

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG
KHOA CÔNG NGHỆ THÔNG TIN



BÀI GIẢNG

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
(OBJECT ORIENTED PROGRAMMING)

BIÊN SOẠN

ThS. NGUYỄN VĂN HIẾU

ThS. TRẦN THỊ CẨM TÚ

ThS. MAI THIÊN THƯ

ThS. NGUYỄN VĂN NĂNG

Vĩnh Long - 2022

LỜI NÓI ĐẦU

-----❧-----

Ngày nay, ngôn ngữ lập trình Python đã trở thành một công cụ thiết yếu cho nhiều lập trình viên. Python là một ngôn ngữ lập trình cấp cao được sử dụng cho mục đích lập trình đa năng với nhiều ưu điểm mạnh là dễ đọc, dễ hiểu và dễ nhớ. Để đáp ứng nhu cầu học tập và nghiên cứu của các bạn sinh viên, đặc biệt là sinh viên chuyên ngành Công nghệ thông tin, chúng tôi đã tiến hành nghiên cứu và biên soạn bài giảng **Lập trình hướng đối tượng**. Bài giảng này được biên soạn dựa trên rất nhiều tài liệu tham khảo về ngôn ngữ lập trình Python đáng tin cậy.

Tài liệu này được biên soạn dựa theo đề cương chi tiết học phần **Lập trình hướng đối tượng** của Khoa Công nghệ thông tin - Trường Đại học Sư phạm Kỹ thuật Vĩnh Long và các tài liệu trong và ngoài nước. Bài giảng gồm 10 chương như sau:

Chương 1: Tổng quan về ngôn ngữ lập trình Python

Chương 2: Các cấu trúc điều khiển

Chương 3: Dữ liệu kiểu chuỗi

Chương 4: Dữ liệu kiểu tập hợp

Chương 5: Hàm và Module

Chương 6: Tổng quan về lập trình hướng đối tượng

Chương 7: Lớp và đối tượng

Chương 8: Kỹ thuật kế thừa

Chương 9: Quá tải toán tử

Chương 10: Làm việc với tập tin và thư mục

Mục tiêu của bài giảng này nhằm giúp các bạn sinh viên chuyên ngành và những ai muốn tìm hiểu về lập trình hướng đối tượng bằng ngôn ngữ Python có một tài liệu cô đọng dùng làm tài liệu học tập và nghiên cứu. Chúng tôi tin rằng các bạn sinh viên chuyên ngành và những ai quan tâm đến lập trình Python sẽ tìm được những kiến thức bổ ích từ quyển bài giảng này.

Mặc dù chúng tôi đã cố gắng biên soạn một cách cô đọng và đầy đủ nhất nhưng chắc chắn vẫn còn thiếu sót. Rất mong được sự đóng góp quý báu của các bạn để quyển bài giảng này ngày càng hoàn thiện hơn.

Chân thành cảm ơn!

Nhóm tác giả

Chương 1: TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH PYTHON



1. GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH PYTHON

1.1. Python là gì

Python là một ngôn ngữ lập trình cấp cao cho các mục đích lập trình đa năng, do Guido Van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất trong sáng, cấu trúc rõ ràng, thuận tiện cho người học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím là ít nhất có thể.

Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát vùng nhớ động. Nó được phát triển trong một dự án mã nguồn mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

1.2. Lịch sử phát triển của ngôn ngữ lập trình Python

Ban đầu, Python được phát triển để chạy trên nền hệ điều hành Unix. Nhưng rồi theo thời gian, Python dần mở rộng sang mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido Van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ yếu trong việc quyết định hướng phát triển của Python.

Sự phát triển của Python đến nay có thể chia thành 3 giai đoạn:

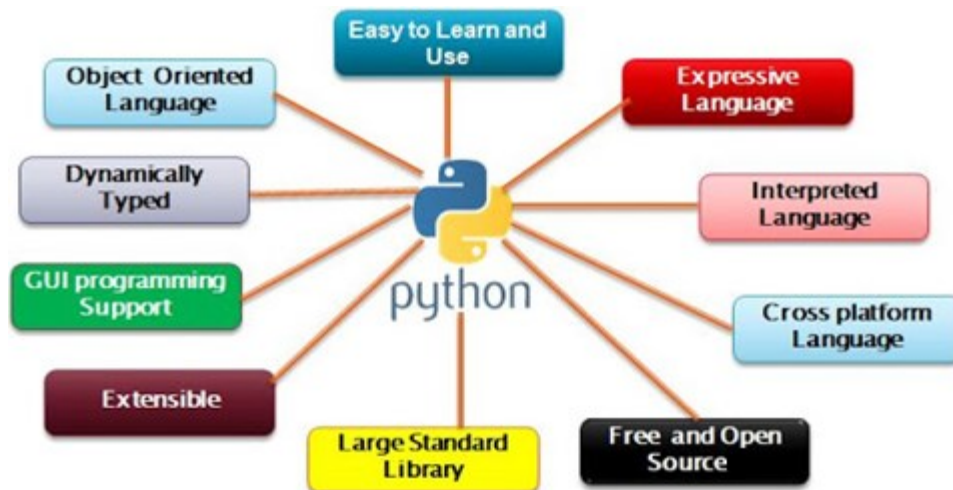
- Python 1: Bao gồm các bản phát hành 1.x. Giai đoạn này kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido làm việc tại CWI (Centrum voor Wiskunde en Informatica - Trung tâm Toán-Tin học tại Amsterdam, Hà Lan), do vậy các bản phát hành đầu tiên đều do CWI phát hành và phiên bản cuối cùng được phát hành tại đây là 1.2. Vào năm 1995, Guido chuyển sang CNRI (Corporation for National Research Initiatives) ở Reston, Virginia. Tại đây ông phát hành một số phiên bản khác và Python 1.6 là phiên bản cuối cùng được phát hành tại CNRI. Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn GPL. Sau đó, CNRI và FSF (Free Software Foundation - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Phiên bản 1.6.1 ra đời, đó là phiên bản đầu tiên tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.

- Python 2: Vào năm 2000, Guido và nhóm phát triển Python dời đến BeOpen.com và thành lập BeOpen PythonLabs Team. Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập Digital Creations. Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của Python Software Foundation (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.

- Python 3: Còn gọi là Python 3000 hoặc Py3K, dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python".

1.3. Các ứng dụng của Python

Python là một ngôn ngữ lập trình đơn giản nhưng hiệu quả, nó gồm các tính năng chính như sau:



- So với Unix Shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.
- So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn, nó có sẵn nhiều kiểu dữ liệu hơn.
- Python thích hợp với các chương trình lớn hơn.
- Python được sử dụng để lập trình Web, nó có thể được sử dụng như một ngôn ngữ kịch bản.
- Python được thiết kế để có thể nhúng và phục vụ như một ngôn ngữ kịch bản để tùy biến và mở rộng các ứng dụng lớn hơn.
- Python được tích hợp sẵn nhiều công cụ và có thư viện chuẩn phong phú.
- Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ đánh dấu khác, và nó cũng cung cấp các thư viện xử lý các giao thức Internet thông dụng.
- Python có khả năng giao tiếp với hầu hết các loại cơ sở dữ liệu, có khả năng xử lý văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.
- Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, Visualization Toolkits, Numeric Python, ...
- Python có thể ứng dụng để phát triển các ứng dụng desktop, người lập trình có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa.
- Python còn được hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, FLTK, Delphi,..., và nó cũng có sẵn một unit testing framework để tạo ra các bộ test.

2. CÁC THÀNH PHẦN CƠ BẢN TRONG PYTHON

2.1. Bảng ký tự của Python

Python cho phép NLT sử dụng các ký tự để tạo nên những câu lệnh trong chương trình. Cụ thể như sau:

- Các chữ cái la tinh (gồm chữ thường và chữ hoa): a..z và A..Z. Và Python là ngôn ngữ lập trình phân biệt chữ hoa và chữ thường. Ví dụ chữ cái ‘a’ là khác với ‘A’.
- Dấu gạch dưới: _
- Các chữ số thập phân: 0..9
- Các ký hiệu toán học: +, -, *, /, %, &, ||, !, >, <, = ...
- Các ký hiệu đặc biệt khác: :, :, [], {}, #, dấu cách, ...

2.2. Từ khóa

Từ khóa là từ dành riêng và đã được định nghĩa trước trong NNLT với một ý nghĩa cố định, thường được chỉ các loại dữ liệu hoặc kết hợp thành câu lệnh. Người lập trình có thể tạo ra những tên mới để chỉ các đối tượng của mình nhưng không được phép trùng với từ khóa.

Đây là một số từ khóa thường gặp trong ngôn ngữ lập trình Python mà ý nghĩa của chúng sẽ được trình bày trong các mục liên quan: if, else, elif, return, while, for, def, import, class, ...

2.3. Câu lệnh và khối lệnh

Trong Python, một câu lệnh được bắt đầu bởi ký tự dòng, nghĩa là một câu lệnh sẽ kết thúc khi nhận được ký tự xuống dòng.

Ví dụ 1: Đoạn chương trình sau đây sẽ được hiểu gồm 4 câu lệnh:

```
x = 5
y = 7
sum = x + y
print(sum)
```

Tuy nhiên chúng ta có thể mở rộng câu lệnh trên nhiều dòng với ký tự tiếp tục dòng lệnh là dấu xỏ phải (\).

Ví dụ 2:

```
sum = 1 + 3 + 5 + \
      7 + 9
print(sum)
```

Đây là trường hợp viết tiếp câu lệnh rất minh bạch. Và Python còn cách viết tiếp câu lệnh trên nhiều dòng bằng cách sử dụng 1 trong 3 cặp dấu ngoặc đơn (), ngoặc vuông [] hoặc ngoặc nhọn {}.

Ví dụ 3: Ví dụ 2 trên có thể được viết lại như sau:

```
sum = (1 + 3 + 5 +  
       7 + 9)  
print(sum)
```

Khác với các ngôn ngữ lập trình khác, Python không sử dụng các cặp từ khóa như: begin và end hay cặp dấu { và } để bao nhiêu câu lệnh thành một khối lệnh mà thay vào đó Python quy ước các lệnh liên tiếp có cùng khoảng cách thụt đầu hàng là thuộc cùng một khối lệnh.

Ví dụ 4:

```
x, y = 0, 0  
if (x==y) :  
    x = 5  
    y = 7  
    sum = x + y  
    print(sum)  
else:  
    print("x và y khác nhau")
```

Nhờ thụt đầu hàng mà code trong Python trông rõ ràng và dễ đọc hơn. Ta có thể bỏ qua việc thụt đầu hàng với những câu lệnh viết trên nhiều dòng, tuy nhiên những nhà lập trình nhiều kinh nghiệm khuyên rằng: Hãy luôn thụt đầu hàng, điều đó làm cho code dễ đọc hơn.

2.4. Chú thích trong chương trình

Một chương trình thường được viết một cách ngắn gọn, do vậy thông thường người lập trình viết vào chương trình các câu chú thích (ghi chú) nhằm giải thích rõ nghĩa hơn các câu lệnh, đoạn chương trình hay chương trình. Một chú thích có thể ghi chú về nhiệm vụ, mục đích và cách thức của các thành phần được chú thích. Các chú thích còn là cách để người viết code giao tiếp với người đọc code, giúp người đọc code không mất quá nhiều thời gian để tìm hiểu và suy đoán.

Trong Python, chúng ta sử dụng ký tự # để bắt đầu một chú thích. Chú thích bắt đầu sau dấu # cho đến khi bắt đầu một dòng mới. Khi thông dịch, Python sẽ bỏ qua những chú thích này.

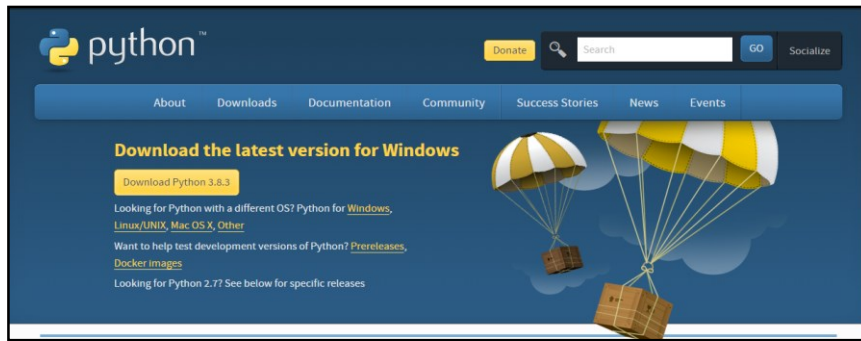
Chúng ta có thể sử dụng cặp 3 dấu nháy đơn ''' hoặc cặp 3 dấu nháy kép """ để chú thích trên nhiều dòng.

3. CÁC BƯỚC ĐỂ TẠO VÀ THỰC THI MỘT CHƯƠNG TRÌNH

3.1. Cài đặt Python

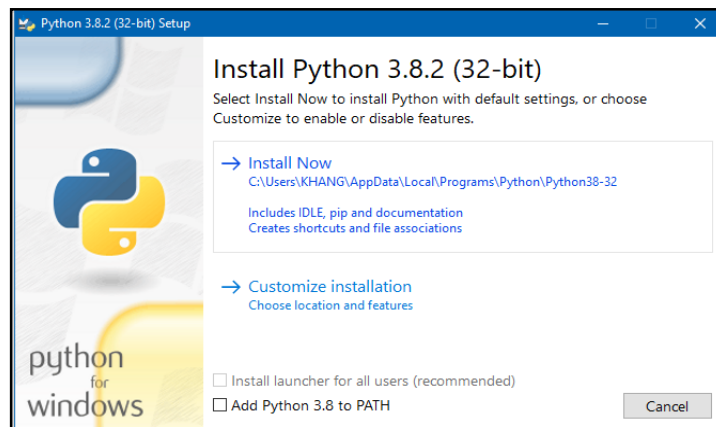
Để bắt đầu làm quen với Python, trước tiên bạn cần cài đặt Python trên máy tính đang dùng, có thể là Windows, macOS hoặc Linux. Python hỗ trợ hầu hết trên các hệ điều hành hiện nay, và cách cài đặt nó cũng khá đơn giản.

Đầu tiên chúng ta cần download bộ cài **Python** từ trang chủ của nó.

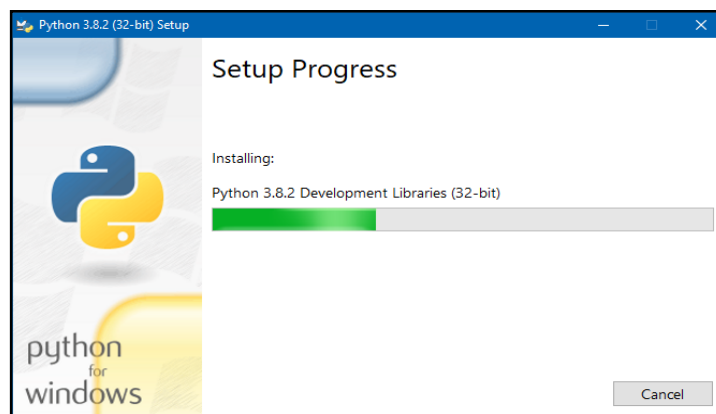


Tiếp tục ta chạy file vừa tải và cài đặt theo các bước sau:

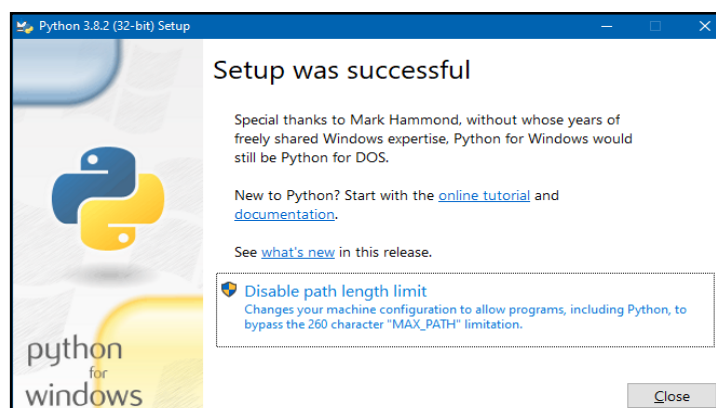
Nhấp đúp vào file vừa tải về để cài đặt.



Tiếp theo ta chọn **Install Now** và chờ đợi quá trình cài đặt hoàn tất.

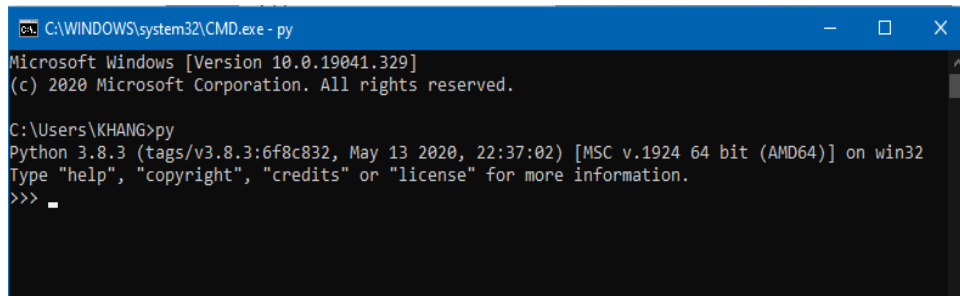


Khi cửa sổ hiển thị **Setup was successful** là ta đã cài đặt thành công môi trường Python và ta chỉ việc nhấp nút **Close**.



Sau khi cài đặt xong môi trường Python ta có thể mở **Command Prompt (CMD)** để kiểm tra.

