1

```
# Khai bao bien thong tin ca nhan
ten = "Nguyen Van A"
tuoi = 20
chieu_cao = 1.75 # met
can_nang = 68     # kg

# Tinh BMI = can_nang / (chieu_cao^2)
bmi = can_nang / (chieu_cao ** 2)

# In ket qua
print(f"Thong tin ca nhan:")
print(f"  Ho ten: {ten}")
print(f"  Tuoi: {tuoi}")
print(f"  Chieu cao: {chieu_cao} m")
print(f"  Can nang: {can_nang} kg")
print(f"  BMI: {bmi:.2f}")

# Danh gia BMI
if bmi < 18.5:
    danh_gia = "Thieu can"
elif bmi < 25:
    danh_gia = "Binh thuong"
elif bmi < 30:
    danh_gia = "Thua can"
else:
    danh_gia = "Beo phi"

print(f"  Danh gia: {danh_gia}")
```

9.1.2 Lời giải Bài tập 2

```
import math

# Hinh chu nhat
chieu_dai = 10
chieu_rong = 5
```

```

dien_tich_hcn = chieu_dai * chieu_rong
chu_vi_hcn = 2 * (chieu_dai + chieu_rong)

print("Hình chu nhat:")
print(f" Chieu dai: {chieu_dai}")
print(f" Chieu rong: {chieu_rong}")
print(f" Dien tich: {dien_tich_hcn}")
print(f" Chu vi: {chu_vi_hcn}")

# Hình tròn
ban_kinh = 7

dien_tich_ht = math.pi * ban_kinh ** 2
chu_vi_ht = 2 * math.pi * ban_kinh

print(f"\nHình tròn:")
print(f" Ban kinh: {ban_kinh}")
print(f" Dien tich: {dien_tich_ht:.2f}")
print(f" Chu vi: {chu_vi_ht:.2f}")

```

9.1.3 Lời giải Bài tập 3

```

# Nhập số nguyên
n = int(input("Nhập một số nguyên: "))

# Kiểm tra số chẵn dương
la_so_chan = (n % 2 == 0)
la_so_duong = (n > 0)
la_so_chan_duong = la_so_chan and la_so_duong

# In kết quả
print(f"So {n}:")
print(f" La so chan: {la_so_chan}")
print(f" La so duong: {la_so_duong}")
print(f" La so chan duong: {la_so_chan_duong}")

# Kiểm tra chi tiết
if la_so_chan_duong:
    print(f" => {n} la so chan duong")
elif n > 0:
    print(f" => {n} la so le duong")
elif la_so_chan:
    print(f" => {n} la so chan khong duong")

```

```

else:
    print(f"  => {n} la so le khong duong")

```

9.1.4 Lời giải Bài tập 4

```

def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32

def celsius_to_kelvin(c):
    return c + 273.15

def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

def kelvin_to_celsius(k):
    return k - 273.15

# Nhập nhiệt độ Celsius
celsius = float(input("Nhập nhiệt độ (Celsius):"))

# Chuyển đổi
fahrenheit = celsius_to_fahrenheit(celsius)
kelvin = celsius_to_kelvin(celsius)

# In kết quả
print(f"\nChuyển đổi nhiệt độ:")
print(f"  {celsius:.2f} do C")
print(f"  {fahrenheit:.2f} do F")
print(f"  {kelvin:.2f} K")

# Ví dụ ngược lại
print(f"\nKiểm tra chuyển đổi ngược:")
c_from_f = fahrenheit_to_celsius(fahrenheit)
c_from_k = kelvin_to_celsius(kelvin)
print(f"  Từ {fahrenheit:.2f} do F -> {c_from_f:.2f} do C")
print(f"  Từ {kelvin:.2f} K -> {c_from_k:.2f} do C")

```

9.1.5 Lời giải Bài tập 5

```

# Nhập chuỗi
chuoi = input("Nhập một chuỗi: ")

# Thống kê cơ bản

```

```

so_ky_tu = len(chuoi)
so_chu_cai = sum(c.isalpha() for c in chuoi)
so_chu_so = sum(c.isdigit() for c in chuoi)
so_khoang_trang = chuoi.count(' ')

print(f"\nThong ke chuoi '{chuoi}':")
print(f" So ky tu: {so_ky_tu}")
print(f" So chu cai: {so_chu_cai}")
print(f" So chu so: {so_chu_so}")
print(f" So khoang trang: {so_khoang_trang}")

# Chuyen doi
print(f"\nChuyen doi:")
print(f" Chu hoa: {chuoi.upper()}")
print(f" Chu thuong: {chuoi.lower()}")
print(f" Chu hoa dau: {chuoi.title()}")

# Kiem tra palindrome (doc xuoi nguoc giuong nhau)
chuoi_sach = ''.join(c.lower() for c in chuoi if
    c.isalnum())
la_palindrome = chuoi_sach == chuoi_sach[::-1]
print(f"\n La palindrome: {la_palindrome}")

# Vi du
if la_palindrome:
    print(f" '{chuoi}' doc xuoi nguoc giuong nhau!")

