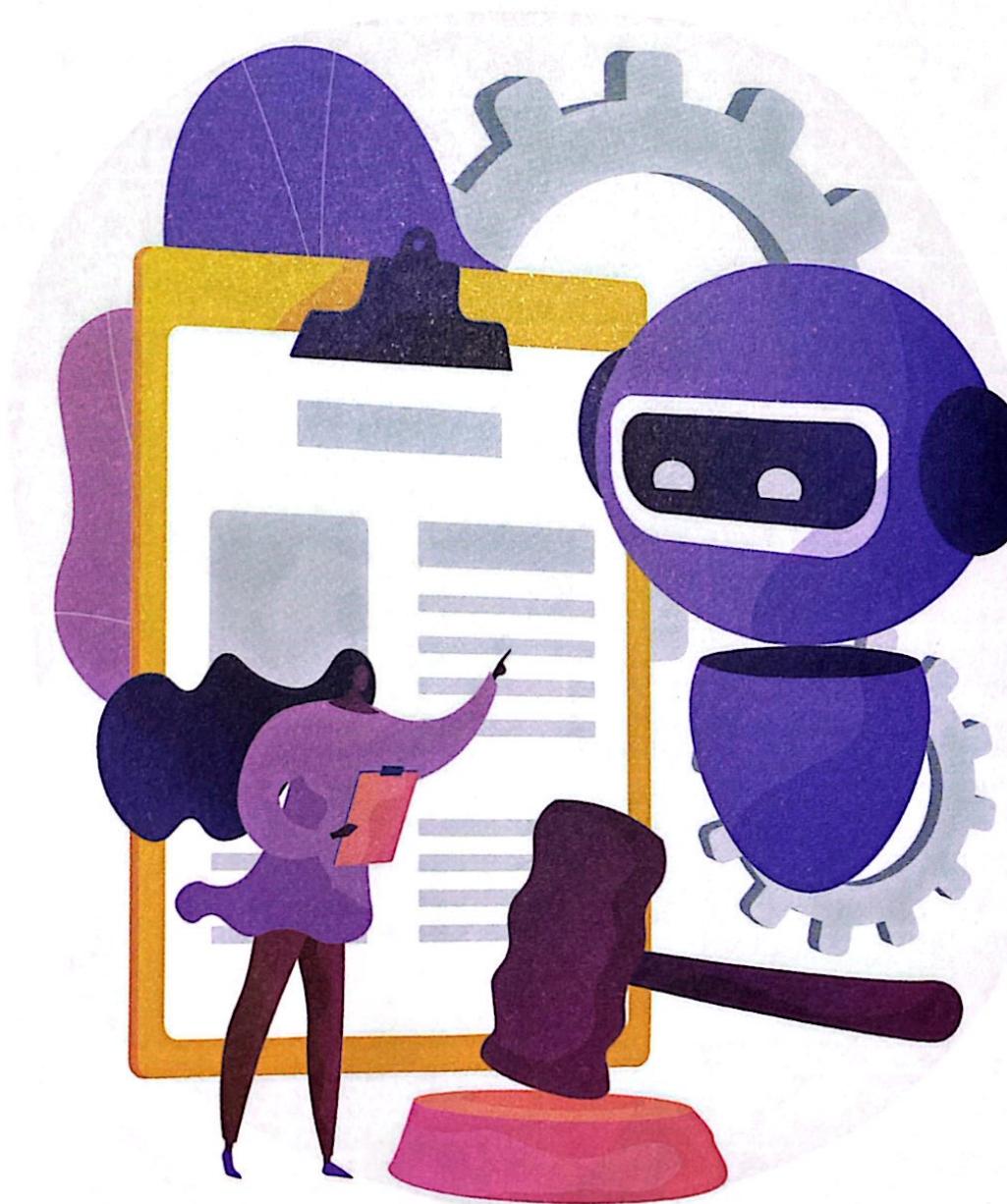


PHẦN 4:

Kiến thức, cách áp dụng, bài tập
thực hành - ngôn ngữ Python



PHẦN 4:

KIẾN THỨC, CÁCH ÁP DỤNG, BÀI TẬP THỰC HÀNH - NGÔN NGỮ PYTHON

Chương 1: Tổng quan về ngôn ngữ Python

I. Tại sao nên học Python?

Python là một ngôn ngữ lập trình thông dịch, đa năng và dễ học, được phát triển bởi Guido van Rossum vào năm 1989 tại Hà Lan.

Ban đầu, ông tạo ra Python là với mục đích để tạo ra một ngôn ngữ lập trình đơn giản và dễ học cho mọi người. Ông đã lấy tên Python từ tên loài rắn Python ở châu Phi, không phải là từ vị tướng Monty Python như nhiều người vẫn nghĩ.

Python ban đầu được phát hành vào năm 1991 với mục đích chính là làm ngôn ngữ lập trình cho các dự án tương tác và truyền thông. Tuy nhiên, với sự phát triển của ngôn ngữ và sự đóng góp của cộng đồng lập trình viên, Python đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, được sử dụng trong nhiều lĩnh vực khác nhau như phát triển web, khoa học dữ liệu, trí tuệ nhân tạo và nhiều lĩnh vực khác.

Python cũng được phát triển và duy trì bởi một cộng đồng lớn và đam mê, với hàng ngàn các thư viện và framework hữu ích, giúp cho việc phát triển phần mềm và giải quyết các vấn đề thực tế trở nên dễ dàng hơn.

Trong những năm gần đây, Python còn được chọn làm ngôn ngữ chính cho giảng dạy lập trình tại nhiều trường đại học và các tổ chức đào tạo khác trên toàn thế giới, bởi vì tính dễ học và sử dụng của nó. Điều này đã giúp cho việc học lập trình trở nên dễ dàng hơn đối với nhiều người, và cũng đã mở ra cơ hội cho nhiều người muốn học lập trình và tham gia vào cộng đồng lập trình viên.

Python có mục đích chính là tăng cường hiệu quả trong việc lập trình, đặc biệt là đối với việc phát triển ứng dụng web, trí tuệ nhân tạo, xử lý dữ liệu và Machine Learning.



II. Python có cú pháp đơn giản, dễ đọc và dễ viết, đồng thời cũng có nhiều tính năng tiên tiến

- › Hỗ trợ đa nền tảng: Python có thể chạy trên nhiều nền tảng và hệ điều hành, bao gồm Windows, Linux, macOS và các thiết bị di động.
- › Có nhiều thư viện và khung làm việc: Python có một số lượng lớn các thư viện và khung làm việc có sẵn, giúp cho việc lập trình nhanh chóng và dễ dàng hơn.
- › Hỗ trợ đa mô hình lập trình: Python hỗ trợ nhiều mô hình lập trình, bao gồm lập trình hướng đối tượng, lập trình hàm và lập trình cấu trúc.
- › Có tính linh hoạt cao: Python cho phép tùy biến và mở rộng với các thư viện bên ngoài, hỗ trợ tích hợp với nhiều ngôn ngữ khác nhau, và có thể được sử dụng trong các dự án lớn và phức tạp.

Python được sử dụng rộng rãi trong nhiều lĩnh vực, bao gồm phát triển web, khoa học dữ liệu, trí tuệ nhân tạo, xử lý ngôn ngữ tự nhiên, hệ thống và mạng máy tính, và phát triển game. Python cũng là một trong những ngôn ngữ lập trình phổ biến nhất hiện nay và được đánh giá cao bởi cộng đồng lập trình viên và các chuyên gia công nghệ.

Để bắt đầu học Python, bạn cần cài đặt môi trường phát triển (IDE) và trình thông dịch Python. Dưới đây là hướng dẫn cài đặt Python trên các hệ điều hành phổ biến như Windows, MacOS và Linux.

III. Cách cài đặt môi trường lập trình Python

1. Cài đặt Python trên Windows

Bước 1: Tải xuống trình cài đặt Python.

Để tải Python, hãy truy cập trang web chính thức của Python tại địa chỉ :

<https://www.python.org/downloads/>

và tải về phiên bản Python phù hợp với hệ điều hành của bạn.

Bước 2: Cài đặt Python.

Mở tệp cài đặt đã tải xuống và chạy file.exe.

Chọn "Add Python to PATH" để có thể sử dụng Python từ bất cứ thư mục

nào trên máy tính của bạn.

Chọn "**Customize installation**" để có thể lựa chọn cài đặt các thành phần khác nhau như trình chỉnh sửa, IDLE và các thư viện tiêu chuẩn.

Bước 3: Kiểm tra cài đặt.

Mở Command Prompt hoặc PowerShell và nhập lệnh "**python --version**" để kiểm tra phiên bản Python đã được cài đặt.

2. Cài đặt Python trên MacOS:

Bước 1: Cài đặt Homebrew.

Homebrew là một trình quản lý gói cho MacOS, giúp bạn cài đặt các ứng dụng và thư viện một cách dễ dàng. Để cài đặt Homebrew, hãy truy cập trang web chính thức của Homebrew tại địa chỉ <https://brew.sh> và làm theo hướng dẫn.