Để kiểm tra môi trường Python đã được cài đặt thành công chưa ta mở cmd và nhập vào lệnh: `py` ↵



```
C:\WINDOWS\system32\CMD.exe - py
Microsoft Windows [Version 10.0.19041.329]
(c) 2020 Microsoft Corporation. All rights reserved.

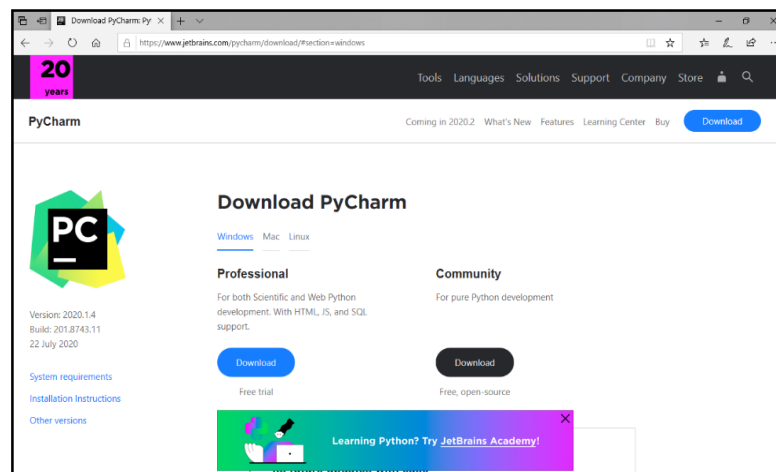
C:\Users\KHANG>py
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Nếu kết quả xuất hiện như hình trên thì chúng ta đã cài đặt Python thành công.

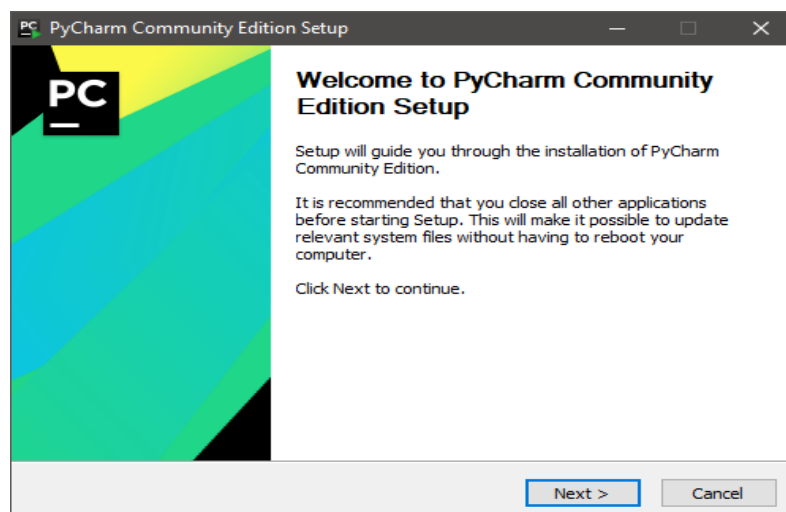
Chúng ta cần cài đặt thêm một phần mềm editor để hỗ trợ trong việc soạn thảo chương trình nguồn như: PyCharm, Sublime Text, Note Pad, ... Trong đó PyCharm thường được sử dụng nhiều hơn.

3.2. Cài đặt PyCharm

Bước 1: Truy cập đến trang chủ của JetBrains để tải PyCharm. PyCharm có 2 phiên bản là Professional và Community. Chọn phiên bản phù hợp và chọn **Download**.



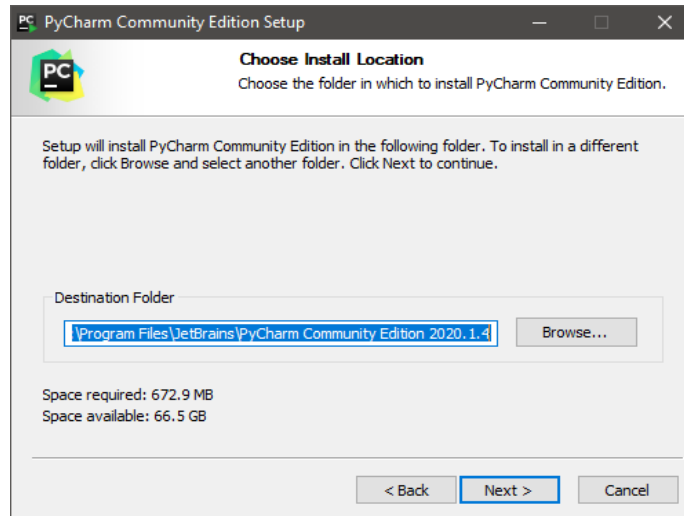
Bước 2: Nhấp đúp file vừa mới tải về để cài đặt.



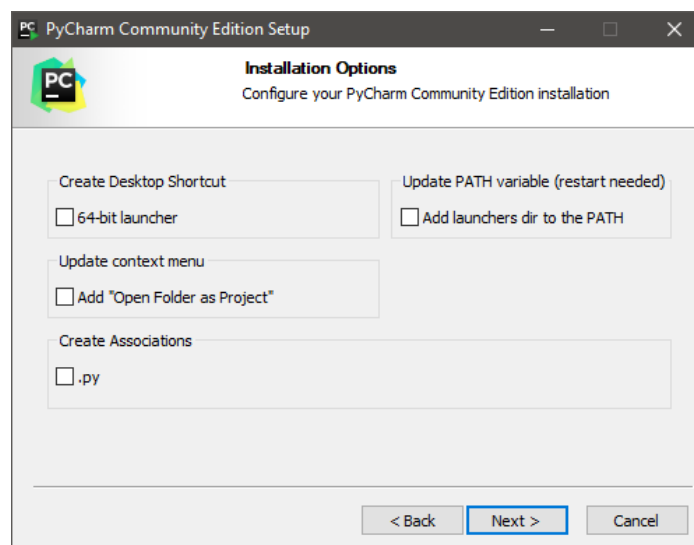
Bước 3: Chọn Next

Bước 4: Chọn đường dẫn để cài đặt PyCharm.

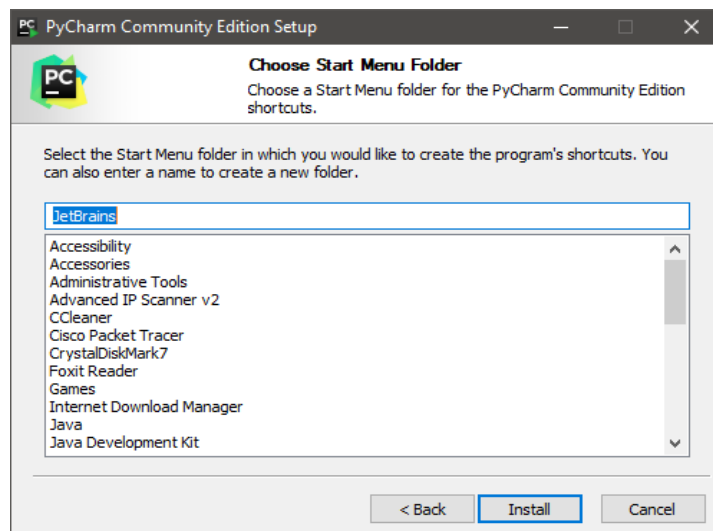
(Mặc định là C:\Program Files\JetBrains\PyCharm Community Edition 2020.1.4)

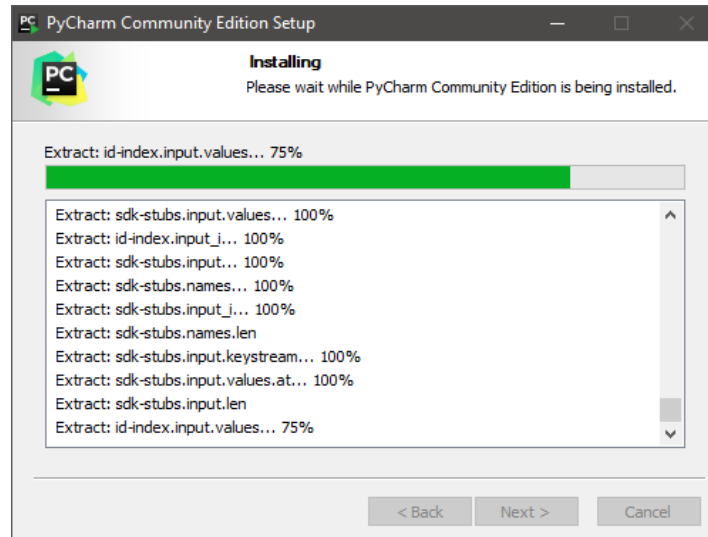
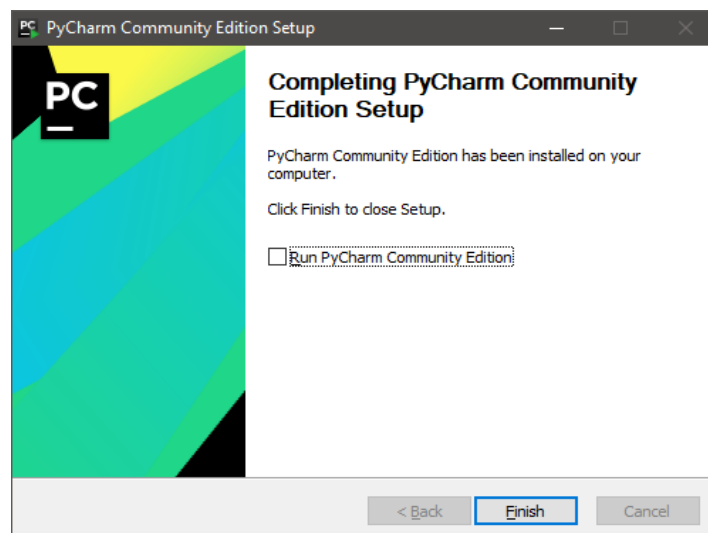


Bước 5: Chọn Next



Bước 6: Chọn các tùy chọn cài đặt theo nhu cầu sử dụng, sau đó chọn Next



Bước 7: Đặt mặc định và chọn Install**Bước 8: Chờ quá trình cài đặt hoàn tất và chọn Finish****3.3. Soạn thảo và thực thi chương trình**

Sau khi cài đặt thành công Python và PyCharm, chúng ta khởi động PyCharm, khai báo các thuộc tính tương ứng và tiến hành soạn thảo chương trình, cuối cùng ta lưu file chương trình có phần mở rộng là .py.

Ta có thể thực thi chương trình Python bằng các cách sau:

Cách 1:

- Mở cửa sổ Command Prompt
- Chuyển đến thư mục có chứa file nguồn vừa tạo ra
- Thực hiện câu lệnh: `python <file nguồn>` ↵

Cách 2:

- Từ cửa sổ soạn thảo file nguồn của PyCharm ta chọn menu Run -> Chọn Run <file nguồn> hoặc ấn tổ hợp phím Shift + F10 hoặc chọn biểu tượng Run trên thanh công cụ.

- Nếu chương trình không có lỗi thì kết quả sẽ được hiển thị trong vùng kết quả của Run, ngược lại sẽ thể hiện dòng lệnh có chứa lỗi trong file nguồn.

4. NHẬP, XUẤT TRONG PYTHON

Trong phần này chúng ta làm quen với một số lệnh cơ bản cho phép người lập trình nhập dữ liệu từ bàn phím hoặc xuất kết quả lên màn hình.

4.1. Nhập dữ liệu từ bàn phím

Trong ngôn ngữ lập trình Python hàm `input()` có vai trò giúp người dùng có thể nhập dữ liệu vào chương trình để thực hiện. Hàm `input()` trong Python sẽ tạm dừng chương trình lại cho đến khi phím enter được nhấn, và nó trả về 1 chuỗi ký tự do người dùng nhập vào từ bàn phím.

Cú pháp: `input([prompt])`

Trong đó `[prompt]` là thông báo sẽ hiển thị ra màn hình, gợi ý thông tin người dùng nhập vào chương trình.

Ví dụ 5:

```
n = input("Nhập số nguyên dương n = ")
```

Lưu ý: Hàm `input()` trả về một chuỗi ký tự, ngay cả khi người dùng nhập vào một số nó cũng chuyển thành chuỗi. Do đó, khi thực hiện tính toán ta cần phải chuyển đổi ngược lại từ chuỗi thành số bằng cách **tên_kiểu_số(chuỗi)**, chẳng hạn `int(n)`.

4.2. Xuất dữ liệu lên màn hình

Hàm `print` trong Python có chức năng hiển thị dữ liệu lên màn hình khi chương trình thực thi.

Cú pháp: `print(content)`

Trong đó:

content là nội dung cần hiển thị lên màn hình, content có thể là hằng, biến hay biểu thức, nếu là hằng ký tự thì đặt trong cặp nháy đơn và hằng chuỗi ký tự thì đặt trong cặp nháy kép.

Nếu muốn hiển thị nhiều nội dung khác nhau trong cùng một hàm `print` thì chúng ta chỉ cần ngăn cách giữa các nội dung hiển thị bằng dấu phẩy.

Ví dụ 6: `print("Giá trị của n là: ", n)`

Mặc định thì mỗi lần chúng ta xuất dữ liệu lên màn hình bởi hàm `print` thì nó sẽ tự động xuống dòng. Nếu chúng ta không muốn xuống dòng khi kết thúc hàm `print` thì ta thêm từ khóa **end** vào tham số cuối cùng của hàm `print` với cú pháp như sau:

print (content, end = "charset")

Trong đó `charset` là tập ký tự mà ta muốn thực hiện khi kết thúc hàm `print`.

Ví dụ 7:

```
print("Lập trình hdt bằng", end = " ")  
print("Python")
```

Kết quả được hiển thị là: Lập trình hết bằng Python

Nếu ta muốn giữa các dữ liệu cần xuất lên màn hình cách nhau bởi charset nào đó thì ta thêm từ khóa *sep* vào tham số cuối cùng của hàm print với cú pháp như sau:

print (content_1, content_2, ..., content_n, sep = “charset”)

Ví dụ 8:

```
print(1, 2, 'ABC', 8.5, sep=';')
```

Kết quả được hiển thị là: 1;2;ABC;8.5

5. CÁC KIỂU DỮ LIỆU CƠ BẢN

5.1. Kiểu int

Kiểu số nguyên (không có chứa dấu chấm thập phân), có thể lưu các số nguyên âm và dương.

Ví dụ 9: -12, 2020

5.2. Kiểu float

Kiểu số thực (có chứa dấu chấm thập phân).

Ví dụ 10: -5.2, 6.3

5.3. Kiểu complex

Kiểu số phức, một số phức có 2 phần là phần thực (real) và phần ảo (imag) và trong Python có 2 dạng để thể hiện một số phức là dùng từ khóa j hoặc hàm complex.

Ví dụ 11:

```
z = 2 + 3j
```

Ví dụ 12:

```
z = complex(2, 3)
```

Trong 2 ví dụ trên thì 2 là phần thực, 3 là phần ảo.

Ví dụ 13:

```
z = 2 + 3j
```

```
print("Phần thực của z = ", z.real)
```

```
print("Phần ảo của z = ", z.imag)
```

Trên màn hình kết quả chúng ta thấy:

```
Phần thực của z = 2.0
```

```
Phần ảo của z = 3.0
```

5.4. Kiểu str

Kiểu chuỗi, được đặt trong cặp dấu nháy đơn hoặc nháy kép.

Ví dụ 14:

```
st1 = "Ngôn ngữ lập trình"
```

```
st2 = 'Python'
```

```
print(st1 + ' ' + st2)
```

5.5. Kiểu bool

Kiểu luận lý hay logic, để lưu 1 trong 2 giá trị False hoặc True.

Ví dụ 15:

```
t1 = False
t2 = True
```

6. BIẾN

6.1. Định nghĩa

Biến được tạo ra với chức năng là nơi để lưu trữ giá trị dữ liệu mà các dữ liệu này có thể thay đổi trong quá trình thực thi chương trình. Biến trong Python không cần khai báo trước mà nó sẽ được tạo ra ngay khi chúng ta gán giá trị cho nó.

Ví dụ 16:

```
x = "Hello world"
y = 10
```

6.2. Khai báo

Không cần thiết để khai báo kiểu dữ liệu cho biến, vì Python sẽ gán kiểu dữ liệu cho biến ngay khi nó nhận được giá trị và chúng ta có thể thay đổi giá trị và kiểu dữ liệu của biến sau đó.

Ví dụ 17:

```
x = 7          # x bây giờ là kiểu int
x = 5.5        # x bây giờ là kiểu float
```

6.3. Gán giá trị cho nhiều biến

Để gán giá trị cho nhiều biến trên cùng một dòng theo cú pháp:

biến 1, biến 2, biến 3 = giá trị 1, giá trị 2, giá trị 3

Ví dụ 18:

```
x, y, z = 5.5, 10, "kiểu số"
print(x)
print(y)
print(z)
```

6.4. Các quy tắc đặt tên cho biến

- Tên biến phải bắt đầu bằng một chữ cái hoặc là ký tự gạch dưới;
- Tên biến không thể bắt đầu bằng một số;
- Tên biến chỉ chứa các ký tự chữ cái (A-Z, a-z), các chữ số (0-9) và ký tự _;
- Tên biến phân biệt chữ hoa và chữ thường.

6.5. Xóa biến

Trong Python có một điểm thú vị là nếu biến đó đang tồn tại mà ta xóa nó đi thì ta không còn sử dụng nó được nữa (tương tự trong C++ khi chúng ta thu hồi bộ nhớ của con trỏ vậy). Cú pháp xóa biến: **del <tên biến>**

Ví dụ 19:

```
x, y = 5.5, 10
print(x)
print(y)
x = y
del y
print(x)
print(y) #NameError: name 'y' is not defined
```

Lưu ý: Nếu chúng ta tạo biến bên ngoài các hàm như các ví dụ trên, thì các biến đó được gọi là biến toàn cục. Còn các biến được tạo bên trong hàm được gọi là biến cục bộ. Trong trường hợp chúng ta tạo 1 biến cục bộ mà có tên trùng với biến toàn cục thì khi đó biến này chỉ sử dụng nội bộ trong hàm và biến toàn cục (trùng tên) vẫn không thay đổi.

7. CÁC TOÁN TỬ TRONG PYTHON

7.1. Các toán tử số học

Gồm các phép toán: +, -, *, /, // (chia lấy phần nguyên), % (chia lấy phần dư), **

7.2. Các toán tử gán

Gồm các phép toán: =, +=, -=, *=, /=, //=, %=, **=

7.3. Các toán tử so sánh

Gồm các phép toán: ==, !=, <, <=, >, >=

7.4. Các toán tử logic

Gồm các phép toán: and, or, not

7.5. Các toán tử nhận dạng

Có 2 phép toán là is và is not: is (trả về True nếu các biến ở hai bên toán tử cùng trỏ tới 1 đối tượng hoặc có cùng giá trị, ngược lại trả về False), is not (ngược lại với toán tử is).

7.6. Các toán tử thành viên

Có 2 phép toán là in và not in: in (trả về True nếu biến thuộc 1 trong các phần tử của tập hợp, ngược lại trả về False), not in (ngược lại với toán tử in).

7.7. Độ ưu tiên của các toán tử

Python có ràng buộc thứ tự ưu tiên của các toán tử. Tuy nhiên cách tốt nhất là ta nên điều khiển nó bằng cách dùng cặp dấu ngoặc đơn () để nó rõ nghĩa hơn.

Nếu các toán tử không nằm trong cặp dấu ngoặc đơn thì nó sẽ được thực hiện theo thứ tự ưu tiên như bảng sau và nếu các toán tử cùng lớp ưu tiên thì chúng sẽ được thực hiện từ trái sang phải.

ĐỘ ƯU TIÊN	TOÁN TỬ	MIÊU TẢ
1	**	Toán tử mũ
2	*, /, //, %	Các toán tử nhân, chia, chia lấy phần nguyên và chia lấy phần dư
3	+, -	Các toán tử cộng, trừ
4	<, <=, >, >=, ==, !=	Các toán tử so sánh
5	=, +=, -=, *=, /=, //=, %=, **=	Các toán tử gán
6	is, is not	Các toán tử so sánh
7	not, or, and	Các toán tử logic

8. CÁC HÀM THƯỜNG DÙNG TRONG PYTHON

Ngôn ngữ lập trình Python có sẵn các hàm mà người lập trình có thể gọi thực hiện trong chương trình. Trong đó có một số hàm thông dụng như sau:

TT	HÀM	MÔ TẢ
1	abs()	Trả về giá trị tuyệt đối của một số
2	all()	Trả về True khi tất cả các phần tử trong tập hợp là đúng
3	any()	Trả về True khi có bất kỳ phần tử nào trong tập hợp là đúng
4	bin()	Chuyển đổi số nguyên hệ 10 sang hệ 2
5	bool()	Chuyển một giá trị sang bool
6	chr()	Trả về một ký tự từ một số nguyên
7	complex()	Tạo một số phức
8	delattr()	Xóa thuộc tính của đối tượng
9	dir()	Trả lại thuộc tính của đối tượng
10	float()	Trả về một số thực từ một số nguyên
11	getattr()	Trả về giá trị thuộc tính của đối tượng
12	hash()	Trả về giá trị hash của đối tượng là một số nguyên

TT	HÀM	MÔ TẢ
13	hex()	Chuyển số nguyên hệ 10 sang hệ 16
14	id()	Trả về định danh của đối tượng
15	int()	Trả về một số nguyên từ số thực
16	issubclass()	Kiểm tra xem đối tượng có là lớp con của một lớp nào đó không
17	max()	Trả về phần tử lớn nhất
18	min()	Trả về phần tử nhỏ nhất
19	next()	Trả về phần tử kế tiếp trong tập hợp
20	oct()	Chuyển số nguyên hệ 10 sang hệ 8
21	ord()	Trả về mã Unicode (số nguyên) cho ký tự Unicode tương ứng
22	pow()	Trả về x mũ y
23	property()	Trả về thuộc tính property của đối tượng
24	round()	Làm tròn số
25	sort()	Trả về list được sắp xếp
26	sum()	Tính tổng các phần tử trong tập hợp
27	super()	Cho phép tham chiếu đến lớp cha

BÀI TẬP CHƯƠNG 1



- Viết chương trình **nhập vào một ký tự**, **in ra số nguyên** là mã ASCII tương ứng của ký tự đó.
- Viết chương trình **nhập vào một số nguyên**, in ra ký tự tương ứng của nó trong bảng mã ASCII.
- Viết chương trình nhập vào hai số nguyên, in ra kết quả của các phép toán chia, chia lấy phần nguyên và chia lấy phần dư trên hai số đó.
- Viết chương trình nhập vào một **số thực x** và một **số nguyên n** , **in ra kết quả của x^n** .

5. Viết chương trình nhập vào chiều dài và chiều rộng của một hình chữ nhật, in ra chu vi và diện tích của hình chữ nhật đó.
6. Viết chương trình nhập vào **bán kính của hình tròn**, in ra diện tích, chu vi của nó.
7. Viết chương trình nhập vào bán kính mặt đáy và chiều cao một hình trụ, tính và in ra thể tích của hình trụ đó ($V = S_{\text{đáy}} * h$).
8. Viết chương trình nhập vào **1 số phức**, cho biết **mô đun** và **số phức liên hợp** của số phức đã nhập.
9. Viết chương trình tính giá trị của $a+aa+aaa+aaaa$ với a là số được nhập từ bàn phím. Giả sử a được nhập vào là 1 thì đầu ra sẽ là: 1234 ($1+11+111+1111 = 1234$).
10. Viết chương trình nhập vào 1 chuỗi gồm các số cách nhau bởi dấu phẩy. In ra các số lẻ trên cùng 1 dòng cách nhau 1 khoảng tab. Nếu input là: 1,2,3,4,5,6,7,8,9 thì output sẽ là: 1 3 5 7 9
11. Viết chương trình nhận chuỗi nhập vào bởi người dùng, in ra "Yes" nếu chuỗi là "yes" hoặc "YES" hoặc "Yes", ngược lại thì in "No".
12. Viết chương trình nhập vào hệ số lương (là số thực có 2 chữ số lẻ) của 1 nhân viên. In ra lương của nhân viên đó, với lương = 1.150.000 * hệ số lương.
13. Viết chương trình nhập vào một số nguyên (có 4 chữ số). In ra tổng của 4 chữ số này và chữ số đầu, chữ số cuối (ví dụ số 3185 có tổng các chữ số là 17, chữ số đầu và cuối là 3 và 5, **in ra** 17, 3, 5).
14. Viết chương trình nhập vào **năm sinh của 1 người**. Cho biết người đó năm nay được **bao nhiêu tuổi**.
15. Viết chương trình tạo ra các số thập phân ngẫu nhiên bằng cách sử dụng module random và các hàm random() và randrange():
 - Có giá trị nằm trong khoảng từ 0 đến 1.
 - Có giá trị nằm trong khoảng từ 10 đến 100.
 - Có giá trị nằm trong khoảng từ 7 đến 97.
 - Có giá trị nằm trong khoảng từ 9 đến 18.
16. Viết chương trình nhập vào cân nặng (ký lô gam) và chiều cao (mét) của 1 người trưởng thành, tính chỉ số BMI và dự báo độ béo phì cho người này theo công thức và bảng sau:

$$\text{BMI} = \text{Cân nặng}/(\text{Chiều cao})^2$$

Phân loại	BMI (kg/m ²) - WHO
Cân nặng thấp (gầy)	< 18,5
Bình thường	18,5 - 24,9
Thừa cân	≥ 25
Tiền béo phì	25 - 29,9
Béo phì độ I	30 - 34,9
Béo phì độ II	35 - 39,9
Béo phì độ III	≥ 40

Chương 2: CÁC CẤU TRÚC ĐIỀU KHIỂN



1. CẤU TRÚC RỄ NHÁNH

Trong thực tế, để giải quyết một vấn đề nào đó chúng ta thường có nhiều phương án lựa chọn khác nhau tương ứng với các điều kiện cụ thể khi thực hiện. Trong ngôn ngữ lập trình Python và các NNLT khác cũng thế, khái niệm này trong lập trình gọi là cấu trúc rẽ nhánh hay lựa chọn. Đặc biệt trong Python không có cấu trúc **case of** hay **switch case** như các NNLT khác.

1.1. Ý nghĩa của cấu trúc if

Một câu lệnh if cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào điều kiện được viết trong câu lệnh là đúng hay sai.

1.2. Cú Pháp

if điều kiện:

 Công việc A (nếu điều kiện đúng)

else:

 Công việc B (nếu điều kiện sai)

Ví dụ 1: Giải phương trình bậc nhất ($ax + b = 0$), chỉ xét hệ số a.