```

9.2 Chương 2: Cấu trúc Dữ liệu

9.2.1 Lời giải Bài tập 1

```

# Tao tuple sinh vien
sinh_vien = ("SV001", "Nguyen Van A", 8.5)

# In toan bo tuple
print("Thong tin sinh vien:", sinh_vien)

# Truy cap tung phan tu
print(f"Ma SV: {sinh_vien[0]}")
print(f"Ten: {sinh_vien[1]}")
print(f"Diem TB: {sinh_vien[2]}")

# Unpacking tuple
ma_sv, ten, diem = sinh_vien

```

```
print(f"\nSinh viên {ten} (Ma: {ma_sv}) có điểm TB: {diem}")
```

9.2.2 Lời giải Bài tập 3

```
# Tao dictionary diem sinh vien
diem_sinh_vien = {
    "SV001": {"ten": "Nguyen Van A", "diem": [8, 7.5, 9]},
    "SV002": {"ten": "Tran Thi B", "diem": [7, 8.5, 8]},
    "SV003": {"ten": "Le Van C", "diem": [9, 9.5, 8.5]}
}

# Truy cap thong tin
print("Thong tin SV001:", diem_sinh_vien["SV001"])

# Them sinh vien moi
diem_sinh_vien["SV004"] = {"ten": "Pham Thi D", "diem": [8.5, 8, 9]}

# Tinh diem trung binh
for ma_sv, thong_tin in diem_sinh_vien.items():
    diem_tb = sum(thong_tin["diem"]) /
        len(thong_tin["diem"])
    print(f"{thong_tin['ten']} (Ma: {ma_sv}): Điểm TB =
        {diem_tb:.2f}")

# Xoa sinh vien
del diem_sinh_vien["SV004"]
print("\nSo sinh vien con lai:", len(diem_sinh_vien))
```

9.2.3 Lời giải Bài tập 5

```
# Hai danh sach mon hoc cua hai sinh vien
mon_hoc_A = {"Toan", "Ly", "Hoa", "Tin", "Anh"}
mon_hoc_B = {"Toan", "Van", "Hoa", "Su", "Anh"}

# Mon hoc chung (giao)
mon_chung = mon_hoc_A & mon_hoc_B
print("Mon hoc chung:", mon_chung)

# Mon hoc chi A hoc (hieu)
mon_chi_A = mon_hoc_A - mon_hoc_B
print("Mon chi A hoc:", mon_chi_A)
```

```

# Mon hoc chi B hoc
mon_chi_B = mon_hoc_B - mon_hoc_A
print("Mon chi B hoc:", mon_chi_B)

# Tat ca cac mon (hop)
tat_ca_mon = mon_hoc_A | mon_hoc_B
print("Tat ca cac mon:", tat_ca_mon)

# Mon khac nhau (symmetric difference)
mon_khac_nhau = mon_hoc_A ^ mon_hoc_B
print("Mon khac nhau:", mon_khac_nhau)

```

9.3 Chương 3: Nhập Xuất Dữ liệu

9.3.1 Lời giải Bài tập 2

```

# Nhập hai số thực
try:
    a = float(input("Nhập số thứ nhất: "))
    b = float(input("Nhập số thứ hai: "))

    # Tính toán
    tong = a + b
    hieu = a - b
    tich = a * b

    # In kết quả
    print(f"\nKết quả tính toán:")
    print(f"  {a} + {b} = {tong:.2f}")
    print(f"  {a} - {b} = {hieu:.2f}")
    print(f"  {a} * {b} = {tich:.2f}")

    # Kiểm tra chia cho 0
    if b != 0:
        thuong = a / b
        print(f"  {a} / {b} = {thuong:.2f}")
    else:
        print(f"  {a} / {b} = Không thể chia cho 0!")

except ValueError:
    print("Lỗi: Vui lòng nhập số hợp lệ!")