Bước 2: Cài đặt Python.

Mở Terminal và nhập lệnh sau: Brew install python.

Chờ đợi quá trình cài đặt hoàn tất.

Bước 3: Kiểm tra cài đặt.

Mở Terminal và nhập lệnh "**python --version**" để kiểm tra phiên bản Python đã được cài đặt.

3. Cài đặt Python trên Linux:

Bước 1: Mở Terminal.

Truy cập Terminal trong hệ điều hành Linux của bạn.

Bước 2: Cài đặt Python.

Nhập lệnh sau để cài đặt Python: Sudo apt-get install python3.

Bước 3: Kiểm tra cài đặt.

Nhập lệnh "**python3 --version**" để kiểm tra phiên bản Python đã được cài đặt.

Sau khi cài đặt Python, bạn có thể sử dụng IDLE hoặc các trình chỉnh sửa mã nguồn khác để viết và chạy các chương trình Python của mình.

Chương 2: Kiến thức, cách áp dụng và bài tập thực hành

I. Khái niệm về ngôn ngữ lập trình Python

Như đã đề cập ở **Chương 1: Tổng quan về ngôn ngữ lập trình Python**, chúng ta hãy hiểu đơn giản: Python là một ngôn ngữ lập trình bậc cao, mã nguồn mở và đa nền tảng.

Để dễ hình dung về những kiến thức mà chúng ta cần nắm để có thể hiểu và sử dụng được Python. Chúng ta cùng xem qua một quy trình cơ bản để học về Python như sau:

- › Học cú pháp cơ bản: Trước khi bắt đầu học bất kỳ ngôn ngữ lập trình nào, bạn cần phải nắm vững cú pháp cơ bản của ngôn ngữ đó.
- › Tập trung vào cấu trúc dữ liệu và giải thuật: Sau khi bạn đã nắm vững cú pháp cơ bản của Python, hãy tập trung vào việc học về cấu trúc dữ liệu và giải thuật. Đây là hai chủ đề quan trọng giúp bạn xây dựng các ứng dụng và giải quyết các vấn đề trong thực tế.
- › Xây dựng các ứng dụng đơn giản: Sau khi bạn đã nắm vững cấu trúc dữ liệu và giải thuật, hãy bắt đầu xây dựng các ứng dụng đơn giản. Bạn có thể bắt đầu bằng các ứng dụng console nhỏ nhưng có tính ứng dụng cao, sau đó chuyển sang các ứng dụng web hoặc ứng dụng di động.
- › Tham gia các dự án mã nguồn mở: Tham gia các dự án mã nguồn mở là cách tuyệt vời để học thêm về cách xây dựng các ứng dụng phức tạp và làm việc trong các dự án phần mềm thực tế.
- › Học các thư viện và framework: Python có nhiều thư viện và framework rất hữu ích để xây dựng các ứng dụng phức tạp. Hãy học các thư viện và framework như NumPy, Pandas, Django, Flask và nhiều hơn nữa để cải thiện kỹ năng của mình.
- › Thực hành thường xuyên: Cuối cùng, để học tốt Python, bạn cần thực hành thường xuyên. Cố gắng tạo ra các dự án mới và đặt mục tiêu học tập hàng tuần để tiếp tục phát triển kỹ năng của mình.

Trước khi tìm hiểu về các cú pháp cơ bản trong Python, hãy cùng nhau tìm hiểu về biến và các kiểu dữ liệu trong Python. Trong Python, biến là một định danh được sử dụng để lưu trữ giá trị hoặc tham chiếu đến một đối tượng. Mỗi biến có một tên riêng, có thể gán cho nó một giá trị hoặc tham chiếu đến một đối tượng. Biến có thể thay đổi giá trị hoặc tham chiếu đến một đối tượng khác trong quá trình chạy chương trình. Kiểu dữ liệu trong Python là các loại dữ liệu khác nhau mà biến có thể lưu trữ.

II. Các kiểu dữ liệu cơ bản trong Python

- **Kiểu số (Numeric):** Kiểu số trong Python bao gồm số nguyên (integer), số thực (float), số phức (complex). **Ví dụ:**

```
a = 5 # Kiểu số nguyên  
b = 3.14 # Kiểu số thực  
c = 2 + 3j # Kiểu số phức
```

- **Kiểu chuỗi (String):** Kiểu chuỗi trong Python là một chuỗi các ký tự được bao quanh bởi dấu nháy đơn hoặc dấu nháy kép. **Ví dụ:**

```
name = "John" # Kiểu chuỗi  
|
```

- **Kiểu Boolean:** Kiểu Boolean trong Python chỉ có hai giá trị: True hoặc False. **Ví dụ:**

```
is_student = True # Kiểu Boolean
```

Trong ngôn ngữ lập trình Python, có nhiều cấu trúc dữ liệu và giải thuật phổ biến được sử dụng để giải quyết các bài toán phức tạp. Sau đây là một số cấu trúc dữ liệu và giải thuật phổ biến trong Python:

- **Kiểu danh sách (List):**

Trong Python, kiểu dữ liệu danh sách (List) là một kiểu dữ liệu dùng để lưu trữ nhiều giá trị khác nhau trong một biến. Kiểu dữ liệu này cho phép lưu trữ các phần tử có kiểu dữ liệu khác nhau trong cùng một danh sách và được đánh số chỉ mục index từ 0.



Cú pháp để khai báo một danh sách trong Python như sau:

```
my_list = [value1, value2, value3, ...]
```

Với value1, value2, value3 là các giá trị cần lưu trữ trong danh sách. Các giá trị này có thể là các kiểu dữ liệu khác nhau như số nguyên, số thực, chuỗi, danh sách, tuple, dictionary, set,...

Ngoài ra, danh sách còn cung cấp một số phương thức để thêm, sửa, xóa phần tử như append(), insert(), extend(), remove(), pop(), clear()....

Ví dụ:

```
my_list = [1, 2, 3, "four", [5, 6]]
```

Trong đó, my_list là tên danh sách được khai báo. Nó chứa 5 phần tử, bao gồm các số nguyên 1, 2, 3, chuỗi "four", và một danh sách có hai phần tử [5, 6].

Để truy cập vào các phần tử của danh sách, ta sử dụng chỉ mục của chúng như sau:

```
my_list = [1, 2, 3, "four", [5, 6]]
print(my_list[0]) # output: 1
print(my_list[3]) # output: four
print(my_list[4][1]) # output: 6
```

› **Kiểu tuple:**

Trong Python, tuple là một kiểu dữ liệu lưu trữ các giá trị tương tự như list. Tuy nhiên, tuple là một kiểu dữ liệu bất biến (immutable), nghĩa là bạn không thể thay đổi nội dung của tuple sau khi tạo nó.

Cú pháp để tạo một tuple trong Python là sử dụng dấu ngoặc đơn () để bao quanh các giá trị của tuple, cách nhau bởi dấu phẩy. **Ví dụ:**

```
my_tuple = (1, 2, 3, "four", 5.0)
```

Các phần tử của một tuple có thể được truy cập bằng cách sử dụng cú pháp tương tự như list, bằng cách sử dụng chỉ số index. **Ví dụ:**

```
my_tuple = (1, 2, 3, "four", 5.0)
```

```
print(my_tuple[0]) # Output: 1  
print(my_tuple[3]) # Output: four
```

Tuple cũng hỗ trợ cắt (slicing) giống như list, bằng cách sử dụng cú pháp [start:end:step]. **Ví dụ:**

```
my_tuple = (1, 2, 3, "four", 5.0)
```

```
print(my_tuple[1:4]) # Output: (2, 3, 'four')
```

Tuy nhiên, bạn không thể thay đổi giá trị của tuple sau khi tạo nó. **Ví dụ:**

```
my_tuple[0] = 10 # Lỗi: TypeError: 'tuple' object does not support item assignment
```

Vì tính bất biến của nó, tuple thường được sử dụng để lưu trữ các giá trị không thay đổi, ví dụ như các giá trị hằng số, các giá trị được sử dụng làm khóa cho các từ điển (dictionaries),...

- › **Kiểu từ điển (Dictionary):**

Trong Python, kiểu dữ liệu từ điển (dictionary) là một cấu trúc dữ liệu mà cho phép lưu trữ các cặp key-value (khóa-giá trị) trong đó mỗi key được liên kết với một giá trị tương ứng. Kiểu dữ liệu từ điển có thể được khởi tạo bằng cách sử dụng cặp ngoặc nhọn '{}' và định nghĩa các cặp key-value bên trong.