```
a = int(input("Nhập a = "))
b = int(input("Nhập b = "))
if a!=0:
    print("Phương trình có nghiệm x = ", -b/a)
else:
    print("Phương trình vô nghiệm")
```

1.3. Cấu trúc if lồng nhau

Đặc điểm chung của các câu lệnh có cấu trúc là bản thân của nó có thể chứa các câu lệnh khác. Điều này cho phép các lệnh if có thể lồng nhau. Nếu nhiều câu lệnh if (có else hoặc không có else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Quy tắc là else sẽ đi với if gần nó nhất mà chưa ghép cặp với else nào khác.

Dạng 1:

if điều kiện 1:

if điều kiện 2:

 Công việc A (nếu điều kiện 1 đúng và điều kiện 2 đúng)

else:

 Công việc B (nếu điều kiện 1 đúng và điều kiện 2 sai)

else:

 Công việc C (nếu điều kiện 1 sai)

Dạng 2:**if** điều kiện 1:

Công việc A (nếu điều kiện 1 đúng)

elif điều kiện 2:

Công việc B (nếu điều kiện 1 sai và điều kiện 2 đúng)

else:

Công việc C (nếu điều kiện 1 sai và điều kiện 2 sai)

Ví dụ 2: Giải phương trình bậc nhất ($ax + b = 0$), xét đủ mọi trường hợp.

```
a = int(input("Nhập a = "))
b = int(input("Nhập b = "))
if a!=0:
    if b==0: print("Phương trình có nghiệm x = 0")
    else: print("Phương trình có nghiệm x = ", -b/a)
elif b!=0: print("Phương trình vô nghiệm")
else: print("Phương trình có vô số nghiệm")
```

1.4. Các lưu ý

- Ta không cần dùng cặp dấu ngoặc đơn () để bao điều kiện sau if hoặc sau elif.
- Ta không dùng cặp dấu ngoặc nhọn {} để bao khối lệnh mà thay vào đó ta phải áp dụng qui tắc thụt đầu hàng cho chính xác.
- Nếu sau if hoặc elif hay else chỉ có 1 câu lệnh thì ta có thể viết ngay sau đó mà không cần phải xuống dòng.
- Đối với kiểu số: 0 đại diện cho False và các giá trị khác 0 đại diện cho True.
- Đối với kiểu chuỗi: chuỗi rỗng đại diện cho False và chuỗi khác rỗng đại diện cho True.

Ví dụ 3: Viết chương trình nhập vào 1 số nguyên, cho biết số đó là chẵn hay lẻ.

```
n = int(input("Nhập n = "))
if n%2==0:
    print(n, "là số chẵn")
else:
    print(n, "là số lẻ")
```

Ví dụ 4: Viết chương trình nhập vào điểm học phần của một môn học bất kỳ, cho biết điểm chữ tương ứng của môn học đó.

```
dhp = float(input("Nhập điểm học phần : "))
print("Điểm chữ tương ứng là",end = " : ")
if dhp>=8.5:
    print("A")
```

```
elif dhp>=7.8:
    print("B+")
elif dhp>=7:
    print("B")
elif dhp>=6.3:
    print("C+")
elif dhp>=5.5:
    print("C")
elif dhp>=4.8:
    print("D+")
elif dhp>=4:
    print("D")
else:
    print("F")
```

2. CẤU TRÚC LẶP

Cấu trúc lặp là một cấu trúc không thể thiếu trong bất kỳ NNLT nào, nó là cấu trúc cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình. Trong Python có 2 cấu trúc lặp là **for in** và **while**. Đặc biệt trong Python không có cấu trúc **do while** hay **repeat until** như các NNLT khác.

2.1. Cấu trúc for in

2.1.1. cú pháp

for <biến lặp> **in** <tập hợp>:

Công việc lặp

2.1.2. Cách thực hiện

- Phần tử đầu tiên của *tập hợp* sẽ được gán cho *biến lặp* và công việc lặp bên trong for sẽ được thực hiện.
- Cứ tiếp tục như vậy lần lượt từng phần tử trong *tập hợp* sẽ được gán cho *biến lặp* và thực hiện công việc lặp sau nó.
- Quá trình được lặp lại cho đến khi thực hiện xong công việc lặp cuối cùng tương ứng với phần tử cuối trong *tập hợp*.

2.1.3. Các lưu ý

- Khác với những ngôn ngữ khác, thay vì lặp qua một dãy số như trong Pascal, hoặc cho phép người dùng có thể tự định nghĩa bước lặp và điều kiện dừng như trong C/C++, Python cho phép duyệt qua từng phần tử trong tập hợp.
- Nếu số lần lặp là cố định (số phần tử trong tập hợp không thay đổi) thì vòng lặp for in được khuyến khích sử dụng.

- Và cũng tương tự như các NNLT khác, Python vẫn cho phép cấu trúc lặp for in lồng nhau.
- Ta có thể sử dụng một số hàm có sẵn để xác định các phần tử trong tập hợp được lặp qua nó.

✓ Hàm **len()** để xác định tổng số phần tử trong tập hợp.

Chẳng hạn:

```
a = [1, 3, 5, 7]
print(len(a)) => 4
st = ['Khoa', 'Công', 'Nghệ', 'Thông', 'Tin']
print(len(st)) => 5
```

✓ Hàm **range()** để tạo một dãy các số bắt đầu từ **0** theo mặc định, tăng thêm **1** (theo mặc định) và kết thúc tại một số được chỉ định.

Chẳng hạn:

```
x = range(10)
print(list(x)) => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Cú pháp đầy đủ của hàm **range()**: **range([start,] stop[, step])**

Chẳng hạn:

```
x = range(1, 10, 2)
print(list(x)) => [1, 3, 5, 7, 9]
x = range(5, 10)
print(list(x)) => [5, 6, 7, 8, 9]
x = range(10, -10, -3)
print(list(x)) => [10, 7, 4, 1, -2, -5, -8]
```

Ví dụ 5: Viết chương trình tính tổng n số nguyên dương đầu tiên với n được nhập từ bàn phím.

```
n = int(input("Nhập số nguyên dương n = "))
S = 0
for i in range(1, n+1):
    S = S + i
print("Tổng của", n, "số nguyên dương là S =", S)
```

Ví dụ 6: Viết chương trình tìm tất cả các số nguyên dương nằm trong đoạn từ 10 đến 100 thỏa điều kiện là bội số của 7 nhưng không là bội số của 5.

```
for i in range(10, 101):
    if (i % 7 == 0 and i % 5 != 0):
        print(i, end = " ")
```

Ví dụ 7: Viết chương trình in ra tất cả các phần tử có trong tập hợp st.

```
st = ['Khoa', 'Công', 'Nghệ', 'Thông', 'Tin']
for i in st:
    print(i)
```

Ví dụ 8: Viết chương trình in ra các phần tử nằm ở các vị trí lẻ trong tập hợp st.

```
st = ['Khoa', 'Công', 'Nghệ', 'Thông', 'Tin']
for i in range(1, len(st), 2):
    print(st[i])
```

Ví dụ 9: Viết chương trình tìm tất cả các phương án đổi N đồng thành các tờ tiền có mệnh giá 2 đồng, 3 đồng và 5 đồng với N là một số nguyên dương ($2 \leq N \leq 20$) được nhập từ bàn phím.

```
N = int(input("Nhập số tiền cần đổi: "))
print("Các phương án đổi tiền (x1,x2,x3) là:")
for i in range(N//2+1):
    for j in range(N//3+1):
        for k in range(N//5+1):
            if (i*2+j*3+k*5==N):
                print('(', i, ', ', j, ', ', k, ')')
```

2.2. Cấu trúc while

2.2.1. Cú pháp

while <điều kiện>:

 Công việc lặp

2.2.2. Cách thực hiện

Trong cấu trúc lặp while, *điều kiện* sẽ được kiểm tra đầu tiên và công việc lặp chỉ được thực hiện nếu *điều kiện* là đúng. Sau một lần lặp, *điều kiện* sẽ được kiểm tra lại và quá trình này sẽ tiếp tục cho đến khi *điều kiện* là sai.

2.2.3. Các lưu ý

- Công việc lặp trong while có thể không được thực hiện nếu *điều kiện* sai ngay từ đầu.
- Giống như if và for, khối lệnh của while cũng được xác định thông qua việc thụt đầu hàng của các câu lệnh.

Ví dụ 10: Viết chương trình tính tổng n số nguyên dương đầu tiên với n nhập từ bàn phím.

```
n = int(input("Nhập số nguyên dương n = "))
S = 0
i = 1
```

```
while i <= n:
    S = S + i
    i = i + 1
print("Tổng của", n, "số nguyên dương là S =", S)
```

Ví dụ 11: Viết chương trình tính tổng các số nguyên lẻ đầu tiên cho đến khi tổng này vừa lớn hơn giá trị M, với M được nhập từ bàn phím.

```
M = int(input("Nhập giá trị M = "))
S, i = 0, 1
while S <= M:
    S += i
    i += 2
print("Tổng vừa lớn hơn", M, "là:", S)
```

2.3. Lệnh break, continue và else trong cấu trúc lặp

Lệnh break và continue trong Python cũng tương tự như trong C và C++. Lệnh break dùng để thoát khỏi cấu trúc lặp nhỏ nhất có chứa nó. Lệnh continue dùng để bỏ qua lần lặp hiện tại và nhảy đến lần lặp kế tiếp.

Điều đặc biệt là trong Python các cấu trúc lặp có thể có về else, khối lệnh sau else sẽ được thực thi khi vòng lặp kết thúc, tuy nhiên nó sẽ không được thực thi khi vòng lặp kết thúc bởi câu lệnh break.

Ví dụ 12: Viết chương trình cho phép nhập lần lượt các số nguyên (dương và âm), việc nhập kết thúc khi nhập số nguyên là 0, tính và in ra kết quả tổng các số nguyên dương đã nhập.

```
S, n = 0, 1
while n!=0:
    n = int(input("Nhập số nguyên n = "))
    if n < 0:
        continue
    S = S + n
print("Tổng các số nguyên dương đã nhập là:", S)
```

Ví dụ 13: Viết chương trình xác định các số nguyên dương $< N$ là số nguyên tố hay hợp số, với N được nhập từ bàn phím.

```
N = int(input("Nhập số nguyên dương N = "))
for i in range(2, N):
    for j in range(2, int(i**0.5)+1):
        if i%j==0:
            print(i, "là hợp số")
            break
```

```
else:
```

```
    print(i, "là số nguyên tố")
```

Tuy nhiên ví dụ trên ta có thể sử dụng hàm `sqrt` để tính căn bậc hai của một số bằng cách import thư viện `math`. Chương trình trên được viết lại như sau.

```
import math
```

```
N = int(input("Nhập số nguyên dương N = "))
```

```
for i in range(2, N):
```

```
    for j in range(2, int(math.sqrt(i))+1):
```

```
        if i%j==0:
```

```
            print(i, "là hợp số")
```

```
            break
```

```
    else:
```

```
        print(i, "là số nguyên tố")
```

3. LỖI, NGOẠI LỆ VÀ XỬ LÝ NGOẠI LỆ

Python thường sinh ra các ngoại lệ (*exception*) khi có lỗi (*error*) xảy ra trong quá trình thực thi chương trình. Như vậy để chương trình không bị xảy ra các lỗi cũng như các ngoại lệ không mong muốn thì NLT phải hiểu rõ và biết cách xử lý các lỗi cũng như các ngoại lệ đó.

3.1. Lỗi

Khi lập trình, việc gặp lỗi (*error*) là điều không thể tránh khỏi. Có một số lỗi do NLT không tuân theo cấu trúc đúng của cú pháp gây ra, các lỗi này được gọi là lỗi cú pháp (*syntax error* hoặc *parsing error*).

Ví dụ 14:

```
n = int(input('Nhập n = '))
```

```
if n%2 == 0
```

```
    print(n, 'là số chẵn')
```

Đoạn chương trình trên sẽ gây ra lỗi cú pháp:

```
File "D:/DriveD/Vidu/Chuong2/Error.py", line 2
```

```
    if n%2 == 0
```

```
        ^
```

```
SyntaxError: invalid syntax
```

3.2. Ngoại lệ

Ngoài các lỗi cú pháp, có một số trường hợp Python phát sinh lỗi khi đang thực thi chương trình (*runtime error*). Những lỗi như thế được gọi là ngoại lệ (*exception*). Ngoại lệ được Python tạo ra để xử lý vấn đề đó nhằm tránh cho chương trình bị hỏng.

Các ngoại lệ thường xảy ra trong chương trình là:

- Chia một số cho 0 (*ZeroDivisionError*).
- Truyền tham số cho hàm, đúng kiểu dữ liệu nhưng giá trị không thích hợp (*ValueError*).
- Không tìm thấy module được import (*ImportError*).
- Mở một tập tin không tồn tại (*FileNotFoundError*).

Bất cứ khi nào có *runtime error* xảy ra, Python sẽ tạo một đối tượng ngoại lệ. Nếu không được xử lý đúng cách thì chương trình sẽ in ra lỗi và chi tiết về lý do lỗi đó xảy ra.

Ví dụ 15:

```
n = int(input('Nhập n = '))
x = n/0
print(x)
```

Đoạn chương trình trên sẽ gây ra *runtime error*:

Traceback (most recent call last):

File "D:/DriveD/Vidu/Chuong2/Exception.py", line 2, in <module>

x = n/0

ZeroDivisionError: division by zero

Có rất nhiều ngoại lệ được Python tạo ra khi gặp các lỗi tương ứng. Sau đây là một số ngoại lệ được xây dựng sẵn trong Python cùng với lỗi gây ra tương ứng.

STT	NGOẠI LỆ	LÝ DO GÂY RA
1	AssertionError	Xảy ra khi câu lệnh assert thất bại
2	AttributeError	Xảy ra khi gán thuộc tính hoặc tham chiếu thất bại
3	EOFError	Xảy ra khi hàm input() chạm vào điều kiện end-of-file
4	FloatingPointError	Xảy ra khi một số thực dấy chấm động thực thi không thành công
5	GeneratorExit	Xảy ra khi phương thức close() của hàm generator được gọi
6	ImportError	Xảy ra khi không tìm thấy module được import
7	IndexError	Xảy ra khi một chỉ số trong chuỗi nằm ngoài phạm vi
8	KeyError	Xảy ra khi không tìm thấy khóa ánh xạ (dictionary) trong tập hợp các khóa hiện có
9	KeyboardInterrupt	Xảy ra khi người dùng nhấn phím ngắt (thông thường là Ctrl-C hoặc Delete)
10	NameError	Xảy ra khi không tìm thấy tên của biến

STT	NGOẠI LỆ	LÝ DO GÂY RA
11	OSError	Xảy ra khi một hàm trả về lỗi liên quan đến hệ thống
12	OverflowError	Xảy ra khi kết quả của phép toán số học quá lớn không thể biểu diễn
13	RuntimeError	Xảy ra khi phát hiện thấy lỗi không thuộc bất kỳ danh mục nào khác
14	StopIteration	Xảy ra bởi phương thức next() của một vòng lặp để báo hiệu rằng không có giá trị nào được trả về bởi iterator
15	SyntaxError	Xảy ra khi gặp lỗi cú pháp
16	IndentationError	Xảy ra khi có lỗi thụt đầu hàng không chính xác
17	TabError	Xảy ra khi thụt đầu hàng sử dụng các tab và dấu cách không nhất quán
18	SystemError	Xảy ra khi trình thông dịch tìm thấy các lỗi nội bộ nhưng tình hình không quá nghiêm trọng
19	SystemExit	Xảy ra bởi hàm sys.exit()
20	TypeError	Xảy ra khi một hàm hoặc phép toán áp dụng kiểu không chính xác cho một đối tượng
21	UnicodeError	Xảy ra khi có lỗi liên quan đến Unicode
22	UnicodeEncodeError	Xảy ra khi lỗi liên quan đến Unicode diễn ra trong quá trình mã hóa
23	UnicodeDecodeError	Xảy ra khi lỗi liên quan đến Unicode diễn ra trong quá trình giải mã
24	UnicodeTranslateError	Xảy ra khi lỗi liên quan đến Unicode trong quá trình dịch
25	ValueError	Xảy ra khi một phép toán hoặc một hàm nhận được một đối số có kiểu đúng nhưng giá trị không phù hợp
26	ZeroDivisionError	Xảy ra khi đối số thứ hai của phép chia là 0

Để xử lý những ngoại lệ này, NLT có thể sử dụng các câu lệnh try, except và finally. Bên cạnh đó NLT có thể tự định nghĩa ngoại lệ của mình trong Python nếu thật sự cần thiết.

3.3. Xử lý ngoại lệ

Khi ngoại lệ xảy ra, Python sẽ tạo ra một exception để xử lý vấn đề đó nhằm tránh cho ứng dụng hay server bị lỗi. Ngoại lệ có thể là bất kỳ điều kiện bất thường làm phá vỡ luồng thực thi chương trình đó. Bất cứ khi nào một ngoại lệ xuất hiện, chương trình sẽ ngừng thực thi, chuyển qua quá trình gọi và in ra lỗi đến khi nó được xử lý.

Trong Python, các ngoại lệ có thể được xử lý bằng khối lệnh *try...except*. Phần thân của *try* sẽ gồm các câu lệnh của chương trình mà chúng có thể gây ra exception. Nếu một exception xuất hiện thì tất cả các câu lệnh trong *try* sẽ bị bỏ qua. Phần thân của *except* được gọi bởi exception handler, vì nó được dùng để bắt lỗi trong chương trình. Khối *except* sẽ thực hiện khi lỗi xuất hiện, không thì sẽ được bỏ qua.

NLT có thể sử dụng các exception có sẵn trong thư viện chuẩn của Python, và để gọi ra các ngoại lệ có sẵn thì NLT phải import module *sys* vào chương trình.

Ví dụ 16:

```
import sys
ts = int(input('Nhập tử số: '))
ms = int(input('Nhập mẫu số: '))
try:
    kq = ts/ms
    print('Kết quả của phép chia là:', kq)
except:
    print('Có ngoại lệ', sys.exc_info(), 'xảy ra.')
    print("Mẫu số bằng 0")
```

Nếu chương trình trên được thực thi với *ts* = 5 và *ms* = 2 thì màn hình kết quả sẽ là:

Kết quả của phép chia là: 2.5

Nếu chương trình trên được thực thi với *ts* = 5 và *ms* = 0 thì màn hình kết quả sẽ là:

Có ngoại lệ (<class 'ZeroDivisionError'>, ZeroDivisionError ('division by zero'), <traceback object at 0x0000005290BB2200>) xảy ra.

Mẫu số bằng 0

Lưu ý: Hàm *sys.exc_info()* trả về 3 thông tin của ngoại lệ tương ứng là kiểu (type), giá trị (value) và dấu vết đã gây ra ngoại lệ (traceback). Nếu NLT chỉ muốn biết 1 hoặc 2 trong 3 thông tin trên thì phải chỉ ra vị trí tương ứng của thông tin đó, chẳng hạn như: *sys.exc_info()[0]*, *sys.exc_info()[1]*, *sys.exc_info()[2]*.

Ở ví dụ 16, không có một ngoại lệ cụ thể nào được đề cập đến trong mệnh đề *except* nên khi chương trình gặp ngoại lệ (dù là bất kỳ exception nào) thì chúng đều được xử lý chỉ theo một cách như vậy. Ngoài ra, NLT có thể chỉ định các ngoại lệ cụ thể cho khối lệnh *except* bằng cách chỉ ra ngoại lệ cụ thể sau từ khóa *except*.

Dạng 1:

```
try:
    #Khối lệnh thực hiện try
except <tên ngoại lệ 1>:
    #Khối lệnh thực hiện ngoại lệ 1
except <tên ngoại lệ 2>:
    #Khối lệnh thực hiện ngoại lệ 2
...
```

Dạng 2:*try:**#Khởi lệnh thuộc try**except(<tên ngoại lệ 1, tên ngoại lệ 2, ..., tên ngoại lệ n>):**#Khởi lệnh thực hiện các ngoại lệ 1, 2, ..., n*

Tên ngoại lệ là các exception mà NLT nghĩ có khả năng xảy ra trong chương trình. Một mệnh đề *try* có thể có nhiều mệnh đề *except* để xử lý chúng khác nhau. Nếu khối lệnh trong *try* có lỗi xảy ra thì chương trình sẽ tìm đến các *except* phía dưới, *except* nào thỏa thì nó sẽ thực thi khối lệnh trong *except* đó.