```

9.3.2 Lời giải Bài tập 5

```

# Doc file va tinh toan
def tinh_toan_tu_file(ten_file):
    try:
        with open(ten_file, 'r', encoding='utf-8') as f:
            # Doc tung dong va chuyen thanh so nguyen
            so_nguyen = []
            for dong in f:
                dong = dong.strip()
                if dong: # Bo qua dong trong
                    so_nguyen.append(int(dong))

    # Tinh toan
    if len(so_nguyen) > 0:
        tong = sum(so_nguyen)
        trung_binh = tong / len(so_nguyen)

        print(f"So luong so: {len(so_nguyen)}")
        print(f"Tong: {tong}")
        print(f"Trung binh: {trung_binh:.2f}")
        print(f"So lon nhat: {max(so_nguyen)}")
        print(f"So nho nhat: {min(so_nguyen)}")
    else:
        print("File rong!")

    except FileNotFoundError:
        print(f"Loi: Khong tim thay file '{ten_file}'")
    except ValueError:
        print("Loi: File chua du lieu khong hop le")

# Su dung
tinh_toan_tu_file('numbers.txt')

```

9.3.3 Lời giải Bài tập 8

```

def thong_ke_file(ten_file):
    try:
        with open(ten_file, 'r', encoding='utf-8') as f:
            noi_dung = f.read()
            dong = noi_dung.split('\n')

            # Dem so dong
            so_dong = len(dong)

```

```

# Dem so tu
tu = noi_dung.split()
so_tu = len(tu)

# Dem so ky tu (bao gom ca khoang trang)
so_ky_tu = len(noi_dung)

# Dem so ky tu (khong khoang trang)
so_ky_tu_khong_space = len(noi_dung.replace(' ', 
→ '' ).replace('\n', ''))

print(f"Thong ke file '{ten_file}':")
print(f" So dong: {so_dong}")
print(f" So tu: {so_tu}")
print(f" So ky tu (co khoang trang):"
→ {so_ky_tu})
print(f" So ky tu (khong khoang trang):"
→ {so_ky_tu_khong_space})

except FileNotFoundError:
    print(f"Loi: Khong tim thay file '{ten_file}'")

# Su dung
thong_ke_file('document.txt')

```

9.4 Chương 4: Điều kiện và Vòng lặp

9.4.1 Lời giải Bài tập 1

```

def la_so_nguyen_to(n):
    """Kiem tra n co phai so nguyen to hay khong"""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Kiem tra cac so le tu 3 den sqrt(n)
    import math
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False

```

```

        return True

# Test
n = int(input("Nhập số nguyên dương: "))
if la_so_nguyen_to(n):
    print(f"{n} là số nguyên tố")
else:
    print(f"{n} không phải số nguyên tố")

# In các số nguyên tố từ 1 đến 100
print("\nCác số nguyên tố từ 1 đến 100:")
so_nguyen_to = [i for i in range(2, 101) if
                la_so_nguyen_to(i)]
print(so_nguyen_to)
print(f"Tổng cộng có {len(so_nguyen_to)} số nguyên tố")

```

9.4.2 Lời giải Bài tập 2

```

def in_tam_giac_so(n):
    """In tam giac so voi n dong"""

    # Tam giac 1: Tang dan
    print("Tam giac 1:")
    for i in range(1, n + 1):
        for j in range(1, i + 1):
            print(j, end=" ")
    print()

    # Tam giac 2: Giam dan
    print("\nTam giac 2:")
    for i in range(n, 0, -1):
        for j in range(1, i + 1):
            print(j, end=" ")
    print()

    # Tam giac 3: Canh can
    print("\nTam giac 3:")
    for i in range(1, n + 1):
        # In khoang trang
        print(" " * (n - i), end="")
        # In so
        for j in range(1, i + 1):
            print(j, end=" ")
    print()

```

```
# Test
n = int(input("Nhập số dong: "))
in_tam_giac_so(n)
```

9.5 Chương 5: Hàm và Module

9.5.1 Lời giải Bài tập 1

```
def giai_thua(n):
    """Tinh n! (giai thua)"""
    if n < 0:
        return None
    if n <= 1:
        return 1
    return n * giai_thua(n - 1)

def to_hop(n, k):
    """Tinh to hop C(n, k) = n! / (k! * (n-k)!)"""
    if k < 0 or k > n:
        return 0
    return giai_thua(n) // (giai_thua(k) * giai_thua(n - k))

def fibonacci(n):
    """Tinh so Fibonacci thu n"""
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)

def fibonacci_iterative(n):
    """Tinh Fibonacci bang phuong phap lap (nhanh hon)"""
    if n <= 0:
        return 0
    if n == 1:
        return 1

    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    return b

# Test
```

```

print("Giai thua:")
for i in range(6):
    print(f" {i}! = {giai_thua(i)}")

print("\nTo hop C(5, k):")
for k in range(6):
    print(f" C(5, {k}) = {to_hop(5, k)}")

print("\nDay Fibonacci (10 so dau):")
for i in range(10):
    print(f" F({i}) = {fibonacci_iterative(i)}")