Cú pháp để khởi tạo một từ điển trong Python như sau:

```
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

Trong đó, 'key1', 'key2', 'key3' là các khóa và 'value1', 'value2', 'value3' là các giá trị tương ứng. Chúng ta có thể truy xuất giá trị của một khóa bằng cách sử dụng cú pháp:

```
my_dict['key1']
```

Nếu khóa không tồn tại trong từ điển, một lỗi sẽ được tạo ra. Chúng ta cũng có thể sử dụng phương thức **get** để truy xuất giá trị của một khóa. Phương thức này sẽ trả về giá trị tương ứng với khóa nếu nó tồn tại trong từ điển và trả về một giá trị mặc định nếu khóa không tồn tại:

```
my_dict.get('key4', 'not found')
```

- › Kiểu set:

Trong Python, kiểu dữ liệu set là một cấu trúc dữ liệu không có thứ tự và không chứa các phần tử trùng lặp.

Set được xây dựng trên kiểu dữ liệu từ khóa của Python, có thể chứa các kiểu dữ liệu như số, chuỗi và tuple.

Cú pháp để tạo một set là sử dụng dấu ngoặc nhọn {} hoặc hàm set():

```
my_set = {1, 2, 3}  
my_set = set([1, 2, 3])
```

Một số phương thức thường sử dụng của set bao gồm:

- **add(element)**: thêm một phần tử vào set.
- **update(other_set)**: thêm nhiều phần tử vào set từ một set khác.
- **remove(element)**: loại bỏ một phần tử khỏi set, nếu không có phần tử này trong set sẽ bị ném ra lỗi KeyError.

- `discard(element)`: loại bỏ một phần tử khỏi set, nếu không có phần tử này trong set sẽ không có hiệu lực gì.
- `pop()`: loại bỏ và trả về một phần tử ngẫu nhiên trong set, nếu set rỗng sẽ bị ném ra lỗi `KeyError`.
- `clear()`: loại bỏ tất cả các phần tử trong set.

Ví dụ minh họa:

```

my_set = {1, 2, 3, 4, 5}
my_set.add(6)
print(my_set) # kết quả: {1, 2, 3, 4, 5, 6}

other_set = {4, 5, 6, 7, 8}
my_set.update(other_set)
print(my_set) # kết quả: {1, 2, 3, 4, 5, 6, 7, 8}

my_set.remove(7)
print(my_set) # kết quả: {1, 2, 3, 4, 5, 6, 8}

my_set.discard(10)
print(my_set) # kết quả: {1, 2, 3, 4, 5, 6, 8}

print(my_set.pop()) # kết quả: một phần tử ngẫu nhiên trong set

my_set.clear()
print(my_set) # kết quả: set()

```

III. Cú pháp

Vậy với các biến và kiểu dữ liệu như trên, chúng ta sẽ sử dụng những cú pháp nào và sử dụng ra sao? Dưới đây là những cú pháp (syntax) cơ bản trong ngôn ngữ lập trình Python mà bạn cần nắm:

1. Cú pháp in ra màn hình:

Để in ra màn hình trong Python, chúng ta sử dụng hàm `print()`. Cú pháp của hàm này như sau:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Trong đó:

- `objects`: Là danh sách các đối tượng cần in ra màn hình. Các đối tượng này được phân tách bằng dấu phẩy (,).



- › sep: Là ký tự phân cách giữa các đối tượng trong danh sách. Giá trị mặc định của sep là khoảng trắng (' ').
- › end: Là ký tự kết thúc khi in xong danh sách các đối tượng. Giá trị mặc định của end là ký tự xuống dòng ('\n').
- › file: Là đối tượng file đích để in ra. Giá trị mặc định là sys.stdout, tức là in ra màn hình.
- › flush: Là biến logic xác định có cần xóa bộ đệm khi in xong hay không. Giá trị mặc định của flush là False.

Ví dụ:

```
print('Xin chào', 'mọi người!', sep='-', end='...')
```

Xin chào-mọi người!...

2. Cú pháp gán giá trị cho biến:

Cú pháp gán giá trị cho biến trong Python là:

```
ten_bien = gia_tri
```

Trong đó:

- › ten_bien là tên của biến mà ta muốn gán giá trị.
- › gia_tri là giá trị mà ta muốn gán cho biến đó.

Ví dụ:

```
x = 10 # gán giá trị 10 cho biến x
```

3. Cú pháp nhập dữ liệu từ bàn phím:

Để nhập dữ liệu từ bàn phím trong Python, ta sử dụng hàm input().

Cú pháp như sau:

input([prompt])

Trong đó:

- › prompt (tùy chọn) là chuỗi được in ra trước khi chờ người dùng nhập dữ liệu.
- › Hàm input() sẽ trả về một chuỗi (string) chứa dữ liệu được người dùng nhập từ bàn phím.

Ví dụ:

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

#output

```
Nhập tên của bạn: Luffy  
Xin chào, Luffy
```

4. Cú pháp điều kiện:

Trong Python, cú pháp điều kiện được sử dụng để thực hiện các lệnh khác nhau tùy thuộc vào giá trị của biểu thức điều kiện. Cú pháp chung cho điều kiện là:

```
if expression:  
    statement(s)  
elif expression:  
    statement(s)  
else:  
    statement(s)
```

Trong đó:

- › expression là một biểu thức điều kiện trả về giá trị True hoặc False.
- › statement(s) là các câu lệnh được thực thi nếu biểu thức điều kiện đúng.

Có thể có một hoặc nhiều câu lệnh trong mỗi khối lệnh, nhưng chúng phải được thụt lề cùng một số khoảng trắng hoặc tab. Thông thường, các nhà lập trình viên sử dụng bốn khoảng trắng để thụt lề.

Ngoài ra, có thể sử dụng các toán tử so sánh và logic trong biểu thức điều kiện, bao gồm:

- == (bằng).
- != (không bằng).
- < (nhỏ hơn).
- > (lớn hơn).
- <= (nhỏ hơn hoặc bằng).
- >= (lớn hơn hoặc bằng).
- and (và).
- or (hoặc).
- not (phủ định).

Ví dụ:

```
x = 10
y = 5

if x > y:
    print("x lớn hơn y")
elif x == y:
    print("x bằng y")
else:
    print("x nhỏ hơn y")

# kết quả là "x lớn hơn y".
```

5. Cú pháp vòng lặp:

Để thực hiện một hành động nhiều lần, ta sử dụng vòng lặp for hoặc while.

Cú pháp vòng lặp for:

```
for <variable> in <sequence>:  
    # block of code
```

Trong đó:

- › <variable>: Tên biến được khai báo để chứa giá trị của từng phần tử trong <sequence>.
- › <sequence>: Một chuỗi, danh sách, hoặc bất kỳ đối tượng nào có thể lặp lại các phần tử (iterable object) như danh sách (list), bộ (tuple), tập hợp (set), chuỗi (string), range, ...
- › block of code: Các câu lệnh được thực hiện cho mỗi phần tử trong <sequence>. Các câu lệnh này được bao bọc bởi một đoạn mã có thụt lề.

Ví dụ:

```
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print(fruit)  
  
#Output  
  
apple  
banana  
cherry
```

Cú pháp vòng lặp while:

```
while <condition>:  
    # block of code
```

Trong đó:

- › <condition>: Một biểu thức logic đúng hoặc sai. Nếu điều kiện này đúng, các câu lệnh trong block of code sẽ được thực hiện. Nếu điều kiện sai, vòng lặp sẽ kết thúc.
- › block of code: Các câu lệnh được thực hiện khi <condition> đúng.

Ví dụ:

```
i = 0
while i < 5:
    print(i)
    i += 1
# output
```

0
1
2
3
4

6. Cú pháp hàm:

Trong Python, cú pháp của một hàm bao gồm các phần sau:

- › Từ khóa **def** để định nghĩa một hàm.
- › Tên của hàm, theo sau bởi cặp dấu ngoặc đơn () .
- › Các tham số đầu vào của hàm, được đặt trong cặp ngoặc đơn () và cách nhau bởi dấu phẩy , .
- › Dấu hai chấm : Để kết thúc khai báo tên hàm và bắt đầu khối lệnh của hàm.
- › Khối lệnh của hàm được thụt lề vào bên trong và được thực thi khi hàm được gọi.
- › Câu lệnh return để trả về giá trị từ hàm (nếu có).

Dưới đây là một ví dụ về cú pháp của một hàm đơn giản trong Python:

```
def square(x):
    return x**2 # trả về bình phương của x

print(square(3)) # in ra giá trị 9
```

7. Cú pháp import module:

Trong Python, để sử dụng các chức năng của một module, chúng ta cần sử dụng cú pháp import. Cú pháp này có thể được sử dụng để import một module hoặc một phần của module. Có hai cách để import một module trong Python:

Import toàn bộ module:

Cú pháp:

```
import module_name
```

Ví dụ minh họa:

```
import math

print(math.pi) # In ra giá trị của hằng số PI trong module math
```

```
3.141592653589793
```

Import một phần của module:

Cú pháp:

```
from module_name import name(s)
```

Ví dụ minh họa:

```
from math import pi  
  
print(pi) # In ra giá trị của hằng số PI trong module math
```

3.141592653589793

Chú ý: Khi sử dụng cú pháp này, bạn không cần phải gọi tên module khi sử dụng các hàm, lớp hay biến trong module. Chỉ cần gọi trực tiếp tên của chúng là được.

Ngoài ra, chúng ta cũng có thể sử dụng cú pháp **as** để đổi tên cho module hoặc phần của module:

```
import module_name as new_name  
from module_name import name as new_name
```

Ví dụ minh họa:

```
import math as m  
from math import pi as p  
  
print(m.sqrt(16)) # In ra kết quả của hàm căn bậc 2 trong module math  
print(p) # In ra giá trị của hằng số PI trong module math
```

4.0
3.141592653589793

Ngoài các cú pháp, khái niệm về “**module**” và “**class**” cũng là những kiến thức quan trọng đối với mỗi một lập trình viên khi học về Python.

IV. Các module trong Python

Module trong Python là một tập hợp các định nghĩa (như hàm, biến, class) được đóng gói lại thành một file. Module cho phép ta tái sử dụng code và phân chia code thành các phần để dễ dàng quản lý.

Python có một số module được tích hợp sẵn như: ‘math’, ‘random’, ‘datetime’, ‘os’, ‘sys’ và nhiều module khác. Ngoài ra, có rất nhiều module được cung cấp bởi cộng đồng Python và có thể được cài đặt thông qua trình quản lý package như pip. Dưới đây là một số các module trong Python:

- Module ‘math’: Là một module chuẩn được sử dụng để thực hiện các tính toán toán học phức tạp. Module math cung cấp các hàm và biến liên quan đến toán học, bao gồm các hàm lượng giác, hàm Logarit, hàm mũ, hàm tính toán số pi và hằng số toán học khác. **Ví dụ:**

```
import math

# Tính căn bậc hai của một số
x = math.sqrt(25)
print(x) # Output: 5.0

# Tính giai thừa của một số
y = math.factorial(5)
print(y) # Output: 120

# Tính cos của một góc
z = math.cos(math.pi/4)
print(z) # Output: 0.7071067811865476
```

- Module ‘random’: Một trong những module chuẩn của Python được sử dụng để tạo ngẫu nhiên các giá trị. Module ‘random’ cung cấp nhiều hàm để tạo ra các giá trị ngẫu nhiên, bao gồm số nguyên, số thực và chuỗi. **Ví dụ:**

```
import random

# Tạo một số nguyên ngẫu nhiên trong khoảng từ 0 đến 9
x = random.randint(0, 9)
print(x) # output: 7

# Tạo một số thực ngẫu nhiên trong khoảng từ 0.0 đến 1.0
y = random.random()
print(y) # output: 0.40295638497670213

# Tạo một chuỗi ngẫu nhiên với độ dài là 5
z = ''.join(random.choices('abcdefghijklmnopqrstuvwxyz', k=5))
print(z) # output: syhvu
```

- Module ‘datetime’: Cung cấp các lớp và hàm cho việc xử lý ngày giờ. Các lớp trong module datetime bao gồm:
 - date: Đại diện cho một ngày trong lịch.

- time: Đại diện cho một thời gian trong ngày.
- datetime: Đại diện cho một thời điểm cụ thể, bao gồm cả ngày và thời gian.
- timedelta: Đại diện cho một khoảng thời gian giữa hai thời điểm.

Module datetime cung cấp nhiều hàm và phương thức để xử lý các đối tượng này. Dưới đây là một số ví dụ về cách sử dụng các lớp và hàm trong module datetime: **Ví dụ:**

```
import datetime

# Tạo một đối tượng date đại diện cho ngày hiện tại
today = datetime.date.today()
print(today) # output: 2023-05-02

# Tạo một đối tượng time đại diện cho thời gian hiện tại
now = datetime.datetime.now().time()
print(now) # output: 21:32:05.304540

# Tạo một đối tượng datetime đại diện cho thời điểm hiện tại
now = datetime.datetime.now()
print(now) # output: 2023-05-02 21:32:05.304588

# Tính khoảng cách thời gian giữa hai thời điểm
delta = datetime.timedelta(days=7)
print(delta) # output: 7 days, 0:00:00

# Tính ngày sau 7 ngày kể từ ngày hiện tại
next_week = today + delta
print(next_week) # output: 2023-05-09
```

Trong ví dụ trên:

- Hàm `datetime.date.today()` được sử dụng để tạo một đối tượng `date` đại diện cho ngày hiện tại.
 - Hàm `datetime.datetime.now().time()` được sử dụng để tạo một đối tượng `time` đại diện cho thời gian hiện tại.
 - Hàm `datetime.datetime.now()` được sử dụng để tạo một đối tượng `datetime` đại diện cho thời điểm hiện tại.
 - Hàm `datetime.timedelta (days=7)` được sử dụng để tạo một đối tượng `timedelta` đại diện cho khoảng thời gian 7 ngày.
 - Và cuối cùng, hàm `today + delta` được sử dụng để tính ngày sau 7 ngày kể từ ngày hiện tại.
- Module '`os`': Cung cấp các phương thức để tương tác với hệ điều hành,

chẳng hạn như tạo và xóa thư mục, điều khiển tiến trình, thực thi các lệnh hệ thống và nhiều hơn nữa.

Các phương thức trong module os cho phép bạn thực hiện các hoạt động như:

- Lấy thông tin về đường dẫn thư mục hiện tại.
- Tạo mới hoặc xóa bỏ các thư mục và tệp tin.
- Thực hiện các hành động trên tệp tin như đổi tên, sao chép, di chuyển.
- Lấy danh sách các tệp tin và thư mục trong một thư mục.
- Thực hiện các hành động liên quan đến quyền truy cập như kiểm tra xem một tệp tin có thể đọc hay không.

Một số ví dụ đối với module os như:

```
import os

# Lấy đường dẫn thư mục hiện tại
current_dir = os.getcwd()
print("Thư mục hiện tại là:", current_dir)

# Tạo thư mục mới
new_dir = "new_directory"
os.mkdir(new_dir)

# Thay đổi tên thư mục
os.rename(new_dir, "renamed_directory")

# Xóa thư mục
os.rmdir("renamed_directory")
```

- › Module ‘**sys**’: Cung cấp cho chúng ta các phương thức để tương tác với trình thông dịch Python và hệ thống hoạt động. Một số chức năng phổ biến của module sys bao gồm:
 - Thay đổi các tham số được truyền vào chương trình khi chúng ta chạy nó.

- Thêm đường dẫn cho Python tìm kiếm các module và package.
- Tạo và kiểm tra thông tin về phiên bản Python hiện tại.
- Tương tác với các đối tượng như ‘`stdout`’ và ‘`stderr`’ để ghi và đọc dữ liệu.

Ví dụ:

Lấy các tham số dòng lệnh được truyền vào khi chạy chương trình bằng cách sử dụng ‘`sys.argv`’.

```
import sys

print("Các tham số được truyền vào chương trình là:")
for arg in sys.argv:
    print(arg)
```

Cách thêm đường dẫn tìm kiếm các module và package vào danh sách tìm kiếm của Python.

```
import sys

sys.path.append("/path/to/module")
```

Lấy thông tin về phiên bản Python bằng cách sử dụng `sys.version` để lấy phiên bản Python hiện tại.

```
import sys

print("Phiên bản Python hiện tại là:", sys.version)
```

Ghi dữ liệu vào `stdout` và `stderr` bằng cách sử dụng `sys.stdout.write()` và `sys.stderr.write()`.

```
import sys

sys.stdout.write("Đây là một dòng trên stdout\n")
sys.stderr.write("Đây là một dòng trên stderr\n")
```

BÀI TẬP VẬN DỤNG

I. Bài tập

Bài tập vận dụng 1: Viết một chương trình Python sử dụng module datetime để hiển thị ngày và giờ hiện tại.

Bên cạnh khái niệm về module, Class là một khái niệm trong lập trình hướng đối tượng (OOP) của Python. Class là một khuôn mẫu để tạo ra các đối tượng (object) có cùng tính chất. Class chứa các thuộc tính (attribute) và phương thức (method) để mô tả đối tượng. Cú pháp để khai báo một class trong Python như sau:

```
class TenClass:  
    def __init__(self, thamso):  
        self.thuoc_tinh_1 = thamso  
        #...  
  
    def phuong_thuc_1(self):  
        #...  
        return ket_qua  
    #...
```

Trong đó:

- › TenClass là tên của class.
- › `__init__` là một phương thức đặc biệt được gọi khi khởi tạo đối tượng từ class.
- › `self` là tham số đại diện cho đối tượng.
- › `thuoc_tinh_1` là một thuộc tính của đối tượng.
- › `phuong_thuc_1` là một phương thức của đối tượng.

Khi tạo một đối tượng từ class, ta sử dụng cú pháp như sau:

```
ten_doi_tuong = TenClass(tham_so)
```

Trong đó:

- › ten_doi_tuong là tên của đối tượng.
- › TenClass là tên của class.
- › tham_so là tham số truyền vào phương thức __init__ để khởi tạo đối tượng.

Ví dụ minh họa cho Class :

```
class HocSinh:  
    def __init__(self, ten, tuoi):  
        self.ten = ten  
        self.tuoi = tuoi  
  
    def gioi_thieu(self):  
        print(f"Xin chao,toi ten la {self.ten}, {self.tuoi} tuoi.")  
  
hs1 = HocSinh("Nam", 20)  
hs1.gioi_thieu() # Xin chao,toi ten la Nam, 20 tuoi.
```

Trong ví dụ trên:

- › class HocSinh có hai thuộc tính là ten và tuoi.
- › một phương thức gioi_thieu để giới thiệu học sinh.
- › Khi tạo đối tượng hs1 từ class HocSinh, ta truyền vào hai tham số ten và tuoi.
- › Sau đó, ta gọi phương thức gioi_thieu của đối tượng hs1 để giới thiệu học sinh.

Bài tập vận dụng 2: Viết một Class Hình chữ nhật trong Python. Class này sẽ có các thuộc tính chiều dài (length) và chiều rộng (width), và các phương thức tính diện tích (area) và chu vi (perimeter) của hình chữ nhật.

II. Thao tác trên tập tin

Như phần Module chúng ta đã tìm hiểu, module ‘os’ giúp chúng ta có thể thao tác với các tệp tin, tập dữ liệu và một số ví dụ minh họa, ngoài ra module ‘pathlib’ cũng hỗ trợ người dùng thực hiện chức năng này. Vậy ngoài những thao tác đã được ví dụ trên, chúng ta có thể thực hiện được những thao tác nào khác. Sau đây là một vài thao tác cơ bản để

người dùng tương tác với tập tin bằng Python như sau:

- › Mở tập tin: Chúng ta có thể mở một tập tin bằng hàm open(). Đối với các tập tin văn bản, chúng ta thường mở chúng với chế độ đọc hoặc ghi:

```
# mở một tập tin văn bản với chế độ đọc  
f = open("filename.txt", "r")
```

```
# mở một tập tin văn bản với chế độ ghi  
f = open("filename.txt", "w")
```

- › Đọc dữ liệu từ tập tin: Chúng ta có thể đọc nội dung của tập tin bằng phương thức read() hoặc readline(). Phương thức read() sẽ đọc toàn bộ nội dung của tập tin và trả về một chuỗi, trong khi readline() sẽ đọc một dòng từ tập tin và trả về một chuỗi:

```
# đọc toàn bộ nội dung của tập tin  
f = open("filename.txt", "r")  
content = f.read()  
print(content)
```

```
# đọc từng dòng của tập tin  
f = open("filename.txt", "r")  
line = f.readline()  
while line:  
    print(line)  
    line = f.readline()
```



- › Ghi dữ liệu vào tập tin: Chúng ta có thể ghi dữ liệu vào tập tin bằng phương thức write(). Phương thức này sẽ ghi một chuỗi vào tập tin:

```
# ghi dữ liệu vào tập tin  
f = open("filename.txt", "w")  
f.write("hello world")  
f.close()
```

- Đóng tập tin: Sau khi sử dụng tập tin, chúng ta nên đóng nó bằng phương thức close():

```
# đóng tập tin
f = open("filename.txt", "r")
content = f.read()
f.close()
```

- Ngoài ra, chúng ta còn có thể sử dụng các phương thức khác để thao tác với tập tin như kiểm tra sự tồn tại của tập tin (exists()), lấy thông tin về tập tin (stat()), tạo thư mục mới (mkdir()), xóa tập tin (remove()) hoặc đổi tên tập tin (rename()).

Sau đây chúng ta có một vài mode dùng cho việc mở file và các phương thức làm việc với file bằng Python mà các bạn có thể tham khảo và tìm hiểu thêm:

1. Mode mở file:

MODE	MÔ TẢ
r	Chỉ được phép đọc.
r+	Được phép đọc và ghi.
rb	Mở file chế độ đọc cho định dạng nhị phân. Con trỏ tại phần bắt đầu của file.
rb+ r+b	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file.
w	Mở file để ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ.
w+	Mở file để đọc và ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ.

wb	Mở file để ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ.
wb+ w+b	Mở file để đọc và ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ.
a	Mở file chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
a+	Mở file chế độ đọc và ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
ab	Mở file chế độ ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
ab+ a+b	Mở file chế độ đọc và ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
x	Mở file chế độ ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi.
x+	Mở file chế độ đọc và ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi.
xb	Mở file chế độ ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi.

xb+	Mở file chế độ đọc và ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi.
b	Mở file ở chế độ nhị phân.
t	Mở file ở chế độ văn bản (mặc định).

2. Phương thức làm việc với file trong Python:

PHƯƠNG THỨC	MÔ TẢ
close()	Đóng một file đang mở. Nó không thực thi được nếu tập tin đã bị đóng.
fileno()	Trả về 1 số nguyên mô tả file (file descriptor).
flush()	Xóa sạch bộ nhớ đệm của luồng file.
isatty()	Trả về True nếu file được kết nối với một thiết bị đầu cuối.
read(n)	Đọc n ký tự trong file.
readable()	Trả về True nếu file có thể đọc được.
readline(n=-1)	Đọc và trả về 1 dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.
readlines(n=-1)	Đọc và trả về một danh sách các dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.

seek(offset,from=SEEK_SET)	Thay đổi vị trí hiện tại bên trong file.
seekable()	Trả về True nếu luồng hỗ trợ truy cập ngẫu nhiên.
tell()	Trả về vị trí hiện tại bên trong file.
truncate(size=None)	Cắt gọn kích cỡ file thành kích cỡ tham số size.
writable()	Trả về True nếu file có thể ghi được.
write(s)	Ghi s ký tự vào trong file và trả về.
writelines(lines)	Ghi một danh sách các dòng và file.

III. Xử lý hình ảnh, file JSON, XML

Với Python, ngoài việc tương tác với các file dữ liệu, chúng ta hoàn toàn có thể thao tác và xử lý với các file hình ảnh hoặc các file JSON, XML,...

Để xử lý hình ảnh trong Python, chúng ta có thể sử dụng thư viện Pillow (PIL), một thư viện mã nguồn mở giúp chúng ta thao tác với hình ảnh và chuyển đổi giữa các định dạng hình ảnh khác nhau. Để sử dụng Pillow, bạn cần cài đặt thư viện bằng lệnh sau: '**pip install Pillow**'.

1. Dưới đây là một số thao tác cơ bản khi xử lý hình ảnh bằng Pillow:

- Đọc và hiển thị hình ảnh:

```
from PIL import Image

# đọc hình ảnh từ tập tin
image = Image.open("image.jpg")

# hiển thị hình ảnh
image.show()
```

- › Chuyển đổi định dạng hình ảnh:

```
from PIL import Image

# đọc hình ảnh từ tập tin
image = Image.open("image.jpg")

# chuyển đổi thành định dạng PNG và lưu vào tập tin mới
image.save("new_image.png", "PNG")
```

- › Thay đổi kích thước hình ảnh:

```
from PIL import Image

# đọc hình ảnh từ tập tin
image = Image.open("image.jpg")

# thay đổi kích thước hình ảnh
new_size = (500, 500)
resized_image = image.resize(new_size)

# lưu hình ảnh đã thay đổi kích thước vào tập tin mới
resized_image.save("resized_image.jpg")
```

- › Xoay hình ảnh:

```
from PIL import Image

# đọc hình ảnh từ tập tin
image = Image.open("image.jpg")

# xoay hình ảnh 90 độ
rotated_image = image.rotate(90)

# lưu hình ảnh xoay mới vào tập tin mới
rotated_image.save("rotated_image.jpg")
```

- › Lọc hình ảnh:

```
from PIL import Image, ImageFilter

# đọc hình ảnh từ tập tin
image = Image.open("image.jpg")

# áp dụng bộ lọc GaussianBlur để làm mờ hình ảnh
blurred_image = image.filter(ImageFilter.GaussianBlur(radius=10))

# lưu hình ảnh đã được làm mờ vào tập tin mới
blurred_image.save("blurred_image.jpg")
```

- › Cắt hình ảnh:

```
from PIL import Image

# mở hình ảnh từ đường dẫn
img = Image.open('path/to/image.jpg')

# cắt hình ảnh
box = (100, 100, 400, 400) # (left, upper, right, lower)
cropped_img = img.crop(box)
```



- › Pillow cũng hỗ trợ nhiều thao tác khác nhau và có thể được sử dụng để thực hiện các công việc phức tạp hơn như phân tích hình ảnh hoặc phát hiện đối tượng trong hình ảnh.

Ngoài ra, chúng ta có thể sử dụng Python để xử lý tập tin XML và JSON.

Trong lập trình ứng dụng web, XML được sử dụng nhiều nhất là xây dựng các API Service. Các API sẽ trả kết quả về dạng XML hoặc JSON để các hệ thống khác có thể nói chuyện với nhau được. Hiện nay tuy JSON được sử dụng phổ biến hơn, nhưng XML cũng vẫn đang được dùng bởi nhiều hệ thống lớn.

Trước tiên chúng ta cần biết, XML là từ viết tắt của từ **Extensible Markup Language** là ngôn ngữ đánh dấu mở rộng. XML có chức năng truyền dữ liệu và mô tả nhiều loại dữ liệu khác nhau. Tác dụng chính của XML là đơn giản hóa việc chia sẻ dữ liệu giữa các nền tảng và các hệ thống được kết nối thông qua mạng Internet.

XML dùng để cấu trúc, lưu trữ và trong trao đổi dữ liệu giữa các ứng dụng và lưu trữ dữ liệu. Ví dụ khi ta xây dựng một ứng dụng bằng PHP và một

ứng dụng bằng Java thì hai ngôn ngữ này không thể hiểu nhau, vì vậy ta sẽ sử dụng XML để trao đổi dữ liệu. Chính vì vậy, XML có tác dụng rất lớn trong việc chia sẻ, trao đổi dữ liệu giữa các hệ thống.

Để xử lý tập tin XML, ta có thể sử dụng thư viện built-in ‘xml’ hoặc thư viện bên ngoài như ‘lxml’ hoặc ‘xml.etree.ElementTree’. Trong đó, thư viện phổ biến nhất là ‘xml.etree.ElementTree’ đây là module chuẩn được tích hợp sẵn trong Python để xử lý các tệp XML.

Đầu tiên, để sử dụng module nói trên, chúng ta cần import module `xml.etree.ElementTree` bằng câu lệnh:

```
import xml.etree.ElementTree as ET
```

Sau đó, để đọc file XML, ta sử dụng hàm `ET.parse()` để tạo một đối tượng `ElementTree`. Ví dụ, để đọc file `example.xml`:

```
tree = ET.parse('example.xml')
```

Đối tượng `ElementTree` có thể truy xuất các phần tử trong tệp XML bằng cách sử dụng phương thức `getroot()`. Phần tử gốc sẽ là nơi bắt đầu cho tất cả các tìm kiếm, truy cập và thay đổi các phần tử trong tệp XML.

```
root = tree.getroot()
```

Sau khi có đối tượng phần tử gốc, ta có thể truy xuất các phần tử con bằng cách sử dụng các phương thức như `find()`, `findall()`, `iter()` hoặc `iterfind()`. Ví dụ, để tìm phần tử `<name>` đầu tiên trong tệp XML:

```
first_name = root.find('name').text
```

Hoặc để tìm tất cả các phần tử `<name>`:

```
for name in root.findall('name'):
    print(name.text)
```

Nếu muốn truy cập các thuộc tính của phần tử, ta có thể sử dụng cú pháp:

```
value = element.get('attribute_name')
```

Nếu muốn thay đổi các phần tử trong tệp XML, chúng ta có thể sử dụng các phương thức của đối tượng phần tử để thực hiện các thay đổi. Ví dụ, để thay đổi nội dung của phần tử `<name>` đầu tiên, ta có thể sử dụng cú pháp:

```
root.find('name').text = 'new_name'
```

Cuối cùng, để lưu các thay đổi của tệp XML, ta sử dụng hàm `ET.ElementTree.write()` và truyền vào tên file cần lưu. Ví dụ:

```
tree.write('new_example.xml')
```

Để có một cái nhìn tổng quát, cụ thể và dễ hiểu hơn. Chúng ta sẽ đi đến một ví dụ cụ thể và các câu lệnh sử dụng để dễ dàng hơn trong việc tiếp cận kiến thức mới. Giả sử chúng ta có một tệp '`data.xml`' như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```



Ta có thể tải nó vào một đối tượng cây cấu trúc XML như sau:

```
tree = ET.parse('data.xml')
root = tree.getroot()
```

Đối tượng root lưu trữ thẻ gốc của cây cấu trúc XML (<data> trong ví dụ trên). Ta có thể lấy các phần tử con của thẻ gốc bằng cách sử dụng thuộc tính children của root:

```
for child in root:
    print(child.tag, child.attrib)
```

Kết quả sẽ là danh sách các phần tử con của thẻ gốc, trong đó mỗi phần tử được biểu diễn bằng tên thẻ và thuộc tính của nó:

```
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

Nếu muốn lấy các giá trị của các phần tử con, ta có thể sử dụng phương thức find() để tìm một phần tử theo tên thẻ và phương thức text để lấy giá trị của phần tử đó:

```
for country in root.findall('country'):
    rank = country.find('rank').text
    name = country.get('name')
    print(name, rank)
```

Kết quả sẽ là danh sách các quốc gia và xếp hạng của chúng:

```
Liechtenstein 1
Singapore 4
Panama
```

2. Một số quy tắc cú pháp cho thuộc tính trong XML:

- Tên thuộc tính trong XML là phân biệt kiểu chữ (không giống như HTML). Tức là, HREF và href là hai thuộc tính khác nhau trong XML.

- Cùng một thuộc tính không thể có hai giá trị trong một cú pháp. Ví dụ sau là sai cú pháp bởi vì thuộc tính b được xác định hai lần:

```
<a b="x" c="y" b="z">...</a>
```

- Tên thuộc tính được định nghĩa không có sự trích dẫn, trong khi giá trị thuộc tính phải luôn luôn trong các dấu trích dẫn. Ví dụ sau là sai cú pháp:

```
<a b=x>...</a>
```

2.1 Quy tắc tham chiếu trong XML:

Tham chiếu (References) thường cho phép bạn thêm hoặc bao phần text hoặc phần đánh dấu bổ sung trong một tài liệu XML. Các tham chiếu luôn luôn bắt đầu với biểu tượng “&”, đây là ký tự dành riêng và kết thúc với ký tự “;”. XML có hai kiểu tham chiếu:

- Tham chiếu thực thể (Entity Reference): Một tham chiếu thực thể chứa một tên giữa dấu tách mở và dấu tách đóng. Ví dụ: & có amp là tên. Tên tham chiếu tới một chuỗi văn bản hoặc đánh dấu đã được định nghĩa trước.
- Tham chiếu ký tự (Character Reference): Chứa các tham chiếu, ví dụ A, chứa một dấu băm (#) được theo sau bởi một số. Số này luôn luôn tham chiếu tới mã hóa Unicode của ký tự. Trong ví dụ này, 65 tham chiếu tới chữ cái “A”.

2.2 Quy tắc về Text trong XML:

- Tên của phần tử XML và thuộc tính XML là phân biệt kiểu chữ, nghĩa là tên của phần tử mở và phần tử đóng phải ở được viết cùng kiểu.
- Để tránh các vấn đề về mã hóa ký tự, tất cả XML file nên được lưu ở dạng Unicode UTF-8 hoặc UTF-16.
- Các ký tự whitespace như khoảng trắng, tab và ngắt dòng giữa các phần tử XML và giữa các thuộc tính XML sẽ bị bỏ qua.
- Một số ký tự được dành riêng trong cú pháp XML. Vì thế, chúng không thể được sử dụng một cách trực tiếp. Để sử dụng chúng, một số

thực thể thay thế được sử dụng, các thực thể này được liệt kê trong bảng dưới:

Không dùng	Thay thế	Mô tả
<	<	Nhỏ hơn.
>	>	Lớn hơn.
&	&	Và.
'	'	Nháy đơn.
"	"	Trích dẫn kép.

Toàn bộ đây là những cách cơ bản để xử lý file XML trong Python bằng module `xml.etree.ElementTree`. Tùy vào mục đích sử dụng và yêu cầu cụ thể, chúng ta có thể sử dụng các thư viện khác như `lxml`, `xml.dom.minidom`, `xml.sax`,... để xử lý XML trong Python.

Tiếp theo, chúng ta sẽ tìm hiểu về JSON (JavaScript Object Notation): Là một định dạng dữ liệu rất phổ biến, được dùng để lưu trữ và thể hiện các dữ liệu có cấu trúc. JSON là định dạng dữ liệu phổ biến được sử dụng để truyền và nhận dữ liệu giữa ứng dụng web và web server.

Để xử lý tệp tin JSON bằng Python, ta cần sử dụng thư viện tích hợp có sẵn trong Python là `JSON`. Thư viện này cung cấp các hàm để encode và decode các đối tượng JSON.