Ví dụ 17:*try:*

```
ts = int(input('Nhập tử số: '))
ms = int(input('Nhập mẫu số: '))
kq = ts/ms
print('Kết quả của phép chia là:', kq)
```

except ValueError:

```
print('Giá trị không hợp lệ.')
```

except ZeroDivisionError:

```
print('Mẫu số không thể là 0.')
```

Chương trình trong ví dụ 17 có thể được viết lại theo dạng 2 như sau:

try:

```
ts = int(input('Nhập tử số: '))
ms = int(input('Nhập mẫu số: '))
kq = ts/ms
print('Kết quả của phép chia là:', kq)
```

except (ValueError, ZeroDivisionError):

```
print('Giá trị không hợp lệ.')
```

```
print('Mẫu số không thể là 0.')
```

Tuy nhiên với cách viết này thì việc thông báo lỗi sẽ không rõ ràng bằng cách viết thứ nhất. Chính vì vậy cú pháp dạng 1 thường được sử dụng nhiều hơn trên thực tế.

Mặc dù Python đã cung cấp sẵn rất nhiều các ngoại lệ, tuy nhiên trong lập trình đôi lúc NLT cần phải bắt các lỗi mà chưa được hỗ trợ. Trong trường hợp này NLT tự tạo một exception cho riêng mình bằng cách sử dụng lệnh *raise*.

Ví dụ 18: Khi nhập dữ liệu là tháng của một năm nào đó thì ta cần kiểm tra tính hợp lệ của giá trị tháng này. Đoạn chương trình có thể được viết như sau:

```
try:
    month = int(input('Nhập vào số tháng trong năm: '))
    if month < 1 or month > 12:
        raise Exception
    else:
        print('Số tháng nhập vào hợp lệ')
except:
    print('Số tháng nhập vào phải thuộc [1 .. 12]')
```

Chương trình trong ví dụ 18 có thể được viết lại như sau:

```
try:
    month = int(input('Nhập vào số tháng trong năm: '))
    if month < 1 or month > 12:
        raise Exception('Số tháng nhập vào phải thuộc [1 .. 12]')
    print('Số tháng nhập vào hợp lệ')
except Exception as message:
    print(message)
```

Đôi khi NLT chỉ sử dụng mệnh đề *try* và *except* là chưa đủ bởi vì khi chương trình có xảy ra ngoại lệ thì các câu lệnh sau đó trong khối lệnh của *try* sẽ không được thực hiện nữa. Trong trường hợp này NLT cần sử dụng thêm mệnh đề *finally* để thực hiện các công việc cần thiết đó. → Kết thúc

Finally còn được gọi là mệnh đề *clean-up* hay *termination* vì nó luôn luôn được thực hiện bất kể có lỗi nào xảy ra trong khối lệnh của *try* hay không. → Đơn đẹp

Thông thường thì các câu lệnh trong *finally* được dùng để thực hiện các công việc đóng tập tin, giải phóng tài nguyên, ... Chẳng hạn khi làm việc với tập tin, đầu tiên là thao tác mở tập tin để làm việc, và sau khi thực hiện xong các thao tác trên tập tin thì NLT cần phải đóng tập tin đó lại để đảm bảo cho việc dữ liệu của tập tin được an toàn. Như vậy, thao tác đóng tập tin được đặt trong mệnh đề *finally* là hợp lý nhất vì khối lệnh của *finally* luôn được thực hiện cho dù chương trình có xảy ra lỗi hay không.

Ví dụ 19:

```
try:
    f = open('Quehuong2.txt', 'r', encoding='utf-8')
    nd = f.read()
    print(nd)
except UnicodeError:
    print('Có lỗi liên quan về mã Unicode')
finally:
    print('Đóng tập tin và giải phóng vùng nhớ')
    f.close()
```

Thực thi chương trình trên kết quả sẽ là:

*Quê hương là chùm khế ngọt
Cho con trèo hái mỗi ngày.*

Đóng tập tin và giải phóng vùng nhớ.

Kết quả cho thấy chương trình không xảy ra lỗi và ngoại lệ nào cả vì thể nội dung tập tin Quehuong2.txt sẽ được đọc và in lên màn hình và cuối cùng là đóng tập tin này và giải phóng vùng nhớ.

Tuy nhiên nếu ta tạo thêm tập tin có tên Quehuong3.txt và lưu với dạng mã hóa là ANSI chứ không phải là utf-8 thì chương trình sẽ gây ra ngoại lệ. Và thực thi chương trình trên kết quả sẽ là:

Có lỗi liên quan về mã Unicode

Đóng tập tin và giải phóng vùng nhớ.

Như vậy, bằng cách thêm mệnh đề *finally* ta có thể yên tâm tập tin sẽ được đóng đúng ngay cả khi phát sinh ngoại lệ khiến chương trình phải dừng đột ngột.

BÀI TẬP CHƯƠNG 2

-----❧-----

1. Viết chương trình nhập vào 2 số phức, cho biết kết quả của 4 phép toán cộng, trừ, nhân, chia trên 2 số phức đó.
2. Viết chương trình nhập vào **số tín chỉ đã tích lũy** của một sinh viên, cho biết sinh viên đó đang là **sinh viên năm mấy**?
3. Viết chương trình tìm số nhỏ nhất trong 4 số thực được nhập từ bàn phím.
4. Viết chương trình nhập vào **điểm trung bình chung của 1 sinh viên** theo thang điểm 4, hãy cho biết sinh đó được **xếp loại** gì?
5. Viết chương trình tính **giai thừa của một số nguyên** dương được nhập vào từ bàn phím.
6. Viết chương trình tính x^n với x là số thực và n là số nguyên dương được nhập vào từ bàn phím.
7. Viết chương trình **đếm số ước số của một số nguyên dương** được nhập vào từ bàn phím.
8. Viết chương trình tính tổng các bội số (từ nhỏ đến lớn) của một số nguyên dương được nhập vào từ bàn phím, công việc kết thúc khi tổng vừa lớn hơn một số nguyên dương cho trước.
9. Viết chương trình tính và in giá trị của Q cho bởi công thức: $Q = \sqrt{(2 * C * D)/H}$. Với giá trị cố định của C là 50, H là 30, D là dãy các giá trị được nhập từ bàn phím với các giá trị của D được **phân cách bằng dấu phẩy**. Giả sử dãy giá trị của D nhập vào là 100,150,180 thì các giá trị của Q sẽ là 18,22,24.

10. Viết chương trình tìm tất cả các số chia hết cho 7 nhưng không phải là bội số của 5, nằm trong đoạn 2000 và 3200. Các số thu được chỉ in trên 1 dòng và được cách nhau bằng dấu phẩy.

11. Viết chương trình **nhập vào lần lượt các số thực**, nếu số nhập vào > 0 thì tính tích, nếu số nhập vào < 0 thì bỏ qua và cho nhập tiếp, nếu số nhập vào $= 0$ thì kết thúc. ✓

12. Viết chương trình tìm tất cả các số trong đoạn 1000 và 3000 sao cho tất cả các chữ số trong số đó là số chẵn. In các số tìm được trên 1 dòng và được cách nhau bằng dấu hai chấm.

13. Viết chương trình tính số tiền thực của một tài khoản ngân hàng dựa trên nhật ký giao dịch được nhập vào từ bàn phím. Khi nhập nhật ký giao dịch phải thỏa 2 điều kiện là: số tiền gửi hoặc rút phải là số dương, số tiền rút không thể lớn hơn số tiền gửi hiện tại. Định dạng nhật ký được hiển thị như sau:

D 200

W 100

Với D là tiền gửi vào, W là tiền rút ra, theo sau là số tiền tương ứng.

14. Viết chương trình để giải 1 câu đố cổ của Trung Quốc: Một trang trại thỏ và gà có 35 đầu, 94 chân, hỏi số thỏ và gà là bao nhiêu?

15. Viết một chương trình tính $1/2 + 2/3 + 3/4 + \dots + n/(n+1)$ với n là số nguyên dương được nhập vào từ bàn phím.

16. Viết chương trình tính tổng $S = 1 + 2 - 3 + 4 + 5 - 6 + 7 + 8 - 9 + \dots \pm n$, với n là số nguyên dương được nhập từ bàn phím.

17. Viết chương trình tính $e = 1 + \frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \dots + (-1)^{n+1} \frac{x^n}{n!}$, với n là số nguyên dương, x là số thực được nhập từ bàn phím.

18. Viết chương trình tính $e = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$, cho đến khi $1/(n+1)!$ nhỏ hơn Epsilon, với Epsilon là 1 lượng khá nhỏ được nhập từ bàn phím.

19. Viết chương trình nhập vào lần lượt các số thực, quá trình nhập kết thúc khi nhập số thực là 0. Cho biết trị trung bình của các số đã nhập.

20. Viết chương trình in ra tất cả các số nguyên tố $\leq N$, N là trị nhập.

21. Viết chương trình phân tích 1 số nguyên dương n thành tích của các thừa số nguyên tố.

22. Viết chương trình tính số hạng thứ n của dãy Fibonacci. Dãy Fibonacci là dãy số gồm các số hạng $F(n)$ với $F(n) = F(n-1) + F(n-2)$ và $F(1) = F(2) = 1$

Ví dụ: Dãy Fibonacci gồm 10 số hạng đầu tiên là: 1 1 2 3 5 8 13 21 34

23. Viết chương trình nhập vào 1 chuỗi là 1 biểu thức toán học, nếu biểu thức hợp lệ thì định trị và in kết quả lên màn hình.

Ví dụ: Input: (5+2)*3-8/(6-4)

Output: 17

Hướng dẫn: Sử dụng try ... except để kiểm tra và hàm eval() để định trị biểu thức.

Chương 3: DỮ LIỆU KIỂU CHUỖI



1. CHUỖI TRONG PYTHON

1.1. Chuỗi ký tự

Chuỗi ký tự (string) là một trong các kiểu dữ liệu thông dụng nhất trong Python. Chuỗi ký tự trong Python được bao trong cặp dấu nháy đơn hoặc cặp dấu nháy kép. Chính vì thế chuỗi 'Hello' tương đương với chuỗi "Hello". Tùy theo ý thích của NLT mà sử dụng 1 trong 2 cặp dấu này đều được. Tên kiểu dữ liệu là **str**.

1.2. Định dạng chuỗi

Với Python có rất nhiều cách định dạng chuỗi. Và trong giáo trình này chúng ta sẽ tìm hiểu một số cách định dạng cơ bản và thường được sử dụng nhiều trong việc lập trình.

1.2.1. Gán chuỗi cho biến

Việc gán một chuỗi cho biến được thực hiện với cú pháp: **<tên biến> = <chuỗi>**

Ví dụ 1:

```
st1 = "Ngôn ngữ lập trình"
st2 = 'Python'
print(st1)
print(st2)
```

1.2.2. Chuỗi đa dòng

Ta có thể gán một chuỗi có nhiều dòng cho một biến bằng cách sử dụng cặp 3 dấu nháy đơn hoặc 3 dấu nháy kép.

Ví dụ 2:

```
st1 = """Quê hương là chùm khế ngọt
Cho con trèo hái mỗi ngày"""
st2 = '''Quê hương là đường đi học
Con về rợp bướm vàng bay.'''
print(st1)
print(st2)
```

1.2.3. Chuỗi là một mảng

Một chuỗi trong Python có thể được xem là một mảng các ký tự là các byte đại diện cho các ký tự Unicode.

Tuy nhiên, Python không có kiểu ký tự mà thay vào đó một ký tự đơn giản là một chuỗi có độ dài bằng 1.

Dấu ngoặc vuông [] được sử dụng để truy cập các ký tự trong chuỗi. Ký tự đầu tiên có chỉ số (index) là 0.

Ví dụ 3:

```
st = "Ngôn ngữ lập trình"  
print(st[0]) # => N  
print(st[9]) # => l
```

1.2.4. Định dạng chuỗi bằng f (f-string)

Phương pháp định dạng này cho ta khả năng thay thế một số vị trí ở trong chuỗi bằng giá trị của các biến mà ta đã khởi tạo. Để có thể sử dụng cách này, ta phải có một chuỗi f, và theo cú pháp: **f "giá trị trong chuỗi"**

Ví dụ 4:

```
a = "chuỗi"  
print(f"Đây là một {a}.")
```

Kết quả in lên màn hình là: Đây là một chuỗi.

Giá trị của biểu thức a được thay thế trong dấu ngoặc nhọn chứa tên của nó.

1.2.5. Định dạng chuỗi bằng hàm format

Cách định dạng này cho phép Python định dạng chuỗi một cách tốt hơn, không chỉ tốt về mặt nội dung mà còn về mặt thẩm mỹ.

Ta sử dụng hàm format theo cú pháp: **format(đối số 1, đối số 2, ... đối số n)**

Ví dụ 5:

```
n = 3.8  
st = "Ngôn ngữ lập trình {} phiên bản {}"  
print(st.format("Python", n))
```

Kết quả in lên màn hình là: Ngôn ngữ lập trình Python phiên bản 3.8

1.2.6. Các ký tự đặc biệt trong chuỗi

Để in được các ký tự đặc biệt thì ta sử dụng ký tự \ trước các ký tự đặc biệt đó.

Chẳng hạn:

Muốn in ra câu *I'm a student.*

Ta phải dùng hàm print là `print("I\'m a student.")`

Các ký tự đặc biệt khác:

- \n ngắt xuống dòng và bắt đầu dòng mới.
- \t đẩy nội dung phía sau nó cách 1 tab.
- \a chuông cảnh báo.
- \b xóa bỏ khoảng trắng phía trước nó.

2. THAO TÁC TRÊN CHUỖI

2.1. Lưu trữ và truy xuất chuỗi

Chuỗi trong Python được lưu trữ trong vùng nhớ với mỗi ô nhớ tương ứng với một ký tự và các ký tự này được bố trí liên tiếp nhau. Do đó ta có thể truy xuất đến từng ký tự trong một chuỗi ký tự thông qua chỉ mục/vị trí của nó với qui định ký tự đầu tiên có chỉ mục/vị trí là 0.

Ví dụ 6: Viết chương trình nhập vào 1 chuỗi bất kỳ, in ra từng ký tự trong chuỗi đó.

```
st = input("Nhập chuỗi bất kỳ: ")
for i in range(0, len(st)):
    print(st[i])
```

2.2. Các toán tử trên chuỗi

TOÁN TỬ	MÔ TẢ	VÍ DỤ
+	Nối/Ghép (concatenate) 2 chuỗi, tạo thành một chuỗi mới. Tuy nhiên 2 chuỗi đứng liền nhau sẽ tự động được nối lại.	"Hello" + "Python" => "Hello Python" st = "Hello" "Python" => st là "HelloPython"
*	Tạo một chuỗi mới bằng cách nối/ghép (concatenate) nhiều lần bản sao của cùng một chuỗi.	"Hello" * 3 => "HelloHelloHello"
[index]	Trả về ký tự tại chỉ mục/vị trí cho bởi index.	st = "Hello" st[0] => "H"; st[1] => "e"; st[len(st)-1] => "o"
[start:end]	Trả về một chuỗi con chứa các ký tự từ start (mặc định là đầu chuỗi) tới end (mặc định là cuối chuỗi), nếu giá trị dương thì tính từ trái còn nếu giá trị âm thì tính từ phải.	st = "Hello" st[1:4] => "ell"; st[:3] => "Hel" st[2:] => "llo"; st[:] => "Hello" st[-2:] => "lo"; st[1:-2] => "el"
in	Trả về True nếu ký tự tồn tại trong chuỗi và ngược lại.	st = "Hello" 'e' in st => True 'h' in st => False
not in	Trả về True nếu ký tự không tồn tại trong chuỗi và ngược lại.	st = "Hello" 'M' not in st => True 'l' not in st => False

TOÁN TỬ	MÔ TẢ	VÍ DỤ
r/R	Chuỗi thô (Raw String) - Ngăn chặn ý nghĩa thực tế của các ký tự đặc biệt như \n, \t, ... Cú pháp cho chuỗi thô giống hệt với chuỗi thông thường ngoại trừ toán tử chuỗi thô là ký tự r/R được đặt trước dấu " hoặc ' của chuỗi.	print("Công nghệ \n thông tin") print(R"Công nghệ \n thông tin") => Công nghệ \n thông tin print('1 \t 2 \t 3') print(r'1 \t 2 \t 3') => 1 \t 2 \t 3
%	%d - Chèn số nguyên đại diện cho đối tượng. %f - Chèn số thực đại diện cho đối tượng. %s - Chèn chuỗi đại diện cho đối tượng. %r - Chèn kiểu mẫu đại diện cho đối tượng, tùy đối tượng là chuỗi hay số.	print('Số này là %d' %9) print('Số này là %f' %9.5) print('Hello %s' %'Python') print('Hello %r' %'Python') print('Hello %r' %9.5)

2.3. Một số hàm xử lý chuỗi

Python có rất nhiều hàm dựng sẵn để xử lý chuỗi, tuy nhiên trong bài giảng này chúng ta chỉ nghiên cứu một số hàm cơ bản thường được sử dụng.

2.3.1. Hàm len()

Cú pháp: len(*st*)

Chức năng: Trả về độ dài (số ký tự) của chuỗi *st*.

Ví dụ 7:

```
st = "Hello Python"
print(len(st)) # => 12
```

2.3.2. Hàm strip()

Cú pháp: *st*.strip()

Chức năng: Loại bỏ tất cả các khoảng trắng ở đầu và cuối chuỗi *st*.

Ngoài ra còn có các hàm:

- *st*.lstrip(): chỉ loại bỏ các khoảng trắng ở đầu chuỗi *st*.
- *st*.rstrip(): chỉ loại bỏ các khoảng trắng ở cuối chuỗi *st*.

Ví dụ 8:

```
st = "  Hello Python!  "
print(st,": chiều dài là",len(st)) # => 18
stdc = st.strip()
print(stdc,": chiều dài là",len(stdc)) # => 13
std = st.lstrip()
print(std,": chiều dài là",len(std)) # => 16
```

```
stc = st.rstrip()
print(stc, ": chiều dài là", len(stc)) # => 15
```

2.3.3. Hàm *replace()*

Cú pháp: *st.replace(stF, stR)*

Chức năng: Tìm chuỗi *stF* trong chuỗi *st* và thay thế bằng chuỗi *stR*.

Ví dụ 9:

```
st = "Hello Python!"
print(st)
sttt = st.replace("Hello", "Hi")
print(sttt)
```

2.3.4. Hàm *find()*

Cú pháp: *st.find(stF, [,start [, end]])*

Chức năng: Tìm chuỗi *stF* trong *st* tính từ vị trí *start* (mặc định là 0) đến vị trí *end* (mặc định là hết chuỗi *st*), nếu tìm thấy thì trả về vị trí tìm thấy chuỗi *stF* đầu tiên, ngược lại (không tìm thấy *stF*) hàm trả về giá trị là -1.

Ngoài ra còn có hàm: *rfind* để tìm từ phải sang trái.

Ví dụ 10:

```
st = "Hello Python!"
vt = st.find('i')
print(vt) # => -1
vt = st.find('o')
print(vt) # => 4
vt = st.find('o', 6)
print(vt) # => 10
vt = st.find('o', 6, 9)
print(vt) # => -1
vt = st.rfind('o')
print(vt) # => 10
vt = st.rfind('o', 2, 9)
print(vt) # => 4
```

2.3.5. Hàm *split()*

Cú pháp: *st.split(chuỗi đại diện)*

Chức năng: Tách chuỗi *st* thành nhiều chuỗi con và đưa chúng vào danh sách được xác định bởi *chuỗi đại diện*, mỗi chuỗi con trở thành 1 phần tử trong danh sách đó.

Ngoài ra còn có hàm: *splitlines()* để tách chuỗi theo từng dòng.

Ví dụ 11:

```
st = "Ngôn ngữ lập trình Python"
arr = st.split(" ")
print(arr)
```

Danh sách arr là: ['Ngôn', 'ngữ', 'lập', 'trình', 'Python']

Ví dụ 12:

```
st = """Xin chào!
Đây là ngôn ngữ lập trình Python
phiên bản 3.8"""
arr = st.splitlines()
print(arr)
```

Danh sách arr là:

```
['Xin chào!', 'Đây là ngôn ngữ lập trình Python', 'phiên bản 3.8']
```

2.3.6. Hàm lower()

Cú pháp: `st.lower()`

Chức năng: Trả về chuỗi chữ thường từ chuỗi st.

Ví dụ 13:

```
st = "Ngôn ngữ lập trình Python"
print(st.lower()) # => ngôn ngữ lập trình python
print(st) # => Ngôn ngữ lập trình Python
st = st.lower()
print(st) # => ngôn ngữ lập trình python
```

2.3.7. Hàm upper()

Cú pháp: `st.upper()`

Chức năng: Trả về chuỗi chữ hoa (chữ in) từ chuỗi st.

Ví dụ 14:

```
st = "Ngôn ngữ lập trình Python"
print(st.upper()) # => NGÔN NGỮ LẬP TRÌNH PYTHON
print(st) # => Ngôn ngữ lập trình Python
st = st.upper()
print(st) # => NGÔN NGỮ LẬP TRÌNH PYTHON
```

2.3.8. Hàm isnumeric()

Cú pháp: `st.isnumeric()`

Chức năng: Trả về giá trị True nếu chuỗi st là chuỗi số nguyên và ngược lại.

Ví dụ 15:

```
st = "7661"
print(st.isnumeric()) # => True
st = "76.61"
print(st.isnumeric()) # => False
st = "64Y1-7661"
print(st.isnumeric()) # => False
```

2.3.9. Hàm isalpha()

Cú pháp: `kt.isalpha()`

Chức năng: Trả về giá trị True nếu ký tự `kt` là chữ cái và ngược lại.