```

9.5.2 Lời giải Bài tập 2

```

def dem_tu(chuoi):
    """Dem so tu trong chuoi"""
    return len(chuoi.split())

def dao_nguoc_chuoi(chuoi):
    """Dao nguoc chuoi"""
    return chuoi[::-1]

def dao_nguoc_tu(chuoi):
    """Dao nguoc tung tu trong chuoi"""
    tu_list = chuoi.split()
    tu_dao = [tu[::-1] for tu in tu_list]
    return ' '.join(tu_dao)

def loai_bo_ky_tu_dac_biet(chuoi):
    """Chi giu lai chu cai, chu so va khoang trang"""
    ket_qua = ''
    for c in chuoi:
        if c.isalnum() or c.isspace():
            ket_qua += c
    return ket_qua

def chuan_hoa_chuoi(chuoi):
    """Loai bo khoang trang thua va chuan hoa"""
    return ' '.join(chuoi.split())

# Test
chuoi_test = " Hoc lap trinh Python rat thu vi! "

print(f"Chuoi goc: '{chuoi_test}'")

```

```

print(f"So tu: {dem_tu(chuoi_test)}")
print(f"Dao nguoc: '{dao_nguoc_chuoi(chuoi_test)}'")
print(f"Dao nguoc tu: '{dao_nguoc_tu(chuoi_test)}'")
print(f"Loai ky tu dac biet:
      '{loai_bo_ky_tu_dac_biet(chuoi_test)}'")
print(f"Chuan hoa: '{chuan_hoa_chuoi(chuoi_test)}'")

```

9.5.3 Lời giải Bài tập 3

```

def thong_ke_list(danh_sach):
    """Thong ke co ban cua list"""
    if not danh_sach:
        return None

    return {
        'min': min(danh_sach),
        'max': max(danh_sach),
        'sum': sum(danh_sach),
        'mean': sum(danh_sach) / len(danh_sach),
        'count': len(danh_sach)
    }

def loc_chan(danh_sach):
    """Loc cac so chan"""
    return [x for x in danh_sach if x % 2 == 0]

def loc_le(danh_sach):
    """Loc cac so le"""
    return [x for x in danh_sach if x % 2 != 0]

def loc_duong(danh_sach):
    """Loc cac so duong"""
    return [x for x in danh_sach if x > 0]

def sap_xep_tang(danh_sach):
    """Sap xep tang dan"""
    return sorted(danh_sach)

def sap_xep_giam(danh_sach):
    """Sap xep giam dan"""
    return sorted(danh_sach, reverse=True)

# Test
so_list = [5, -3, 8, 12, -7, 0, 15, 4, -1, 9]

```

```

print("Danh sach goc:", so_list)
print("\nThong ke:")
stats = thong_ke_list(so_list)
for key, value in stats.items():
    print(f" {key}: {value}")

print(f"\nSo chan: {loc_chan(so_list)}")
print(f"So le: {loc_le(so_list)}")
print(f"So duong: {loc_duong(so_list)}")
print(f"Sap xep tang: {sap_xep_tang(so_list)}")
print(f"Sap xep giam: {sap_xep_giam(so_list)}")

```

9.5.4 Lời giải Bài tập 4

```

from functools import reduce

# Danh sach sinh vien
sinh_vien = [
    {'ten': 'An', 'diem': 8.5},
    {'ten': 'Binh', 'diem': 6.0},
    {'ten': 'Cuong', 'diem': 7.5},
    {'ten': 'Dung', 'diem': 9.0},
    {'ten': 'Em', 'diem': 5.5}
]

# Map: Lay danh sach ten
ten_list = list(map(lambda sv: sv['ten'], sinh_vien))
print("Danh sach ten:", ten_list)

# Map: Tang diem them 0.5
diem_tang = list(map(lambda sv: {**sv, 'diem':
    min(sv['diem'] + 0.5, 10)},
    sinh_vien))
print("\nDiem sau khi tang:")
for sv in diem_tang:
    print(f" {sv['ten']}: {sv['diem']}")

# Filter: Loc sinh vien dat (diem >= 7)
sv_dat = list(filter(lambda sv: sv['diem'] >= 7, sinh_vien))
print("\nSinh vien dat (diem >= 7):")
for sv in sv_dat:
    print(f" {sv['ten']}: {sv['diem']}")

```

```

# Reduce: Tinh tong diem
tong_diem = reduce(lambda acc, sv: acc + sv['diem'],
                     sinh_vien, 0)
diem_tb = tong_diem / len(sinh_vien)
print(f"\nDiem trung binh lop: {diem_tb:.2f}")

# Ket hop: Danh sach diem cao nhat
diem_max = max(sinh_vien, key=lambda sv: sv['diem'])
print(f"\nSinh vien diem cao nhat: {diem_max['ten']} - "
      f"{diem_max['diem']}")
```