3. JSON

3.1 Đọc tệp tin JSON:

Dùng hàm `json.load()` để đọc nội dung tệp JSON và chuyển đổi thành đối tượng Python. Ví dụ, nếu ta có tệp tin JSON như sau:

```
{  
    "name": "John",  
    "age": 30,  
    "city": "New York"  
}
```

Ta có thể đọc tệp tin này và lưu vào một biến dictionary bằng cách sử dụng đoạn code sau:

```
import json

# Mở tệp tin JSON
with open('data.json') as f:
    # Load dữ liệu từ tệp tin vào biến data
    data = json.load(f)

# In ra biến data
print(data)

# Kết quả đạt được
{'name': 'John', 'age': 30, 'city': 'New York'}
```

3.2 Viết tệp tin JSON:

Dùng hàm `json.dump()` để chuyển đổi đối tượng Python thành định dạng JSON và ghi vào tệp tin. Ví dụ, nếu ta có một dictionary như sau:

```
import json

# Tạo đối tượng Python
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Ghi đối tượng Python thành tệp JSON
with open('data.json', 'w') as f:
    json.dump(data, f)
```

3.3 Truy cập đến các phần tử trong JSON:

Để truy cập đến các phần tử trong JSON, ta có thể sử dụng cú pháp tương tự như truy cập đến các phần tử trong dictionary.

- Dùng cú pháp `[key]` để truy cập đến giá trị của từ khóa trong đối tượng JSON.

- › Dùng cú pháp [index] để truy cập đến phần tử trong mảng JSON.

Ví dụ, nếu ta có một tệp tin JSON như sau:

```
{  
    "name": "John",  
    "age": 30,  
    "city": "New York",  
    "pets": [  
        { "name": "Buddy", "type": "dog" },  
        { "name": "Fluffy", "type": "cat" }  
    ]  
}
```

Ta có thể truy cập đến các phần tử trong JSON như sau:

```
import json  
  
# Mở tệp tin JSON  
with open('data.json') as f:  
    # Load dữ liệu từ tệp tin vào biến data  
    data = json.load(f)  
  
    # Truy cập đến các phần tử trong JSON  
    name = data['name']  
    age = data['age']  
    city = data['city']  
    pets = data['pets']  
  
    # Truy cập đến các phần tử trong mảng pets  
    pet1 = pets[0]  
    pet2 = pets[1]  
    pet1
```

3.4 Chỉnh sửa dữ liệu trong tệp JSON:

Dùng cú pháp [key] hoặc [index] để truy cập đến giá trị cần chỉnh sửa. Sau đó, gán giá trị mới cho phần tử đó. Ví dụ minh họa cho trường hợp này như sau:

```
import json

# Đọc tệp JSON
with open('data.json', 'r') as f:
    data = json.load(f)

# Chỉnh sửa giá trị của từ khóa "name"
data["name"] = "Tom"

# Chỉnh sửa giá trị của phần tử thứ hai trong mảng "scores"
data["scores"][1] = 90

# Ghi đổi tượng Python mới vào tệp JSON
with open('data.json', 'w') as f:
    json.dump(data, f)
```

Khi xử lý file JSON trong Python, có một số lưu ý quan trọng như sau:

- › Kiểm tra định dạng JSON: Trước khi bắt đầu xử lý file JSON, hãy đảm bảo rằng định dạng của file đó là JSON. Nếu không, chương trình sẽ không thể đọc được file và gây ra lỗi.
- › Sử dụng module JSON: Python cung cấp module JSON để hỗ trợ việc đọc và ghi file JSON. Hãy sử dụng module này để đảm bảo tính đúng đắn và hiệu quả của chương trình.
- › Kiểm tra dữ liệu đầu vào: Khi xử lý file JSON, đặc biệt là khi đọc file, hãy kiểm tra kỹ dữ liệu đầu vào. Nếu dữ liệu không đúng định dạng hoặc không đầy đủ, chương trình sẽ không hoạt động đúng.
- › Sử dụng try-except: Khi xử lý file JSON, hãy sử dụng try-except để bắt các lỗi có thể xảy ra trong quá trình xử lý. Điều này sẽ giúp chương trình hoạt động ổn định hơn và dễ dàng sửa lỗi.

- › Sử dụng encoding đúng: khi ghi file JSON, hãy đảm bảo sử dụng encoding đúng để tránh lỗi khi đọc lại file.
- › Sử dụng các thao tác đọc và ghi file đúng cách: đối với các tập tin JSON lớn, nên sử dụng các thao tác đọc và ghi file đúng cách để tối ưu hiệu suất của chương trình. Ví dụ, nên sử dụng đọc và ghi file theo từng phần để tránh tải toàn bộ dữ liệu vào bộ nhớ đồng thời.
- › Sử dụng các công cụ hỗ trợ: Có nhiều công cụ hỗ trợ xử lý file JSON trong Python như:
 - jq: Là một công cụ dòng lệnh miễn phí và mã nguồn mở được sử dụng để phân tích và xử lý các tệp dữ liệu JSON. Nó cung cấp các tính năng mạnh mẽ để truy vấn, lọc và biến đổi dữ liệu JSON. Công cụ này thường được sử dụng trong các kịch bản tự động hóa, đặc biệt là trong các hệ thống đám mây và DevOps, để giải quyết các vấn đề liên quan đến dữ liệu JSON.
 - json.tool: Một thuật ngữ trong lập trình và có liên quan đến JSON, một định dạng dữ liệu phổ biến trong truyền thông và lập trình web. "json.tool" có thể được hiểu là một công cụ, thư viện hoặc chương trình hỗ trợ trong việc xử lý và định dạng các file JSON. Cụ thể, json.tool giúp các lập trình viên đọc, ghi, sửa và tạo các file JSON một cách dễ dàng và hiệu quả hơn.
 - jsonlint: Một công cụ hỗ trợ xử lý file JSON trong Python, giúp kiểm tra tính hợp lệ của cú pháp JSON và đưa ra thông báo lỗi cụ thể. Nó có thể được sử dụng để đảm bảo rằng các file JSON được tạo ra hoặc sử dụng bởi các ứng dụng Python là đúng cú pháp và có thể được phân tích đúng. Công cụ jsonlint có thể được cài đặt bằng cách sử dụng pip, một trình quản lý gói cho Python.

Nên sử dụng các công cụ này để giúp đỡ trong quá trình xử lý file JSON.

IV. Ví dụ thực tế

Đi qua những kiến thức về Python mà chúng ta đã cùng tìm hiểu ở trên, một ví dụ cụ thể, thực tế về ngôn ngữ lập trình Python có thể thực hiện được.

Đó chính là ứng dụng ngôn ngữ lập trình Python để lập trình một phần mềm đơn giản dùng để chuyển văn bản thành giọng nói.

Để viết một ứng dụng Python để chuyển đổi văn bản thành giọng nói, chúng ta cần sử dụng một số thư viện như gTTS để chuyển đổi văn bản thành âm thanh và pygame để phát lại âm thanh đó.

Dưới đây là một ví dụ về một ứng dụng đơn giản để chuyển đổi văn bản thành giọng nói.

Ứng dụng này sử dụng thư viện gTTS để chuyển đổi văn bản thành âm thanh, lưu lại file âm thanh dưới dạng mp3. Sau đó, thư viện pygame được sử dụng để phát lại file âm thanh này. Cuối cùng, file âm thanh được xóa khỏi hệ thống để tránh lưu trữ không cần thiết.

```
# Import các thư viện cần thiết
from gtts import gTTS
import pygame
import os

# Xác định đoạn văn bản cần chuyển đổi
text = "Hello, how are you?"

# Tạo đối tượng gTTS và lưu tệp âm thanh
tts = gTTS(text=text, lang='en')
tts.save("audio.mp3")

# Khởi tạo pygame
pygame.mixer.init()

# Tải tệp âm thanh và phát
pygame.mixer.music.load("audio.mp3")
pygame.mixer.music.play()

# Đợi âm thanh phát xong
while pygame.mixer.music.get_busy():
    continue

# Xóa tệp âm thanh
os.remove("audio.mp3")
```



Vận dụng những kiến thức trên, cùng những hiểu biết và những thông tin mà các bạn đã tự tìm hiểu. Chúng ta hãy cùng nhau thử thách với một bài tập lớn sau đây:

Bài tập vận dụng 3: Tính tuổi dựa vào ngày tháng năm sinh nhập vào.

Bài tập vận dụng 4: Viết một chương trình chấp nhận chuỗi từ do người dùng nhập vào, phân tách nhau bởi dấu phẩy và in những từ đó thành chuỗi theo thứ tự bảng chữ cái, phân tách nhau bằng dấu phẩy.

Giả sử đầu vào được nhập là: without, hello, bag, world.

Thì đầu ra sẽ là: bag, hello, without, world.

Gợi ý: Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.

Bài tập vận dụng 5: Tạo một chương trình quản lý đồ uống sử dụng Python.

Yêu cầu:

- › Sử dụng module json để lưu thông tin đồ uống vào tập tin beverages.json.
- › Sử dụng class để định nghĩa đối tượng đồ uống với các thuộc tính sau:
 - Tên đồ uống.
 - Giá tiền.
 - Số lượng.
 - Mô tả.
- › Cung cấp các chức năng sau:
 - Thêm đồ uống mới vào danh sách.
 - Xóa đồ uống khỏi danh sách.
 - Sửa thông tin đồ uống.
 - Tìm kiếm đồ uống theo tên.
 - Hiển thị danh sách đồ uống và tổng giá trị.