Ví dụ 16:

```
st = "Tin học 123"
print(st[0].isalpha()) # => True
print(st[9].isalpha()) # => False
```

2.3.10. Hàm isdigit()

Cú pháp: `kt.isdigit()`

Chức năng: Trả về giá trị True nếu ký tự `kt` là chữ số và ngược lại.

Ví dụ 17:

```
st = "Tin học 123"
print(st[2].isdigit()) # => False
print(st[8].isdigit()) # => True
```

2.4. Một số lưu ý

- Không như các ngôn ngữ lập trình khác, chuỗi trong Python không thể bị thay đổi trực tiếp (immutable). Chẳng hạn như phép gán 1 ký tự vào 1 vị trí/chỉ mục trong chuỗi sẽ gây ra lỗi `TypeError: 'str' object does not support item assignment`.

Ví dụ 18:

```
st = "Tin học 123"
st[8]='A'

# => Câu lệnh st[8]='A' bị báo lỗi sau:
TypeError: 'str' object does not support item assignment
```

Để thực hiện được công việc trên ta phải thực hiện như sau:

```
st = "Tin học 123"
st = st[:8]+'A'+st[9:]
```

- `st[:i] + st[i:]` bằng chính `st`.

Ví dụ 19:

```
st = "Tin học 123"
stcpy = st[:5]+st[5:]
print(stcpy) # => Tin học 123
```

- Một chỉ mục quá lớn sẽ được thay bằng kích thước chuỗi, một giới hạn trên nhỏ hơn giới hạn dưới sẽ trả về chuỗi rỗng.

Ví dụ 20:

```
st = "Tin học 123"
stcpy = st[8:50]
print(stcpy) # => 123
stcpy = st[8:5]
print(stcpy) # => ''
```

- Các vị trí/chỉ mục có thể là số âm, để bắt đầu đếm từ bên phải. Trong trường hợp này ký tự cuối cùng trong chuỗi có vị trí/chỉ mục là -1.

Ví dụ 21:

```
st = "Tin học 123"
print(st[-1]) # => 3
print(st[4:-2]) # => học 1
print(st[-3:]) # => 123
print(st[:-3]) # => Tin học
```

3. CÁC CHƯƠNG TRÌNH ỨNG DỤNG**3.1. Chương trình 1**

Viết chương trình nhập vào một chuỗi. Đếm số ký tự chữ và ký tự số trong chuỗi đó.

```
st=input("Nhập vào chuỗi bất kỳ:")
dc=0
ds=0
for kt in st:
    if kt.isalpha():
        dc=dc+1
    elif kt.isdigit():
        ds=ds+1
print("Số chữ cái là:", dc)
print("Số chữ số là:", ds)
```

Tuy nhiên chương trình trên ta có thể viết theo cách tiếp cận tương tự như các NNLT C/C++ hay Pascal như sau:

```
st=input("Nhập vào chuỗi bất kỳ:")
dc=0
ds=0
for i in range(0,len(st)):
    if st[i].isalpha():
        dc=dc+1
    elif st[i].isdigit():
        ds=ds+1
print("Số chữ cái là:", dc)
print("Số chữ số là:", ds)
```

3.2. Chương trình 2

Viết chương trình nhập vào một chuỗi bất kỳ, đổi thành chuỗi dạng UPPER, thay thế các khoảng trắng trong chuỗi thành các dấu '-', in ra từng từ trong chuỗi trên các dòng khác nhau và cho biết trong chuỗi có tổng cộng bao nhiêu từ.

```
st=input("Nhập vào chuỗi bất kỳ:")
print("Chuỗi vừa nhập là:",st)
st=st.upper()
print("Chuỗi sau khi đổi là:",st)
st=st.replace(' ','-')
print("Chuỗi sau khi thay thế là:",st)
a=st.split('-')
dem=0
print("Các từ có trong chuỗi là:")
for i in a:
    if(i!=''):
        dem +=1
        print(i)
print("Trong chuỗi có",dem,"từ")
```

3.3. Chương trình 3

Viết chương trình nhập vào họ tên của một người; cắt bỏ các khoảng trắng thừa có trong họ tên; cho biết trong họ tên có xuất hiện ký tự kt không, với kt được nhập từ bàn phím; in ra họ và tên của người này.

```
ht=input("Nhập vào họ tên:")
print("Họ tên vừa nhập là:",ht)
ht=ht.strip()
```



```

i=1
while (i<len(ht)-1):
    if (ht[i]==' ' and ht[i+1]!=' '):
        ht=ht[:i]+ht[i+1:]
    else:
        i+=1
print("Họ tên sau khi bỏ khoảng trắng thừa là:",ht)
kt=input("Nhập ký tự cần kiểm tra:")
if (ht.find(kt)!=-1):
    print("Có ký tự",kt,"trong họ tên")
else:
    print("Không có ký tự", kt, "trong họ tên")
ktd=ht.find(' ')
ho=ht[:ktd]
ktc=ht.rfind(' ')
ten=ht[ktc+1:]
print("Người này có họ là",ho,"và có tên là",ten)

```

BÀI TẬP CHƯƠNG 3

-----❖-----

- Viết chương trình đếm số ký tự đặc biệt (không phải là chữ cái và chữ số) và số từ có trong một chuỗi được nhập vào từ bàn phím.
- Viết chương trình xóa tất cả các ký tự không phải là ký số ra khỏi chuỗi.
- Viết chương trình nhập vào một chuỗi bất kỳ, cho biết số lần xuất hiện của tất cả các ký tự có trong chuỗi đó.

Ví dụ: Input: "Apple"

Output: 'A': 1, 'p': 2, 'l': 1, 'e': 1

- Viết một chương trình nhập vào 1 chuỗi gồm nhiều từ được cách nhau bằng dấu phẩy, hãy in những từ đó thành chuỗi theo thứ tự bảng chữ cái và cách nhau bằng dấu bằng dấu hai chấm.

Ví dụ: Input: without,hello,bag,world

Output: bag:hello:without:world

- Viết chương trình nhập vào một chuỗi bất kỳ, cho biết chuỗi này có đối xứng không?

Ví dụ: Input: "CNTT202TTNC"

Output: Đối xứng

6. Viết chương trình nhập vào 1 chuỗi, in chuỗi theo thứ tự ngược lại, in các ký tự có chỉ số chẵn.

7. Viết chương trình nhập vào một chuỗi với sự kết hợp của chữ thường và chữ hoa, sắp xếp các ký tự theo cách mà tất cả các chữ thường đứng trước, chữ hoa đứng sau.

Ví dụ: Input: “PyNaTive”

Output: yaivePNT

8. Viết chương trình nhập vào 1 chuỗi, cho biết chuỗi này có phải là sự lặp lại của một chuỗi con nào đó hay không, nếu phải thì in ra chuỗi con đó?

Ví dụ: Input: “CNTTCNTTCNTT”

Output: Chuỗi con là CNTT

9. Viết chương trình nhập vào 1 chuỗi gồm nhiều từ được cách nhau bằng khoảng trống, hãy in ra các chuỗi số có trong chuỗi nhập.

Ví dụ: Input: “Ngôn ngữ lập trình Python 3 tháng 8 năm 2021”

Output: ‘3’, ‘8’, ‘2021’

10. Giả sử có một website yêu cầu người dùng nhập username và password để đăng ký. Viết chương trình để kiểm tra tính hợp lệ của password mà người dùng nhập vào.

Các tiêu chí kiểm tra password hợp lệ gồm:

- Ít nhất 1 chữ cái nằm trong [a-z]
- Ít nhất 1 số nằm trong [0-9]
- Ít nhất 1 ký tự nằm trong [A-Z]
- Ít nhất 1 ký tự nằm trong [\$ # @]
- Độ dài mật khẩu tối thiểu: 6
- Độ dài mật khẩu tối đa: 12

11. Giả sử chúng ta có một vài địa chỉ email dạng username@companyname.com, hãy viết chương trình để in username và companyname của một địa chỉ email cụ thể được nhập vào từ bàn phím (chỉ bao gồm chữ cái).

Ví dụ: Input: BinhPT@fit.com

Output: username là BinhPT và companyname là fit

Hướng dẫn: Sử dụng \w để kiểm tra chữ cái.

12. Tương tự như bài 11 nhưng địa chỉ email là bất kỳ.

13. Viết chương trình nhập vào 1 chuỗi, hãy in chuỗi nhập theo mã Unicode, sau đó chuyển chuỗi nhập sang một chuỗi Unicode được mã hóa bằng UTF-8.

Hướng dẫn: Sử dụng định dạng 'u:string' để định nghĩa chuỗi Unicode hoặc sử dụng hàm **format('Unicode')** và sử dụng hàm **encode()** hoặc **encode('utf-8')**.

14. Viết chương trình nhập vào 1 chuỗi, hãy nén và sau đó giải nén chuỗi nhập.

Hướng dẫn: Sử dụng hàm **compress()** và **decompress()** trong module zlib để nén và giải nén chuỗi.

15. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược theo từng từ.

Ví dụ: Input: ‘Cấm Không Được Câu Cá’

Output: ‘Cá Câu Được Không Cấm’

16. Viết chương trình nhập vào một số nguyên dương nhỏ hơn 1000, đọc số đó ra chữ.

Ví dụ: Input: 245

Output: ‘Hai trăm bốn mươi lăm’

17. Viết chương trình nhập vào một chuỗi ký số (các ký tự từ 1 đến 9), hãy cho biết trong chuỗi có bao nhiêu dãy con tăng dần, in ra dãy con tăng dần dài nhất có trong chuỗi này.

18. Viết chương trình nhập vào một chuỗi bất kỳ, cho biết từ nào xuất hiện nhiều nhất.

19. Viết chương trình nhập vào một chuỗi bất kỳ, cho biết trong chuỗi có bao nhiêu ký tự không phải là chữ cái. Đổi tất cả các chữ cái thành dạng chuẩn Upper (toàn bộ là ký tự hoa). Cho biết trong chuỗi các chữ số nào xuất hiện nhiều nhất và chúng xuất hiện bao nhiêu lần.

20. Viết chương trình nhập vào họ tên của một người, kiểm tra tính hợp lệ của họ tên đã nhập (chỉ gồm chữ cái và khoảng trắng nhưng không toàn là khoảng trắng), cắt bỏ các khoảng trắng thừa trong họ tên, chuyển họ tên về dạng Proper, cho biết trong họ tên ký tự nào xuất hiện nhiều nhất, in ra tên của người này.

Chương 4: DỮ LIỆU KIỂU TẬP HỢP



Trong Python có nhiều kiểu dữ liệu được sử dụng gần giống nhau được gọi chung là kiểu tập hợp. Các kiểu dữ liệu đó là str, list, tuple, set, ... Tuy nhiên ở mỗi kiểu này sẽ có một số đặc trưng riêng.

1. DỮ LIỆU KIỂU LIST

1.1. Định nghĩa

List trong Python là cấu trúc dữ liệu mà có khả năng lưu trữ nhiều kiểu dữ liệu khác nhau trong cùng một danh sách.

List có tính chất thay đổi được (mutable), nghĩa là nó sẽ không tạo ra một List mới nếu ta thay đổi các phần tử trong nó.

1.2. Khai báo và khởi tạo

Một List trong Python được khai báo và khởi tạo theo cú pháp:

<Tên_List> = [Giá trị_1, Giá trị_2, Giá trị_3, ..., Giá trị_n]

Ví dụ 1:

```
ds_sn = [1, 2, 3, 4, 5]
tt_sv = ['19004001', "Phan Thanh Bình", 2001, 8.5, True]
```

1.3. Các tính chất của List

List là một tập hợp hay là một container được sử dụng rất nhiều trong Python. List có các tính chất sau:

- Các phần tử của List được chứa trong cặp dấu ngoặc [].
- Các phần tử được cách nhau bởi dấu phẩy (,).
- Vị trí/chỉ mục của các phần tử trong List được tính từ 0.
- List có thể chứa mọi giá trị, ngay cả chính nó.
- List tương tự như mảng 1 chiều trong các NNLT khác.

Ví dụ 2:

```
ds_sn = [1, 2, 3, 4, 5]
print(ds_sn[2])
tt_sv = ['19004001', "Phan Thanh Bình", 2001, 8.5, True]
print("Sinh viên", tt_sv[1], "có ĐTB là", tt_sv[3])
ds_sn_sv = [ds_sn, tt_sv]
print(ds_sn_sv[0])
print(ds_sn_sv[1])
print(ds_sn_sv[0][2]) # => 3
print(ds_sn_sv[1][0]) # => 19004001
```

1.4. Truy cập đến các phần tử

Các phần tử trong List được truy cập dựa vào vị trí hay chỉ mục của nó.

Ví dụ 3:

```
ds_sn = [1, 2, 3, 4, 5]
print(ds_sn[0]) # => 1
print(ds_sn[2]) # => 3
print(ds_sn[len(ds_sn)-1]) # => 5
for i in range(0, len(ds_sn)):
    print(ds_sn[i], end='\t') # => 1    2    3    4    5
```

1.5. Các toán tử trên List

Vì List và chuỗi có cách truy xuất đến các phần tử tương tự nhau là dựa vào vị trí hay chỉ mục, vì thế trên List cũng có các toán tử tương tự như trên chuỗi. Các toán tử thường dùng đó là:

TOÁN TỬ	MÔ TẢ
+	Nối/Ghép (concatenate) 2 List lại và trả về List mới.
*	Tạo một List mới bằng cách nối/ghép (concatenate) nhiều lần bản sao của cùng một List.
[index]	Trả về phần tử tại chỉ mục/vị trí cho bởi index.
[start:end]	Trả về một List con chứa các phần tử từ start tới end, nếu giá trị dương thì tính từ trái còn nếu giá trị âm thì tính từ phải.
in	Trả về True nếu phần tử tồn tại trong List và ngược lại.
not in	Trả về True nếu phần tử không tồn tại trong List và ngược lại.
del	Xóa một phần tử trong List được chỉ ra bởi vị trí.

Ví dụ 4:

```
sn_le = [1, 3, 5, 7, 9]
sn_chan = [2, 4, 6, 8]
sn = sn_le + sn_chan
print(sn) # => [1, 3, 5, 7, 9, 2, 4, 6, 8]
chanx2 = sn_chan * 2
print(chanx2) # => [2, 4, 6, 8, 2, 4, 6, 8]
print(sn_le[3]) # => 7
print(sn_le[1:4]) # => [3, 5, 7]
print(sn_chan[:-2]) # => [2, 4]
```

```
print(sn_chan[-3:3]) # => [4, 6]
print(6 in chanx2) # => True
print(7 in chanx2) # => False
del chanx2[1]
print(chanx2) # => [2, 6, 8, 2, 4, 6, 8]
```

1.6. Một số hàm thường dùng trên List

Trong Python có rất nhiều hàm được xây dựng sẵn để giúp NLT giải quyết các vấn đề liên quan đến List, tuy nhiên trong bài giảng này chúng ta chỉ nghiên cứu một số hàm thường được sử dụng.

1.6.1. Hàm len()

Cú pháp: len(List)

Chức năng: Trả về độ dài (số phần tử) của List.

Ví dụ 5:

```
sn = [5, 2, 7, 6, 4]
print(len(sn)) # => 5
```

1.6.2. Hàm max()

Cú pháp: max(List)

Chức năng: Trả về giá trị lớn nhất của các phần tử có trong List.

Ví dụ 6:

```
sn = [5, 2, 7, 6, 4]
print(max(sn)) # => 7
```

1.6.3. Hàm min()

Cú pháp: min(List)

Chức năng: Trả về giá trị nhỏ nhất của các phần tử có trong List.

Ví dụ 7:

```
sn = [5, 2, 7, 6, 4]
print(min(sn)) # => 2
```

1.6.4. Hàm list()

Cú pháp: list(Seq)

Chức năng: Chuyển một Seq thành một List, trong đó Seq có thể là str, list, tuple, ...

Ví dụ 8:

```
st = "ABCDEF"
kt = list(st)
print(kt) # => ['A', 'B', 'C', 'D', 'E', 'F']
```

1.6.5. Hàm count()

Cú pháp: `List.count(key)`

Chức năng: Đếm số phần tử có giá trị là key có trong *List*.

Ví dụ 9:

```
sn = [5, 6, 7, 6, 4]
print(sn.count(4)) # => 1
print(sn.count(6)) # => 2
print(sn.count(8)) # => 0
```

1.6.7. Hàm index()

Cú pháp: `List.index(key [, start[, end]])`

Chức năng: Trả về vị trí/chỉ mục của phần tử có giá trị là key xuất hiện đầu tiên trong *List*, nếu không có phần tử thì hàm sẽ báo lỗi.

Ví dụ 10:

```
sn = [5, 6, 7, 6, 4]
print(sn.index(6)) # => 1
print(sn.index(8)) # => Báo lỗi ValueError: 8 is not in list
print(sn.index(6, 2, 5)) # => 3
```

1.6.8. Hàm append()

Cú pháp: `List.append(key)`

Chức năng: Thêm 1 phần tử có giá trị là key vào cuối *List*.

Ví dụ 11:

```
sn = [5, 2, 7, 6, 4]
sn.append(9)
print(sn) # => [5, 2, 7, 6, 4, 9]
```

1.6.9. Hàm insert()

Cú pháp: `List.insert(index, key)`

Chức năng: Thêm 1 phần tử có giá trị là key vào tại vị trí index trong *List*.

Ví dụ 12:

```
sn = [5, 2, 7, 6, 4]
sn.insert(2, 9)
print(sn)
sn.insert(0, 3)
print(sn)
sn.insert(len(sn), 8)
print(sn)
```

```
sn.insert(-5, 75)
print(sn)
sn.insert(10, 75)
print(sn)
# => Kết quả lần lượt là:
[5, 2, 9, 7, 6, 4]
[3, 5, 2, 9, 7, 6, 4]
[3, 5, 2, 9, 7, 6, 4, 8]
[3, 5, 2, 75, 9, 7, 6, 4, 8]
[3, 5, 2, 75, 9, 7, 6, 4, 8, 75]
```

1.6.10. Hàm *remove()*

Cú pháp: *List.remove(key)*

Chức năng: Xóa 1 phần tử có giá trị là key xuất hiện đầu tiên trong *List*, nếu không có phần tử thì hàm sẽ báo lỗi.

Ví dụ 13:

```
sn = [5, 6, 7, 6, 4]
sn.remove(6)
print(sn) # => [5, 7, 6, 4]
sn.remove(4)
print(sn) # => [5, 7, 6]
sn.remove(9)
print(sn) # => Báo lỗi ValueError: list.remove(x): x not in list
```

1.6.11. Hàm *pop()*

Cú pháp: *List.pop([index])*

Chức năng: Xóa và trả về giá trị của phần tử ở vị trí index, nếu không có index thì phần tử cuối cùng trong *List* sẽ bị xóa.

Ví dụ 14:

```
sn = [5, 2, 7, 6, 4]
print(sn.pop()) # => 4
print(sn) # => [5, 2, 7, 6]
print(sn.pop(2)) # => 7
print(sn) # => [5, 2, 6]
```

1.6.12. Hàm *copy()*

Cú pháp: *List.copy()*

Chức năng: Trả về một List hoàn toàn giống với *List* ban đầu.

Ví dụ 15:

```
sn = [5, 2, 7, 6, 4]
sn_copy = sn.copy()
print(sn_copy) # => [5, 2, 7, 6, 4]
```

1.6.13. Hàm clear()

Cú pháp: `List.clear()`

Chức năng: Xóa tất cả các phần tử có trong *List*.

Ví dụ 16:

```
sn = [5, 2, 7, 6, 4]
sn.clear()
print(sn) # => []
```

1.6.14. Hàm reverse()

Cú pháp: `List.reverse()`

Chức năng: Đảo ngược *List*.

Ví dụ 17:

```
sn = [5, 2, 7, 6, 4]
sn.reverse()
print(sn) # => [4, 6, 7, 2, 5]
```

1.6.15. Hàm sort()

Cú pháp: `List.sort([reverse=False/True])`

Chức năng: Sắp xếp *List* theo thứ tự giảm dần nếu `reverse=True`, ngược lại hoặc không có `reverse` hàm sẽ sắp xếp tăng dần.

Ví dụ 18:

```
sn = [5, 2, 7, 6, 4]
sn.sort()
print(sn) # => [2, 4, 5, 6, 7]
sn.sort(reverse=True)
print(sn) # => [7, 6, 5, 4, 2]
```

1.6.16. Hàm extend()

Cú pháp: `List.extend(Seq)`

Chức năng: Thêm nội dung của *Seq* vào cuối *List*, trong đó *Seq* có thể là str, list, tuple, ...

Ví dụ 19:

```
sn = [5, 2, 7, 6, 4]
st = 'ABCD'
sn.extend(st)
print(sn) # => [5, 2, 7, 6, 4, 'A', 'B', 'C', 'D']
```

1.7. List comprehension

Sử dụng list comprehension là cách viết chương trình ngắn gọn để tạo một danh sách. Trong Python, nó chỉ đơn giản là cách nhanh chóng để lọc hoặc chỉnh sửa một danh sách dựa trên điều kiện nào đó được cụ thể hóa bằng một biểu thức. List comprehension sẽ rất hiệu quả khi ta đang sử dụng vòng lặp.