9.5.5 Lời giải Bài tập 5

File: *math_utils.py*

```

"""
Module chua cac ham toan hoc tien ich
"""

import math

def la_so_nguyen_to(n):
    """Kiem tra so nguyen to"""
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

def gcd(a, b):
    """Tinh uoc chung lon nhat"""
    while b:
        a, b = b, a % b
    return a

def lcm(a, b):
    """Tinh boi chung nho nhat"""
    return abs(a * b) // gcd(a, b)

def giao_phuong_trinh_bac_2(a, b, c):
    """Giao phuong trinh ax^2 + bx + c = 0"""
    if a == 0:
        if b == 0:
```

```

        return None # Vo nghiem hoac vo so nghiem
        return [-c / b] # Phuong trinh bac nhat

delta = b**2 - 4*a*c
if delta < 0:
    return [] # Vo nghiem thuc
elif delta == 0:
    return [-b / (2*a)] # Nghiem kep
else:
    sqrt_delta = math.sqrt(delta)
    x1 = (-b + sqrt_delta) / (2*a)
    x2 = (-b - sqrt_delta) / (2*a)
    return [x1, x2]

# Hang so
PI = math.pi
E = math.e

```

File: main.py (su dung module)

```

import math_utils

# Test cac ham
print("Kiem tra so nguyen to:")
for n in [7, 10, 17, 20]:
    print(f" {n}: {math_utils.la_so_nguyen_to(n)}")

print("\nUCLN va BCNN:")
a, b = 12, 18
print(f" GCD({a}, {b}) = {math_utils.gcd(a, b)}")
print(f" LCM({a}, {b}) = {math_utils.lcm(a, b)}")

print("\nGiai phuong trinh bac 2:")
# x^2 - 5x + 6 = 0
nghiem = math_utils.giai_phuong_trinh_bac_2(1, -5, 6)
print(f" x^2 - 5x + 6 = 0: {nghiem}")

# 2x^2 + 3x - 5 = 0
nghiem = math_utils.giai_phuong_trinh_bac_2(2, 3, -5)
print(f" 2x^2 + 3x - 5 = 0: {nghiem}")

print(f"\nHang so: PI = {math_utils.PI:.4f}, E =
→ {math_utils.E:.4f}")

```

9.6 Chương 6: Lập trình Hướng Đối tượng

9.6.1 Lời giải Bài tập 1

```

class SinhVien:
    """Class quan ly thong tin sinh vien"""

    def __init__(self, ma_sv, ho_ten, nam_sinh,
                 diem_list=None):
        self.ma_sv = ma_sv
        self.ho_ten = ho_ten
        self.nam_sinh = nam_sinh
        self.diem_list = diem_list if diem_list else []

    def them_diem(self, diem):
        """Them diem moi"""
        if 0 <= diem <= 10:
            self.diem_list.append(diem)
        else:
            print("Diem khong hop le!")

    def tinh_diem_tb(self):
        """Tinh diem trung binh"""
        if not self.diem_list:
            return 0
        return sum(self.diem_list) / len(self.diem_list)

    def xep_loai(self):
        """Xep loai hoc luc"""
        dtb = self.tinh_diem_tb()
        if dtb >= 9.0:
            return "Xuat sac"
        elif dtb >= 8.0:
            return "Gioi"
        elif dtb >= 7.0:
            return "Kha"
        elif dtb >= 5.0:
            return "Trung binh"
        else:
            return "Yeu"

    def __str__(self):
        """String representation"""
        dtb = self.tinh_diem_tb()
        xl = self.xep_loai()

```

```

        return f"SV {self.ma_sv} - {self.ho_ten} - DTB:
        ↳ {dtb:.2f} - {xl}"

# Test
sv1 = SinhVien("SV001", "Nguyen Van A", 2005)
sv1.them_diem(8.5)
sv1.them_diem(7.0)
sv1.them_diem(9.0)
print(sv1)

sv2 = SinhVien("SV002", "Tran Thi B", 2004, [9.5, 9.0, 8.5])
print(sv2)

```

9.6.2 Lời giải Bài tập 2

```

import math

class Hinh:
    """Class co so cho cac hinh hoc"""

    def dien_tich(self):
        raise NotImplementedError

    def chu_vi(self):
        raise NotImplementedError

    def __str__(self):
        return f"{self.__class__.__name__} - DT:
        ↳ {self.dien_tich():.2f}, CV: {self.chu_vi():.2f}"

class HinhTron(Hinh):
    def __init__(self, ban_kinh):
        self.ban_kinh = ban_kinh

    def dien_tich(self):
        return math.pi * self.ban_kinh ** 2

    def chu_vi(self):
        return 2 * math.pi * self.ban_kinh

class HinhChuNhat(Hinh):
    def __init__(self, chieu_dai, chieu_rong):
        self.chieu_dai = chieu_dai
        self.chieu_rong = chieu_rong

