Lập trình Python Cơ bản

Contents

Trường: Đại học Trà Vinh

Khoa: Kỹ thuật và Công nghệ

Bộ môn: Công nghệ thông tin

Tên tài liệu: **Lập trình Python Cơ bản**

Tác giả: Nguyễn Thiên Ân + ChatGPT

Mục lục

Contents

Chương 1. Giới thiệu về Python

Chương 2. Cấu trúc cơ bản của chương trình Python

Chương 3. Các kiểu dữ liệu cơ bản trong Python

Chương 4. Biến và hằng số

Chương 1. Giới thiệu về Python

Contents

1.1. Python là gì?

Python là một ngôn ngữ lập trình bậc cao, thông dịch, và hướng đối tượng, được thiết kế với mục tiêu dễ học, dễ đọc và dễ sử dụng. Python được Guido van Rossum phát triển lần đầu tiên vào năm 1991 và đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới. Python có cú pháp rõ ràng, dễ hiểu và hỗ trợ nhiều thư viện mạnh mẽ, giúp lập trình viên có thể nhanh chóng xây dựng các ứng dụng mà không phải lo lắng quá nhiều về chi tiết kỹ thuật. Python có thể được sử dụng cho nhiều mục đích khác nhau, từ phát triển phần mềm, khoa học dữ liệu, học máy (machine learning), trí tuệ nhân tạo (AI) đến các ứng dụng web và tự động hóa.

1.2. Lịch sử phát triển của Python

- **1980s:** Python được tạo ra bởi Guido van Rossum tại Trung tâm Nghiên cứu Khoa học và Máy tính Quốc gia (CWI) ở Hà Lan. Nguyên mẫu đầu tiên của Python được phát triển dựa trên ngôn ngữ lập trình ABC, một ngôn ngữ được thiết kế để dễ học và dễ sử dụng. Tuy nhiên, Python đã vượt qua những hạn chế của ABC và trở thành một ngôn ngữ phổ dụng hơn.

- **1991:** Python chính thức được phát hành lần đầu tiên với phiên bản 0.9.0, bao gồm các tính năng cơ bản như xử lý chuỗi, kiểu dữ liệu danh sách (list) và các câu lệnh điều kiện.
- **2000:** Phiên bản Python 2.0 được phát hành, và đây là giai đoạn Python bắt đầu trở nên phổ biến trong cộng đồng lập trình. Python 2.x tiếp tục được duy trì cho đến năm 2020.
- **2008:** Phiên bản Python 3.0 được phát hành, mang lại những cải tiến về cú pháp và hiệu suất, mặc dù không tương thích ngược với các phiên bản trước. Điều này gây ra sự phân chia trong cộng đồng, với nhiều người vẫn sử dụng Python 2 trong khi một số khác chuyển sang Python 3.
- **2020:** Python 2 chính thức hết hạn hỗ trợ, và Python 3 trở thành phiên bản chính thức và được duy trì. Python 3.x hiện tại đã trở thành phiên bản chính thức của ngôn ngữ này và có sự phát triển liên tục với nhiều cải tiến mạnh mẽ.

1.3. Sự khác biệt của ngôn ngữ Python với các ngôn ngữ khác

- **Cú pháp đơn giản và dễ đọc:** Cú pháp của Python rất rõ ràng và dễ đọc, gần gũi với ngôn ngữ tự nhiên, không yêu cầu dấu chấm phẩy (`;`) để kết thúc câu lệnh và không cần dấu ngoặc nhọn (`{}`) để phân biệt các khối mã (block of code). Thay vào đó, Python sử dụng thụt lề để xác định phạm vi (scope) của các khối mã.

```
if x > 10:
    print("x is greater than 10")
```

- **Kiểu dữ liệu động (Dynamic Typing):** Python là ngôn ngữ kiểu dữ liệu động (dynamically typed), tức là bạn không cần phải khai báo kiểu dữ liệu của biến trước khi sử dụng. Kiểu dữ liệu sẽ được gán tự động khi bạn gán giá trị cho biến.

```
x = 10  # x là kiểu int
x = "Hello"  # x thay đổi thành kiểu string
```

- **Quản lý bộ nhớ:** Python tự động quản lý bộ nhớ thông qua garbage collection (GC), nghĩa là bộ nhớ sẽ được giải phóng tự động khi các đối tượng không còn được tham chiếu.
- **Lập trình hướng đối tượng:** Python hỗ trợ lập trình hướng đối tượng (OOP) một cách linh hoạt và dễ dàng, nhưng không bắt buộc. Bạn có thể sử dụng các tính năng OOP (như kế thừa, đóng gói, và đa hình) mà không cần khai báo quá phức tạp.

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

  def bark(self):
      print(f"{self.name} barks!")
```

- **Quản lý thư viện và gói (Package Management):** Python có hệ thống quản lý gói rất mạnh mẽ và dễ sử dụng với `pip`, cho phép bạn dễ dàng cài đặt và quản lý thư viện bên ngoài. Python cũng có PyPI (Python Package Index), nơi lưu trữ hàng ngàn thư viện và công cụ sẵn có.
- **Tính tương thích đa nền tảng:** Python hỗ trợ đa nền tảng tuyệt vời, có thể chạy trên Windows, macOS, Linux mà không gặp phải nhiều vấn đề tương thích. Mã Python thường không cần thay đổi khi chuyển giữa các hệ điều hành.
- **Quản lý lỗi (Exception Handling):** Python sử dụng hệ thống xử lý ngoại lệ đơn giản và dễ hiểu. Các khối `try-except` giúp xử lý lỗi một cách mạch lạc.

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

- **Tốc độ thực thi:** Python là một ngôn ngữ diễn dịch (interpreted language), tức là mã nguồn sẽ được biên dịch và thực thi trực tiếp bởi trình thông dịch (interpreter) mà không cần biên dịch trước. Điều này khiến Python chậm hơn so với các ngôn ngữ biên dịch như C hay C++.
- **Sử dụng trong các lĩnh vực:** Python thường được sử dụng trong các lĩnh vực như khoa học dữ liệu, trí tuệ nhân tạo, tự động hóa, phát triển web (Django, Flask), và các ứng dụng học máy. Python nổi bật trong việc nhanh chóng phát triển các nguyên mẫu và các dự án nghiên cứu.

1.4. Tính ưu việt và ứng dụng của Python

Python có nhiều đặc điểm nổi bật giúp ngôn ngữ này được yêu thích trong nhiều lĩnh vực lập trình:

- **Cú pháp đơn giản, dễ học:** Python dễ tiếp cận đối với những người mới bắt đầu học lập trình, giúp họ nhanh chóng làm quen và phát triển các ứng dụng mà không cần phải học

cú pháp phức tạp.

- **Tính linh hoạt và đa năng:** Python có thể được sử dụng cho nhiều mục đích khác nhau, bao gồm phát triển phần mềm, tạo các ứng dụng web, phân tích dữ liệu, khoa học dữ liệu, học máy, trí tuệ nhân tạo, và tự động hóa.
- **Cộng đồng phát triển mạnh mẽ:** Python có một cộng đồng người dùng và lập trình viên đông đảo, giúp người dùng dễ dàng tìm kiếm sự hỗ trợ và học hỏi từ những người có kinh nghiệm.
- **Hỗ trợ thư viện phong phú:** Python có một kho thư viện đồ sộ, từ những thư viện hỗ trợ phát triển web như Django, Flask, đến các thư viện khoa học dữ liệu như NumPy, Pandas, Matplotlib.
- **Khả năng tương thích và tích hợp:** Python có khả năng tích hợp tốt với các ngôn ngữ lập trình khác và hỗ trợ nhiều nền tảng khác nhau. Bạn có thể sử dụng Python để giao tiếp với các hệ thống khác như C, C++, Java hoặc thậm chí tích hợp với các dịch vụ web.

Ứng dụng của Python rất rộng và bao gồm:

- **Phát triển ứng dụng web:** Với các framework như Django, Flask, Pyramid.
- **Khoa học dữ liệu và phân tích dữ liệu:** Python rất phổ biến trong việc xử lý, phân tích và trực quan hóa dữ liệu, với các thư viện như Pandas, Matplotlib, NumPy.
- **Học máy (Machine Learning) và Trí tuệ nhân tạo (AI):** Các thư viện như TensorFlow, Keras, và Scikit-learn giúp phát triển các mô hình học máy.
- **Tự động hóa:** Python có thể sử dụng để tự động hóa các tác vụ đơn giản hoặc phức tạp trên máy tính, chẳng hạn như xử lý tệp, gửi email tự động, v.v.
- **Phát triển phần mềm:** Python cũng có thể được dùng để phát triển các ứng dụng phần mềm với giao diện người dùng (GUI) bằng các thư viện như Tkinter hoặc PyQt.

1.5. Cài đặt Python trên các hệ điều hành

Để bắt đầu lập trình Python, bạn cần cài đặt Python trên máy tính của mình. Quá trình cài đặt khác nhau tùy vào hệ điều hành mà bạn đang sử dụng.

- **Cài đặt trên Windows:**
    1. Truy cập trang web chính thức của Python tại python.org.
    2. Tải phiên bản Python mới nhất dành cho Windows.

3. Chạy tệp cài đặt và chọn tùy chọn “Add Python to PATH” trước khi nhấn nút “Install Now”.
4. Sau khi cài đặt, mở Command Prompt và gõ `python` để kiểm tra xem Python đã được cài đặt thành công hay chưa.

- **Cài đặt trên macOS:**
  1. Python thường đã được cài sẵn trên macOS, nhưng bạn có thể tải bản mới nhất từ python.org.
  2. Chạy tệp .pkg để cài đặt Python.
  3. Kiểm tra cài đặt bằng cách mở Terminal và gõ `python3 --version`.
- **Cài đặt trên Linux:**
  1. Mở terminal và sử dụng trình quản lý gói của hệ điều hành để cài đặt Python. Ví dụ, trên Ubuntu, bạn có thể dùng lệnh:

     ```
     sudo apt update
     sudo apt install python3
     ```

  2. Kiểm tra cài đặt bằng lệnh:

     ```
     python3 --version
     ```

Hoặc có thể cài đặt đặt Python qua Anaconda như sau:

Để cài đặt Python thông qua **Anaconda**, bạn có thể làm theo các bước sau. Anaconda là một môi trường quản lý các thư viện và môi trường Python, giúp bạn quản lý các gói thư viện và môi trường ảo rất tiện lợi, đặc biệt trong các dự án khoa học dữ liệu, học máy, hoặc phân tích dữ liệu.

- **Cài đặt Anaconda trên Windows:**

1. Truy cập trang web Anaconda: Truy cập trang web chính thức của Anaconda tại https://www.anaconda.com.
2. Tải Anaconda: Chọn phiên bản Anaconda phù hợp với hệ điều hành của bạn (Windows), sau đó tải tệp cài đặt **Anaconda Distribution**.
3. Chạy tệp cài đặt:
   - Mở tệp tải về và chạy chương trình cài đặt.
   - Trong quá trình cài đặt, bạn có thể chọn các tùy chọn sau:

- **Add Anaconda to my PATH environment variable** (để dễ dàng sử dụng Anaconda từ Command Prompt).
- **Install for me only** (cài đặt cho người dùng hiện tại).
- Bạn cũng có thể chọn **Install Microsoft VSCode** nếu cần.

4. Hoàn tất cài đặt: Sau khi cài đặt hoàn tất, mở **Anaconda Navigator** (giao diện đồ họa) hoặc sử dụng **Anaconda Prompt** (dòng lệnh) để bắt đầu làm việc với Python.
5. Kiểm tra cài đặt: Mở **Command Prompt** và gõ lệnh sau để kiểm tra xem Anaconda và Python đã được cài đặt thành công:

```
conda --version
python --version
```

- **Cài đặt Anaconda trên macOS:**

1. Truy cập trang web Anaconda: Truy cập trang web chính thức của Anaconda tại https://www.anaconda.com.
2. Tải Anaconda: Chọn phiên bản Anaconda phù hợp với macOS, sau đó tải tệp cài đặt **Anaconda Distribution**.
3. Chạy tệp cài đặt: Mở tệp tải về (.pkg) và làm theo các bước hướng dẫn cài đặt để cài đặt Anaconda trên hệ điều hành macOS.
4. Hoàn tất cài đặt: Sau khi cài đặt xong, bạn có thể mở **Anaconda Navigator** hoặc sử dụng **Terminal** với các lệnh conda.
5. Kiểm tra cài đặt: Mở **Terminal** và gõ lệnh sau để kiểm tra:

```
conda --version
python --version
```

- **Cài đặt Anaconda trên Linux** (ví dụ với Ubuntu):

1. Truy cập trang web Anaconda: Truy cập trang web chính thức của Anaconda tại https://www.anaconda.com.
2. Tải Anaconda: Chọn phiên bản phù hợp cho Linux (tệp .sh), sau đó tải về.
3. Cài đặt Anaconda:
   - Mở **Terminal** và di chuyển đến thư mục chứa tệp cài đặt `.sh`.
   - Chạy lệnh sau để cài đặt:

```
bash Anaconda3-<version>-Linux-x86_64.sh
```

   - Làm theo các hướng dẫn cài đặt trong terminal.
4. Hoàn tất cài đặt: Sau khi cài đặt hoàn tất, bạn có thể sử dụng **Anaconda Navigator** hoặc sử dụng **Terminal** để làm việc với môi trường Python.
5. Kiểm tra cài đặt: Mở **Terminal** và kiểm tra cài đặt bằng lệnh:

```
conda --version
python --version
```

Lưu ý thêm:

- **Anaconda Navigator** là một ứng dụng GUI (giao diện đồ họa người dùng) giúp bạn dễ dàng quản lý môi trường, cài đặt các gói và thư viện.
- Bạn cũng có thể sử dụng **Anaconda Prompt** (hoặc Terminal trên macOS/Linux) để chạy các lệnh conda và quản lý các môi trường Python của mình.
- Khi sử dụng **Anaconda**, bạn có thể tạo các **môi trường ảo** riêng biệt cho từng dự án để tránh xung đột giữa các gói thư viện. Ví dụ:

```
conda create --name myenv python=3.9
conda activate myenv
```

1.6. Cài đặt và sử dụng môi trường phát triển (IDE)

Một môi trường phát triển tích hợp (IDE) là công cụ rất hữu ích giúp lập trình viên viết mã Python một cách dễ dàng và hiệu quả. Dưới đây là một số IDE phổ biến:

- **PyCharm:** PyCharm là một IDE mạnh mẽ được phát triển bởi JetBrains, rất phổ biến trong cộng đồng lập trình Python. Nó hỗ trợ các tính năng như gỡ lỗi, tự động hoàn thành mã, kiểm tra mã, và tích hợp với Git.
   - Cài đặt: Truy cập PyCharm để tải và cài đặt.
- **Visual Studio Code (VS Code):** VS Code là một trình soạn thảo mã nguồn miễn phí và phổ biến, hỗ trợ Python thông qua các tiện ích mở rộng. Đây là một lựa chọn nhẹ nhàng nhưng mạnh mẽ, phù hợp với lập trình viên Python.

    - Cài đặt: Truy cập Visual Studio Code và cài đặt Python extension.
- **Jupyter Notebook:** Jupyter Notebook là một công cụ lý tưởng cho việc phân tích dữ liệu và làm việc với mã Python trong các bài toán khoa học dữ liệu. Nó cho phép bạn viết mã và chạy thử ngay trong trình duyệt, dễ dàng kết hợp giữa mã nguồn và văn bản.
    - Cài đặt: Cài đặt thông qua Anaconda hoặc sử dụng pip:

```
pip install notebook
```

- **IDLE (Integrated Development and Learning Environment):** Đây là IDE cơ bản đi kèm với Python. IDLE nhẹ và dễ sử dụng, phù hợp cho những người mới bắt đầu học Python.
    - Cài đặt: IDLE đã được cài đặt sẵn cùng với Python.

Sau khi cài đặt một IDE, bạn có thể bắt đầu viết và chạy các chương trình Python đầu tiên của mình trong môi trường này. Mỗi IDE sẽ có những tính năng hỗ trợ khác nhau, do đó bạn có thể lựa chọn IDE phù hợp với nhu cầu và sở thích cá nhân.

Chương 2. Cấu trúc cơ bản của chương trình Python

Contents

Python là một ngôn ngữ lập trình dễ học và dễ sử dụng, với cú pháp đơn giản, giúp người mới bắt đầu dễ dàng tiếp cận. Mặc dù đơn giản, nhưng Python lại mạnh mẽ và linh hoạt, hỗ trợ nhiều loại ứng dụng khác nhau, từ phát triển web đến khoa học dữ liệu. Trong phần này, chúng ta sẽ tìm hiểu cấu trúc cơ bản của một chương trình Python, cách thức sử dụng câu lệnh in và nhập dữ liệu từ người dùng.

2.1. Cấu trúc một chương trình Python

Chương trình Python thường được xây dựng từ các dòng lệnh thực thi, bao gồm các phần cơ bản sau:

- **Câu lệnh khai báo (Import Statements)**: Python cho phép bạn sử dụng các thư viện và module ngoài, giúp mở rộng tính năng của chương trình. Ví dụ: `import math` để sử dụng các hàm toán học có sẵn.
- **Biến và kiểu dữ liệu (Variables and Data Types)**: Python hỗ trợ nhiều kiểu dữ liệu như số nguyên, số thực, chuỗi, danh sách, tuple, từ điển và nhiều hơn nữa. Các biến có thể được khai báo và gán giá trị một cách dễ dàng.
- **Câu lệnh điều kiện (Conditional Statements)**: Bạn có thể kiểm tra điều kiện trong chương trình với các câu lệnh `if`, `elif`, và `else`.

- **Vòng lặp (Loops)**: Python hỗ trợ vòng lặp `for` và `while` giúp bạn lặp lại các hành động.
- **Hàm (Functions)**: Để tổ chức mã nguồn và giảm thiểu sự lặp lại, bạn có thể định nghĩa các hàm trong Python.

Ví dụ về một chương trình Python cơ bản:

```
# Khai báo một biến
x = 10

# Kiểm tra điều kiện
if x > 5:
    print("x lớn hơn 5")
else:
    print("x nhỏ hơn hoặc bằng 5")
```

Chương trình này đơn giản, nhưng đã thể hiện đầy đủ cấu trúc cơ bản của một chương trình Python.

2.2. Câu lệnh in (print)

Câu lệnh `print` là một trong những câu lệnh cơ bản nhất trong Python, dùng để hiển thị thông tin lên màn hình.

Cú pháp: `print(<dữ liệu>)`

Hàm `print` có thể in ra bất kỳ kiểu dữ liệu nào (chuỗi, số, danh sách, v.v.), đồng thời có thể kết hợp nhiều đối tượng trong một câu lệnh. Chúng ta có thể sử dụng dấu phẩy (`,`) hoặc chuỗi định dạng để hiển thị các giá trị.

Ví dụ:

```
# In một chuỗi
print("Hello, World!")