Để sử dụng list comprehension ta thực hiện theo cú pháp:

[<biểu thức> for <biến> in <danh sách> if <điều kiện>]

Ví dụ 20: Tạo một danh sách mới với các phần tử là bình phương của các phần tử trong danh sách đã có.

```
old_list = [1, 2, 5, 8, 10]
print(old_list)
new_list = [x**2 for x in old_list]
print(new_list)
```

Nếu không sử dụng list comprehension thì yêu cầu của ví dụ trên ta có thể phải viết như sau:

```
old_list = [1, 2, 5, 8, 10]
print(old_list)
new_list = []
for i in range(len(old_list)):
    new_list.append(old_list[i]**2)
print(new_list)
```

Cả 2 chương trình trên đều cho cùng một kết quả là:

```
[1, 2, 5, 8, 10]
[1, 4, 25, 64, 100]
```

Ví dụ 21: Tương tự ví dụ 20 nhưng chỉ tạo ra danh sách mới với các phần tử trong danh sách đã có có giá trị là chẵn.

```
old_list = [1, 2, 5, 8, 10]
print(old_list)
new_list = [x**2 for x in old_list if x%2==0]
print(new_list)
```

Chương trình trên đều cho cùng một kết quả là:

```
[1, 2, 5, 8, 10]
[4, 64, 100]
```

Sử dụng list comprehension rất hiệu quả, tuy nhiên không phải trường hợp nào cũng có thể sử dụng nó. Chúng ta không nên sử dụng list comprehension khi giải thuật phức tạp.

1.8. Các chương trình ứng dụng

1.8.1. Chương trình 1

Viết chương trình nhập vào một danh sách gồm n phần tử kiểu số nguyên, in ra danh sách vừa nhập. Cho biết giá trị của phần tử lớn nhất và nhỏ nhất. Cho biết có bao nhiêu phần tử có giá trị là k , với k nhập từ bàn phím. Nhập vào một giá trị bất kỳ, thêm nó vào chính giữa, đầu và cuối danh sách, in ra danh sách sau khi thêm.

```
n = int(input("Nhập số lượng phần tử:"))
ds = []
for i in range(0, n):
    s = int(input("Nhập số nguyên:"))
    ds.append(s)
print("Danh sách sau khi nhập là:")
for i in ds:
    print(i, end='\t')
print("\nGiá trị của phần tử lớn nhất là:", max(ds))
print("Giá trị của phần tử nhỏ nhất là:", min(ds))
k = int(input("Nhập giá trị để đếm:"))
print("Trong ds có", ds.count(k), "pt có giá trị là", k)
k = int(input("Nhập giá trị để thêm:"))
vtg = int(len(ds)/2)
ds.insert(vtg, k)
ds.insert(0, k)
ds.insert(len(ds), k)
print("Danh sách sau khi thêm là:", ds)
```

1.8.2. Chương trình 2

Viết chương trình nhập vào một danh sách gồm m phần tử kiểu số thực, in ra danh sách vừa nhập. Cho biết vị trí xuất hiện đầu tiên của phần tử có giá trị là x , với x nhập từ bàn phím. Xóa phần tử có giá trị là y xuất hiện cuối cùng trong danh sách, với y nhập từ bàn phím. Sao chép n phần tử từ vị trí k trong danh sách hiện có thành một danh sách mới, in ra danh sách vừa sao chép.

```
m = int(input("Nhập số lượng phần tử:"))
ds = []
for i in range(0, m):
    s = float(input("Nhập số thực:"))
    ds.append(s)
print("Danh sách sau khi nhập là:")
print(ds)
x = float(input("Nhập số thực x cần tìm:"))
```

```

if x in ds:
    vt = ds.index(x)
    print("Vị trí xuất hiện đầu tiên của", x, "là:", vt)
else:
    print("Không tồn tại", x, "trong danh sách")
y = float(input("Nhập số thực y cần xóa:"))
if y in ds:
    vt = len(ds)-1
    while ds[vt] != y:
        vt = vt - 1
    del ds[vt]
    print("Danh sách sau khi xóa", y, "sau cùng là:")
    print(ds)
else:
    print("Không tồn tại", y, "trong danh sách")
n = int(input("Nhập số lượng phần tử cần sao chép:"))
k = int(input("Nhập vị trí cần sao chép:"))
dsm = ds[k:k+n]
print("Danh sách được sao chép là:")
print(dsm)

```

1.8.3. Chương trình 3

Viết chương trình nhập vào một danh sách gồm n phần tử có kiểu chuỗi, in ra danh sách vừa nhập. Tạo ra danh sách đảo ngược của danh sách hiện có. Cho biết danh sách đã nhập có đối xứng không? Xóa tất cả các phần tử có giá trị là chuỗi st, với st nhập từ bàn phím. Sắp xếp danh sách theo thứ tự giảm dần, in ra danh sách sau khi sắp xếp.

```

n = int(input("Nhập số lượng phần tử:"))
ds = []
for i in range(0, n):
    s = input("Nhập chuỗi ký tự:")
    ds.append(s)
print("Danh sách sau khi nhập là:")
print(ds)
dsn = ds.copy()
dsn.reverse()
print("Danh sách đảo ngược của danh sách hiện có là:")
print(dsn)
i = 0
kt = True

```

```
while i < len(ds) and kt == True:
    if(ds[i] == dsn[i]):
        i = i + 1
    else:
        kt = False
if kt == True:
    print("Danh sách đối xứng")
else:
    print("Danh sách không đối xứng")
st = input("Nhập chuỗi ký tự:")
while st in ds:
    ds.remove(st)
print("Danh sách sau khi xóa chuỗi", st, ":")
print(ds)
ds.sort(reverse=True)
print("Danh sách sau khi sắp xếp là:")
print(ds)
```

2. DỮ LIỆU KIỂU TUPLE

2.1. Định nghĩa

Tuple cũng là một kiểu dữ liệu tương tự như List, tuy nhiên nó khác với List ở chỗ là giá trị các phần tử trong Tuple là không thể thay đổi được (immutable) và không có hỗ trợ một số hàm như append, pop,...

2.2. Khai báo và khởi tạo

Một Tuple trong Python được khai báo và khởi tạo theo cú pháp:

<Tên_Tuple> = (Giá trị_1, Giá trị_2, Giá trị_3, ..., Giá trị_n)

Ví dụ 22:

```
tp_sn = (1, 2, 3, 4, 5)
tt_sv = ('19004001', "Phan Thanh Bình", 2001, 8.5, True)
```

Tuple có thể chứa mọi giá trị, ngay cả chính nó.

Nếu khởi tạo 1 Tuple chỉ có 1 phần tử thì ta phải thêm 1 dấu phẩy sau phần tử đó.

<Tên_Tuple> = (Giá trị,)

Ví dụ 23:

```
tp_st = (7.8, )
ds_sn = [5, 2, 7, 6, 4]
tp_sn = (5, 2, 7, 6, 4)
ds_sn[2] = 100
```

```
print(ds_sn) # => [5, 2, 100, 6, 4]
print(tp_sn[2]) # => 7
tp_sn[2] = 100 # => Báo lỗi TypeError: 'tuple' object does
not support item assignment
```

2.3. Các tính chất của Tuple

Tuple cũng là một tập hợp hay là một container trong Python và có các tính chất sau:

- Vị trí/chỉ mục của các phần tử trong Tuple được tính từ 0.
- Tuple tương tự như List trong Python hay mảng 1 chiều trong các NNLT khác. Mặc dù các phần tử nằm trong cặp dấu () nhưng khi truy xuất đến các phần tử thì ta dùng cặp dấu [].
- Tốc độ truy xuất của Tuple nhanh hơn List.
- Chiếm vùng nhớ ít hơn List.
- Bảo vệ cho dữ liệu không bị thay đổi.
- Có thể dùng làm key của Dictionary.

2.4. Các toán tử trên Tuple

Hoàn toàn tương tự như List. Tuy nhiên trên Tuple không có toán tử del.

2.5. Một số hàm thường dùng trên Tuple

Tuple cũng có một số hàm đã được xây dựng sẵn nhưng không nhiều bằng List. Trên Tuple ta có thể sử dụng các hàm: len, max, min, tuple(Seq), count, index tương tự như List. Còn các hàm khác như: append, insert, remove, pop, copy,... thì không được hỗ trợ.

Ví dụ 24:

Viết chương trình nhập vào tọa độ của 3 điểm trong hệ tọa độ Descartes, in ra các điểm đã nhập. Kiểm tra xem 3 điểm đó có tạo thành một tam giác không? Nếu tạo thành tam giác thì cho biết chu vi và diện tích của tam giác đó.

```
import math
xA = float(input("Nhập hoành độ cho A:"))
yA = float(input("Nhập tung độ cho A:"))
DiemA = (xA, yA)
xB = float(input("Nhập hoành độ cho B:"))
yB = float(input("Nhập tung độ cho B:"))
DiemB = (xB, yB)
xC = float(input("Nhập hoành độ cho C:"))
yC = float(input("Nhập tung độ cho C:"))
DiemC = (xC, yC)
print("Tọa độ 3 điểm đã nhập là:")
print("Điểm A =", DiemA)
```

```

print("Điểm B =", DiemB)
print("Điểm C =", DiemC)
AB = math.sqrt(pow(xB - xA, 2) + pow(yB - yA, 2))
AC = math.sqrt(pow(xC - xA, 2) + pow(yC - yA, 2))
BC = math.sqrt(pow(xC - xB, 2) + pow(yC - yB, 2))
if (AB + AC > BC and AB + BC > AC and AC + BC > AB):
    print("3 điểm này tạo thành một tam giác.")
    cv = AB + AC + BC
    print("Chu vi của tam giác =", cv)
    P = cv/2
    dt = math.sqrt(P*(P - AB)*(P - AC)*(P - BC))
    print("Diện tích của tam giác =", dt)
else:
    print("3 điểm này KHÔNG tạo thành một tam giác.")

```

3. DỮ LIỆU KIỂU SET

3.1. Định nghĩa

Python cũng có một kiểu dữ liệu gọi là Set. Một set là một nhóm các phần tử không lặp. Ứng dụng cơ bản của nó là dùng để kiểm tra hội viên và loại bỏ các phần tử trùng nhau.

3.2. Khai báo

Để khai báo kiểu tập hợp ta dùng từ khóa set và cặp dấu (). Nhưng ta không thể khai báo trực tiếp một biến kiểu set vì nó chỉ có 1 đối số có kiểu là str, list hoặc tuple.

Ví dụ 25:

```

mau_sac = ['xanh', 'đỏ', 'tím', 'vàng', 'tím', 'xanh']
th_mau = set(mau_sac)
print(th_mau)
# => {'xanh', 'tím', 'đỏ', 'vàng'}
the_thao = ('bóng đá', 'bóng rổ', 'bóng chuyền', 'cầu lông')
th_tt = set(the_thao)
print(th_tt)
# => {'bóng rổ', 'bóng đá', 'cầu lông', 'bóng chuyền'}

```

3.3. Các toán tử trên Set

Trên Set có các toán tử cơ bản là: hợp (|), giao (&), hiệu (-) và xor (^)

Ví dụ 26:

```

th1 = set('NNLT Python')
th2 = set('NNLT Pascal')
print(th1)
# => {'T', 'P', 'N', 'L', 'y', 't', ' ', 'o', 'h', 'n'}

```

```

print(th2)
# => {'T', 'P', 'l', 'N', 'L', 'c', 's', 'a', ' '}
print(th1 | th2)
# => {'P', 'l', 'N', 'L', 'c', 's', 'y', ' ', 'h', 'T',
      'a', 't', 'o', 'n'}
print(th1 & th2)
# => {'T', 'P', 'N', 'L', ' '}
print(th1 - th2)
# => {'y', 't', 'o', 'h', 'n'}
print(th1 ^ th2)
# => {'l', 'c', 's', 'y', 'a', 't', 'o', 'h', 'n'}

```

4. DỮ LIỆU KIỂU DICT

4.1. Định nghĩa

Dict cũng là 1 trong các kiểu tập hợp được gọi chung container trong Python. Tuy nhiên Dict (Dictionary) có điều khác biệt so với List và Tuple là các phần tử không có index để xác định vị trí/chỉ mục của nó. Trong Dict các phần tử được phân biệt bởi key (khóa) của chúng.

Dict hoạt động như quyển từ điển, tức là chúng ta muốn tra từ nào đó thì kết quả là nghĩa của từ đó. Dict cũng tương tự như vậy, khi chúng ta tra một key nào đó thì kết quả trả về là value (giá trị) của key đó.

4.2. Khai báo và khởi tạo

Dict được khai báo và khởi tạo theo cú pháp:

<Tên_dict> = {key_1: value_1, key_2: value_2, ..., key_n: value_n}

Ví dụ 27:

```

td = {}
print(td)
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)

```

4.3. Các tính chất của Dict

- Các phần tử được đặt trong cặp dấu {}.
- Các phần tử được ngăn cách nhau bởi dấu phẩy (,).
- Mỗi phần tử là một cặp key:value.
- Key và value ngăn cách nhau bằng dấu hai chấm (:).

4.4. Sử dụng Dict Comprehension

Ta có thể sử dụng Dict Comprehension để tạo ra một Dict theo cú pháp:

```
<Tên_dict> = {key : value for key, value in [(key_1, value_1),  
                                             (key_2, value_2), ..., (key_n, value_n)]}
```

Ví dụ 28:

```
td = {key: value for key, value in [("mssv", '19004001'),  
                                   ("ht", "Phan Thanh Bình"), ("ns", 2001), ("dtb", 8.5), ("gt", True)]}  
print(td)
```

4.5. Sử dụng Constructor Dict

- Khởi tạo Dict rỗng bằng **dict()**

Ví dụ 29:

```
empty_dict = dict()  
print(empty_dict) # => {}
```

- Khởi tạo Dict bằng **dict(iterable)**

Trong đó iterable có thể là List, Tuple hay bất cứ container nào có thể chứa cặp key:value. Điều quan trọng là container phải có cặp giá trị key:value.

Ví dụ 30:

```
ctn = [("mssv", '19004001'), ("ht", "Phan Thanh Bình"),  
       ("ns", 2001), ("dtb", 8.5), ("gt", True)]  
td = dict(ctn)  
print(td)
```

4.6. Sử dụng phương thức fromkeys

Ta cũng có thể khởi tạo Dict bằng phương thức fromkeys theo cú pháp:

dict.fromkeys(iterable, value)

Cách này giúp chúng ta khởi tạo một Dict với các key nằm trong một iterable. Tất cả các khóa đều nhận cùng 1 giá trị là value, giá trị mặc định của value là None.

Ví dụ 31:

```
key_1 = ('key1', 'key2', 'key3')  
val_1 = 'abc'  
td1 = dict.fromkeys(key_1, val_1)  
print(td1) # => {'key1': 'abc', 'key2': 'abc', 'key3': 'abc'}  
key_2 = ('key4', 'key5')  
td2 = dict.fromkeys(key_2)  
print(td2) # => {'key4': None, 'key5': None}
```

4.6. Truy cập các phần tử trong Dict

Để xác định được giá trị của các phần tử trong Dict thì ta truy cập key của chúng theo cú pháp:

<Tên_dict>[key]

Ví dụ 32:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
hoten = td["ht"]
dtb = td["dtb"]
print("Sinh viên", hoten, "có điểm TB là", dtb)
# => Sinh viên Phan Thanh Bình có điểm TB là 8.5
for x in td:
    print(x, ': ', td[x])
# => mssv : 19004001
      ht : Phan Thanh Bình
      ns : 2001
      dtb : 8.5
      gt : True
```

4.7. Các thao tác trên Dict

4.7.1. Thay đổi giá trị của phần tử trong Dict

Để thay đổi giá trị của phần tử trong Dict ta chỉ cần truy cập vào key của phần tử đó và gán lại giá trị mới cho value.

Ví dụ 33:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
td["ns"] = 2002
print(td)
# => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',
      'ns': 2002, 'dtb': 8.5, 'gt': True}
```

4.7.2. Thêm phần tử vào Dict

Thêm 1 phần tử vào Dict tức là ta cần thêm 1 cặp key:value theo cú pháp:

<Tên_dict>[key] = value

Ta lưu ý là phần tử thêm vào phải có key không trùng với các phần tử đã có, nếu key đã có thì nó chỉ thay đổi giá trị của value mà không thêm phần tử mới.

Ví dụ 34:

```
td = {"mssv": "19004001", "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
td["qq"] = "Vĩnh Long"
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',
               'ns': 2001, 'dtb': 8.5, 'gt': True, 'qq': 'Vĩnh Long'}
td["ns"] = 1999
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',
               'ns': 1999, 'dtb': 8.5, 'gt': True, 'qq': 'Vĩnh Long'}
```

4.7.3. Xóa phần tử trong Dict

Để xóa 1 phần tử trong Dict có khóa là key thì ta dùng cú pháp:

```
del <Tên_dict>[key]
```

Ví dụ 35:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
del td["gt"]
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',
               'ns': 2001, 'dtb': 8.5}
```

4.7.4. Xóa Dict

Để xóa Dict thì ta dùng cú pháp:

```
del <Tên_dict>
```

Ví dụ 36:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
del td
print(td)
# => Báo lỗi NameError: name 'td' is not defined
```

4.8. Một số hàm thường dùng trên Dict**4.8.1. Hàm len()**

Cú pháp: len(Dict)

Chức năng: Trả về độ dài (số phần tử) của Dict.

Ví dụ 37:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
print(len(td)) # => 5
```

4.8.2. Hàm str()

Cú pháp: str(*Dict*)

Chức năng: Chuyển một *Dict* sang một chuỗi.

Ví dụ 38:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
print(td[2]) # => Báo lỗi KeyError: 2  
st = str(td)  
print(st) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',  
              'ns': 2001, 'dtb': 8.5, 'gt': True}  
print(st[2]) # => m
```

✧ Tương tự chúng ta cũng có thể sử dụng các hàm list(*Dict*), tuple(*Dict*), set(*Dict*) để chuyển *Dict* sang các kiểu tương ứng.

4.8.3. Hàm copy()

Cú pháp: *Dict*.copy()

Chức năng: Trả về một *Dict* mới hoàn toàn giống với *Dict* ban đầu.

Ví dụ 39:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
td_cpy = td.copy()  
print(td_cpy)
```

4.8.4. Hàm clear()

Cú pháp: *Dict*.clear()

Chức năng: Xóa tất cả các phần tử có trong *Dict*.

Ví dụ 40:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)
```

```
td.clear()
print(td) # => {}
```

4.8.5. Hàm get()

Cú pháp: *Dict*.get(key[, default])

Chức năng: Trả về giá trị value của key tương ứng trong *Dict*, nếu không có key thì trả về giá trị default, nếu không có default thì trả về None.

Ví dụ 39:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
print(td.get("ns")) # => 2001
print(td.get("qq")) # => None
print(td.get("qq", "Vĩnh Long")) # => Vĩnh Long
```

4.8.6. Hàm keys()

Cú pháp: *Dict*.keys()

Chức năng: Trả về một giá trị thuộc lớp dict_keys, giá trị của dict_keys sẽ là tất cả các key có trong *Dict*.

Ví dụ 40:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
print(td.keys())
# => dict_keys(['mssv', 'ht', 'ns', 'dtb', 'gt'])
```

4.8.7. Hàm values()

Cú pháp: *Dict*.values()

Chức năng: Trả về một giá trị thuộc lớp dict_values, giá trị của dict_values sẽ là tất cả các value có trong *Dict*.

Ví dụ 41:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
print(td.values())
# => dict_values(['19004001', 'Phan Thanh Bình', 2001, 8.5, True])
```

4.8.8. Hàm items()

Cú pháp: *Dict.items()*

Chức năng: Trả về một giá trị thuộc lớp dict_items, giá trị của dict_items sẽ là tất cả các phần tử là các Tuple (key : value) có trong *Dict*.

Ví dụ 42:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
print(td.items())  
# => dict_items([('mssv', '19004001'), ('ht', 'Phan Thanh Bình'),  
                 ('ns', 2001), ('dtb', 8.5), ('gt', True)])
```

4.8.9. Hàm pop()

Cú pháp: *Dict.pop(key[, default])*

Chức năng: Trả về giá trị của phần tử có khóa là key trong *Dict* đồng thời xóa bỏ phần tử này khỏi *Dict*. Nếu phần tử có khóa là key không tồn tại thì trả về giá trị None nếu không có giá trị default, ngược lại trả về giá trị default.

Ví dụ 43:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
gt = td.pop("dtb")  
print(gt) # => 8.5  
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',  
               'ns': 2001, 'gt': True}
```

4.8.10. Hàm popitem()

Cú pháp: *Dict.popitem()*

Chức năng: Trả về giá trị của phần tử cuối cùng trong *Dict* là 1 Tuple (key : value) đồng thời xóa bỏ phần tử này khỏi *Dict*. Nếu *Dict* rỗng thì hàm sẽ báo lỗi.

Ví dụ 44:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",  
      "ns": 2001, "dtb": 8.5, "gt": True}  
print(td)  
tp = td.popitem()  
print(tp) # => ('gt', True)  
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',  
               'ns': 2001, 'dtb': 8.5}
```

```
td_empty = {}
tp = td_empty.popitem()
# => Báo lỗi KeyError: 'popitem(): dictionary is empty'
```

4.8.11. Hàm update()

Cú pháp: *Dict*.update(obj)

Chức năng: Nối thêm obj vào cuối *Dict*, với obj có thể là một Dict khác gồm một hay nhiều phần tử là các cặp key : value. Nếu key tồn tại trong *Dict* thì nó cập nhật trường value tương ứng, ngược lại thì nó thêm 1 phần tử vào cuối *Dict*.