```

```

def dien_tich(self):
    return self.chieu_dai * self.chieu_rong

def chu_vi(self):
    return 2 * (self.chieu_dai + self.chieu_rong)

class TamGiac(Hinh):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def dien_tich(self):
        # Công thức Heron
        p = (self.a + self.b + self.c) / 2
        return math.sqrt(p * (p - self.a) * (p - self.b) *
                         (p - self.c))

    def chu_vi(self):
        return self.a + self.b + self.c

# Test
hinh_list = [
    HinhTron(5),
    HinhChuNhat(10, 6),
    TamGiac(3, 4, 5)
]

print("Danh sach cac hinh:")
for hinh in hinh_list:
    print(f" {hinh}")

# Tính tổng diện tích
tong_dt = sum(hinh.dien_tich() for hinh in hinh_list)
print(f"\nTổng diện tích: {tong_dt:.2f}")

```

9.6.3 Lời giải Bài tập 3

```

from datetime import datetime

class TaiKhoan:
    """Class quan ly tai khoan ngan hang"""

```

```

lai_suat_nam = 0.05 # Class variable: 5% per year

def __init__(self, so_tai_khoan, chu_tai_khoan,
             so_du_ban_dau=0):
    self.so_tai_khoan = so_tai_khoan
    self.chu_tai_khoan = chu_tai_khoan
    self.__so_du = so_du_ban_dau # Private attribute
    self.lich_su = []

def nap_tien(self, so_tien):
    """Nap tien vao tai khoan"""
    if so_tien > 0:
        self.__so_du += so_tien
        self._ghi_lich_su('Nap tien', so_tien)
        print(f"Nap {so_tien:.0f} VND thanh cong. So
              du: {self.__so_du:.0f}")
    else:
        print("So tien khong hop le!")

def rut_tien(self, so_tien):
    """Rut tien tu tai khoan"""
    if so_tien > 0:
        if so_tien <= self.__so_du:
            self.__so_du -= so_tien
            self._ghi_lich_su('Rut tien', -so_tien)
            print(f"Rut {so_tien:.0f} VND thanh cong.
                  So du: {self.__so_du:.0f}")
        else:
            print("So du khong du!")
    else:
        print("So tien khong hop le!")

def chuyen_tien(self, tai_khoan_nhan, so_tien):
    """Chuyen tien den tai khoan khac"""
    if so_tien > 0 and so_tien <= self.__so_du:
        self.__so_du -= so_tien
        tai_khoan_nhan.__so_du += so_tien
        self._ghi_lich_su('Chuyen tien', -so_tien)
        tai_khoan_nhan._ghi_lich_su('Nhan tien',
                                     so_tien)
        print(f"Chuyen {so_tien:.0f} VND thanh cong")
    else:
        print("Giao dich that bai!")

```

```

def xem_so_du(self):
    """Xem so du tai khoan"""
    return self.__so_du

def _ghi_lich_su(self, loai, so_tien):
    """Ghi lich su giao dich (protected method)"""
    giao_dich = {
        'thoi_gian': datetime.now().strftime("%Y-%m-%d
                                     ↪ %H:%M:%S"),
        'loai': loai,
        'so_tien': so_tien,
        'so_du': self.__so_du
    }
    self.lich_su.append(giao_dich)

def in_lich_su(self):
    """In lich su giao dich"""
    print(f"\nLich su giao dich -
          ↪ {self.chu_tai_khoan}:")
    for gd in self.lich_su:
        print(f"  {gd['thoi_gian']} | {gd['loai']}:
              ↪ {gd['so_tien']:+,.0f} | So du:
              ↪ {gd['so_du']:+,.0f}")

def __str__(self):
    return f"TK {self.so_tai_khoan} -
           ↪ {self.chu_tai_khoan} - So du:
           ↪ {self.__so_du:+,.0f} VND"

# Test
tk1 = TaiKhoan("001", "Nguyen Van A", 1000000)
tk2 = TaiKhoan("002", "Tran Thi B", 500000)

print(tk1)
print(tk2)

tk1.nap_tien(500000)
tk1.rut_tien(200000)
tk1.chuyen_tien(tk2, 300000)

print(f"\n{tk1}")
print(tk2)

tk1.in_lich_su()