# In nhiều đối tượng cùng lúc
name = "Alice"
age = 25
print("Name:", name, "Age:", age)

# Sử dụng f-string để định dạng chuỗi
print(f"Name: {name}, Age: {age}")
```

Câu lệnh `print` cũng có thể được sử dụng để in kết quả từ các phép toán hoặc kết quả từ hàm.

2.3. Câu lệnh nhập (input)

Câu lệnh `input` trong Python cho phép chương trình nhận dữ liệu từ người dùng. Khi sử dụng câu lệnh `input`, chương trình sẽ tạm dừng và chờ người dùng nhập dữ liệu từ bàn phím.

Cú pháp: `input(<chuỗi yêu cầu>)`

Câu lệnh `input` nhận dữ liệu dưới dạng chuỗi (`string`), và bạn có thể chuyển đổi dữ liệu đó thành các kiểu dữ liệu khác nếu cần, ví dụ như số nguyên hay số thực.

Ví dụ:

```
# Nhập tên người dùng
name = input("Enter your name: ")
print("Hello,", name)

# Nhập một số và chuyển đổi thành số nguyên
age = int(input("Enter your age: "))
print("Your age is", age)
```

Lưu ý rằng tất cả dữ liệu nhập vào từ người dùng qua câu lệnh `input` đều là kiểu chuỗi, nên nếu muốn làm việc với các kiểu dữ liệu khác (ví dụ: số học), bạn cần chuyển đổi chúng.

2.4. Bài tập thực hành

- **Bài tập với câu lệnh `print`:**

1. In ra dòng chữ “Hello, World!”.
2. In ra ba dòng riêng biệt với nội dung: “Python”, “is”, “fun”.
3. In ra dòng “Python is awesome!” bằng một câu lệnh print.
4. In ra kết quả phép cộng 5 + 3.
5. In ra thông báo “Tôi có 5 quả táo”.

- **Bài tập với câu lệnh `input`:**

1. Nhập tên người dùng và in ra “Chào bạn, [Tên người dùng]!”.
2. Nhập hai số và in ra tổng của chúng.
3. Nhập tuổi và in ra thông báo “Bạn [tuổi] tuổi”.
4. Nhập tên và in ra câu “Xin chào, [Tên người dùng]!”.
5. Nhập chiều cao và cân nặng, sau đó in ra thông tin đó.

Chương 3. Các kiểu dữ liệu cơ bản trong Python

Contents

Python hỗ trợ nhiều kiểu dữ liệu cơ bản khác nhau, mỗi kiểu dữ liệu phục vụ một mục đích cụ thể trong việc lưu trữ và xử lý thông tin. Việc hiểu rõ các kiểu dữ liệu này là nền tảng để lập trình hiệu quả trong Python. Trong phần này, chúng ta sẽ đi qua các kiểu dữ liệu cơ bản trong Python.

3.1. Kiểu số (`int`, `float`, `complex`)

Kiểu số nguyên (`int`):

- Kiểu `int` trong Python dùng để đại diện cho các số nguyên (không có phần thập phân).

- Ví dụ: `a = 10`, `b = -3`

Kiểu số thực (`float`):

- Kiểu `float` dùng để biểu diễn các số thực (có phần thập phân).
- Ví dụ: `x = 3.14`, `y = -0.001`

Kiểu số phức (`complex`):

- Kiểu `complex` trong Python đại diện cho số phức, có phần thực và phần ảo.
- Cú pháp: `a + bj`, trong đó `a` là phần thực và `b` là phần ảo.
- Ví dụ: `z = 2 + 3j`

Ví dụ sử dụng các kiểu số:

```
# Kiểu int
a = 10
# Kiểu float
b = 3.14
# Kiểu complex
c = 1 + 2j
print(type(a))  # <class 'int'>
print(type(b))  # <class 'float'>
print(type(c))  # <class 'complex'>
```

3.2. Kiểu chuỗi (`String`)

Kiểu chuỗi trong Python dùng để lưu trữ các dãy ký tự, có thể bao gồm chữ cái, số, dấu câu, và các ký tự đặc biệt.

Chuỗi có thể được tạo ra bằng dấu nháy đơn (`'`) hoặc dấu nháy kép (`"`).
Chuỗi có thể chứa các ký tự đặc biệt, chẳng hạn như khoảng trắng, dòng mới (`\n`), tab (`\t`), v.v.

Ví dụ:

```
# Kiểu string
s1 = "Hello, World!"
s2 = 'Python Programming'
print(type(s1))  # <class 'str'>
```

Các thao tác trên chuỗi:

- Kết hợp chuỗi: `"Hello" + " " + "World"`
- Lặp chuỗi: `"Python" * 3`
- Cắt chuỗi: `s1[0:5]`

3.3. Kiểu Boolean (`bool`)

Kiểu Boolean chỉ có hai giá trị: `True` và `False`. Kiểu này thường được sử dụng trong các biểu thức điều kiện, vòng lặp, và so sánh.

- `True`: Đại diện cho giá trị đúng.
- `False`: Đại diện cho giá trị sai.

Ví dụ:

```
# Kiểu bool
is_active = True
is_empty = False
print(type(is_active))  # <class 'bool'>
```

3.4. Kiểu danh sách (`List`)

Danh sách (`List`) trong Python là một kiểu dữ liệu cho phép lưu trữ một tập hợp các phần tử có thể thay đổi (mutable). Các phần tử trong danh sách có thể là bất kỳ kiểu dữ liệu nào, bao gồm các kiểu dữ liệu khác nhau.

Danh sách được khai báo trong dấu ngoặc vuông (`[]`).
Các phần tử trong danh sách có thể thay đổi, bổ sung hoặc xóa.

Ví dụ:

```
# Kiểu list
my_list = [1, 2, 3, 'Python', 3.14]
print(type(my_list))  # <class 'list'>
```

Các thao tác với danh sách:

- Truy xuất phần tử: `my_list[0]`
- Thêm phần tử: `my_list.append(5)`
- Sửa đổi phần tử: `my_list[1] = 10`

3.5. Kiểu `tuple`

`Tuple` là một kiểu dữ liệu giống như danh sách nhưng bất biến (immutable). Sau khi tạo ra, các phần tử trong tuple không thể thay đổi được.

`Tuple` được khai báo trong dấu ngoặc đơn (`()`).
Vì là bất biến, tuple thường được sử dụng để lưu trữ dữ liệu không cần thay đổi.

Ví dụ:

```
# Kiểu tuple
my_tuple = (1, 2, 3, 'Python', 3.14)
print(type(my_tuple))  # <class 'tuple'>
```

Các thao tác với `tuple`:

- Truy xuất phần tử: `my_tuple[0]`
- Tuple không thể thay đổi phần tử: `my_tuple[1] = 10` sẽ gây lỗi.

3.6. Kiểu từ điển (`Dictionary`)

`Dictionary` trong Python là một tập hợp các cặp khóa-giá trị (`key-value pairs`). Mỗi khóa (`key`) là duy nhất, và mỗi giá trị (`value`) có thể là bất kỳ kiểu dữ liệu nào.

`Dictionary` được khai báo trong dấu ngoặc nhọn (`{}`).
Các khóa trong `dictionary` phải là bất biến (immutable), ví dụ như chuỗi, số nguyên, hoặc `tuple`.

Ví dụ:

```
# Kiểu dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}
print(type(my_dict))  # <class 'dict'>
```

Các thao tác với `dictionary`:

- Truy xuất giá trị: `my_dict['name']`
- Thêm cặp khóa-giá trị: `my_dict['country'] = 'USA'`
- Xóa cặp khóa-giá trị: `del my_dict['city']`

3.7. Kiểu `set`

`Set` trong Python là một tập hợp các phần tử không có thứ tự và không chứa các phần tử trùng lặp. `Set` được sử dụng khi bạn cần lưu trữ một tập hợp các giá trị duy nhất và không quan tâm đến thứ tự của chúng.

`Set` được khai báo trong dấu ngoặc nhọn (`{}`), giống như `dictionary` nhưng không có cặp khóa-giá trị.

Ví dụ:

```
# Kiểu set
my_set = {1, 2, 3, 4, 5}
print(type(my_set))  # <class 'set'>
```

Các thao tác với `set`:

- Thêm phần tử: `my_set.add(6)`
- Xóa phần tử: `my_set.remove(3)`
- Tập hợp không có phần tử trùng lặp.

3.8. Kiểu `None`

`None` là một kiểu dữ liệu đặc biệt trong Python, dùng để biểu thị sự vắng mặt hoặc không có giá trị. `None` thường được sử dụng để đánh dấu sự thiếu sót hoặc chưa được khởi tạo.

Ví dụ:

```
# Kiểu None
x = None
print(type(x))  # <class 'NoneType'>
```

`None` không phải là một kiểu dữ liệu số, chuỗi hay bất kỳ kiểu nào khác. Nó đại diện cho "không có giá trị". Thường dùng trong các trường hợp kiểm tra điều kiện hoặc khi muốn khởi tạo một biến mà không gán giá trị ngay lập tức.

3.9. Kiểu `Sorting`

Bạn có thể sắp xếp một danh sách tại chỗ (không tạo đối tượng mới) bằng cách gọi hàm `sort` của nó:

```
# Kiểu None
In [61]: a = [7, 2, 5, 1, 3]
In [62]: a.sort()
In [63]: a
Out[63]: [1, 2, 3, 5, 7]
```

Hàm `sort` có một vài tùy chọn sẽ rất hữu ích trong một số tình huống. Một trong những tùy chọn đó là khả năng truyền vào một khóa sắp xếp phụ—tức là một hàm tạo ra giá trị để dùng trong việc sắp xếp các đối tượng. Ví dụ, ta có thể sắp xếp một tập hợp các chuỗi theo độ dài của chúng:

```
In [64]: b = ['saw', 'small', 'He', 'foxes', 'six']
In [65]: b.sort(key=len)
In [66]: b
Out[66]: ['He', 'saw', 'six', 'small', 'foxes']
```

Chúng ta sẽ sớm tìm hiểu về hàm `sorted`, có thể tạo ra một bản sao đã sắp xếp của một dãy chung.

Tìm kiếm nhị phân và duy trì danh sách đã sắp xếp Mô-đun tích hợp `bisect` triển khai tìm kiếm nhị phân và chèn vào một danh sách đã sắp xếp. Hàm `bisect.bisect` tìm vị trí mà một phần tử nên được chèn vào để giữ cho danh sách đã sắp xếp, trong khi `bisect.insort` thực hiện việc chèn phần tử vào vị trí đó:

```
In [67]: import bisect
In [68]: c = [1, 2, 2, 2, 3, 4, 7]
In [69]: bisect.bisect(c, 2)
Out[69]: 4
In [70]: bisect.bisect(c, 5)
Out[70]: 6
In [71]: bisect.insort(c, 6)
In [72]: c
Out[72]: [1, 2, 2, 2, 3, 4, 6, 7]
```

3.10. Kiểu Slicing

Bạn có thể chọn các phần của hầu hết các kiểu dữ liệu dãy bằng cách sử dụng cú pháp cắt dãy, trong đó dạng cơ bản là `start:stop` được truyền vào toán tử chỉ mục `[]`:

```
In [73]: seq = [7, 2, 3, 7, 5, 6, 0, 1]
In [74]: seq[1:5]
Out[74]: [2, 3, 7, 5]
```

Các phần tử có thể được gán lại thông qua cắt dãy:

```
In [75]: seq[3:4] = [6, 3]
In [76]: seq
Out[76]: [7, 2, 3, 6, 3, 5, 6, 0, 1]
```

Mặc dù phần tử ở chỉ mục bắt đầu được bao gồm, nhưng chỉ mục dừng không được bao gồm, vì vậy số lượng phần tử trong kết quả là `stop - start`. Bạn có thể bỏ qua `start` hoặc `stop`, trong trường hợp đó chúng mặc định là bắt đầu của dãy và kết thúc của dãy:

```
In [77]: seq[:5]
Out[77]: [7, 2, 3, 6, 3]
In [78]: seq[3:]
Out[78]: [6, 3, 5, 6, 0, 1]
```

Chỉ mục âm sẽ cắt dãy dựa trên độ dài của dãy từ cuối:

```
In [79]: seq[-4:]
Out[79]: [5, 6, 0, 1]
In [80]: seq[-6:-2]
Out[80]: [6, 3, 5, 6]
```

Một bước (`step`) cũng có thể được sử dụng sau dấu hai chấm thứ hai để lấy, ví dụ, mỗi phần tử cách nhau 1 phần:

```
In [81]: seq[::2]
Out[81]: [7, 3, 3, 6, 1]
```

Một cách sử dụng thông minh là truyền vào `-1`, điều này có tác dụng hữu ích là đảo ngược danh sách hoặc `tuple`:

```
In [82]: seq[::-1]
Out[82]: [1, 0, 6, 5, 3, 6, 3, 2, 7]
```

3.11. Bài tập thực hành

- **Kiểu số:**

1. Khai báo hai biến `x` và `y` với giá trị lần lượt là `25` và `3.14`. Hãy in ra kiểu của các biến đó.
2. Tạo một số phức `z` với phần thực là `5` và phần ảo là `7`. In ra giá trị của `z` và kiểu dữ liệu của nó.

- **Kiểu chuỗi `String`:**

1. Khai báo một chuỗi bất kỳ và in ra độ dài của chuỗi đó.
2. Cắt chuỗi từ chỉ mục thứ 2 đến chỉ mục thứ 5, rồi in ra kết quả.
3. Kết hợp hai chuỗi “Hello” và “World” thành một chuỗi duy nhất và in ra kết quả.

- **Kiểu Boolean `bool`:**

1. Khai báo hai biến kiểu `Boolean` `is_valid` và `is_active` với giá trị `True` và `False`. In ra kết quả của phép toán `and`, `or` giữa hai biến này.
2. Sử dụng kiểu `bool` để kiểm tra xem một số có phải là số âm hay không. Ví dụ, kiểm tra giá trị của biến `a = -5`.

- **Kiểu danh sách `List`:**

1. Khai báo một danh sách có ít nhất 5 phần tử, bao gồm cả số và chuỗi. In ra phần tử đầu tiên và phần tử cuối cùng của danh sách.

2. Thêm một phần tử mới vào danh sách và in lại danh sách.
3. Sửa đổi phần tử ở chỉ mục 2 và in ra kết quả.

- **Kiểu `tuple`:**

1. Tạo một `tuple` gồm 4 phần tử, in ra phần tử thứ 2 và phần tử cuối cùng.
2. Sử dụng cú pháp `slicing` để lấy các phần tử từ chỉ mục 1 đến chỉ mục 3 của `tuple` và in ra kết quả.

- **Kiểu từ điển `Dictionary`:**

1. Tạo một `dictionary` với ít nhất 3 cặp khóa-giá trị. In ra giá trị của một khóa bất kỳ.
2. Thêm một cặp khóa-giá trị mới vào `dictionary` và in lại toàn bộ `dictionary`.
3. Xóa một cặp khóa-giá trị khỏi `dictionary` và in lại kết quả.

- **Kiểu `set`:**

1. Khai báo một `set` gồm các số từ 1 đến 5. In ra `set` đó.
2. Thêm một số mới vào `set` và in lại `set`.
3. Xóa một phần tử khỏi `set` và in lại kết quả.

- **Kiểu `None`:**

1. Khai báo một biến `a` với giá trị là `None`. In ra kiểu của biến này.
2. Kiểm tra xem biến `a` có phải là `None` hay không và in ra kết quả.

- **Kiểu `Sorting`:**

1. Khai báo một danh sách có các số hỗn hợp và sắp xếp danh sách theo thứ tự tăng dần.
2. Sắp xếp danh sách các chuỗi theo độ dài chuỗi.

- **Kiểu `Slicing`:**

1. Tạo một danh sách số nguyên và sử dụng `slicing` để lấy các phần tử từ chỉ mục 3 đến chỉ mục 7.
2. Sử dụng chỉ mục âm để lấy phần tử từ chỉ mục -5 đến hết danh sách.

Chương 4. Biến và hằng số

Contents

Trong lập trình, biến và hằng số là các khái niệm quan trọng trong việc lưu trữ và xử lý dữ liệu. Python cung cấp các cơ chế linh hoạt để làm việc với chúng. Trong phần này, chúng ta sẽ tìm hiểu cách khai báo và sử dụng biến, quy tắc đặt tên biến, cũng như cách sử dụng hằng số trong Python.

4.1. Khai báo và sử dụng biến

Biến là một tên đại diện cho một giá trị trong chương trình. Khi bạn khai báo một biến, bạn đang yêu cầu chương trình dành ra một không gian bộ nhớ để lưu trữ dữ liệu và gán giá trị cho biến đó.

- **Cú pháp khai báo biến** trong Python là rất đơn giản: `tên_biến = giá_trị`
- Python là ngôn ngữ không yêu cầu khai báo kiểu dữ liệu của biến, do đó, bạn chỉ cần gán giá trị cho biến, Python sẽ tự động xác định kiểu dữ liệu của biến đó.

Ví dụ:

```
# Khai báo và sử dụng biến
name = "Alice"  # Biến name lưu trữ giá trị chuỗi
age = 25  # Biến age lưu trữ giá trị số nguyên
height = 5.5  # Biến height lưu trữ giá trị số thực

# Sử dụng biến
print("Name:", name)
print("Age:", age)
print("Height:", height)
```

Lưu ý:

- Python cho phép thay đổi giá trị của biến bất kỳ lúc nào.
- Bạn có thể sử dụng biến trong các phép toán, biểu thức, hay trong các hàm.

4.2. Quy tắc đặt tên biến

Khi đặt tên cho biến trong Python, bạn cần tuân thủ một số quy tắc sau:

- Tên biến phải bắt đầu bằng chữ cái (a-z, A-Z) hoặc dấu gạch dưới (_). Không thể bắt đầu bằng số.

  Ví dụ hợp lệ: `name`, `_age`, `height1`
  Ví dụ không hợp lệ: `1name`, `@age`

- Tên biến có thể bao gồm chữ cái, số, hoặc dấu gạch dưới (_), nhưng không được chứa khoảng trắng.

  Ví dụ hợp lệ: `first_name`, `user_age_1`
  Ví dụ không hợp lệ: `first name`, `user age`

- Tên biến phân biệt chữ hoa và chữ thường.

  Ví dụ, `name` và `Name` là hai biến khác nhau.

- Không được sử dụng từ khóa hoặc tên của các đối tượng đặc biệt trong Python. Python có một số từ khóa nhất định không thể sử dụng làm tên biến, chẳng hạn như `if`, `else`, `while`, `def`, `return`, v.v.

  Ví dụ không hợp lệ:

  ```
  if = 10  # Không hợp lệ vì 'if' là từ khóa
  ```

- Khuyến cáo sử dụng quy tắc đặt tên theo kiểu "snake_case" (tức là các từ được nối với nhau bằng dấu gạch dưới _, như: user_name, total_amount), mặc dù Python cho phép bạn dùng kiểu "camelCase" (như userName, totalAmount), kiểu "PascalCase", hoặc kiểu "UPPERCASE" cho các hằng số.

  Ví dụ về tên biến hợp lệ:

```
age = 30
user_name = "John"
totalAmount = 100.5
user_age_1 = 25
```

4.3. Hằng số trong Python

Hằng số là một giá trị không thay đổi trong suốt chương trình. Python không hỗ trợ trực tiếp hằng số như một số ngôn ngữ khác, nhưng bạn có thể tạo ra các hằng số bằng cách sử dụng quy ước đặt tên.

- Quy ước về hằng số: Trong Python, hằng số thường được khai báo bằng tên biến viết hoa hoàn toàn và ngăn cách bởi dấu gạch dưới (`_`). Điều này chỉ là một quy ước, và Python không có cơ chế để ngăn chặn việc thay đổi giá trị của hằng số.
- Các hằng số có thể là bất kỳ kiểu dữ liệu nào, bao gồm số, chuỗi, hoặc các kiểu dữ liệu phức tạp khác.

Ví dụ:

```
# Hằng số
PI = 3.14159
GRAVITY = 9.8
MAX_SPEED = 120

# Sử dụng hằng số
print("Giá trị của PI:", PI)
print("Gia tốc trọng trường:", GRAVITY)
print("Tốc độ tối đa:", MAX_SPEED)
```

Lưu ý về hằng số:

- Trong Python, việc thay đổi giá trị của hằng số không bị hạn chế, vì vậy đây chỉ là quy ước giúp lập trình viên nhận biết rằng giá trị của biến đó không nên thay đổi trong quá trình

thực thi chương trình.

- Hằng số có thể được sử dụng trong các phép toán, cấu trúc điều kiện, và các tình huống mà giá trị không thay đổi.

Chương 5. Câu lệnh điều kiện

Contents

Câu lệnh điều kiện là một trong những cấu trúc cơ bản trong lập trình, cho phép chương trình đưa ra quyết định dựa trên các điều kiện cụ thể. Trong Python, câu lệnh điều kiện giúp bạn kiểm tra các biểu thức và thực hiện hành động khác nhau tùy vào kết quả của biểu thức đó. Phần này sẽ giới thiệu về các câu lệnh điều kiện trong Python, bao gồm câu lệnh `if`, `else`, `elif`, và cách sử dụng chúng một cách hiệu quả.

5.1. Câu lệnh `if`

Câu lệnh `if` trong Python dùng để kiểm tra một điều kiện và thực hiện một khối mã khi điều kiện đó là đúng (True). Cú pháp cơ bản của câu lệnh `if` như sau:

```
if điều_kiện:
    # Khối mã thực thi khi điều kiện là True
```

Trong đó, `điều_kiện` là một biểu thức trả về giá trị Boolean (True hoặc False).

Ví dụ:

```
# Kiểm tra tuổi
age = 20
if age >= 18:
    print("Bạn đã đủ tuổi để tham gia chương trình.")
```

Trong ví dụ trên, chương trình sẽ kiểm tra xem `age` có lớn hơn hoặc bằng 18 hay không. Nếu đúng, câu lệnh `print` sẽ được thực thi.

5.2. Câu lệnh `else` và `elif`

5.2.1. Câu lệnh else:

Câu lệnh `else` dùng để xác định hành động khi điều kiện trong câu lệnh `if` không thỏa mãn (False). Câu lệnh `else` không cần điều kiện, nó sẽ thực thi khi tất cả các điều kiện trước đó đều sai.

Cú pháp:

```
if điều_kiện:
    # Khối mã thực thi khi điều kiện là True
else:
    # Khối mã thực thi khi điều kiện là False
```

Ví dụ:

```
age = 15
if age >= 18:
    print("Bạn đã đủ tuổi.")
else:
    print("Bạn chưa đủ tuổi.")
```

Kết quả: `Bạn chưa đủ tuổi.`

5.2.2. Câu lệnh elif:

Câu lệnh `elif`: Dùng khi bạn có nhiều điều kiện để kiểm tra. Nếu điều kiện trong câu lệnh `if` không thỏa mãn, Python sẽ tiếp tục kiểm tra các điều kiện trong câu lệnh `elif`. Nếu điều kiện nào thỏa mãn, chương trình sẽ thực hiện khối mã của điều kiện đó.

Cú pháp:

```
if điều_kiện_1:
    # Khối mã thực thi khi điều kiện 1 là True
elif điều_kiện_2:
    # Khối mã thực thi khi điều kiện 2 là True
else:
    # Khối mã thực thi khi tất cả điều kiện đều False
```

Ví dụ:

```
age = 20
if age < 18:
    print("Bạn chưa đủ tuổi.")
elif age == 18:
    print("Chúc mừng bạn vừa đủ tuổi.")
else:
    print("Bạn đã đủ tuổi.")
```

Kết quả: `Bạn chưa đủ tuổi.`

5.3. Câu lệnh điều kiện lồng nhau

Câu lệnh điều kiện lồng nhau (nested if) là khi bạn đặt một câu lệnh `if` bên trong câu lệnh `if` hoặc `else`. Điều này giúp bạn kiểm tra các điều kiện phức tạp hơn.

Cú pháp:

```
if điều_kiện_1:
    if điều_kiện_2:
        # Khối mã thực thi khi điều kiện 1 và điều kiện 2 đều đúng
    else:
        # Khối mã thực thi khi điều kiện 1 đúng nhưng điều kiện 2 sai
else:
    # Khối mã thực thi khi điều kiện 1 sai
```

Ví dụ:

```
age = 20
has_permission = True

if age >= 18:
    if has_permission:
        print("Bạn được phép tham gia chương trình.")
    else:
        print("Bạn chưa có sự cho phép.")
else:
    print("Bạn chưa đủ tuổi.")
```

Kết quả: `Bạn được phép tham gia chương trình.`

Trong ví dụ trên, đầu tiên Python sẽ kiểm tra điều kiện `age >= 18`. Nếu điều kiện này đúng, nó tiếp tục kiểm tra thêm điều kiện `has_permission`. Nếu điều kiện này cũng đúng, nó sẽ thực hiện hành động tương ứng.

5.4. Bài tập thực hành

- **Bài tập với câu lệnh `if`:**

1. Nhập một số và kiểm tra xem nó là số chẵn hay số lẻ.
2. Kiểm tra xem một người có đủ tuổi để lái xe (tuổi >= 18).
3. Kiểm tra điểm của một học sinh và in ra kết quả: “Đạt” nếu điểm >= 5, “Không đạt” nếu điểm < 5.

- **Bài tập với câu lệnh `else`:**

1. Kiểm tra điểm thi và in ra thông báo “Đạt” nếu điểm >= 5, ngược lại in “Không đạt”.
2. Kiểm tra tuổi đủ điều kiện tham gia chương trình (>= 18).

- **Bài tập với câu lệnh `elif`:**

1. Kiểm tra độ tuổi và in ra các thông báo:
    - Nếu tuổi < 18, in “Bạn chưa đủ tuổi.”
    - Nếu tuổi == 18, in “Chúc mừng bạn vừa đủ tuổi.”
    - Nếu tuổi > 18, in “Bạn đã đủ tuổi.”
2. Kiểm tra ngày trong tuần và in ra tên ngày:
    - 1: “Thứ Hai”

    - 2: “Thứ Ba”
    - 3: “Thứ Tư”
    - ...

- **Bài tập với câu lệnh điều kiện lồng nhau (Nested if):**

1. Kiểm tra nếu tuổi >= 18, sau đó kiểm tra xem người dùng có sự cho phép hay không (True/False).
2. Kiểm tra nếu điểm >= 8, sau đó kiểm tra nếu người đó có tham gia các hoạt động ngoại khóa hay không.

- **Bài tập tổng hợp:**
    - Kiểm tra tình trạng học sinh:
        - Nếu điểm >= 7 và hành vi tốt (yes), in “Học sinh giỏi.”
        - Nếu điểm < 7 và hành vi tốt (yes), in “Học sinh trung bình.”
        - Nếu điểm >= 7 và hành vi xấu (no), in “Học sinh cần cải thiện hành vi.”
        - Nếu điểm < 7 và hành vi xấu (no), in “Học sinh yếu.”

Chương 6. Câu lệnh vòng lặp

Contents

Vòng lặp là một cấu trúc điều khiển cho phép lập trình viên thực thi một khối mã nhiều lần. Trong Python, có hai loại vòng lặp chính: vòng lặp `for` và vòng lặp `while`. Các câu lệnh vòng lặp giúp bạn tự động hóa các tác vụ lặp đi lặp lại mà không cần phải viết mã lặp lại nhiều lần. Trong phần này, chúng ta sẽ tìm hiểu về các câu lệnh vòng lặp trong Python, bao gồm cách sử dụng vòng lặp `for`, vòng lặp `while`, cũng như các câu lệnh `break`, `continue` và cách lặp qua các cấu trúc dữ liệu.

6.1. Vòng lặp for

Vòng lặp `for` trong Python được sử dụng để lặp qua các đối tượng có thể lặp được (`iterable`), chẳng hạn như danh sách (`list`), `tuple`, chuỗi (`string`), hoặc một phạm vi số (`range`). Câu lệnh `for` rất hữu ích khi bạn muốn lặp qua một tập hợp các giá trị mà không cần biết trước số lần lặp.

Cú pháp cơ bản:

```
for biến in đối_tượng:
    # Khối mã sẽ được thực thi trong mỗi lần lặp
```

Ví dụ:

```
# Vòng lặp for qua một danh sách
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Kết quả:

```
apple
banana
cherry
```

Trong ví dụ này, vòng lặp `for` sẽ lặp qua từng phần tử trong danh sách `fruits` và in ra tên quả. Vòng lặp `for` cũng có thể được sử dụng với `range()` để lặp qua một dãy số:

```
for i in range(5):
    print(i)
```

Kết quả:

```
0
1
2
3
4
```

6.2. Vòng lặp while

Vòng lặp `while` trong Python lặp lại một khối mã miễn là điều kiện kiểm tra của vòng lặp là đúng. Khi điều kiện này trở thành sai, vòng lặp sẽ dừng lại. Đây là lựa chọn tốt khi số lần lặp không được xác định trước mà phụ thuộc vào một điều kiện.

Cú pháp cơ bản:

```
while điều_kiện:
    # Khối mã sẽ được thực thi miễn là điều kiện đúng
```

Ví dụ:

```
# Vòng lặp while
count = 0
while count < 5:
    print(count)
    count += 1  # Tăng giá trị của count mỗi lần lặp
```

Kết quả:

```
0
1
2
3
4
```

Trong ví dụ này, vòng lặp sẽ tiếp tục chạy miễn là `count` nhỏ hơn 5. Mỗi lần lặp, giá trị của `count` sẽ tăng lên 1, và khi `count` đạt 5, vòng lặp sẽ dừng.

6.3. Câu lệnh break, continue

Trong vòng lặp, bạn có thể sử dụng các câu lệnh `break` và `continue` để điều khiển luồng thực thi của vòng lặp.

- `break`: Dùng để thoát ra khỏi vòng lặp khi một điều kiện nào đó được thỏa mãn.
- `continue`: Dùng để bỏ qua phần còn lại của vòng lặp trong lần lặp hiện tại và tiếp tục với lần lặp tiếp theo.

Ví dụ sử dụng `break`:

```
# Sử dụng break để thoát vòng lặp
for i in range(10):
    if i == 5:
        break  # Thoát vòng lặp khi i bằng 5
    print(i)
```

Kết quả:

```
0
1
2
3
4
```

Khi `i` bằng 5, câu lệnh `break` sẽ dừng vòng lặp.

Ví dụ sử dụng `continue`:

```
# Sử dụng continue để bỏ qua một lần lặp
for i in range(10):
    if i == 5:
        continue  # Bỏ qua lần lặp này khi i bằng 5
    print(i)
```

Kết quả:

```
0
1
2
3
4
6
7
8
9
```

Ở đây, khi `i` bằng 5, câu lệnh `continue` sẽ bỏ qua việc in giá trị `i` và tiếp tục với lần lặp tiếp theo.

6.4. Lặp qua các cấu trúc dữ liệu

Python cung cấp khả năng lặp qua các cấu trúc dữ liệu như danh sách, `tuple`, từ điển, tập hợp (`set`), và chuỗi. Câu lệnh vòng lặp `for` có thể được sử dụng để truy cập từng phần tử trong những cấu trúc này.

- **Lặp qua danh sách, `tuple`, và chuỗi:**

```
# Lặp qua danh sách
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

- **Lặp qua từ điển:** Khi lặp qua một từ điển, bạn có thể lặp qua các khóa (`keys`), giá trị (`values`), hoặc cả khóa và giá trị.

```
# Lặp qua các khóa của từ điển
person = {"name": "John", "age": 30, "city": "New York"}
for key in person:
    print(key)

# Lặp qua các khóa và giá trị của từ điển
for key, value in person.items():
    print(key, value)
```

- **Lặp qua `set`:** `Set` là một tập hợp các phần tử không trùng lặp. Bạn có thể lặp qua các phần tử của `set` giống như các kiểu dữ liệu khác.