Ví dụ 45:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình",
      "ns": 2001, "dtb": 8.5, "gt": True}
print(td)
td_ud = {"ns": 2000, "dt": "Kinh"}
td.update(td_ud)
print(td) # => {'mssv': '19004001', 'ht': 'Phan Thanh Bình',
               'ns': 2000, 'dtb': 8.5, 'gt': True, 'dt': 'Kinh'}
```

BÀI TẬP CHƯƠNG 4



- Viết chương trình nhập vào một **list** các phần tử kiểu số nguyên, in ra list đã nhập. Hãy lọc ra các số lẻ từ list đã nhập thành một list mới. Sau đó loại bỏ các phần tử trùng trong list mới này.
- Viết chương trình nhập vào một **list** các phần tử kiểu số thực, in ra list đã nhập. Xóa phần tử ở trí k với k nhập từ bàn phím. Tách list đang có thành 2 list: list1 có phần lẻ ≥ 0.5 và list2 có phần lẻ < 0.5 , in ra 2 list vừa tách. Sắp xếp list1 tăng dần và list2 giảm dần, trộn list1 và list2 thành một list mới với các phần tử xen kẽ nhau.
- Viết chương trình nhập 1 chuỗi gồm nhiều dòng, hãy chuyển các dòng trong chuỗi thành chữ in hoa và in ra màn hình.

Ví dụ: Input: Hello world!

Đây là ngôn ngữ lập trình Python

Verion 3.0 năm 2021

Output: HELLO WORLD!

ĐÂY LÀ NGÔN NGỮ LẬP TRÌNH PYTHON

VERION 3.0 NĂM 2021

4. Một Robot di chuyển trong mặt phẳng bắt đầu từ điểm đầu tiên (0,0). Robot có thể di chuyển theo hướng UP, DOWN, LEFT và RIGHT với những bước nhất định. Dấu di chuyển của robot được qui ước như sau:

UP 5

DOWN 3

LEFT 3

RIGHT 2

Từ đầu tiên chỉ hướng di chuyển, số đứng sau chính là số bước đi.

Hãy viết chương trình để tính toán khoảng cách từ vị trí hiện tại đến vị trí đầu tiên (0,0), sau khi robot đã di chuyển một quãng đường.

Ví dụ: Input: UP 5
DOWN 3
LEFT 3
RIGHT 2

Output: Khoảng cách di chuyển là 2.23606797749979

5. Viết chương trình nhập vào 2 số nguyên dương a và b thỏa điều kiện $a < b$, chọn và in một số chẵn ngẫu nhiên trong $[a, b]$, chọn và in một số ngẫu nhiên vừa chia hết cho 2 vừa chia hết cho 3 trong $[a, b]$.

Hướng dẫn: Sử dụng **list comprehension** và hàm **choice()** trong module random.

6. Viết chương trình nhập vào 2 số nguyên dương a và b thỏa điều kiện $a < b$, tạo ra list1 với 5 số ngẫu nhiên từ a đến b, tạo ra list2 với 10 số lẻ ngẫu nhiên từ a đến b, tạo ra list3 với k số ngẫu nhiên chia hết cho cả 5 và 7 từ a đến b, với k nhập từ bàn phím, từ list3 hãy tạo thêm list4 bằng cách trộn các phần tử trong list3, in ra các list đã tạo.

Hướng dẫn: Sử dụng **list comprehension** và hàm **sample()** và hàm **shuffle()** trong module random.

7. Viết chương trình nhập vào 1 **list** gồm n phần tử, tạo list1 sau khi xóa các phần tử chẵn trong list, tạo list2 sau khi xóa các phần tử là bội số của 3 trong list, tạo list3 sau khi xóa các phần tử nằm ở vị trí lẻ trong list, tạo list4 sau khi xóa các phần tử nằm ở các vị trí 0, 3 và 4 trong list, tạo list5 sau khi xóa tất cả các phần tử có giá trị là k trong list, với k nhập từ bàn phím.

Hướng dẫn: Sử dụng **list comprehension** để xóa nhiều phần tử và hàm **enumerate()** để lấy vị trí/index của phần tử.

8. Viết chương trình nhập vào một **list** các phần tử là chuỗi các số nhị phân 6 gồm chữ số, in ra list đã nhập. Chuyển các phần tử chia hết cho 5 sang một **tuple** mới. In kết quả của tuple mới đó. Sau đó thêm vào tuple này tất cả các phần tử chia hết cho 5 và < 30 mà chưa có trong tuple.

Ví dụ: Input: ['000101', '010100', '001111', '001010', '100011', '011110']
Tuple mới: ('000101', '001010', '010100', '100011')
Tuple đã thêm: ('000101', '001010', '010100', '100011', '001111', '011001')

9. Viết chương trình nhập vào chuỗi gồm nhiều số được cách nhau bởi dấu phẩy, hãy tạo ra một **list** và một **tuple** chứa các số trong chuỗi nhập.

Ví dụ: Input: 34,67,55,33,12,98

Output: ['34', '67', '55', '33', '12', '98'] và ('34', '67', '55', '33', '12', '98')

10. Viết chương trình nhập 1 chuỗi gồm nhiều dòng, mỗi dòng gồm name, age, height được cách nhau bởi dấu phẩy, sau đó chuyển mỗi dòng thành 1 phần tử trong **tuple** (name, age, height). Hãy sắp xếp các phần tử trong tuple theo thứ tự tăng dần, với tiêu chí sắp xếp là: name => age => height. In ra tuple trước và sau khi sắp xếp.

Ví dụ: Input: Jsn,21,1.60

John,20,1.70

Jony,19,1.71

Jony,18,1.73

Jsn,21,1.65

Output: [('Jsn', '21', '1.60'), ('John', '20', '1.70'), ('Jony', '19', '1.71'), ('Jony', '18', '1.73'), ('Jsn', '21', '1.65')]

[('John', '20', '1.70'), ('Jony', '18', '1.73'), ('Jony', '19', '1.71'), ('Jsn', '21', '1.60'), ('Jsn', '21', '1.65')]

11. Viết chương trình tạo một **tuple** tp gồm n phần tử với giá trị từ 1 đến n. Tạo 2 tuple mới gồm nửa số phần tử đầu và nửa số phần tử cuối từ tp. Tạo thêm 2 tuple khác để chứa các phần tử có giá trị lẻ và chẵn từ tp.

12. Viết chương trình nhập vào 1 **tuple** gồm n phần tử là các chữ cái in hoa, tạo ra tuple1 là tất cả hoán vị của n phần tử trong tuple, tạo ra tuple2 là tất cả hoán vị của k trong n phần tử trong tuple, với k nhập từ bàn phím và $0 \leq k \leq n$.

Ví dụ: tuple: ('A', 'B', 'C')

tuple1: (('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'), ('C', 'B', 'A'))

tuple2 (k=2): (('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B'))

Hướng dẫn: Sử dụng hàm **permutations()** trong module itertools để lấy tất cả các hoán vị trong tuple/list.

13. Viết chương trình nhập 1 chuỗi gồm nhiều từ được phân cách bởi khoảng trắng, hãy loại bỏ các từ trùng lặp, sau đó sắp xếp theo thứ tự bảng chữ cái và in ra chúng.

Ví dụ: Input: hello world and practice makes perfect and hello world again

Output: again and hello makes perfect practice world

Hướng dẫn: Sử dụng kiểu **set** để xóa phần tử trùng và hàm **sorted()** để sắp xếp **list**.

14. Viết chương trình nhập vào một số nguyên dương n, hãy tạo ra một **dictionary** chứa các cặp (i : i + i) với i = 1 .. n. Sau đó hiển thị dictionary lên màn hình.

Ví dụ: Input: n = 5

Output: {(1 : 2), (2 : 4), (3 : 6), (4 : 8), (5 : 10)}

15. Viết chương trình tạo một **dictionary** Anh – Việt bằng cách nhập lần lượt các từ tiếng Anh làm khóa (key) và nghĩa tiếng Việt tương ứng làm giá trị (value), việc tạo dictionary ngừng khi người nhập khóa là khoảng trắng. Hãy thực hiện công việc tra nghĩa của các từ có trong dictionary.

16. Viết một chương trình nhập vào một chuỗi bất kỳ, hãy đếm số chữ số, số chữ cái hoa và số chữ cái thường trong câu đó với yêu cầu phải sử dụng kiểu **dictionary** để thực hiện.

17. Viết chương trình nhập vào 1 chuỗi gồm nhiều từ. Hãy thống kê tần suất của từng từ trong chuỗi nhập và xuất ra sau khi đã sắp xếp theo bảng chữ cái.

Ví dụ: Input: New to Python or choosing between Python 2 and Python 3. Read Python 2 or Python 3.

Output: 2:2, 3.:2, New:1, Python:5, Read:1, and:1, between:1, choosing:1, or:2, to:1

Hướng dẫn: Sử dụng kiểu **dictionary** để thống kê với key là từ và value là tần suất.

18. Viết chương trình tạo ra một **list nested** (danh sách lồng) là ma trận (mảng 2 chiều) gồm m hàng, n cột với giá trị các phần tử là $i * j$ (chỉ số hàng * chỉ số cột). In ra ma trận vừa tạo. Đếm số phần tử là ước số của k, với k được nhập từ bàn phím. Tính tổng các phần tử nằm trên cột c, với c được nhập từ bàn phím. Sắp xếp giảm dần các phần tử nằm trên hàng h, với h được nhập từ bàn phím.

19. Viết chương trình nhập vào một **list nested** là ma trận vuông cấp N. In ra ma trận vừa nhập. In các phần tử nằm trên đường chéo chính. Tính tích các phần tử dương nằm trên đường chéo phụ. In ra nội dung của phần tử max. Cho biết vị trí của các phần tử min. Sắp xếp tăng dần các phần tử nằm trên cột c, với c được nhập từ bàn phím.

Chương 5: HÀM VÀ MODULE



1. HÀM

1.1. Khái niệm

Hàm (Function) là một chương trình con trong chương trình lớn. Hàm có thể có tham số/đối số hoặc không. Hàm có thể trả về giá trị hoặc không, trong trường hợp không trả về giá trị thì hàm hoạt động như một thủ tục trong các NNLT khác.

Một chương trình là một tập hợp các hàm, trong nhiều NNLT có một hàm chính để thực thi chương trình cũng như gọi thực hiện các hàm đã định nghĩa. Tuy nhiên đối với NNLT Python thì không có hàm chính, mà các hàm đã được định nghĩa trong chương trình sẽ được thực thi khi được gọi.

Hàm giúp cho việc phân đoạn chương trình thành những mô-đun riêng lẻ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa là một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, dễ dàng cho việc kiểm tra và bảo trì chương trình.

1.2. Khai báo

Một hàm trong Python được khai báo và định nghĩa theo cú pháp:

```
def <Tên_hàm> (TSHT_1, TSHT_2, ..., TSHT_n):  
    <khởi_lệnh>  
    [return <biểu_thức>]
```

Lưu ý:

- Nếu hàm không có giá trị trả về thì không cần có lệnh return.
- Một hàm có thể không có tham số nào.
- Nếu chưa nghĩ ra các lệnh thích hợp cho hàm thì ta có thể dùng lệnh **pass** để giữ chỗ cho khối lệnh trong hàm khi khai báo nhằm tránh việc để trống khối lệnh thì sẽ báo lỗi.

1.3. Gọi hàm

Để hàm thực hiện ta gọi hàm theo cú pháp:

```
<Tên_hàm> (TSTT_1, TSTT_2, ..., TSTT_n)
```

Tùy thuộc hàm có bao nhiêu TSHT mà khi gọi hàm ta truyền số lượng TSTT tương ứng.

Ví dụ 1: Xây dựng hàm tính giai thừa một số nguyên dương và gọi thực hiện hàm này.

```
def GiaiThua(n):  
    kq = 1  
    for i in range(1, n+1):  
        kq = kq * i  
    return kq
```

```
n = int(input("Nhập số nguyên dương n = "))
print(n, "! =", GiaiThua(n))
```

Ví dụ 2: Xây dựng hàm in ra các ước số của một số nguyên dương và gọi thực hiện hàm này.

```
def UocSo(n):
    for i in range(1, n+1):
        if n%i==0:
            print(i, end='\t')
m = int(input("Nhập số nguyên dương m = "))
print("Các ước số của", m, "là:")
UocSo(m)
```

Ví dụ 3: Xây dựng hàm in ra các số là bội số của 7 trong 100 số nguyên dương đầu tiên và gọi thực hiện hàm này.

```
def BoiSo7():
    i = 7
    while i < 101:
        print(i, end='\t')
        i += 7
print("Các bội số của", 7, "là:")
BoiSo7()
```

Ví dụ 4:

Nếu ta chưa viết được các câu lệnh cho hàm thì ta thay thế khối lệnh đó bằng lệnh pass, và khi hàm này được gọi thực hiện thì kết quả nhận được là None hoặc không có gì.

```
def GiaiThua(n):
    pass
print(5, "! =", GiaiThua(5))
# => Kết quả: 5 ! = None
def BoiSo7():
    pass
print("Các bội số của", 7, "là:")
BoiSo7()
# => Kết quả: Các bội số của 7 là:
```

1.4. Truyền tham số cho hàm

Tham số trong định nghĩa hàm được gọi là tham số hình thức (TSHT), còn tham số được truyền trong lời gọi hàm được gọi là tham số thực tế (TSTT). TSTT có thể là hằng giá trị, là biến hoặc là biểu thức.

Tương tự như các>NNLT khác, các TSHT trong Python có thể là tham số tự do hoặc tham số mặc định. Tham số tự do và tham số mặc định thì ta đều có thể truyền TSTT tương ứng cho chúng. Tuy nhiên ta có thể không truyền TSTT cho tham số mặc định và điều này sẽ báo lỗi nếu nó là tham số tự do. Điều cần lưu ý là các tham số mặc định phải luôn nằm phía sau (bên phải) các tham số tự do.

Ví dụ 5: Xây dựng hàm tính x lũy thừa n với x là số thực, n là số nguyên dương và gọi thực hiện hàm này.

```
def LuyThua(x, n = 2):
    kq = 1
    for i in range(1, n+1):
        kq = kq * x
    return kq
x = float(input("Nhập số thực x = "))
n = int(input("Nhập số nguyên n = "))
print(x, "^", n, "=", LuyThua(x, n))
print(x, "^", 2, "=", LuyThua(x))
```

Bên cạnh đó, Python còn hỗ trợ cách truyền TSTT là keyword bằng cách chỉ ra cụ thể từng cặp <Tên_TSHT> = <TSTT_tương ứng>, và ta có thể thay đổi thứ tự các cặp này so với lúc định nghĩa hàm.

Ví dụ 6:

```
def LuyThua(x, n):
    kq = 1
    for i in range(1, n+1):
        kq = kq * x
    return kq
a = LuyThua(5, 3)
b = LuyThua(3, 5)
print(a) # => 125
print(b) # => 243
y = LuyThua(x = 5, n = 3)
z = LuyThua(n = 3, x = 5)
print(y) # => 125
print(z) # => 125
```

Ngoài ra, trong Python còn có một loại tham số được gọi là tham số có số lượng thay đổi được. Nếu ta khai báo tham số có số lượng thay đổi được trong định nghĩa hàm thì khi gọi hàm đó ta có thể truyền nhiều/ít tham số hơn số lượng tham số mà ta đã khai báo trong hàm.

Ví dụ 7:

```
def Info(num_1, *num_n):
    print(num_1, end = '\t')
    for i in num_n:
        print(i, end = '\t')
Info(1) # => 1
Info(1, 2) # => 1    2
Info(1, 2, 3) # => 1    2    3
Info() # => Báo lỗi TypeError: Info() missing 1 required
positional argument: 'num_1'
```

Các TSHT và các biến mà chúng ta khai báo trong hàm là các biến cục bộ của hàm, các truy xuất từ bên ngoài đến các biến này đều không thực hiện được. Nhưng đối với các biến toàn cục, là các biến được khai báo bên ngoài hàm, thì trong hàm ta vẫn có thể gọi sử dụng chúng, tuy nhiên ta cần phải cẩn thận khi thực hiện điều này.

Ví dụ 8:

```
x = 5
def Tong(num):
    t = x + num
    return t
print(Tong(10)) # => 15
y = x + t # => Báo lỗi NameError: name 't' is not defined
```

Lưu ý: Trong Python không có hình thức truyền tham biến khi NLT muốn thay đổi giá trị của các tham số trong hàm như các NNLT khác. Vì thế nếu cần thiết thì NLT phải tìm cách xử lý khác chẳng hạn như sử dụng biến toàn cục.

Ví dụ 9: Hàm Swap sau đây **không** cho kết quả như mong muốn là hoán vị giá trị cho 2 tham số a và b.

```
def Tang(a):
    a = a + 10
x = 5
print("Trước khi gọi hàm x =", x) # => 5
Tang(x)
print("Sau khi gọi hàm x =", x) # => 5
Ta sử dụng biến toàn cục x để có thể thay đổi giá trị sau khi thực hiện hàm như sau:
def Tang(a):
    a = a + 10
    return a
x = 5
print("Trước khi gọi hàm x =", x) # => 5
x = Tang(x)
print("Sau khi gọi hàm x =", x) # => 15
```

1.5. Packing và Unpacking arguments

1.5.1. Unpacking arguments

Giả sử ta có một dãy các phần tử nằm trong một list, một tuple hay một container nào đó và chúng là các tham số cho 1 hàm nào đó. Cách đơn giản là ta sẽ dùng vị trí/chỉ mục (index) để gọi từng phần tử vào hàm. Tuy nhiên vấn đề đặt ra ở đây là nếu số phần tử này quá nhiều (hàng trăm, hàng ngàn) thì ta giải quyết như thế nào? Khi đó unpacking arguments sẽ giải quyết giúp chúng ta.

Unpacking arguments có 2 cách là: 1 sao (*) và 2 sao (**). Nếu unpacking arguments 1 sao đối với kiểu **dict** thì chỉ nhận được key, nếu muốn nhận được giá trị value thì ta phải dùng unpacking arguments 2 sao và khi đó ta phải truyền tham số theo cách keyword, tức là ta phải đảm bảo tham số và key của dict phải toàn hoàn giống nhau.

Ví dụ 10:

```
ls = ("Python", "C++", "Java")
def Xuat(t1, t2, t3):
    print (t1)
    print ("Hello " + t2)
    print ("Hi " + t3)
Xuat(ls[0], ls[1], ls[2])
# => Kết quả hiển thị lên màn hình là:
Python
Hello C++
Hi Java
```

Lưu ý: Ta có thể gọi hàm Xuat() với unpacking arguments 1 sao: Xuat(*ls)

=> Kết quả hiển thị lên màn hình hoàn toàn giống với lời gọi: Xuat(ls[0], ls[1], ls[2])

Ví dụ 11:

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình"}
def Info(a, b):
    print(a)
    print(b)
Info(*td) # => Kết quả hiển thị lên màn hình là:
mssv
ht
```

Để in được thông tin là giá trị của value thì ta phải unpacking arguments 2 sao.

```
td = {"mssv": '19004001', "ht": "Phan Thanh Bình"}
def Info(a, b):
    print(a)
```

```

    print(b)
Info(**td)
# => Báo lỗi TypeError: Info() got an unexpected keyword
argument 'mssv'

```

Mặc dù hàm trên được unpacking arguments 2 sao nhưng bị báo lỗi. Để không bị báo lỗi trên ta cần đặt tên 2 tham số trong hàm Info giống với 2 key trong dictionary td là mssv và ht. Ví dụ 5.11 được viết và gọi lại như sau:

```

td = {"mssv": '19004001', "ht": "Phan Thanh Bình"}
def Info(mssv, ht):
    print(mssv)
    print(ht)
Info(**td) # => Kết quả hiển thị lên màn hình là:
19004001
Phan Thanh Bình

```

1.5.2. Packing arguments

Unpacking có nghĩa là mở gói thì packing có nghĩa là đóng gói. Khi ta sử dụng packing argument là đồng nghĩa với việc ta nhờ một tham số đóng gói tất cả các giá trị truyền vào cho hàm thành một tuple hoặc dict. Nếu không có gì để đóng gói, tức là ta không truyền vào bất kỳ tham số nào thì ta sẽ nhận được một tuple rỗng hoặc dict rỗng. Để giao nhiệm vụ cho một tham số làm công việc này ta đặt 1 hoặc 2 dấu * trước tham số đó.

Cũng tương tự như unpacking arguments, packing 1 sao dùng để đóng gói các tham số thường và kết quả là tuple còn packing 2 sao dùng để đóng gói các tham số truyền bằng keyword và kết quả là dict.

Ví dụ 12:

```

def Xuat(*ts_thuong):
    print(ts_thuong)
    print(type(ts_thuong))
Xuat("Python", "C++", "Java")
# => Kết quả hiển thị lên màn hình là:
('Python', 'C++', 'Java')
<class 'tuple'>

```

Ví dụ 13:

```

def Info(**ts_keyword):
    for key, value in ts_keyword.items():
        print(key, ":", value)
    print(type(ts_keyword))
Info(mssv="19004001", ht="Phan Thanh Bình")

```



```
# => Kết quả hiển thị lên màn hình là:
mssv : 19004001
ht : Phan Thanh Bình
<class 'dict'>
```

Lưu ý:

- Nếu sau packing arguments còn tham số nào khác thì khi truyền giá trị cho nó ta phải dùng keyword argument.
- Ta có thể đặt các tham số thường sau packing arguments 1 sao nhưng điều này là không thể đối với packing arguments 2 sao.

Ví dụ 14:

```
def Xuat(*ts_thuong, ts):
    print(ts_thuong)
    print(type(ts_thuong))
    print(ts)

Xuat("Python", "C++", "Java", ts = "Pascal")

# => Kết quả hiển thị lên màn hình là:
('Python', 'C++', 'Java')
<class 'tuple'>
Pascal
```

Ví dụ 15:

```
def Info(**ts_keyword, ts):
    for key, value in ts_keyword.items():
        print(key, ":", value)
    print(type(ts_keyword))
    print(ts)

Info(mssv="19004001", ht="Phan Thanh Bình", ts = 2001)

# => Báo lỗi tại line 1
def Info(**ts_keyword, ts):
    ^
SyntaxError: invalid syntax
```

1.6. Hàm đệ quy

Một hàm gọi đến hàm khác là việc gọi hàm một cách thông thường, tuy nhiên nếu một hàm lại gọi đến chính bản thân nó thì ta gọi hàm là đệ quy. Hàm đệ quy sẽ thực hiện lâu hơn và tốn nhiều vùng nhớ hơn. Vì vậy đối với những bài toán có thể giải được bằng giải thuật lặp thì không nên dùng giải thuật đệ quy.