Bài tập vận dụng 6: Viết chương trình Python để nhập điểm trung bình của n sinh viên và tính toán tổng, trung bình, điểm cao nhất và điểm thấp nhất. Sau đó, phân loại điểm trung bình của sinh viên thành các hạng A, B, C, D và F theo bảng điểm sau:

- › A: Điểm trung bình từ 9.0 đến 10.0.
- › B: Điểm trung bình từ 8.0 đến 8.9.
- › C: Điểm trung bình từ 7.0 đến 7.9.
- › D: Điểm trung bình từ 6.0 đến 6.9.
- › F: Điểm trung bình từ 0 đến 5.9.

Yêu cầu:

- › Sử dụng module math để tính toán trung bình, điểm cao nhất và điểm thấp nhất.
- › Sử dụng module random để tạo ngẫu nhiên điểm trung bình của sinh viên.
- › Sử dụng câu lệnh điều kiện if-elif-else để phân loại điểm trung bình của sinh viên.
- › Sử dụng vòng lặp for để lặp qua danh sách điểm trung bình và tính toán tổng.

V. Lời giải

Bài tập vận dụng 1:

Để giải quyết bài toán, chúng ta cần sử dụng module datetime có sẵn trong Python. Module datetime cung cấp các lớp để đại diện cho các thời gian khác nhau (ngày, giờ, phút, giây, micro giây,...) và các chức năng để xử lý chúng.

Để bắt đầu, trước tiên chúng ta cần nhập module datetime bằng cách sử dụng lệnh import datetime. Sau đó, chúng ta có thể sử dụng lớp datetime để biểu diễn ngày và giờ hiện tại.

Đoạn mã dưới đây là ví dụ cho chương trình hiển thị ngày và giờ hiện tại:

```
import datetime  
  
now = datetime.datetime.now()  
  
print("Ngày và giờ hiện tại:")  
print(now.strftime("%Y-%m-%d %H:%M:%S"))
```

Giải thích bài giải trên như sau:

- › Trong đó, `datetime.datetime.now()` trả về thời gian hiện tại và định dạng thời gian được xác định bằng phương thức `strftime()`. Trong ví dụ này, chúng ta sử dụng định dạng `%Y-%m-%d %H:%M:%S` để hiển thị ngày và giờ hiện tại.
- › Kết quả khi chạy chương trình sẽ là một dòng chữ hiển thị ngày và giờ hiện tại, ví dụ: "**Ngày và giờ hiện tại: 2021-12-01 14:30:00**".
- › Tùy vào nhu cầu sử dụng, chúng ta có thể sửa đổi định dạng hiển thị ngày và giờ bằng cách sử dụng các chuỗi định dạng khác nhau trong phương thức `strftime()`.

Bài tập vận dụng 2:

Để giải quyết bài toán trên, ta sử dụng class trong Python để tạo ra một đối tượng Hình chữ nhật. Một lớp đối tượng trong Python được định nghĩa bằng cách sử dụng từ khóa `class` và tên lớp. Sau đó, các thuộc tính và phương thức được định nghĩa bên trong lớp.

Ví dụ, chương trình dưới đây minh họa cách định nghĩa lớp Hình chữ nhật và tính diện tích và chu vi của một hình chữ nhật cụ thể:

```

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

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

# tạo một đối tượng Rectangle với chiều dài 4 và chiều rộng 3
rectangle = Rectangle(4, 3)

# tính diện tích và chu vi của đối tượng Rectangle
print("Diện tích của hình chữ nhật là:", rectangle.area())
print("Chu vi của hình chữ nhật là:", rectangle.perimeter())

```

Kết quả đạt được sẽ là :

- › Diện tích của hình chữ nhật là: 12.
- › Chu vi của hình chữ nhật là: 14.

Giải thích bài giải trên như sau:

- › Trong chương trình trên, chúng ta định nghĩa một lớp Rectangle với hai thuộc tính là length và width (chiều dài và chiều rộng) được khởi tạo trong phương thức `__init__`. Phương thức `area()` tính diện tích của hình chữ nhật bằng cách nhân chiều dài với chiều rộng, và phương thức `perimeter()` tính chu vi bằng cách cộng hai lần chiều dài và chiều rộng.
- › Sau đó, chúng ta tạo một đối tượng Rectangle với chiều dài là 4 và chiều rộng là 3. Cuối cùng, chúng ta tính diện tích và chu vi của đối tượng Rectangle bằng cách sử dụng các phương thức của lớp Rectangle.

Bài tập vận dụng 3: Để giải quyết bài toán này, ta dùng đoạn code như sau:

```
import datetime

print("Mời bạn vui lòng nhập ngày tháng năm sinh để tính tuổi")
birth_day = int(input("Ngày sinh:"))
birth_month = int(input("Tháng sinh:"))
birth_year = int(input("Năm sinh:"))

current_year = datetime.date.today().year
current_month = datetime.date.today().month
current_day = datetime.date.today().day

age_year = current_year - birth_year
age_month = abs(current_month - birth_month)
age_day = abs(current_day - birth_day)

print("### Tuổi của bạn chính xác là:##\n",
      age_year, "tuổi",
      age_month, "tháng và",
      age_day, "ngày")
```

Bài tập vận dụng 4: Để giải quyết bài toán này, ta dùng đoạn code như sau:

```
items=[x for x in input("Nhập một chuỗi: ").split(',')]

items.sort()

print(', '.join(items))
```

Bài tập vận dụng 5 :

Để bắt đầu, ta cần tạo file beverages.json trống và tạo class Beverage để định nghĩa đối tượng đồ uống.

```

import json

class Beverage:
    def __init__(self, name, price, quantity, description):
        self.name = name
        self.price = price
        self.quantity = quantity
        self.description = description

    def to_dict(self):
        return {
            'name': self.name,
            'price': self.price,
            'quantity': self.quantity,
            'description': self.description
        }

```

Trong đó, phương thức `to_dict` sẽ chuyển đổi đối tượng `Beverage` thành một từ điển để lưu vào file json.

Tiếp theo, ta tạo các chức năng thêm, xóa, sửa, tìm kiếm, hiển thị danh sách.

```

class BeverageManager:
    def __init__(self, file_path):
        self.file_path = file_path
        self.beverages = []
        self.load_data()

    def load_data(self):
        try:
            with open(self.file_path, 'r') as f:
                data = json.load(f)
                for b in data:
                    self.beverages.append(Beverage(b['name'], b['price'], b['quantity'], b['description']))
        except FileNotFoundError:
            pass

    def save_data(self):
        with open(self.file_path, 'w') as f:
            data = [b.to_dict() for b in self.beverages]
            json.dump(data, f, indent=4)

    def add_beverage(self, name, price, quantity, description):
        beverage = Beverage(name, price, quantity, description)
        self.beverages.append(beverage)
        self.save_data()

    def remove_beverage(self, name):
        self.beverages = [b for b in self.beverages if b.name != name]
        self.save_data()

```

```
def update_beverage(self, name, price, quantity, description):
    for beverage in self.beverages:
        if beverage.name == name:
            beverage.price = price
            beverage.quantity = quantity
            beverage.description = description
    self.save_data()

def search_beverage(self, name):
    for beverage in self.beverages:
        if beverage.name == name:
            return beverage
    return None

def display_beverages(self):
    total_value = 0
    for beverage in self.beverages:
        total_value += beverage.price * beverage.quantity
    print(f"{beverage.name} - {beverage.price} USD - {beverage.quantity} pcs")
    print(f"Total value: {total_value} USD")

# Example usage:
manager = BeverageManager('beverages.json')
manager.add_beverage('Coffee', 2.5, 10, 'Hot coffee')
manager.add_beverage('Tea', 1.5, 15, 'Green tea')
manager.add_beverage('Cola', 1.0, 20, 'Soft drink')
manager.display_beverages()
```

```
import math
import random

# Nhập số lượng sinh viên
n = int(input("Nhập số lượng sinh viên: "))

# Tạo danh sách điểm trung bình ngẫu nhiên
diem_trung_binh = [random.uniform(0, 10) for _ in range(n)]

# Tính tổng, trung bình, điểm cao nhất và điểm thấp nhất
tong = sum(diem_trung_binh)
trung_binh = tong / n
cao_nhat = max(diem_trung_binh)
thap_nhat = min(diem_trung_binh)

print("Tổng điểm trung bình:", tong)
print("Điểm trung bình:", trung_binh)
print("Điểm cao nhất:", cao_nhat)
print("Điểm thấp nhất:", thap_nhat)

# Phân loại điểm trung bình của sinh viên
for diem in diem_trung_binh:
    if diem >= 9.0:
        print("Hạng A")
    elif diem >= 8.0:
        print("Hạng B")
    elif diem >= 7.0:
        print("Hạng C")
    elif diem >= 6.0:
        print("Hạng D")
    else:
        print("Hạng F")
```