```
# Lặp qua set
fruits_set = {"apple", "banana", "cherry"}
for fruit in fruits_set:
    print(fruit)
```

6.5. Bài tập thực hành

- **Bài tập với vòng lặp `for`:**

1. Lặp qua danh sách các số từ 1 đến 5 và in từng phần tử.
2. Nhập một chuỗi từ người dùng và in từng ký tự của chuỗi.
3. Sử dụng vòng lặp `for` với `range()` để in các số từ 0 đến 9.
4. Tính tổng các số trong danh sách `[1, 2, 3, 4, 5]`.

- **Bài tập với vòng lặp `while`:**

1. Sử dụng vòng lặp `while` để in các số từ 1 đến 5.
2. Sử dụng vòng lặp `while` để đếm ngược từ 10 đến 1.
3. Nhập một số nguyên N và tính tổng các số từ 1 đến N bằng vòng lặp `while`.

4. Kiểm tra xem một số có phải là số nguyên tố không bằng vòng lặp `while`.

- **Bài tập với câu lệnh `break` và `continue`:**

1. Lặp qua các số từ 1 đến 10 và dừng khi gặp số 6 (sử dụng `break`).
2. Lặp qua các số từ 1 đến 10 và bỏ qua số 5 (sử dụng `continue`).
3. Sử dụng `break` và `continue` trong vòng lặp `while` để bỏ qua số 3 và dừng khi gặp số 7.

- **Lặp qua các cấu trúc dữ liệu:**

1. Lặp qua các phần tử trong danh sách `["apple", "banana", "cherry"]`.
2. Lặp qua các phần tử trong `tuple` `(1, 2, 3, 4, 5)`.
3. Lặp qua các khóa và giá trị trong từ điển `{"name": "Alice", "age": 25, "city": "Paris"}`.
4. Lặp qua các phần tử trong `set` `{1, 2, 3, 4, 5}`.

- **Bài tập tổng hợp:**

1. In ra các số chia hết cho 3 trong khoảng từ 1 đến 30.
2. In ra bảng cửu chương 2 bằng vòng lặp `for`.

Chương 7. Hàm trong Python

Contents

Hàm là một khối mã được định nghĩa để thực hiện một nhiệm vụ cụ thể. Trong Python, hàm cho phép chúng ta tái sử dụng mã nguồn và tổ chức chương trình một cách hợp lý. Phần này sẽ giới thiệu về cách định nghĩa và sử dụng hàm trong Python, các khái niệm liên quan đến tham số, giá trị trả về, và các kiểu hàm đặc biệt như hàm vô danh và hàm đệ quy.

7.1. Định nghĩa hàm (`def`)

- **Khái niệm hàm**: Là một khối mã có thể được gọi để thực hiện một tác vụ nào đó.
- **Cấu trúc định nghĩa hàm**: Cách sử dụng từ khóa `def` để định nghĩa hàm trong Python.

```
def ten_ham(tham_so1, tham_so2):
    # Thực hiện tác vụ
    return ket_qua
```

- **Ví dụ**:

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

print(tong(3, 5))  # Output: 8
```

- **Các điểm lưu ý**:
  - Tên hàm phải tuân thủ quy tắc đặt tên biến trong Python.
  - Một hàm có thể không có tham số (hàm không đối số) hoặc có nhiều tham số.
  - Câu lệnh `return` dùng để trả về giá trị từ hàm.

7.2. Tham số và đối số trong hàm

- **Tham số (parameter)**: Là các biến được định nghĩa trong dấu ngoặc khi khai báo hàm.
- **Đối số (argument)**: Là các giá trị thực tế được truyền vào khi gọi hàm.
- **Ví dụ**:

```
def chao(ten):  # 'ten' là tham số
    print(f"Chào {ten}!")

chao("Minh")  # "Minh" là đối số
```

- **Các loại tham số:**:
  - **Tham số bắt buộc**: Là các tham số phải được truyền giá trị khi gọi hàm.
  - **Tham số mặc định**: Có thể có giá trị mặc định nếu không được truyền đối số khi gọi hàm.

```
def chao(ten="Bạn"):  # tham số có giá trị mặc định
    print(f"Chào {ten}!")
chao()  # Output: Chào Bạn!
chao("Hoa")  # Output: Chào Hoa!
```

  - **Tham số không xác định (Arbitrary arguments)**: Dùng `*args` để nhận số lượng tham số không xác định.

```
def tinh_tong(*args):
    return sum(args)

print(tinh_tong(1, 2, 3))  # Output: 6
```

  - **Tham số từ khóa (Keyword arguments)**: Dùng `**kwargs` để nhận tham số theo cặp khóa - giá trị.

```
def thong_tin(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

thong_tin(ten="Minh", tuoi=22)  # Output: ten: Minh, tuoi: 22
```

7.3. Giá trị trả về từ hàm

- **Khái niệm**: Một hàm có thể trả về một giá trị (hoặc không có giá trị trả về). Sử dụng từ khóa `return` để trả về giá trị từ hàm.
- **Hàm không có giá trị trả về**: Nếu không có câu lệnh `return`, hàm sẽ trả về `None` mặc định.

```
def khong_tra_ve():
    print("Hàm này không có giá trị trả về.")

result = khong_tra_ve()  # result sẽ là None
print(result)  # Output: None
```

- **Trả về nhiều giá trị**: Có thể trả về nhiều giá trị dưới dạng tuple.

```
def tinh_tong_hieu(a, b):
    return a + b, a - b

tong, hieu = tinh_tong_hieu(5, 3)
print(tong, hieu)  # Output: 8 2
```

7.4. Hàm vô danh (`Lambda`)

- **Khái niệm**: Hàm vô danh (hay còn gọi là hàm `lambda`) là một hàm không có tên, thường dùng cho các tác vụ đơn giản mà không cần phải định nghĩa hàm thông thường.
- **Cấu trúc của hàm `lambda`**:

```
dlambda tham_so1, tham_so2: biểu_thức
```

- **Ví dụ**:

```
# Hàm lambda tính tổng hai số
tong = lambda a, b: a + b
print(tong(3, 4))  # Output: 7
```

- **Ứng dụng**:
  - Sử dụng trong các hàm như `map()`, `filter()`, `sorted()`, v.v.
  - Ví dụ với `sorted()`:

```
# Hàm lambda tính tổng hai số
danh_sach = [(1, 2), (3, 4), (2, 3)]
danh_sach_sort = sorted(danh_sach, key=lambda x: x[1])
print(danh_sach_sort)  # Output: [(1, 2), (2, 3), (3, 4)]
```

7.5. Hàm đệ quy

- **Khái niệm**: Hàm đệ quy là hàm gọi chính nó trong quá trình thực hiện. Đệ quy thường được sử dụng để giải quyết các bài toán có tính chất lặp lại.
- **Cấu trúc**:
  - Hàm đệ quy cần có một điều kiện dừng (`base case`) để tránh gây ra vô hạn.
  - **Ví dụ**: Tính giai thừa

```
# Hàm lambda tính tổng hai số
def giai_thua(n):
    if n == 0:  # Điều kiện dừng
        return 1
    else:
        return n * giai_thua(n - 1)

print(giai_thua(5))  # Output: 120
```

- **Các điểm lưu ý**:
  - Đệ quy có thể rất hiệu quả cho các bài toán chia để trị hoặc các bài toán có cấu trúc cây.
  - Phải chắc chắn rằng mỗi lần gọi đệ quy làm giảm dần giá trị và có điều kiện dừng hợp lý.

7.6. Bài tập thực hành

- **Đề bài về định nghĩa hàm:**

1. Viết một hàm `tinh_tong(a, b)` để tính tổng của hai số và trả về kết quả.
2. Viết một hàm `kiem_tra_so_chan(n)` kiểm tra xem một số có phải là số chẵn hay không. Hàm này trả về `True` nếu là số chẵn, `False` nếu là số lẻ.
3. Viết một hàm `chao(ten)` để in ra lời chào theo tên người dùng. Ví dụ, nếu tên người dùng là “Minh”, hàm sẽ in “Chào Minh!”.

- **Đề bài về tham số và đối số trong hàm:**

1. Viết một hàm `hieu_so(a, b)` tính hiệu giữa hai số a và b và trả về kết quả.
2. Viết một hàm `giai_thuong(a, b=1)` tính tích của hai số, với b có giá trị mặc định là 1.
3. Viết một hàm `tinh_tong(*args)` nhận vào một số lượng tham số không xác định và trả về tổng của tất cả các tham số đó.

- **Đề bài về giá trị trả về từ hàm:**

1. Viết một hàm `tinh_tong_hieu(a, b)` trả về cả tổng và hiệu của hai số a và b dưới dạng tuple.
2. Viết một hàm `kiem_tra_somoi(x)` kiểm tra xem một số có phải là số nguyên tố không. Hàm này trả về `True` nếu là số nguyên tố và `False` nếu không phải.
3. Viết một hàm `giai_thua(n)` tính giá trị giai thừa của một số n. Nếu không có giá trị trả về, hàm sẽ in thông báo “Hàm không trả về giá trị”.

- **Đề bài về hàm vô danh (`Lambda`):**

1. Viết một hàm `lambda square` nhận một tham số x và trả về bình phương của x.
2. Dùng hàm `lambda` trong `sorted()` để sắp xếp một danh sách các `tuple` theo phần tử thứ 2 của mỗi `tuple`.
3. Viết một hàm `lambda` `is_even` để kiểm tra xem một số có phải là số chẵn không.

- **Đề bài về hàm đệ quy:**

1. Viết một hàm đệ quy `giai_thua(n)` tính giai thừa của một số n.
2. Viết một hàm đệ quy `fibonacci(n)` để tính số Fibonacci thứ n.
3. Viết một hàm đệ quy `tong_cac_so(n)` tính tổng các số từ 1 đến n.

Chương 8. Xử lý Ngoại lệ

Contents

Xử lý ngoại lệ (exception handling) là một cơ chế giúp chương trình Python có thể xử lý các lỗi hoặc tình huống bất ngờ mà không làm gián đoạn toàn bộ chương trình. Việc sử dụng xử lý ngoại lệ giúp nâng cao tính ổn định và dễ bảo trì của chương trình. Phần này sẽ trình bày các kỹ thuật xử lý ngoại lệ cơ bản, xử lý các lỗi đặc biệt và cách tạo exception tùy chỉnh trong Python.

8.1. Cấu trúc `try-except`

- **Khái niệm**: Xử lý ngoại lệ trong Python chủ yếu sử dụng cấu trúc `try-except` để bắt và xử lý các lỗi xảy ra trong khi chương trình đang thực thi.
- **Cấu trúc cơ bản**:

```
try:
    # Đoạn mã có thể gây ra lỗi
except Lỗi as e:
    # Xử lý khi có lỗi
```

- **Trong đó**:
  - `try`: Chứa đoạn mã mà bạn nghĩ có thể gây ra lỗi.
  - `except`: Xử lý lỗi nếu lỗi xảy ra trong phần `try`. Bạn có thể chỉ định loại lỗi hoặc để chung cho tất cả lỗi.
- **Ví dụ**:

```
try:
    x = 5 / 0  # Lỗi chia cho 0
except ZeroDivisionError as e:
    print(f"Lỗi: {e}")
```

- **Output**:

```
Lỗi: division by zero
```

- **Các điều kiện xử lý**:
  - Bạn có thể sử dụng nhiều `except` để xử lý các loại lỗi khác nhau.
  - Có thể sử dụng `else` để chỉ định đoạn mã thực thi nếu không có lỗi xảy ra trong phần `try`.
  - Có thể dùng `finally` để thực hiện một số tác vụ cần thiết như đóng file, kết nối cơ sở dữ liệu, v.v., dù có lỗi hay không.
- **Ví dụ với `else` và `finally`**:

```
try:
    x = 5 / 1
except ZeroDivisionError:
    print("Lỗi chia cho 0")
else:
    print("Không có lỗi")
finally:
    print("Đoạn mã này luôn được thực thi")
```

- **Output**:

```
Không có lỗi
Đoạn mã này luôn được thực thi
```

8.2. Xử lý lỗi đặc biệt

Python cung cấp một số lỗi đặc biệt (built-in exceptions) có thể giúp chương trình xử lý các tình huống đặc biệt. Một số lỗi thông dụng bao gồm:

- `ZeroDivisionError`: Lỗi khi chia cho 0.
- `FileNotFoundError`: Lỗi khi không tìm thấy tệp tin.

- `ValueError`: Lỗi khi truyền tham số không đúng loại.
- `IndexError`: Lỗi khi truy cập vào chỉ mục không tồn tại trong danh sách.
- `KeyError`: Lỗi khi truy cập vào một khóa không tồn tại trong từ điển.

Ví dụ xử lý lỗi `ValueError`:

```
try:
    x = int("abc")  # Sẽ gây ra lỗi ValueError
except ValueError as e:
    print(f"Lỗi: {e}")
```

- **Output**

```
Lỗi: invalid literal for int() with base 10: 'abc'
```

Ví dụ xử lý lỗi `FileNotFoundError`:

```
try:
    with open("non_existent_file.txt", "r") as file:
        content = file.read()
except FileNotFoundError as e:
    print(f"Lỗi: {e}")
```

- **Output**

```
Lỗi: [Errno 2] No such file or directory: 'non_existent_file.txt'
```

Xử lý nhiều loại lỗi: Nếu bạn muốn xử lý nhiều loại lỗi trong một khối `try-except`, bạn có thể liệt kê chúng trong một `except`:

```
try:
    # Giả sử bạn nhập dữ liệu không hợp lệ
    num = int(input("Nhập số: "))
except ValueError:
    print("Lỗi: Vui lòng nhập một số hợp lệ.")
except KeyboardInterrupt:
    print("Lỗi: Bạn đã hủy thao tác.")
```

8.3. Tạo và sử dụng exception tùy chỉnh

Ngoài các lỗi có sẵn trong Python, bạn cũng có thể tạo ra các lỗi tùy chỉnh (custom exceptions) để xử lý các tình huống cụ thể trong chương trình của mình.

- **Cấu trúc tạo exception tùy chỉnh:** Để tạo một exception tùy chỉnh, bạn cần kế thừa từ lớp `Exception` hoặc các lớp kế thừa của nó.

```
class TenException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

- **Ví dụ tạo exception tùy chỉnh:**

```
class NegativeNumberError(Exception):
    def __init__(self, message="Số không thể âm"):
        self.message = message
        super().__init__(self.message)

def kiem_tra_so_duong(x):
    if x < 0:
        raise NegativeNumberError("Số âm không hợp lệ!")
    return x

try:
    print(kiem_tra_so_duong(-5))
except NegativeNumberError as e:
    print(f"Lỗi: {e}")
```

- **Output:**