```

9.7 Chương 7: NumPy trong Kỹ thuật

9.7.1 Lời giải Bài tập 1

```

import numpy as np

# Tao cac loai mang
a = np.array([1, 2, 3, 4, 5])
print("Mang 1D:", a)

# Mang 2D
b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("\nMang 2D:\n", b)

# Mang zeros, ones
zeros = np.zeros((3, 4))
ones = np.ones((2, 3))
print("\nZeros:\n", zeros)
print("Ones:\n", ones)

# Mang ngau nhien
random = np.random.randint(0, 100, size=(3, 3))
print("\nRandom:\n", random)

# Thao tac
print("\nHinh dang:", b.shape)
print("Kich thuoc:", b.size)
print("Kieu du lieu:", b.dtype)

# Indexing va slicing
print("\nPhan tu [1,2]:", b[1, 2])
print("Hang thu 2:\n", b[1, :])
print("Cot thu 3:\n", b[:, 2])

# Thong ke
print("\nTong:", np.sum(b))
print("Trung binh:", np.mean(b))
print("Max:", np.max(b))
print("Min:", np.min(b))

```

9.7.2 Lời giải Bài tập 2

```

import numpy as np

# Tao hai ma tran
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

print("Ma tran A:\n", A)
print("\nMa tran B:\n", B)

# Cong, tru
print("\nA + B:\n", A + B)
print("\nA - B:\n", A - B)

# Nhan phan tu (element-wise)
print("\nA * B (element-wise):\n", A * B)

# Nhan ma tran (matrix multiplication)
print("\nA @ B (matrix mult):\n", A @ B)

# Chuyen vi
print("\nChuyen vi A:\n", A.T)

# Dinh thuc (neu ma tran vuong)
if A.shape[0] == A.shape[1]:
    det_A = np.linalg.det(A)
    print(f"\nDet(A) = {det_A:.4f}")

# Ma tran nghich dao (neu ton tai)
try:
    A_inv = np.linalg.inv(A)
    print("\nMa tran nghich dao A:\n", A_inv)
    # Kiem tra
    print("\nKiem tra A * A_inv:\n", A @ A_inv)
except np.linalg.LinAlgError:
    print("\nMa tran khong kha nghich")

# Tri rieng va vector rieng
C = np.array([[4, -2], [1, 1]])
eigenvalues, eigenvectors = np.linalg.eig(C)
print("\nTri rieng:", eigenvalues)
print("Vector rieng:\n", eigenvectors)

```

9.8 Chương 8: Visualization

9.8.1 Lời giải Bài tập 1

```

import matplotlib.pyplot as plt
import numpy as np

# Du lieu
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Tao figure
plt.figure(figsize=(10, 6))

# Ve do thi
plt.plot(x, y1, 'b-', label='sin(x)', linewidth=2)
plt.plot(x, y2, 'r--', label='cos(x)', linewidth=2)

# Tuy chinh
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Ham sin va cos', fontsize=14, fontweight='bold')
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)

# Luu va hien thi
plt.savefig('bai_tap_plot.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

9.8.2 Lời giải Bài tập 2

```

import matplotlib.pyplot as plt
import numpy as np

# Tao du lieu ngau nhien phan phoi chuan
np.random.seed(42)
data1 = np.random.normal(100, 15, 1000) # mean=100, std=15
data2 = np.random.normal(120, 20, 1000) # mean=120, std=20

# Tao figure voi 2 subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Histogram

```

```

ax1.hist(data1, bins=30, alpha=0.7, color='blue',
         edgecolor='black',
         label='Group 1')
ax1.hist(data2, bins=30, alpha=0.7, color='red',
         edgecolor='black',
         label='Group 2')
ax1.set_xlabel('Gia tri', fontsize=12)
ax1.set_ylabel('Tần suất', fontsize=12)
ax1.set_title('Histogram', fontsize=14, fontweight='bold')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Density plot
ax2.hist(data1, bins=30, density=True, alpha=0.5,
         color='blue',
         label='Group 1')
ax2.hist(data2, bins=30, density=True, alpha=0.5,
         color='red',
         label='Group 2')
ax2.set_xlabel('Gia tri', fontsize=12)
ax2.set_ylabel('Mật độ', fontsize=12)
ax2.set_title('Distribution Plot', fontsize=14,
              fontweight='bold')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('bai_tap_histogram.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

9.9 Chương 9: SciPy

9.9.1 Lời giải Bài tập 1

```

import numpy as np
from scipy import integrate, optimize
import matplotlib.pyplot as plt

# Định nghĩa các hàm
def f1(x):
    return x**2

def f2(x):