```
Lỗi: Số âm không hợp lệ!
```

Khi nào nên sử dụng exception tùy chỉnh?

- Khi bạn muốn thông báo một lỗi rõ ràng hơn về các tình huống đặc biệt trong ứng dụng.
- Khi có các điều kiện hoặc logic nghiệp vụ phức tạp mà không thể xử lý chỉ bằng các exception có sẵn.

8.4. Bài tập thực hành

- **Đề bài về cấu trúc `try-except:`**

1. Viết một đoạn mã để thực hiện phép chia hai số và sử dụng `try-except` để bắt lỗi khi chia cho 0.
2. Viết một chương trình yêu cầu người dùng nhập một số và sử dụng `try-except` để bắt lỗi nếu người dùng nhập vào một giá trị không phải là số.
3. Viết một chương trình thực hiện phép tính trên hai số (cộng, trừ, nhân, chia) và dùng `try-except` để xử lý các lỗi phổ biến như chia cho 0 và nhập sai dữ liệu.

- **Đề bài về xử lý lỗi đặc biệt:**

1. Viết một đoạn mã để đọc một tệp tin từ ổ đĩa và sử dụng `try-except` để bắt lỗi `FileNotFoundError` khi tệp không tồn tại.
2. Viết một chương trình yêu cầu người dùng nhập một số và kiểm tra xem giá trị đó có phải là số hợp lệ hay không bằng cách bắt lỗi `ValueError`.
3. Viết một chương trình thực hiện phép toán với các chỉ số của danh sách và sử dụng `try-except` để bắt lỗi `IndexError` khi chỉ mục không tồn tại.

- **Đề bài về tạo và sử dụng `exception` tùy chỉnh:**

1. Viết một chương trình có hàm `kiem_tra_so_duong(x)` kiểm tra xem một số có là số dương hay không. Nếu số âm, hãy tạo và ném một `exception` tùy chỉnh có tên `NegativeNumberError`.
2. Viết một chương trình yêu cầu người dùng nhập vào một số dương, nếu người dùng nhập số âm, chương trình sẽ `raise exception` `NegativeNumberError` và in ra thông báo lỗi.
3. Tạo một `exception` tùy chỉnh có tên `InvalidAgeError` để kiểm tra độ tuổi người dùng nhập vào. Nếu độ tuổi nhỏ hơn 0 hoặc lớn hơn 150, chương trình sẽ `raise exception` này và in thông báo lỗi.

Chương 9. Công cụ hỗ trợ lập trình

Contents

Trong quá trình phát triển phần mềm với Python, việc sử dụng các công cụ hỗ trợ sẽ giúp lập trình viên tăng hiệu quả công việc, từ việc viết mã nguồn đến gỡ lỗi và quản lý thư viện. Dưới đây là các công cụ phổ biến được sử dụng trong lập trình Python:

9.1. Các công cụ IDE cho Python

IDE (Integrated Development Environment) là môi trường phát triển tích hợp giúp lập trình viên viết mã, kiểm tra và gỡ lỗi ứng dụng một cách hiệu quả. Đối với Python, có một số IDE phổ biến được sử dụng rộng rãi:

9.1.1. PyCharm

- Giới thiệu: PyCharm là một IDE mạnh mẽ và phổ biến, được phát triển bởi JetBrains, hỗ trợ đầy đủ các tính năng cho lập trình Python, bao gồm tự động hoàn thành mã, kiểm tra lỗi cú pháp, tích hợp hệ thống kiểm tra mã (unit tests), và tích hợp với các công cụ quản lý mã nguồn như Git.
- Tính năng nổi bật:
  - Hỗ trợ phân tích mã tĩnh và kiểm tra lỗi trong thời gian thực.
  - Tích hợp tốt với các công cụ quản lý dự án và hệ thống kiểm tra mã.

- Cung cấp giao diện gỡ lỗi mạnh mẽ và công cụ quản lý cơ sở dữ liệu.
- Cung cấp hỗ trợ cho việc phát triển web với Django, Flask.
- Hỗ trợ tích hợp với Docker và các công cụ ảo hóa khác.

9.1.2. VS Code (Visual Studio Code)

- Giới thiệu: VS Code là một trình soạn thảo mã nguồn mã nguồn mở, phát triển bởi Microsoft, hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Python. VS Code không phải là một IDE đầy đủ nhưng có thể được mở rộng thành một IDE mạnh mẽ nhờ các tiện ích mở rộng (extensions).
- Tính năng nổi bật:
  - Giao diện nhẹ, nhanh và dễ sử dụng.
  - Hỗ trợ tích hợp với Git và các dịch vụ kiểm soát phiên bản.
  - Hỗ trợ nhiều tiện ích mở rộng để nâng cao trải nghiệm lập trình Python (ví dụ: Python Extension for VS Code).
  - Hỗ trợ tính năng IntelliSense (hoàn thành mã tự động) và gỡ lỗi.
  - Có thể chạy các notebook Jupyter trực tiếp trong VS Code.

9.1.3. Jupyter Notebook

- Giới thiệu: Jupyter Notebook là một công cụ mã nguồn mở cho phép lập trình viên viết và thực thi mã Python trong một môi trường tương tác. Nó đặc biệt hữu ích trong các lĩnh vực khoa học dữ liệu, học máy (machine learning), và nghiên cứu khoa học.
- Tính năng nổi bật:
  - Tạo và chia sẻ các tài liệu chứa mã Python, biểu đồ, văn bản mô tả (markdown).
  - Chạy mã trực tiếp trong các cell, giúp dễ dàng thử nghiệm và phân tích dữ liệu.
  - Hỗ trợ nhiều thư viện khoa học dữ liệu như Pandas, NumPy, Matplotlib, và Scikit-learn.
  - Dễ dàng chia sẻ mã và kết quả với đồng nghiệp hoặc cộng đồng.

9.2. Các công cụ gỡ lỗi (debugging)

Gỡ lỗi là một quá trình quan trọng trong lập trình giúp tìm và sửa các lỗi trong mã nguồn. Python cung cấp nhiều công cụ gỡ lỗi hiệu quả:

9.2.1. Gỡ lỗi với PyCharm

- PyCharm cung cấp một công cụ gỡ lỗi mạnh mẽ với giao diện đồ họa, giúp lập trình viên dễ dàng theo dõi và kiểm tra các biến, luồng thực thi, và lỗi trong chương trình.
- Tính năng nổi bật:
  - Điểm dừng (breakpoints) để dừng chương trình và kiểm tra các giá trị.
  - Khả năng theo dõi các thay đổi trong các biến và bước qua từng dòng mã để hiểu rõ quá trình thực thi.

9.2.2. PDB (Python Debugger)

- Giới thiệu: PDB là công cụ gỡ lỗi dòng lệnh mặc định của Python. PDB cho phép lập trình viên gỡ lỗi mã Python thông qua các lệnh trong terminal.
- Tính năng nổi bật:
  - Dừng chương trình tại một điểm dừng và thực hiện các lệnh kiểm tra (print các giá trị của biến, bước qua mã, tiếp tục thực thi, v.v.).
  - Có thể được tích hợp trực tiếp vào mã Python bằng cách sử dụng lệnh `import pdb`; `pdb.set_trace()`.

9.2.3. VS Code Debugger

- Giới thiệu: VS Code cung cấp một công cụ gỡ lỗi tích hợp với giao diện đồ họa, dễ sử dụng, hỗ trợ việc gỡ lỗi chương trình Python. Bạn chỉ cần đặt điểm dừng (breakpoints) trong mã và chạy chương trình để gỡ lỗi.
- Tính năng nổi bật:
  - Tích hợp chặt chẽ với các công cụ kiểm soát phiên bản như Git.
  - Hỗ trợ điều khiển quá trình gỡ lỗi thông qua giao diện người dùng trực quan.
  - Có thể theo dõi và thay đổi giá trị các biến trong suốt quá trình gỡ lỗi.

9.3. Quản lý thư viện và cài đặt (pip, venv)

Việc quản lý thư viện và môi trường cài đặt trong Python là rất quan trọng để duy trì tính ổn định và khả năng tái sử dụng của mã nguồn.

9.3.1. pip (Python Package Installer)

- Giới thiệu: pip là công cụ quản lý gói (package manager) mặc định của Python, giúp cài đặt và quản lý các thư viện (packages) từ Python Package Index (PyPI).
- Tính năng nổi bật:
  - Cài đặt và nâng cấp các thư viện Python với cú pháp đơn giản (`pip install <package>`).
  - Quản lý các gói đã cài đặt trong dự án và tạo các file yêu cầu cài đặt (`requirements.txt`).
  - Hỗ trợ cài đặt thư viện từ các nguồn khác ngoài PyPI như GitHub.

9.3.2. venv (Virtual Environment)

- Giới thiệu: venv là công cụ giúp tạo ra các môi trường ảo (virtual environment) trong Python, giúp cô lập các gói và thư viện cho từng dự án mà không ảnh hưởng đến hệ thống toàn cục.
- Tính năng nổi bật:
  - Giúp tạo một môi trường Python độc lập, ngăn không cho các gói xung đột với nhau giữa các dự án.
  - Có thể tạo một môi trường ảo với lệnh `python -m venv <directory>`.
  - Cho phép cài đặt các gói riêng biệt cho từng môi trường mà không thay đổi hệ thống Python chính.

9.3.3. Pipenv và Poetry

- Giới thiệu: Đây là các công cụ bổ sung giúp quản lý thư viện và môi trường ảo, đặc biệt trong các dự án phức tạp, giúp tự động tạo và quản lý các file yêu cầu cài đặt gói (Pipfile, pyproject.toml).
- Tính năng nổi bật:
  - Tích hợp pip và venv để dễ dàng quản lý môi trường và thư viện cài đặt.
  - Pipenv tự động tạo và quản lý các file `Pipfile` và `Pipfile.lock`.
  - Poetry cung cấp các tính năng quản lý dự án mạnh mẽ, bao gồm quản lý phiên bản và cài đặt thư viện.

Tóm tắt

Các công cụ hỗ trợ lập trình Python là những phần không thể thiếu giúp lập trình viên làm việc hiệu quả hơn. Các IDE như PyCharm, VS Code, và Jupyter Notebook cung cấp môi trường làm việc lý tưởng cho lập trình viên Python. Các công cụ gỡ lỗi như PyCharm Debugger, PDB, và VS Code Debugger giúp phát hiện và sửa lỗi nhanh chóng. Cuối cùng, việc quản lý thư viện và môi trường cài đặt thông qua `pip`, `venv`, và các công cụ như Pipenv và Poetry đảm bảo rằng các dự án Python luôn ổn định và dễ bảo trì.

Chương 10. Làm việc với tệp (File I/O)

Contents

Trong lập trình Python, việc làm việc với tệp (Input/Output – I/O) là một kỹ năng quan trọng để xử lý dữ liệu ngoài bộ nhớ, như đọc, ghi và quản lý tệp tin trên hệ thống. Python cung cấp một bộ công cụ mạnh mẽ để làm việc với tệp tin và thư mục, từ việc mở và đóng tệp đến quản lý các thao tác đọc/ghi với tệp văn bản và tệp nhị phân.

10.1. Mở và đóng tệp

Để làm việc với tệp trong Python, bạn cần mở tệp trước khi thực hiện các thao tác I/O và đóng tệp khi hoàn tất công việc để giải phóng tài nguyên hệ thống. Python sử dụng hàm open() để mở tệp và hàm close() để đóng tệp.

10.1.1. Mở tệp với hàm `open()`

Hàm `open()` được sử dụng để mở tệp. Cú pháp:

```
file = open("tên_tệp", "mode")
```

Trong đó:

- `"mode"`: chỉ định chế độ mở tệp (ví dụ: `"r"` cho đọc, `"w"` cho ghi, `"a"` cho ghi thêm dữ liệu, `"b"` cho tệp nhị phân, v.v.).

10.1.2. Chế độ mở tệp

Các chế độ mở tệp phổ biến bao gồm:

- `"r"`: Mở tệp để đọc (mặc định). Nếu tệp không tồn tại, sẽ gây lỗi.
- `"w"`: Mở tệp để ghi (nếu tệp đã tồn tại, sẽ ghi đè nội dung cũ).
- `"a"`: Mở tệp để ghi thêm dữ liệu vào cuối tệp.
- `"b"`: Chế độ nhị phân, dùng khi làm việc với tệp nhị phân (ví dụ: `"rb"`, `"wb"`).

10.1.3. Đóng tệp với hàm `close()`

Sau khi hoàn tất các thao tác I/O, bạn nên đóng tệp để bảo vệ tài nguyên:

```
file.close()
```

10.1.4. Sử dụng khối with để quản lý tệp

Thay vì sử dụng `open()` và `close()` thủ công, Python hỗ trợ cách tiếp cận an toàn và dễ dàng hơn bằng cách sử dụng câu lệnh `with`, giúp tự động đóng tệp khi thoát khỏi khối lệnh:

```
with open("tên_tệp", "r") as file:
    nội_dung = file.read()
# Tệp tự động được đóng khi thoát khỏi `with`
```

10.2. Đọc và ghi tệp văn bản

Khi làm việc với tệp văn bản, bạn có thể đọc và ghi dữ liệu vào tệp sử dụng các phương thức như `read()`, `readline()`, `write()` và `writelines()`.

10.2.1. Đọc tệp văn bản

- `read()`: Đọc toàn bộ nội dung tệp:

```
with open("tên_tệp", "r") as file:
    nội_dung = file.read()
```

- `readline()`: Đọc một dòng trong tệp:

```
with open("tên_tệp", "r") as file:
    dòng = file.readline()
```

- `readlines()`: Đọc toàn bộ tệp và trả về một danh sách các dòng:

```
with open("tên_tệp", "r") as file:
    dòng = file.readlines()
```

10.2.2. Ghi tệp văn bản

- `write()`: Ghi một chuỗi vào tệp:

```
with open("tên_tệp", "w") as file:
    file.write("Nội dung mới")
```

- `writelines()`: Ghi một danh sách các chuỗi vào tệp:

```
with open("tên_tệp", "w") as file:
    file.writelines(["Dòng 1\n", "Dòng 2\n", "Dòng 3\n"])
```

10.3 Làm việc với tệp nhị phân

Khi làm việc với dữ liệu không phải là văn bản, như hình ảnh, âm thanh hoặc tệp dữ liệu phức tạp, bạn sẽ cần làm việc với tệp nhị phân. Các tệp nhị phân được mở trong chế độ `"rb"` (đọc nhị phân) hoặc `"wb"` (ghi nhị phân).

10.3.1. Đọc tệp nhị phân

Khi đọc tệp nhị phân, nội dung tệp được trả về dưới dạng các byte:

```
with open("tệp_nhị_phân", "rb") as file:
    dữ_liệu = file.read()
```

10.3.2. Ghi tệp nhị phân

Khi ghi tệp nhị phân, bạn có thể ghi các byte vào tệp:

```
with open("tệp_nhị_phân", "wb") as file:
    file.write(dữ_liệu)
```

10.3.3. Sử dụng `struct` để làm việc với các kiểu dữ liệu nhị phân

Python cung cấp thư viện `struct` để thao tác với dữ liệu nhị phân phức tạp hơn, ví dụ như đọc và ghi các kiểu dữ liệu cơ bản (số nguyên, số thực, v.v.) dưới dạng nhị phân.

10.4. Quản lý tệp và thư mục

Ngoài việc mở và thao tác với tệp, Python cung cấp các công cụ để quản lý các tệp và thư mục trên hệ thống.

10.4.1. Thư viện `os`

Thư viện `os` cung cấp các phương thức để làm việc với tệp và thư mục:

- `os.rename(old_name, new_name)`: Đổi tên tệp hoặc thư mục.
- `os.remove(path)`: Xóa tệp.
- `os.rmdir(path)`: Xóa thư mục (chỉ khi thư mục rỗng).
- `os.mkdir(path)`: Tạo thư mục mới.

10.4.2. Thư viện `shutil`

Thư viện `shutil` cung cấp các phương thức cao cấp hơn để làm việc với tệp và thư mục:

- `shutil.copy(src, dst)`: Sao chép tệp từ vị trí `src` đến `dst`.
- `shutil.move(src, dst)`: Di chuyển tệp hoặc thư mục từ `src` đến `dst`.
- `shutil.rmtree(path)`: Xóa thư mục và toàn bộ nội dung của nó.

10.4.3. Duyệt thư mục với `os.walk()`

Phương thức `os.walk()` giúp duyệt qua các thư mục và tệp con trong một thư mục:

```
for root, dirs, files in os.walk("thư_mục"):
    for file in files:
        print(os.path.join(root, file))
```

10.4.4. Kiểm tra sự tồn tại của tệp/thư mục

Bạn có thể sử dụng các phương thức từ thư viện `os.path` để kiểm tra xem tệp hoặc thư mục có tồn tại không:

- `os.path.exists(path)`: Kiểm tra xem tệp/thư mục có tồn tại không.
- `os.path.isfile(path)`: Kiểm tra xem có phải là tệp không.
- `os.path.isdir(path)`: Kiểm tra xem có phải là thư mục không.

10.5. Bài tập thực hành

- **Mở và đóng tệp:**

1. Viết một chương trình mở một tệp văn bản (chế độ đọc) và in ra toàn bộ nội dung của tệp.
2. Viết một chương trình mở một tệp văn bản (chế độ ghi) và ghi một dòng văn bản vào tệp.
3. Viết một chương trình sử dụng câu lệnh `with` để mở tệp, đọc dữ liệu và tự động đóng tệp khi hoàn thành.

- **Đọc và ghi tệp văn bản:**

1. Viết một chương trình đọc tệp văn bản và in ra nội dung của từng dòng (sử dụng `readline()`).
2. Viết một chương trình đọc toàn bộ nội dung của tệp văn bản (sử dụng `read()`).
3. Viết một chương trình ghi một danh sách các dòng vào tệp văn bản (sử dụng `writelines()`).
4. Viết một chương trình thêm một dòng mới vào tệp (sử dụng chế độ `"a"`).

- **Làm việc với tệp nhị phân:**

1. Viết một chương trình đọc một tệp nhị phân và in ra các byte của tệp (sử dụng `rb`).
2. Viết một chương trình ghi một chuỗi byte vào tệp nhị phân (sử dụng `wb`).
3. Viết một chương trình sao chép một tệp nhị phân từ một thư mục này sang thư mục khác.

- **Quản lý tệp và thư mục:**

1. Viết một chương trình để kiểm tra xem một tệp có tồn tại hay không (sử dụng `os.path.exists()`).
2. Viết một chương trình tạo một thư mục mới (sử dụng `os.mkdir()`).
3. Viết một chương trình xóa một tệp (sử dụng `os.remove()`).
4. Viết một chương trình sao chép một tệp từ thư mục này sang thư mục khác (sử dụng `shutil.copy()`).
5. Viết một chương trình di chuyển một tệp từ thư mục này sang thư mục khác (sử dụng `shutil.move()`).

- **Duyệt thư mục:**

1. Viết một chương trình duyệt qua các tệp trong một thư mục và in ra đường dẫn đầy đủ của chúng (sử dụng `os.walk()`).
2. Viết một chương trình liệt kê tất cả các thư mục con trong một thư mục gốc.

Chương 11. Thư viện chuẩn của Python

Contents

Python cung cấp một bộ thư viện chuẩn phong phú, giúp lập trình viên thực hiện nhiều tác vụ khác nhau mà không cần phải cài đặt thêm thư viện bên ngoài. Các thư viện này bao gồm các công cụ toán học, làm việc với thời gian, quản lý tệp và hệ thống, sinh số ngẫu nhiên, và xử lý dữ liệu cấu trúc. Dưới đây là một số thư viện chuẩn quan trọng trong Python:

11.1. Thư viện `math`

Thư viện `math` cung cấp các hàm toán học cơ bản và các hằng số toán học hữu ích, giúp thực hiện các phép toán phức tạp hơn mà không cần phải triển khai thủ công.

- **Các hàm toán học cơ bản**:
  - `math.sqrt(x)` - Tính căn bậc hai của một số.
  - `math.pow(x, y)` - Tính x mũ y.
  - `math.fabs(x)` - Trả về giá trị tuyệt đối của x.
  - `math.factorial(x)` - Tính giai thừa của x.

- `math.log(x, base)` - Tính logarithm của x với cơ số `base` (logarithm tự nhiên nếu không chỉ định cơ số).
- `math.sin(x)`, `math.cos(x)`, `math.tan(x)` - Các hàm lượng giác.

- **Hằng số**:
  - `math.pi` - Giá trị của π (khoảng 3.14159).
  - `math.e` - Hằng số Euler (khoảng 2.718).

Ví dụ sử dụng thư viện `math`:

```
import math
print(math.sqrt(16))  # In ra 4.0
print(math.factorial(5))  # In ra 120
```

11.2. Thư viện `datetime`

Thư viện `datetime` cho phép lập trình viên làm việc với ngày tháng và thời gian. Nó cung cấp các lớp và hàm để xử lý ngày giờ, tính toán các khoảng thời gian, và định dạng ngày giờ theo cách linh hoạt.

- **Các lớp chính trong `datetime`:**
  - `datetime.date`: Đại diện cho ngày (năm, tháng, ngày).
  - `datetime.time`: Đại diện cho thời gian (giờ, phút, giây, mili giây).
  - `datetime.datetime`: Đại diện cho cả ngày và thời gian.
  - `datetime.timedelta`: Đại diện cho khoảng thời gian giữa hai đối tượng datetime.
- **Các phương thức hữu ích:**
  - `datetime.now()` - Trả về ngày và giờ hiện tại.
  - `datetime.today()` - Trả về ngày hiện tại.
  - `datetime.strptime(date_string, format)` - Chuyển đổi chuỗi thành đối tượng datetime.
  - `datetime.strftime(format)` - Định dạng đối tượng datetime thành chuỗi.

Ví dụ sử dụng thư viện `datetime`:

```
import datetime
now = datetime.datetime.now()
print(now)  # In ra ngày và giờ hiện tại
print(now.strftime("%Y-%m-%d %H:%M:%S"))  # Định dạng ngày giờ
```

11.3. Thư viện `os` và `sys`

- `os`: Thư viện `os` cung cấp các phương thức để làm việc với hệ điều hành, giúp tương tác với tệp, thư mục, và môi trường hệ thống.
  - `os.getcwd()` - Lấy thư mục hiện tại.
  - `os.listdir(path)` - Lấy danh sách các tệp và thư mục trong thư mục path.
  - `os.mkdir(path)` - Tạo thư mục mới.
  - `os.remove(path)` - Xóa tệp.
  - `os.path.join(path, name)` - Kết hợp đường dẫn và tên tệp.
- `sys`: Thư viện `sys` cung cấp các phương thức để tương tác với hệ điều hành thông qua Python.
  - `sys.argv` - Danh sách các tham số dòng lệnh.
  - `sys.exit()` - Thoát khỏi chương trình.
  - `sys.path` - Danh sách các thư mục chứa các mô-đun Python.

Ví dụ sử dụng thư viện `os` và `sys`:

```
import os
import sys

print(os.getcwd())  # In ra thư mục hiện tại
print(sys.argv)  # In ra các tham số dòng lệnh
```

11.4. Thư viện `random`

Thư viện `random` cung cấp các hàm để sinh số ngẫu nhiên và lựa chọn ngẫu nhiên trong các dãy số, chuỗi, hoặc danh sách.

- Các hàm phổ biến trong `random`:
  - `random.randint(a, b)` - Sinh một số nguyên ngẫu nhiên trong khoảng từ a đến b.

- `random.random()` - Sinh một số thực ngẫu nhiên từ 0 đến 1.
- `random.choice(seq)` - Lựa chọn một phần tử ngẫu nhiên từ danh sách `seq`.
- `random.shuffle(seq)` - Trộn ngẫu nhiên các phần tử trong danh sách `seq`.

Ví dụ sử dụng thư viện `random`:

```
import random
print(random.randint(1, 10))  # In ra một số ngẫu nhiên từ 1 đến 10
print(random.choice([1, 2, 3, 4, 5]))  # Lựa chọn một số ngẫu nhiên trong danh sách
```

11.5. Thư viện `collections`

Thư viện `collections` cung cấp các kiểu dữ liệu mở rộng và các cấu trúc dữ liệu hữu ích hơn so với các kiểu dữ liệu cơ bản trong Python như `list`, `dict`, `set`.

- Các kiểu dữ liệu trong `collections`:
  - `Counter`: Đếm số lần xuất hiện của các phần tử trong một dãy.
  - `defaultdict`: Một dạng mở rộng của từ điển (`dict`), cho phép gán giá trị mặc định cho khóa chưa có trong từ điển.
  - `namedtuple`: Cung cấp một lớp con của tuple với các trường có tên.
  - `deque`: Một danh sách có thể thay đổi, cho phép chèn và loại bỏ phần tử ở cả hai đầu.

Ví dụ sử dụng thư viện `collections`:

```
from collections import Counter

data = ['a', 'b', 'c', 'a', 'b', 'a']
counter = Counter(data)
print(counter)  # In ra: Counter({'a': 3, 'b': 2, 'c': 1})
```

11.6. Thư viện `json`

Thư viện `json` cung cấp các phương thức để chuyển đổi giữa dữ liệu Python và định dạng JSON (JavaScript Object Notation), một định dạng phổ biến để lưu trữ và trao đổi dữ liệu.

Các hàm chính trong `json`:

- `json.dumps(obj)` - Chuyển đổi đối tượng Python thành chuỗi JSON.
- `json.loads(s)` - Chuyển đổi chuỗi JSON thành đối tượng Python.
- `json.dump(obj, file)` - Ghi đối tượng Python vào tệp dưới dạng JSON.
- `json.load(file)` - Đọc dữ liệu JSON từ tệp và chuyển thành đối tượng Python.

Ví dụ sử dụng thư viện `json`:

```
import json

# Chuyển đổi tượng Python thành chuỗi JSON
data = {'name': 'Alice', 'age': 25}
json_data = json.dumps(data)
print(json_data)  # In ra: '{"name": "Alice", "age": 25}'

# Chuyển chuỗi JSON thành đối tượng Python
parsed_data = json.loads(json_data)
print(parsed_data)  # In ra: {'name': 'Alice', 'age': 25}
```

11.7. Bài tập thực hành

- **Thực hành với thư viện `math`:**

1. Tính căn bậc hai: Viết một chương trình nhận vào một số và tính căn bậc hai của nó bằng cách sử dụng `math.sqrt()`.
2. Tính giai thừa: Viết một chương trình nhận vào một số nguyên và tính giai thừa của nó bằng cách sử dụng `math.factorial()`.
3. Tính giá trị tuyệt đối: Viết một chương trình nhận vào một số và in ra giá trị tuyệt đối của nó bằng cách sử dụng `math.fabs()`.
4. Tính số mũ: Viết một chương trình nhận vào hai số và tính số mũ (x^y) của chúng bằng cách sử dụng `math.pow()`.
5. Tính sin, cos, tan: Viết một chương trình nhận vào một góc (đơn vị độ) và tính các giá trị sin, cos, tan của góc đó.

- **Thực hành với thư viện `datetime`:**

1. In ngày giờ hiện tại: Viết một chương trình in ra ngày giờ hiện tại sử dụng `datetime.now()`.
2. Định dạng ngày giờ: Viết một chương trình in ngày giờ hiện tại theo định dạng "Ngày-Tháng-Năm Giờ:Phút:Giây".

3. Chuyển chuỗi thành ngày: Viết một chương trình nhận vào một chuỗi ngày theo định dạng "DD/MM/YYYY" và chuyển nó thành đối tượng `datetime.date` bằng `datetime.strptime()`.
4. Tính toán khoảng thời gian: Viết một chương trình sử dụng `datetime.timedelta()` để tính toán khoảng thời gian giữa hai ngày.

- **Thực hành với thư viện `os` và `sys`:**

1. Lấy thư mục hiện tại: Viết một chương trình in ra thư mục làm việc hiện tại bằng cách sử dụng `os.getcwd()`.
2. Liệt kê tệp trong thư mục: Viết một chương trình liệt kê tất cả các tệp trong một thư mục bằng cách sử dụng `os.listdir()`.
3. Kiểm tra tồn tại của tệp: Viết một chương trình kiểm tra xem một tệp có tồn tại hay không bằng cách sử dụng `os.path.exists()`.
4. Sử dụng tham số dòng lệnh: Viết một chương trình nhận các tham số dòng lệnh và in ra giá trị của chúng sử dụng `sys.argv`.

- **Thực hành với thư viện `random`:**

1. Sinh số ngẫu nhiên: Viết một chương trình sinh một số nguyên ngẫu nhiên trong khoảng từ 1 đến 100 sử dụng `random.randint()`.
2. Lựa chọn ngẫu nhiên từ danh sách: Viết một chương trình lựa chọn ngẫu nhiên một phần tử từ danh sách sử dụng `random.choice()`.
3. Trộn ngẫu nhiên danh sách: Viết một chương trình trộn ngẫu nhiên các phần tử trong một danh sách sử dụng `random.shuffle()`.

- **Thực hành với thư viện `collections`:**

1. Đếm số lần xuất hiện của các phần tử: Viết một chương trình đếm số lần xuất hiện của các phần tử trong một danh sách sử dụng `Counter`.
2. Sử dụng `defaultdict`: Viết một chương trình sử dụng defaultdict để tạo một từ điển với giá trị mặc định là 0. Sau đó, tăng giá trị của các khóa trong từ điển.
3. Sử dụng `namedtuple`: Viết một chương trình tạo một `namedtuple` để lưu trữ thông tin sinh viên gồm tên, tuổi và điểm, sau đó tạo một đối tượng sinh viên và in ra thông tin của họ.

- **Thực hành với thư viện `json`:**

1. Chuyển đổi đối tượng Python thành JSON: Viết một chương trình chuyển một từ điển Python thành chuỗi JSON sử dụng `json.dumps()`.

2. Chuyển đổi JSON thành đối tượng Python: Viết một chương trình chuyển chuỗi JSON thành đối tượng Python sử dụng `json.loads()`.
3. Ghi và đọc dữ liệu JSON từ tệp: Viết một chương trình ghi một đối tượng Python vào tệp dưới dạng JSON bằng `json.dump()`, và sau đó đọc dữ liệu từ tệp đó sử dụng `json.load()`.

Chương 12. Lập trình hướng đối tượng (OOP)

Contents

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một phương pháp lập trình tổ chức mã nguồn thành các đối tượng (objects) và lớp (classes). OOP giúp quản lý mã nguồn dễ dàng hơn, cải thiện khả năng tái sử dụng và bảo trì phần mềm. Trong Python, OOP là một khái niệm quan trọng, và ngôn ngữ này hỗ trợ đầy đủ các tính năng của lập trình hướng đối tượng, bao gồm lớp, đối tượng, kế thừa, đa hình, đóng gói và trừu tượng hóa.

12.1. Các khái niệm cơ bản về OOP

Các khái niệm cơ bản của OOP bao gồm:

- **Lớp (Class):** Là một khuôn mẫu hoặc bản thiết kế để tạo ra các đối tượng. Lớp định nghĩa các thuộc tính (data) và phương thức (functions) chung cho các đối tượng.
- **Đối tượng (Object):** Là một thể hiện cụ thể của lớp. Mỗi đối tượng có thể có trạng thái (giá trị của thuộc tính) và hành vi (phương thức).
- **Thuộc tính (Attributes):** Là dữ liệu hoặc thông tin mà mỗi đối tượng chứa. Thuộc tính có thể là biến của lớp hoặc đối tượng.

- **Phương thức (Methods):** Là các hàm được định nghĩa trong lớp, mô tả hành vi hoặc các thao tác mà đối tượng có thể thực hiện.
- **Kế thừa (Inheritance):** Cho phép một lớp con kế thừa các thuộc tính và phương thức của lớp cha, giúp tái sử dụng mã và mở rộng chức năng.
- **Đa hình (Polymorphism):** Cho phép các đối tượng thuộc các lớp khác nhau có thể sử dụng cùng một phương thức nhưng thực thi theo cách khác nhau.
- **Đóng gói (Encapsulation):** Ẩn đi các chi tiết cài đặt bên trong lớp và chỉ cho phép truy cập thông qua các phương thức công khai.
- **Trừu tượng hóa (Abstraction):** Ẩn bớt các chi tiết phức tạp của hệ thống và chỉ tập trung vào những khía cạnh quan trọng, cung cấp một giao diện đơn giản hơn cho người dùng.

12.2. Lớp và đối tượng trong Python

- **Lớp (Class)** trong Python được định nghĩa bằng từ khóa `class`. Lớp có thể chứa các thuộc tính và phương thức. Một đối tượng được tạo từ lớp thông qua việc gọi tên lớp như một hàm.

Cú pháp định nghĩa lớp:

```
class MyClass:
    def __init__(self, name):
        self.name = name  # Thuộc tính

    def greet(self):
        return f"Hello, {self.name}!"  # Phương thức
```

- **Đối tượng (Object):** Khi lớp đã được định nghĩa, bạn có thể tạo một đối tượng (instance) từ lớp đó:

```
obj = MyClass("Alice")
print(obj.greet())  # In ra "Hello, Alice!"
```

Trong ví dụ trên:

- `MyClass` là lớp.
- `obj` là đối tượng của lớp `MyClass`.

12.3. Thuộc tính và phương thức

- **Thuộc tính (Attributes):** Là các biến trong lớp, dùng để lưu trữ dữ liệu của đối tượng.
  - **Thuộc tính thể hiện (Instance attributes):** Là các thuộc tính của một đối tượng cụ thể. Chúng được gán thông qua từ khóa self.
  - **Thuộc tính lớp (Class attributes):** Là các thuộc tính chung cho tất cả các đối tượng của lớp và không thay đổi khi tạo đối tượng mới.
- **Phương thức (Methods):** Là các hàm định nghĩa trong lớp, thực hiện các hành động trên đối tượng hoặc lớp.
  - **Phương thức thể hiện (Instance methods):** Là các phương thức hoạt động trên các thuộc tính thể hiện và được gọi thông qua đối tượng.
  - **Phương thức lớp (Class methods):** Là các phương thức hoạt động trên lớp thay vì đối tượng cụ thể, được khai báo với @classmethod.
  - **Phương thức tĩnh (Static methods):** Là các phương thức không phụ thuộc vào đối tượng hay lớp, được khai báo với @staticmethod.

Ví dụ:

```
class Dog:
    species = "Canis familiaris"  # Thuộc tính lớp

    def __init__(self, name, age):
        self.name = name  # Thuộc tính thể hiện
        self.age = age

    def description(self):  # Phương thức thể hiện
        return f"{self.name} is {self.age} years old."

    @classmethod
    def species_info(cls):  # Phương thức lớp
        return f"All dogs are of species {cls.species}."

    @staticmethod
    def sound():  # Phương thức tĩnh
        return "Woof!"
```

12.4. Kế thừa và đa hình

- **Kế thừa (Inheritance):** Cho phép một lớp con kế thừa các thuộc tính và phương thức từ lớp cha, giúp tái sử dụng mã và mở rộng chức năng mà không cần viết lại mã cũ.

Cú pháp kế thừa:

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):  # Lớp Dog kế thừa lớp Animal
    def speak(self):
        print("Woof!")

dog = Dog()
dog.speak()  # In ra "Woof!"
```

Trong ví dụ trên, lớp `Dog` kế thừa từ lớp `Animal`, và phương thức `speak()` trong Dog ghi đè phương thức `speak()` của lớp cha `Animal`, thể hiện tính đa hình (polymorphism).

- **Đa hình (Polymorphism):** Là khả năng các đối tượng của các lớp khác nhau có thể thực hiện phương thức cùng tên nhưng với cách thực thi khác nhau.

Ví dụ về đa hình:

```
class Cat(Animal):
    def speak(self):
        print("Meow!")

animals = [Dog(), Cat()]
for animal in animals:
    animal.speak()  # In ra "Woof!" và "Meow!"
```

12.5. Đóng gói và trừu tượng hóa

- **Đóng gói (Encapsulation):** Là kỹ thuật ẩn các chi tiết thực thi bên trong lớp và chỉ cho phép truy cập thông qua các phương thức công khai. Đóng gói giúp bảo vệ dữ liệu của lớp khỏi bị thay đổi một cách trái phép.
  - **Thuộc tính riêng tư:** Thuộc tính hoặc phương thức bắt đầu bằng dấu gạch dưới (_) hoặc hai gạch dưới (__) để chỉ ra rằng chúng không nên bị truy cập trực tiếp từ ngoài lớp.

Ví dụ đóng gói:

```python
class Person:
    def __init__(self, name):
        self._name = name  # Thuộc tính riêng tư

    def get_name(self):  # Phương thức công khai
        return self._name

person = Person("John")
print(person.get_name())  # In ra "John"
```

- **Trừu tượng hóa (Abstraction):** Là quá trình ẩn chi tiết cài đặt và chỉ cung cấp các giao diện đơn giản cho người sử dụng. Trừu tượng hóa giúp người lập trình tập trung vào các vấn đề quan trọng mà không phải lo lắng về cách thức hoạt động bên trong.

Trong Python, trừu tượng hóa có thể được thực hiện thông qua các lớp trừu tượng (abstract classes) và phương thức trừu tượng (abstract methods) bằng cách sử dụng mô-đun abc (Abstract Base Classes).

12.6. Các lớp đặc biệt trong Python (dunder methods)

Các phương thức đặc biệt, còn gọi là dunder methods (double underscore methods), là các phương thức có tên bắt đầu và kết thúc bằng dấu gạch dưới đôi (__). Chúng cho phép bạn định nghĩa cách thức hoạt động của các phép toán và hành vi đặc biệt của đối tượng.

Một số phương thức đặc biệt thông dụng:

- `__init__(self)` - Hàm khởi tạo, được gọi khi tạo đối tượng.
- `__str__(self)` - Định nghĩa cách thức hiển thị đối tượng dưới dạng chuỗi khi sử dụng `print()` hoặc `str()`.
- `__repr__(self)` - Định nghĩa cách thức hiển thị đối tượng khi gọi `repr()`.
- `__add__(self, other)` - Định nghĩa phép cộng đối với hai đối tượng.
- `__len__(self)` - Định nghĩa cách tính độ dài đối tượng khi sử dụng `len()`.

Ví dụ sử dụng dunder methods:

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"Point({self.x}, {self.y})"

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

p1 = Point(1, 2)
p2 = Point(3, 4)
print(p1)  # In ra "Point(1, 2)"
p3 = p1 + p2  # Sử dụng phương thức __add__
print(p3)  # In ra "Point(4, 6)"
```

Tóm tắt

Lập trình hướng đối tượng là một phương pháp mạnh mẽ và phổ biến trong Python, giúp tổ chức mã nguồn và tăng cường tính linh hoạt của phần mềm. Việc hiểu và áp dụng các khái niệm như lớp, đối tượng, kế thừa, đa hình, đóng gói, trừu tượng hóa, và các phương thức đặc biệt là rất quan trọng để phát triển các ứng dụng phức tạp và dễ bảo trì.

Chương 13. Quản lý bộ nhớ và tối ưu hóa

Contents

Quản lý bộ nhớ và tối ưu hóa là những yếu tố quan trọng trong lập trình Python, giúp cải thiện hiệu suất và giảm thiểu sự tiêu tốn tài nguyên hệ thống. Python cung cấp các công cụ và kỹ thuật để lập trình viên có thể tối ưu hóa bộ nhớ và hiệu suất của ứng dụng. Mục này sẽ tập trung vào các phương pháp quản lý bộ nhớ, tối ưu hóa hiệu suất và sử dụng các cấu trúc dữ liệu sao cho hợp lý.

13.1. Garbage Collection

Garbage Collection (GC) trong Python là cơ chế tự động quản lý bộ nhớ, giúp giải phóng bộ nhớ mà không còn được sử dụng nữa, qua đó giảm thiểu rủi ro của việc rò rỉ bộ nhớ (memory leaks). Python sử dụng **Garbage Collector** để theo dõi các đối tượng không còn được tham chiếu đến và tự động thu hồi bộ nhớ của chúng.

- **Cơ chế hoạt động:**
  - Python sử dụng tham chiếu đếm (reference counting) để theo dõi số lượng tham chiếu đến một đối tượng. Khi số lượng tham chiếu bằng 0, đối tượng sẽ được xóa và bộ nhớ sẽ được giải phóng.
  - Tuy nhiên, tham chiếu vòng (circular references) (khi các đối tượng tham chiếu lẫn nhau) có thể không bị phát hiện và giải phóng thông qua tham chiếu đếm. Để xử lý

tình huống này, Python sử dụng một bộ thu gom rác bổ sung, sử dụng thuật toán generational garbage collection.

- **Bộ thu gom rác:**
  - Python phân chia bộ nhớ thành ba thế hệ (generations):
    - **Thế hệ 0:** Chứa các đối tượng mới được tạo ra.
    - **Thế hệ 1:** Chứa các đối tượng sống sót qua một chu kỳ thu gom rác.
    - **Thế hệ 2:** Chứa các đối tượng sống sót qua nhiều chu kỳ thu gom rác.
  - Các đối tượng trong **thế hệ 0** sẽ được kiểm tra và thu hồi bộ nhớ khi có sự kiện thu gom rác diễn ra. Các đối tượng sống sót sẽ được di chuyển sang thế hệ cao hơn.
- **Quản lý Garbage Collection:**
  - Thư viện `gc` trong Python cho phép lập trình viên điều khiển việc thu gom rác, như kích hoạt hoặc tắt quá trình thu gom, theo dõi các đối tượng không còn được sử dụng, và dọn dẹp bộ nhớ khi cần thiết.

Ví dụ sử dụng thư viện `gc`:

```
import gc

# Bật / Tắt garbage collection
gc.enable()  # Bật
gc.disable()  # Tắt

# Kiểm tra các đối tượng không còn tham chiếu
gc.collect()  # Thu gom rác thủ công
```

13.2. Các kỹ thuật tối ưu hóa hiệu suất

Tối ưu hóa hiệu suất là quá trình cải thiện tốc độ và hiệu quả sử dụng tài nguyên của ứng dụng. Trong Python, có một số kỹ thuật tối ưu hóa mà lập trình viên có thể sử dụng để tăng hiệu suất của chương trình:

- **Tránh việc tạo và xóa đối tượng quá thường xuyên:** Việc tạo và xóa các đối tượng có thể gây tốn kém về mặt hiệu suất. Cần hạn chế việc sử dụng quá nhiều đối tượng tạm thời và thay vào đó, tái sử dụng chúng.
- **Sử dụng các cấu trúc dữ liệu hiệu quả:** Cấu trúc dữ liệu như danh sách (`list`), từ điển (`dict`), và tập hợp (`set`) có sự khác biệt rõ rệt về hiệu suất. Cần chọn đúng cấu trúc dữ

liệu phù hợp với yêu cầu công việc (ví dụ: sử dụng `set` để kiểm tra sự tồn tại của phần tử nhanh hơn `list`).

- **Tránh việc sử dụng vòng lặp không cần thiết:** Các vòng lặp quá dài hoặc lặp lại nhiều lần có thể làm giảm tốc độ của chương trình. Cần tối ưu hóa vòng lặp, ví dụ, bằng cách giảm thiểu số lần lặp không cần thiết hoặc sử dụng các công cụ như **list comprehensions** thay vì vòng lặp truyền thống.
- **Sử dụng thư viện chuẩn và thư viện bên ngoài hiệu quả:** Các thư viện như **NumPy**, **Pandas**, **Cython** hay **multiprocessing** có thể giúp tối ưu hóa hiệu suất trong các tác vụ tính toán nặng hoặc khi cần xử lý đồng thời (concurrent processing).
- **Profile mã nguồn:** Sử dụng công cụ **cProfile** hoặc **timeit** để xác định các điểm tắc nghẽn (bottleneck) trong mã và tối ưu hóa chúng.

Ví dụ tối ưu hóa với `timeit`:

```
import timeit

# Thực hiện đo thời gian một đoạn mã
execution_time = timeit.timeit('x = sum(range(100))', number=1000)
print(f"Thời gian thực thi: {execution_time} giây")
```

- **Sử dụng các kỹ thuật biên dịch và tối ưu hóa mã máy:**
  - **Cython:** Biên dịch mã Python sang mã C để tăng tốc độ thực thi.
  - **PyPy:** Sử dụng trình thông dịch Python thay thế PyPy, một trình thông dịch JIT (Just-In-Time compiler) giúp cải thiện hiệu suất cho các chương trình Python dài hạn.

13.3. Sử dụng các cấu trúc dữ liệu tối ưu

Python cung cấp nhiều cấu trúc dữ liệu khác nhau với hiệu suất và tính năng tối ưu cho từng trường hợp sử dụng. Việc chọn đúng cấu trúc dữ liệu là rất quan trọng trong việc tối ưu hóa bộ nhớ và hiệu suất.

- **Danh sách (`List`):**
  - Là cấu trúc dữ liệu cơ bản trong Python. Tuy nhiên, việc thao tác với danh sách (như thêm hoặc xóa phần tử ở đầu danh sách) có thể tốn thời gian.
  - Tối ưu: Sử dụng **list comprehensions** thay vì vòng lặp for truyền thống để cải thiện hiệu suất.
- **Từ điển (`Dictionary`):**

- Cung cấp khả năng tra cứu nhanh chóng thông qua các khóa. Việc thêm, xóa, và tìm kiếm một khóa trong từ điển có độ phức tạp thời gian là O(1) trung bình.
- Tối ưu: Sử dụng từ điển cho các tác vụ tra cứu dữ liệu thay vì danh sách khi cần kiểm tra sự tồn tại của phần tử.

- **Tập hợp (`Set`):**
  - Là một cấu trúc dữ liệu không chứa các phần tử trùng lặp và hỗ trợ các phép toán tập hợp nhanh chóng (như giao, hợp, và hiệu).
  - Tối ưu: Sử dụng `set` khi cần kiểm tra sự tồn tại của phần tử nhanh hơn danh sách.
- **Hàng đợi (`Queue`) và Ngăn xếp (`Stack`):**
  - Python không có cấu trúc dữ liệu hàng đợi và ngăn xếp trực tiếp, nhưng có thể sử dụng `collections.deque` để thay thế. Đây là một cấu trúc dữ liệu hiệu quả cho việc thêm và xóa phần tử từ hai đầu danh sách.

Ví dụ sử dụng `deque`:

```
from collections import deque

queue = deque([1, 2, 3])
queue.append(4)  # Thêm phần tử vào cuối hàng đợi
queue.popleft()  # Lấy phần tử ra khỏi đầu hàng đợi
```

- Tối ưu hóa bộ nhớ với các cấu trúc dữ liệu đặc biệt:
  - `array` **trong mô-đun** `array`: Cung cấp cấu trúc dữ liệu mảng, phù hợp với các phép toán số học và tiết kiệm bộ nhớ khi so với danh sách.
  - `heapq`: Thư viện này cung cấp một cấu trúc dữ liệu **heap** cho phép bạn quản lý các phần tử theo thứ tự ưu tiên.

Ví dụ sử dụng `heapq`:

```
import heapq

heap = []
heapq.heappush(heap, 10)
heapq.heappush(heap, 20)
heapq.heappush(heap, 5)
print(heapq.heappop(heap))  # In ra 5 (phần tử nhỏ nhất)
```

Tóm tắt

Quản lý bộ nhớ và tối ưu hóa hiệu suất là yếu tố quan trọng trong lập trình Python, đặc biệt khi làm việc với các ứng dụng quy mô lớn. Hiểu và áp dụng các kỹ thuật như garbage collection, tối ưu hóa mã nguồn, và sử dụng các cấu trúc dữ liệu phù hợp sẽ giúp tăng hiệu quả hoạt động của chương trình. Việc chọn cấu trúc dữ liệu tối ưu và tối thiểu hóa việc sử dụng tài nguyên là những kỹ năng cần thiết để phát triển ứng dụng hiệu quả trong Python.

Chương 14. Giới thiệu về lập trình với thư viện bên ngoài

Contents

Trong Python, việc sử dụng các thư viện bên ngoài là một phần quan trọng giúp mở rộng khả năng lập trình và giải quyết các bài toán phức tạp một cách hiệu quả. Các thư viện này không chỉ giúp tăng tốc quá trình phát triển phần mềm mà còn cung cấp các công cụ mạnh mẽ để xử lý các nhiệm vụ như phân tích dữ liệu, trực quan hóa, tính toán khoa học, và giao tiếp với các dịch vụ web. Mục này sẽ giới thiệu về cách cài đặt và sử dụng thư viện bên ngoài trong Python, cũng như các thư viện phổ biến và cách làm việc với API web.

14.1. Cài đặt và sử dụng thư viện với `pip``

`pip` (Python Package Installer) là công cụ tiêu chuẩn dùng để cài đặt và quản lý các thư viện và gói phần mềm trong Python. Việc cài đặt các thư viện bên ngoài thông qua pip là rất đơn giản và nhanh chóng, giúp lập trình viên có thể tiếp cận các thư viện mạnh mẽ để mở rộng chức năng của Python.

- **Cài đặt thư viện với `pip`:**
  - Để cài đặt một thư viện, bạn chỉ cần sử dụng lệnh `pip install <tên_thư_viện>` trong terminal hoặc command prompt. Ví dụ:

```
pip install numpy
```

- Sau khi cài đặt, bạn có thể nhập thư viện vào trong mã Python bằng câu lệnh `import`:

```
import numpy as np
```

- **Cài đặt phiên bản cụ thể:**
  - Bạn có thể cài đặt một phiên bản cụ thể của thư viện bằng cách chỉ định phiên bản sau tên thư viện:

```
pip install numpy==1.21.2
```

- **Cập nhật thư viện:**
  - Để cập nhật một thư viện lên phiên bản mới nhất, sử dụng lệnh:

```
pip install --upgrade numpy
```

- **Xóa thư viện:**
  - Nếu bạn không còn sử dụng thư viện nào, có thể gỡ bỏ nó bằng lệnh:

```
pip uninstall numpy
```

- **Quản lý các gói trong môi trường ảo (virtual environment):**
  - Để tránh xung đột giữa các thư viện trong các dự án khác nhau, bạn có thể sử dụng môi trường ảo (virtual environment) để quản lý các thư viện.
  - Để tạo một môi trường ảo, sử dụng lệnh:

```
python -m venv myenv
```

  - Sau đó, kích hoạt môi trường ảo và cài đặt thư viện trong đó:

```
source myenv/bin/activate  # Trên macOS/Linux
myenv\Scripts\activate  # Trên Windows
pip install numpy
```

14.2. Các thư viện phổ biến trong Python: numpy, pandas, matplotlib

Python có rất nhiều thư viện mạnh mẽ được sử dụng phổ biến trong các lĩnh vực như khoa học dữ liệu, phân tích dữ liệu, và trực quan hóa. Sau đây là ba thư viện rất phổ biến mà mọi lập trình viên Python nên biết:

- **Numpy:**
  - **Numpy** là thư viện cơ bản cho tính toán khoa học với Python, cung cấp cấu trúc mảng (array) mạnh mẽ và các phép toán số học nhanh chóng.
  - Nó cho phép bạn thực hiện các phép toán vector hóa và làm việc với mảng đa chiều, hỗ trợ các phép toán ma trận, tính toán thống kê, và nhiều hàm số học khác.
  - **Ví dụ sử dụng Numpy:**

```
import numpy as np
# Tạo một mảng numpy
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
# Phép cộng mảng
c = a + b
print(c)  # [6 8 10 12]
```

- **Pandas:**
  - **Pandas** là thư viện rất mạnh mẽ để xử lý và phân tích dữ liệu. Nó cung cấp hai cấu trúc dữ liệu chính là `Series` và `DataFrame` giúp dễ dàng xử lý dữ liệu bảng (dataframe) và chuỗi (series).
  - Pandas giúp bạn đọc và ghi dữ liệu từ các định dạng tệp khác nhau (CSV, Excel, SQL), lọc, nhóm, và xử lý dữ liệu một cách dễ dàng.
  - **Ví dụ sử dụng Pandas:**

```
import pandas as pd
# Tạo một DataFrame từ một từ điển
data = {'Tên': ['Alice', 'Bob', 'Charlie'], 'Tuổi': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

Kết quả

```
    Tên  Tuổi
0  Alice    25
1    Bob    30
2 Charlie    35
```

- **Matplotlib:**
  - Matplotlib là thư viện phổ biến nhất trong Python để vẽ đồ thị và trực quan hóa dữ liệu. Thư viện này cho phép bạn tạo các loại biểu đồ khác nhau như đồ thị đường, đồ thị cột, histograms, scatter plots, và nhiều loại khác.
  - **Ví dụ sử dụng Matplotlib:**

```
import matplotlib.pyplot as plt
# Dữ liệu
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
# Vẽ đồ thị
plt.plot(x, y)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Biểu đồ đơn giản')
plt.show()
```

14.3. Làm việc với các API web

Các API web (Application Programming Interface) là một cách quan trọng để ứng dụng Python giao tiếp với các dịch vụ bên ngoài như các ứng dụng web, cơ sở dữ liệu từ xa, hoặc các dịch vụ đám mây. Thư viện `requests` trong Python giúp bạn dễ dàng gửi yêu cầu HTTP đến các API và nhận dữ liệu phản hồi từ chúng.