```

```

    return 2*x + 3

# Tim giao diem
def diff(x):
    return f1(x) - f2(x)

# Su dung fsolve tim 2 nghiem
x_intersect = optimize.fsolve(diff, [-2, 4])
x1, x2 = sorted(x_intersect)
print(f"Giao diem: x1 = {x1:.4f}, x2 = {x2:.4f}")

# Tinh dien tich
# A = integral[x1, x2] (f2(x) - f1(x)) dx
area, error = integrate.quad(lambda x: f2(x) - f1(x), x1,
                                x2)
print(f"Dien tich: {area:.4f}")

# Ve do thi
x = np.linspace(x1 - 1, x2 + 1, 500)
y1 = f1(x)
y2 = f2(x)

plt.figure(figsize=(10, 8))
plt.plot(x, y1, 'b-', label='y = x^2', linewidth=2)
plt.plot(x, y2, 'r-', label='y = 2x + 3', linewidth=2)

# To mau vung dien tich
x_fill = np.linspace(x1, x2, 100)
y1_fill = f1(x_fill)
y2_fill = f2(x_fill)
plt.fill_between(x_fill, y1_fill, y2_fill, alpha=0.3,
                 color='green',
                 label=f'Dien tich = {area:.4f}')

# Danh dau giao diem
plt.plot([x1, x2], [f1(x1), f1(x2)], 'ko', markersize=8,
         label='Giao diem')

plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Tinh dien tich hinh phang', fontsize=14,
          fontweight='bold')
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)

```

```

plt.axhline(y=0, color='k', linewidth=0.5)
plt.axvline(x=0, color='k', linewidth=0.5)

plt.savefig('bai_tap_dien_tich.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

9.9.2 Lời giải Bài tập 4

```

import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

# Du lieu nhiet do trung binh thang
thang = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
nhiet_do = np.array([17, 18, 21, 25, 29, 30, 30, 29, 28, 25,
                     22, 18])

# Noi suy tuyen tinh
f_linear = interpolate.interp1d(thang, nhiet_do,
                                 kind='linear')

# Noi suy cubic
f_cubic = interpolate.interp1d(thang, nhiet_do,
                                kind='cubic')

# Noi suy cubic spline
f_spline = interpolate.CubicSpline(thang, nhiet_do)

# Tao luoi chi tiet
thang_fine = np.linspace(1, 12, 365)

# Tinh nhiet do noi suy
nhiet_do_linear = f_linear(thang_fine)
nhiet_do_cubic = f_cubic(thang_fine)
nhiet_do_spline = f_spline(thang_fine)

# Ve do thi
plt.figure(figsize=(14, 8))

plt.plot(thang, nhiet_do, 'ko', markersize=8, label='Du lieu
            thuc te')
plt.plot(thang_fine, nhiet_do_linear, 'b-', linewidth=2,
         label='Noi suy tuyen tinh')

```

```

plt.plot(thang_fine, nhiet_do_cubic, 'r--', linewidth=2,
          label='Noi suy cubic')
plt.plot(thang_fine, nhiet_do_spline, 'g-.', linewidth=2,
          label='Cubic spline')

plt.xlabel('Thang', fontsize=12)
plt.ylabel('Nhiệt độ (do C)', fontsize=12)
plt.title('Noi suy du lieu nhiet do Ha Noi',
           fontsize=14, fontweight='bold')
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.xticks(thang)

plt.savefig('bai_tap_noi_suy.png', dpi=300,
            bbox_inches='tight')
plt.show()

# Tinh nhiet do ngay 15 cua moi thang
print("\nNhiệt độ uoc luong ngay 15:")
for i, t in enumerate(thang):
    ngay = t + 0.5 # Giua thang (ngay 15)
    if ngay <= 12:
        temp_spline = f_spline(ngay)
        print(f" Thang {t}: {temp_spline:.2f} do C")

# Tinh dao ham (toc do thay doi nhiet do)
dao_ham = f_spline.derivative()
toc_do_thay_doi = dao_ham(thang)
print("\nToc do thay doi nhiet do (do C/thang):")
for i, t in enumerate(thang):
    print(f" Thang {t}: {toc_do_thay_doi[i]:.2f}")

```

9.10 Lưu ý chung

Khi giải các bài tập, sinh viên cần chú ý:

- **Hiểu đề bài:** Đọc kỹ yêu cầu và xác định input/output cần thiết
- **Phân tích bài toán:** Chia nhỏ bài toán thành các bước xử lý
- **Chọn cấu trúc dữ liệu phù hợp:** List, tuple, dict, set, array...
- **Xử lý ngoại lệ:** Sử dụng try-except để bắt lỗi
- **Kiểm tra kết quả:** Test với nhiều trường hợp khác nhau
- **Tối ưu code:** Sử dụng list comprehension, NumPy operations
- **Comment code:** Giải thích logic và thuật toán

- **Tuân thủ PEP 8:** Coding style chuẩn của Python

Sinh viên nên tự thực hiện các bài tập trước khi xem lời giải, sau đó so sánh và rút kinh nghiệm. Có thể có nhiều cách giải khác nhau cho cùng một bài toán, quan trọng là hiểu logic và áp dụng đúng kiến thức đã học.