- **Cài đặt thư viện requests:** Đầu tiên, bạn cần cài đặt thư viện `requests` bằng pip:

```
pip install requests
```

- **Gửi yêu cầu GET đến API:** Bạn có thể sử dụng phương thức get của thư viện requests để gửi yêu cầu đến API và nhận dữ liệu.

```
import requests
url = 'https://jsonplaceholder.typicode.com/posts'
response = requests.get(url)
# Hiển thị dữ liệu trả về từ API
print(response.json())
```

- **Gửi yêu cầu POST đến API:** Bạn cũng có thể gửi yêu cầu POST để gửi dữ liệu đến API, ví dụ như khi gửi dữ liệu biểu mẫu (form) hoặc JSON:

```
url = 'https://jsonplaceholder.typicode.com/posts'
data = {'title': 'foo', 'body': 'bar', 'userId': 1}
response = requests.post(url, json=data)
print(response.json())  # Dữ liệu phản hồi từ API
```

- **Xử lý phản hồi từ API:** Dữ liệu trả về từ API thường là dữ liệu JSON. Bạn có thể sử dụng phương thức `.json()` của đối tượng `response` để chuyển đổi dữ liệu này thành một đối tượng Python như từ điển.

Tóm tắt

Việc làm việc với thư viện bên ngoài là một phần không thể thiếu trong lập trình Python, giúp giải quyết các vấn đề phức tạp mà không cần phải viết lại mã từ đầu. Các thư viện như **Numpy**, **Pandas**, và **Matplotlib** cung cấp những công cụ mạnh mẽ để xử lý và phân tích dữ liệu, trong khi việc làm việc với **API web** mở rộng khả năng giao tiếp giữa ứng dụng Python và các dịch vụ trực tuyến. Hiểu cách cài đặt và sử dụng các thư viện này sẽ giúp bạn tăng tốc quá trình phát triển phần mềm và mở rộng khả năng của ứng dụng Python.

Chương 15. Bài tập thực hành và dự án nhỏ

Contents

Việc áp dụng lý thuyết vào thực hành là một trong những cách hiệu quả nhất để học lập trình. Trong phần này, chúng ta sẽ làm quen với một số bài tập thực hành cơ bản giúp củng cố kiến thức về các khái niệm trong Python, cũng như các dự án nhỏ giúp sinh viên rèn luyện kỹ năng lập trình thực tế. Các bài tập và dự án này sẽ giúp sinh viên hiểu sâu hơn về cách sử dụng Python để giải quyết các vấn đề trong cuộc sống thực.

15.1. Bài tập về tính toán số học

Bài tập này sẽ giúp sinh viên thực hành các phép toán cơ bản trong Python, đồng thời làm quen với việc sử dụng các biểu thức số học và các hàm toán học có sẵn trong Python.

- **Mục tiêu:**
  - Hiểu cách sử dụng các phép toán cơ bản trong Python như cộng, trừ, nhân, chia, lũy thừa.
  - Làm quen với các hàm toán học trong thư viện math.
- **Bài tập:**
  1. Viết chương trình nhập vào một số và tính toán các phép toán sau: cộng, trừ, nhân, chia, lũy thừa.

2. Sử dụng thư viện math để tính căn bậc hai, giá trị tuyệt đối, và logarit của một số.
3. Tính diện tích và chu vi của một hình tròn khi biết bán kính.

Ví dụ:

```
import math

# Bài tập 1: Tính diện tích và chu vi của hình tròn
radius = float(input("Nhập bán kính hình tròn: "))
area = math.pi * radius ** 2
circumference = 2 * math.pi * radius
print(f"Diện tích hình tròn: {area:.2f}")
print(f"Chu vi hình tròn: {circumference:.2f}")
```

15.2. Xây dựng một trò chơi đơn giản

Việc xây dựng trò chơi đơn giản là một cách tuyệt vời để thực hành các kỹ năng lập trình như điều khiển luồng chương trình, xử lý dữ liệu nhập vào, và sử dụng vòng lặp.

- **Mục tiêu:**
  - Cải thiện kỹ năng sử dụng các cấu trúc điều khiển như vòng lặp, điều kiện, và nhập xuất dữ liệu.
  - Làm quen với việc xây dựng các ứng dụng tương tác với người dùng.
- **Bài tập:**
  1. Xây dựng trò chơi “Đoán số” trong đó máy tính chọn một số ngẫu nhiên trong khoảng từ 1 đến 100, và người chơi phải đoán số đó. Sau mỗi lần đoán, chương trình sẽ cho biết số đó lớn hơn hay nhỏ hơn số đúng.
  2. Xây dựng trò chơi “Từ điển” trong đó người chơi sẽ đoán nghĩa của các từ, chương trình sẽ cung cấp gợi ý hoặc câu trả lời đúng nếu người chơi không trả lời chính xác.

Ví dụ “Đoán số”:

```python
import random

# Chương trình đoán số
print("Chào mừng bạn đến với trò chơi Đoán số!")
number = random.randint(1, 100)
guess = None
while guess != number:
    guess = int(input("Nhập số bạn đoán (từ 1 đến 100): "))
    if guess < number:
        print("Số bạn đoán nhỏ hơn!")
    elif guess > number:
        print("Số bạn đoán lớn hơn!")
    else:
        print("Chúc mừng bạn, đoán đúng rồi!")
```

15.3. Xử lý chuỗi và danh sách

Xử lý chuỗi và danh sách là một phần rất quan trọng trong lập trình Python. Bài tập này sẽ giúp sinh viên làm quen với các thao tác cơ bản trên chuỗi và danh sách, bao gồm việc thao tác với các phần tử trong chuỗi, cắt chuỗi, tìm kiếm, thay thế, và lặp qua các phần tử của danh sách.

- **Mục tiêu:**
  - Làm quen với các phương thức xử lý chuỗi như cắt chuỗi, nối chuỗi, và tìm kiếm chuỗi.
  - Thực hành các thao tác với danh sách như thêm, xóa, sắp xếp và lặp qua các phần tử.
- **Bài tập:**
  1. Viết chương trình kiểm tra xem một chuỗi có phải là palindrome (đọc xuôi giống như đọc ngược) hay không.
  2. Viết chương trình tìm tất cả các số nguyên tố trong một danh sách cho trước.
  3. Viết chương trình để tìm số lớn nhất và số nhỏ nhất trong một danh sách và hoán đổi chúng.

Ví dụ xử lý chuỗi:

```
# Kiểm tra chuỗi palindrome
def is_palindrome(s):
    return s == s[::-1]

string = input("Nhập một chuỗi: ")
if is_palindrome(string):
    print(f"Chuỗi '{string}' là palindrome!")
else:
    print(f"Chuỗi '{string}' không phải là palindrome.")
```

15.4. Dự án quản lý tệp và thư mục

Trong dự án này, sinh viên sẽ học cách làm việc với hệ thống tệp, bao gồm việc đọc, ghi tệp, và quản lý thư mục. Việc này sẽ giúp sinh viên làm quen với các thao tác cơ bản trong lập trình ứng dụng thực tế, nơi việc xử lý tệp và thư mục rất quan trọng.

- **Mục tiêu:**
  - Làm quen với các thao tác đọc, ghi và tạo tệp trong Python.
  - Quản lý và thao tác với thư mục, bao gồm tạo, xóa, và liệt kê các tệp trong thư mục.
- **Bài tập:**
  1. Viết chương trình quản lý danh bạ, nơi người dùng có thể thêm, sửa, xóa và tìm kiếm thông tin liên lạc. Dữ liệu sẽ được lưu trong tệp văn bản.
  2. Viết chương trình sao lưu tệp, trong đó người dùng có thể sao lưu tệp từ một thư mục này sang một thư mục khác.
  3. Viết chương trình đọc dữ liệu từ một tệp CSV và tính toán tổng và trung bình của các giá trị trong tệp.

Ví dụ quản lý tệp:

```python
# Quản lý danh bạ: Thêm, sửa, xóa thông tin
def add_contact(filename, name, phone):
    with open(filename, 'a') as file:
        file.write(f"{name}, {phone}\n")

def list_contacts(filename):
    with open(filename, 'r') as file:
        contacts = file.readlines()
        for contact in contacts:
            print(contact.strip())

# Thêm thông tin liên lạc
add_contact('contacts.txt', 'Alice', '123-456-789')
list_contacts('contacts.txt')
```

Tóm tắt

Các bài tập thực hành và dự án nhỏ không chỉ giúp sinh viên củng cố kiến thức lý thuyết mà còn giúp phát triển các kỹ năng lập trình thực tế. Những bài tập như tính toán số học, xây dựng trò chơi đơn giản, xử lý chuỗi và danh sách, và quản lý tệp giúp sinh viên làm quen với việc áp dụng các kỹ thuật lập trình cơ bản vào giải quyết vấn đề thực tế. Những dự án này sẽ chuẩn bị cho sinh viên những kỹ năng cần thiết để phát triển phần mềm và làm việc trong môi trường lập trình chuyên nghiệp.

Chương 16. Tài liệu và nguồn học Python nâng cao

Contents

Khi đã nắm vững những kiến thức cơ bản về Python, việc tiếp tục học hỏi và nâng cao kỹ năng lập trình là rất quan trọng. Trong phần này, chúng ta sẽ khám phá các tài liệu và nguồn học Python nâng cao, giúp sinh viên có thể mở rộng kiến thức, tìm hiểu các kỹ thuật lập trình tiên tiến, và tham gia vào cộng đồng lập trình Python toàn cầu. Những nguồn tài nguyên này sẽ là cầu nối giúp sinh viên phát triển và hoàn thiện kỹ năng lập trình của mình.

16.1. Học từ tài liệu chính thức của Python

Tài liệu chính thức của Python là nguồn tài nguyên quan trọng và đáng tin cậy nhất cho việc học và hiểu sâu về ngôn ngữ lập trình này. Python cung cấp tài liệu chi tiết về cú pháp, thư viện chuẩn, các kỹ thuật lập trình nâng cao và nhiều tính năng hữu ích khác. Đây là điểm khởi đầu lý tưởng cho những ai muốn nâng cao kiến thức về Python.

- **Tài liệu chính thức của Python:**
  - Trang web chính thức của Python là python.org, nơi có tài liệu về Python, bao gồm các phiên bản khác nhau của Python, thông tin về các thư viện và API.
  - **Tài liệu Python:** Tài liệu này bao gồm tất cả các khái niệm cơ bản và nâng cao về ngôn ngữ, từ cấu trúc dữ liệu, các tính năng ngôn ngữ, đến các thư viện và công cụ hỗ trợ.
  - **Hướng dẫn sử dụng thư viện chuẩn:** Tại Python Standard Library Documentation, sinh viên có thể tìm thấy hướng dẫn chi tiết về các thư viện chuẩn của Python, từ cơ

bản đến nâng cao.

- **Python Enhancement Proposals (PEPs):** Đây là các đề xuất về các cải tiến và thay đổi trong Python, giúp người học hiểu về quá trình phát triển và cải tiến Python. Đặc biệt, PEP 8 cung cấp hướng dẫn về phong cách lập trình Python chuẩn.

- **Các chủ đề nâng cao trong tài liệu chính thức:**
  - Quản lý bộ nhớ và garbage collection.
  - Xử lý đồng thời và lập trình bất đồng bộ.
  - Thực thi mã động và metaprogramming.
  - Thư viện nâng cao như `asyncio`, `multiprocessing`, `concurrent.futures`.

16.2. Các khóa học và cộng đồng học Python trực tuyến

Ngoài tài liệu chính thức, có rất nhiều khóa học trực tuyến và cộng đồng hỗ trợ giúp bạn học Python một cách hiệu quả. Những nền tảng học trực tuyến cung cấp các khóa học từ cơ bản đến nâng cao, các bài tập thực hành, và các dự án để sinh viên áp dụng kiến thức đã học vào thực tế. Các cộng đồng lập trình Python giúp bạn giao lưu, học hỏi từ những người có kinh nghiệm và nhận sự hỗ trợ khi gặp khó khăn.

- **Các nền tảng khóa học trực tuyến:**
  - **Coursera:** Cung cấp các khóa học Python từ các trường đại học và tổ chức uy tín, như "Python for Everybody" của Đại học Michigan.
  - **edX:** Có các khóa học Python từ các trường như MIT, Harvard, và các tổ chức hàng đầu khác. Một số khóa học nổi bật là "CS50's Introduction to Computer Science" của Harvard.
  - **Udemy:** Cung cấp các khóa học Python với giá cả hợp lý, bao gồm các khóa học cho người mới bắt đầu và các khóa học nâng cao như "Python for Data Science" và "Python for Web Development".
  - **Real Python:** Đây là một nền tảng chuyên biệt dành cho việc học Python, cung cấp nhiều bài học từ cơ bản đến nâng cao, bao gồm các kỹ thuật lập trình hiện đại và các thư viện Python phổ biến.
- **Cộng đồng học Python trực tuyến:**
  - **Stack Overflow:** Là nơi tuyệt vời để đặt câu hỏi và tìm kiếm các câu trả lời cho các vấn đề lập trình Python. Đây là cộng đồng lập trình lớn nhất, nơi bạn có thể giao lưu với

các lập trình viên từ khắp nơi trên thế giới.

- **Reddit:** Các subreddit như r/learnpython và r/Python là các cộng đồng nơi người học Python có thể chia sẻ kiến thức, giải quyết vấn đề và học hỏi từ những lập trình viên khác.
- **GitHub:** Là nền tảng chia sẻ mã nguồn phổ biến, nơi bạn có thể tìm thấy hàng nghìn dự án mã nguồn mở sử dụng Python. Tham gia vào các dự án mã nguồn mở sẽ giúp bạn học hỏi từ các lập trình viên khác và cải thiện kỹ năng của mình.

- **Các diễn đàn và nhóm mạng xã hội:**
  - **Python Discord:** Là một cộng đồng Python trên Discord, nơi bạn có thể tham gia các cuộc thảo luận, đặt câu hỏi và giao lưu với các lập trình viên Python.
  - **Python Meetup:** Là các nhóm gặp gỡ trực tiếp hoặc trực tuyến dành cho người học và lập trình viên Python. Đây là nơi bạn có thể tham gia các sự kiện, hội thảo và chia sẻ kinh nghiệm với cộng đồng.

16.3. Tham gia vào các dự án mã nguồn mở

Tham gia vào các dự án mã nguồn mở là cách tuyệt vời để rèn luyện kỹ năng lập trình và làm việc nhóm. Bạn có thể học hỏi từ những lập trình viên có kinh nghiệm, cải thiện khả năng giải quyết vấn đề và hiểu rõ hơn về cách phát triển phần mềm trong môi trường thực tế. Python có một cộng đồng mã nguồn mở rất mạnh mẽ, với nhiều dự án lớn được phát triển và duy trì bởi cộng đồng toàn cầu.

- **Tìm kiếm các dự án mã nguồn mở:**
  - **GitHub:** GitHub là nơi chứa hầu hết các dự án mã nguồn mở. Bạn có thể tìm kiếm các dự án Python, đóng góp mã nguồn và tham gia vào các vấn đề (issues) hoặc pull requests của các dự án.
  - **GitLab:** Đây cũng là một nền tảng mã nguồn mở nổi bật, nơi bạn có thể tìm thấy nhiều dự án Python và tham gia vào việc phát triển các dự án đó.
  - **SourceForge:** Mặc dù không phổ biến như GitHub, SourceForge vẫn là một nền tảng tuyệt vời để khám phá các dự án mã nguồn mở và tham gia đóng góp.
- **Các dự án mã nguồn mở Python nổi bật:**
  - **Django:** Một framework web Python phổ biến. Tham gia vào dự án này sẽ giúp bạn học về phát triển web với Python.
  - **Flask:** Một framework web nhẹ, dễ học và linh hoạt. Đây là lựa chọn tốt cho những ai muốn xây dựng các ứng dụng web nhỏ hoặc đơn giản.

    - **Pandas:** Một thư viện mạnh mẽ cho phân tích dữ liệu. Tham gia vào dự án này sẽ giúp bạn học về xử lý dữ liệu với Python.
    - **NumPy:** Thư viện nền tảng cho tính toán khoa học trong Python. Bạn có thể đóng góp vào các tính năng mới hoặc cải thiện hiệu suất của thư viện này.
    - **TensorFlow:** Một thư viện học sâu nổi tiếng, tham gia vào dự án này sẽ giúp bạn học về trí tuệ nhân tạo và học máy.
- **Lợi ích khi tham gia mã nguồn mở:**
    - **Cải thiện kỹ năng lập trình:** Bạn sẽ học được các phương pháp lập trình hay và cải tiến khả năng viết mã chất lượng cao.
    - **Học hỏi từ người khác:** Làm việc cùng với các lập trình viên có kinh nghiệm sẽ giúp bạn mở rộng kiến thức và cải thiện kỹ năng giải quyết vấn đề.
    - **Mở rộng mạng lưới:** Bạn sẽ được kết nối với cộng đồng lập trình viên toàn cầu và có cơ hội học hỏi từ các chuyên gia trong ngành.

Tóm tắt

Việc học Python không dừng lại ở việc nắm vững các kiến thức cơ bản mà còn cần sự phát triển liên tục thông qua việc tham gia vào các khóa học trực tuyến, cộng đồng học lập trình, và các dự án mã nguồn mở. Các tài liệu chính thức của Python là nguồn tài nguyên tuyệt vời để học và hiểu sâu về ngôn ngữ này. Tham gia các cộng đồng trực tuyến và các dự án mã nguồn mở không chỉ giúp bạn cải thiện kỹ năng lập trình mà còn giúp bạn xây dựng một mạng lưới chuyên gia và học hỏi những kiến thức mới trong quá trình phát triển sự nghiệp lập trình.