Tuy nhiên trên thực tế có nhiều bài toán mà khi ta tìm giải thuật đệ quy thì rất dễ nhưng tìm giải thuật lặp thì rất khó. Trong các trường hợp như thế thì đệ quy là phương pháp khá hữu dụng mà chúng ta cần biết đến.

Một hàm đệ quy là hàm có gọi lại chính bản thân nó trong khi định nghĩa hàm.

Ví dụ 16: Xây dựng hàm tính giai thừa của một số nguyên dương bằng giải thuật đệ quy.

```
def GiaiThua(n):
    if n==1:
        return 1
    else:
        return GiaiThua(n-1)*n
n = int(input("Nhập số nguyên dương n = "))
print(n, "! =", GiaiThua(n))
```

Ví dụ 17: Xây dựng hàm tìm ước chung lớn nhất của hai số nguyên dương bằng giải thuật đệ quy.

```
def UCLN(a, b):
    if a==b:
        return a
    elif a<b:
        return UCLN(a, b-a)
    else:
        return UCLN(a-b, b)
x = int(input("Nhập số nguyên dương x = "))
y = int(input("Nhập số nguyên dương y = "))
print("UCLN(", x, ', ', y, ") =", UCLN(x, y))
```

Ví dụ 18: Xây dựng hàm in ra các ước số của một số nguyên dương theo thứ tự giảm dần bằng giải thuật đệ quy.

```
def In_UocSo(n, i):
    if i>0:
        if n%i==0:
            print(i)
            In_UocSo(n, i-1)
n = int(input("Nhập số nguyên dương n = "))
print("Các ước số của", n, "là:")
In_UocSo(n, n)
```

1.7 Hàm nặc danh

Hàm nặc danh còn được gọi là hàm vô danh hay hàm lambda, hiểu theo cách đơn giản, là hàm không có tên và chúng không được khai báo theo cách chính thức bởi từ khóa **def**. Để khai báo hàm nặc danh ta dùng từ khóa **lambda**. Lambda nhận danh sách các tham số bất kỳ và chỉ trả về một giá trị dưới dạng một biểu thức đã được ước lượng. Chúng ta không thể gọi trực tiếp hàm nặc danh trong lệnh print mà cần phải gán vào 1 biến và biến này tương tự như tên hàm khi gọi.

Khai báo hàm nặc danh theo cú pháp:

lambda [TSHT_1, TSHT_2, ... TSHT_n]: Biểu_thức

Ví dụ 19: Xây dựng hàm nặc danh để tính x lập phương.

```
lp = lambda x: x**3  
print(lp(5))
```

Hàm lambda là một công cụ nhanh, gọn để ta có thể tạo ra một hàm và sử dụng nó hoặc khi ta chỉ cần khởi tạo một hàm cấu trúc đơn giản và tái sử dụng nhiều lần. Việc sử dụng nó thay cho một hàm có tên là không quan trọng. Tất nhiên ta có thể chỉ sử dụng hàm có tên thôi là đủ. Và trong trường hợp ta cần xây dựng hàm có giải thuật phức tạp thì ta không thể sử dụng hàm nặc danh được.

1.8. Hàm sinh tự động

Trong Python có một loại hàm đặc biệt được gọi là **hàm sinh tự động (generator)** dùng để **trả về một đối tượng (iterator) mà chúng ta có thể lặp lại (một giá trị tại một thời điểm)**. Nó tạo ra một đối tượng kiểu danh sách, nhưng ta chỉ có thể duyệt qua các phần tử của generator một lần duy nhất vì generator không lưu dữ liệu trong bộ nhớ mà cứ mỗi lần lặp thì chúng sẽ tạo phần tử tiếp theo trong dãy và trả về chính phần tử đó.

1.8.1. Cách tạo

Generator cũng tương tự như hàm bình thường nên cũng được khai báo bắt đầu bởi từ khóa **def**. Tuy nhiên trong generator phải có ít nhất một lệnh **yield <biểu thức>**. Trong generator có thể có nhiều lệnh **yield** và có cả lệnh **return**. Trong trường hợp này, cả **yield** và **return** đều trả về các giá trị từ hàm.

Điều khác biệt ở đây là **return** sẽ chấm dứt hoàn toàn một hàm, còn **yield** sẽ chỉ tạm dừng các trạng thái bên trong hàm và sau đó vẫn có thể tiếp tục khi được gọi trong các lần sau.

Chẳng hạn khi ta gọi phương thức **__next__()** lần thứ nhất, generator thực hiện các công việc tính toán giá trị rồi gặp từ khóa **yield** nào thì nó sẽ trả về các phần tử tại vị trí đó, khi ta gọi phương thức **__next__()** lần thứ hai thì generator không bắt đầu chạy tại vị trí đầu tiên mà nó sẽ bắt đầu ngay phía sau từ khóa **yield** thứ nhất. Cứ như thế generator tạo ra các phần tử trong dãy, cho đến khi không còn gặp từ khóa **yield** nào nữa thì kết thúc và giải phóng đối tượng.

1.8.2. Sự khác biệt của hàm generator và hàm thông thường

- Hàm generator chứa một hoặc nhiều câu lệnh **yield**.

- Khi được gọi, generator trả về một đối tượng (*iterator*) nhưng không bắt đầu thực thi ngay lập tức.
- Phương thức `__next__()` được thực thi một cách tự động, vì thế ta có thể lặp qua các mục bằng cách sử dụng `next()`.
- Lệnh `yield` sẽ tạm dừng hàm, các biến cục bộ và trạng thái của chúng được ghi nhớ giữa các lệnh gọi liên tiếp. Mỗi lần lệnh `yield` được chạy, nó sẽ sinh ra một giá trị mới.

Ví dụ 20:

```
def my_gen():
    n = 1
    print('Đoạn CT đầu tiên')
    yield n
    n += 1
    print('Đoạn CT ở giữa')
    yield n
    n += 1
    print('Đoạn CT cuối cùng')
    yield n

ob = my_gen()
print(next(ob))
print(next(ob))
print(next(ob))
print(next(ob))
```

Kết quả hiển thị của chương trình trên là:

Đoạn CT đầu tiên

1

Đoạn CT ở giữa

2

Đoạn CT cuối cùng

3

Traceback (most recent call last): line 15, in <module>

print(next(ob))

StopIteration

Ví dụ trên ta thấy rằng giá trị của biến `n` được ghi nhớ giữa các lần gọi, không giống như các hàm bình thường sẽ kết thúc ngay sau mỗi lần gọi.

Khi ta gọi phương thức `next()` lần thứ nhất, generator thực hiện các công việc tính toán giá trị rồi trả về phần tử tại vị trí đó (`n = 1`), khi gọi phương thức `next()` lần thứ hai

thì generator không bắt đầu chạy tại vị trí đầu tiên mà bắt đầu ngay sau từ khóa yield thứ nhất. Cứ như thế generator tạo ra các phần tử trong dãy, cho đến khi không còn gặp từ khóa yield nào nữa thì giải phóng đối tượng (*StopIteration*).

Để khởi động lại quá trình ta cần tạo một đối tượng generator khác bằng câu lệnh tạo ra đối tượng tương ứng.

Lưu ý: Ta có thể sử dụng generator trực tiếp cho các vòng lặp for. Khi đó vòng lặp for lấy một iterator và lặp lại nó bằng hàm *next()*, tự động kết thúc khi *StopIteration* xảy ra.

Ví dụ 21:

```
def my_gen():
    n = 1
    print('Đoạn CT đầu tiên')
    yield n
    n += 1
    print('Đoạn CT ở giữa')
    yield n
    n += 1
    print('Đoạn CT cuối cùng')
    yield n
for ob in my_gen():
    print(ob)
```

Kết quả hiển thị của chương trình trên là:

```
Đoạn CT đầu tiên
1
Đoạn CT ở giữa
2
Đoạn CT cuối cùng
3
```

1.8.3. Biểu thức generator

Generator có thể dễ dàng được tạo ra khi sử dụng biểu thức generator. Giống như hàm lambda, generator cũng tạo một biểu thức generator vô danh.

Cú pháp tương tự như cú pháp của list comprehension, nhưng dấu ngoặc vuông được thay thế bằng dấu ngoặc tròn như sau:

(<biểu thức> for <biến> in <danh sách> if <điều kiện>)

List comprehension thì trả về một list, còn biểu thức generator trả về một generator tại một thời điểm khi được yêu cầu. Vì lý do này, biểu thức generator sử dụng ít bộ nhớ hơn, đem lại hiệu quả hơn so với list comprehension.

Ví dụ 22:

```
old_list = [1, 2, 5, 8, 10]
print(old_list)
new_list = (x**2 for x in old_list if x%2==0)
for ob in new_list:
    print(ob, end = '\t')
```

Ở ví dụ trên, biểu thức generator không tạo ra kết quả cần thiết ngay lập tức mà trả về đối tượng generator, cứ mỗi lần lặp thì chúng sẽ tạo phần tử tiếp theo trong new_list và trả về phần tử đó.

1.8.4. Các ưu điểm khi sử dụng generator

- *Đơn giản hóa code, dễ triển khai:* Generator có thể giúp cho việc viết chương trình được triển khai rõ ràng và ngắn gọn hơn.
- *Sử dụng ít bộ nhớ:* Một hàm thông thường khi trả về list sẽ lưu toàn bộ list trong bộ nhớ vì thế sẽ sử dụng dung lượng bộ nhớ nhiều hơn. Generator sẽ sử dụng ít bộ nhớ hơn vì chúng chỉ thực sự tạo kết quả khi được gọi đến, sinh ra một phần tử tại một thời điểm, đem lại hiệu quả nếu chúng ta không có nhu cầu duyệt nó quá nhiều lần.
- *Tạo ra các list vô hạn:* Generator là phương tiện hữu hiệu để tạo ra một luồng dữ liệu vô hạn. Các luồng vô hạn này không cần lưu trữ toàn bộ trong bộ nhớ vì generator chỉ tạo ra một phần tử tại một thời điểm.

Ví dụ 23:

```
def PowTwoGen(max = 1):
    n = 0
    while n < max:
        yield 2 ** n
        n += 1
ob = PowTwoGen()
while ob:
    print(next(ob))
```

Ví dụ 24:

```
def Sochan():
    n = 0
    while True:
        yield n
        n += 2
ob = Sochan()
while ob:
    print(next(ob))
```

2. MODULE

2.1. Khái niệm

Tất cả các ví dụ và chương trình mà chúng ta đã biết từ đầu tới giờ đều được thực thi trong command line hoặc từ một file python có phần mở rộng là .py, .dll, .so, ... Hạn chế của cách này là đối với các ứng dụng lớn, chương trình dài, phức tạp, có nhiều chức năng thì rất khó kiểm tra lỗi, bảo trì và tái sử dụng. Để giải quyết được các khó khăn này, NNLT Python cung cấp một cách đặt các định nghĩa (thường nhất là hàm) vào một tập tin và dùng chúng trong các chương trình khi cần thiết. Tập tin này được gọi là module. Các định nghĩa trong module có thể được đưa vào chương trình để thực thi bằng lệnh import. Module là một tập tin chứa các định nghĩa và các câu lệnh. Ta có thể truy cập tên của module thông qua biến toàn cục `__name__`.

2.2. Phân loại

Trong Python có 3 loại module thường được sử dụng đó là:

- Module được viết bằng NNLT Python, có phần mở rộng là .py.
- Các thư viện liên kết động, có phần mở rộng là .dll, .pyd, .so, .sl, ...
- C-Module liên kết với trình thông dịch.

2.3. Cách sử dụng

Muốn đưa module vào chương trình để thực thi ta thực hiện theo cú pháp:

import <Tên_module_1> [, <Tên_module_2>, ..., <Tên_module_n>]

Khi gặp câu lệnh trên thì trình thông dịch sẽ tiến hành tìm kiếm module tương ứng theo thứ tự thư mục sau:

- i. Thư mục hiện hành mà chương trình đang gọi.
- ii. Các thư mục trong PYTHONPATH (một biến môi trường với danh sách thư mục).
- iii. Thư mục mặc định có vị trí phụ thuộc vào chọn lựa trong quá trình cài đặt.

Để biết được đường dẫn mà một module đã được đưa vào chương trình thì ta thực thi đoạn code sau:

```
import Tên_module  
print(Tên_module.__file__)
```

Ví dụ 20:

```
import Module_hamDQ  
print(Module_hamDQ.__file__)  
# => D:\DriveD\Bai tap thuc hanh\LTHDT by Python\Vi_du\Module_hamDQ.py
```

Để biết được danh sách các thuộc tính và hàm (phương thức) mà một module hỗ trợ thì ta dùng lệnh `dir` với cú pháp:

dir(Tên_module)

Ví dụ 21:

```
import Module_hamDQ
print(dir(Module_hamDQ))
# => ['GiaiThua', 'In_UocSo', 'UCLN', '__builtins__',
      '__cached__', '__doc__', '__file__', '__loader__',
      '__name__', '__package__', '__spec__']
```

Ví dụ 22: Tạo một module có tên là Module_hamDQ gồm có 3 hàm: tính giai thừa của một số nguyên dương, tìm ước chung lớn nhất của hai số nguyên dương, in ra các ước số của một số nguyên dương theo thứ tự giảm dần. Sau đó viết chương trình sử dụng module này để gọi và thực thi các hàm có trong module.

Bước 1: Tạo module với nội dung sau và lưu với tên là Module_hamDQ.py.

```
def GiaiThua(n):
    if n==1:
        return 1
    else:
        return GiaiThua(n-1)*n
def UCLN(a, b):
    if a==b:
        return a
    elif a<b:
        return UCLN(a, b-a)
    else:
        return UCLN(a-b, b)
def In_UocSo(n, i):
    if i>0:
        if n%i==0:
            print(i)
        In_UocSo(n, i-1)
```

Bước 2: Tạo file chương trình với nội dung sau và lưu với tên là Sudung_Module.py.

```
import Module_hamDQ
n = int(input("Nhập số nguyên dương n = "))
print(n,"! =",Module_hamDQ.GiaiThua(n))
print("Các ước số của", n, "là:")
Module_hamDQ.In_UocSo(n, n)
x = int(input("Nhập số nguyên dương x = "))
y = int(input("Nhập số nguyên dương y = "))
print("UCLN(",x,',',y,") =",Module_hamDQ.UCLN(x, y))
```


Lưu ý:

- Khi ta đã import module vào chương trình thì ta có thể sử dụng các thành phần của module đó với cú pháp: **<Tên_module>.<Tên_thành_phần>**
- Ta có thể đặt bí danh cho module mà ta import vào file chương trình bằng lệnh: **import <Tên_module> as <Bí_danh>** và khi đó muốn sử dụng các thành phần của module thì ta viết: **<Bí_danh>.<Tên_thành_phần>**
- Nếu ta chỉ cần import một vài thành phần của module vào chương trình để sử dụng thì ta thực hiện theo cú pháp: **from <Tên_module> import <Các_thành_phần>**. Nếu muốn import tất cả các thành phần thì sử dụng dấu * là đại diện.
- Ta có thể tham khảo các module chuẩn tại <https://docs.python.org/3/py-modindex.html>
- Ta chỉ có thể import lại module (có cập nhật) khi trình thông dịch được khởi động lại. Điều này không thuận tiện, vì thế Python cung cấp cách thuận tiện hơn đó là:
import imp
imp.reload(<Tên_module>)

Ví dụ 23: Ta viết lại file chương trình Sudung_Module.py bằng cách sử dụng bí danh là mdDQ cho module Module_hamDQ.

```
import Module_hamDQ as mdDQ
n = int(input("Nhập số nguyên dương n = "))
print(n, "! =", mdDQ.GiaiThua(n))
print("Các ước số của", n, "là:")
mdDQ.In_UocSo(n, n)
x = int(input("Nhập số nguyên dương x = "))
y = int(input("Nhập số nguyên dương y = "))
print("UCLN(", x, ', ', y, ") =", mdDQ.UCLN(x, y))
```

Ví dụ 24: Ta viết lại file chương trình Sudung_Module.py chỉ cần sử dụng 2 hàm tính giai thừa và in ước số của module Module_hamDQ.

```
from Module_hamDQ import GiaiThua, In_UocSo
n = int(input("Nhập số nguyên dương n = "))
print(n, "! =", GiaiThua(n))
print("Các ước số của", n, "là:")
In_UocSo(n, n)
```

2.4. Đóng gói module

Trong Python, một số module trong cùng thư mục (kể cả trong thư con) có thể kết hợp lại để tạo ra một package, gọi là đóng các module thành một gói. Package giúp cho việc quản lý và sử dụng các module có liên quan đơn giản và hiệu quả hơn.

Để đóng các module trong 1 gói (package) ta thực hiện các bước sau:

Bước 1: Tạo thư mục để chứa các module cần đóng gói.

Bước 2: Tạo các module cần đóng gói trong thư mục đã tạo ở bước 1.

Bước 3: Tạo file `__init__.py` trong thư mục chứa các module.

Bước 4: Viết code cho file `__init__.py` như sau:

```
import <Tên_thư_mục>.<Tên_module_1>
import <Tên_thư_mục>.<Tên_module_2>
...
import <Tên_thư_mục>.<Tên_module_n>
```

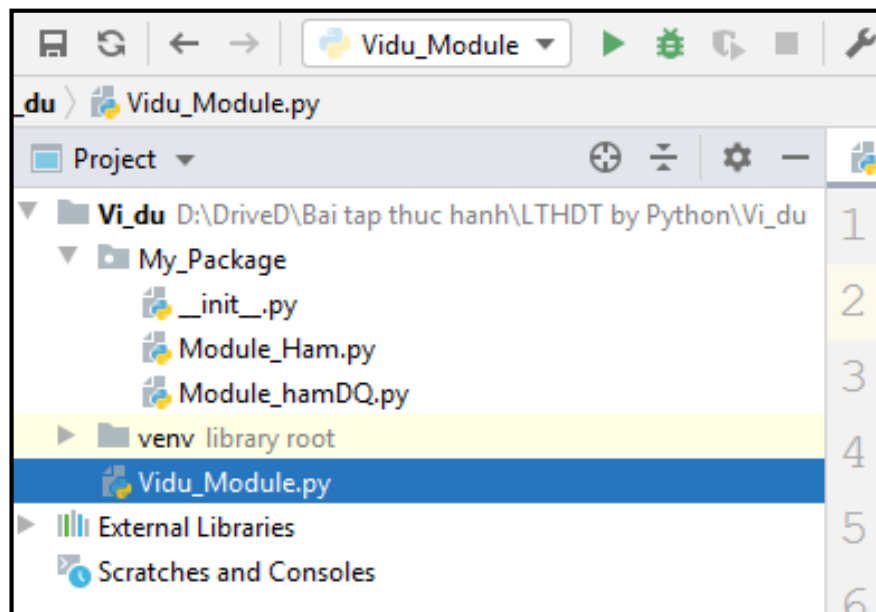
Bước 5: Tạo file để import package và sử dụng các thành phần trong các module đã được đóng gói trong package đó.

Lưu ý:

- File `__init__.py` có ý nghĩa đặc biệt trong Python. Nếu Python phát hiện thư mục nào có chứa file với tên `__init__.py` thì nó sẽ tự động xem thư mục đó là một package chứ không còn là một thư mục thông thường nữa.
- Các lệnh import ở bước 4 có thể viết bằng các cách khác mà chúng ta đã biết. Tuy nhiên ta nên sử dụng cùng 1 cách import ở cả 2 tập tin `__init__.py` và tập tin sử dụng package.

Ví dụ 25: Tạo package có tên `My_Package` để đóng gói 2 module là `Module_Ham` và `Module_hamDQ`. Trong đó `Module_Ham` chứa các hàm `UocSo`, `BoiSo7` và `LuyThua` và `Module_hamDQ` chứa các hàm `GiaiThua`, `UCLN` và `In_UocSo`. Sau đó tạo file `Vidu_Module.py` để sử dụng các hàm trong 2 modules của gói `My_Package`.

Cấu trúc thư mục và các tập tin như trong hình ảnh sau:



Nội dung 2 module `Module_Ham` và `Module_hamDQ` tương tự như các ví dụ phần hàm và module.

Nội dung file `__init__.py` như sau:

```
import My_Package.Module_Ham
import My_Package.Module_hamDQ
```

Nội dung file `Vidu_Module.py` như sau:

```
import My_Package
n = int(input("Nhập số nguyên dương n = "))
print(n, "! =", My_Package.Module_hamDQ.GiaiThua(n))
print("Các ước số của", n, "là:")
My_Package.Module_Ham.UocSo(n)
```

Lưu ý:

- File `Vidu_Module.py` ta có thể gọi thêm các hàm khác trong 2 module để tự kiểm chứng và làm quen với việc sử dụng hàm, module và package.
- Để có thể đặt bí danh cho các module thì ta phải import từng module trong package. Khi đó lệnh `import My_Package` được thay thế bởi:

```
import My_Package.Module_Ham as mdh
import My_Package.Module_hamDQ as mdDQ
```

BÀI TẬP CHƯƠNG 5

-----❧-----

1. Xây dựng **hàm** tính giai thừa của một số nguyên dương bằng giải thuật đệ quy, sau đó xây dựng tính tổng $S = 1! + 2! + 3! + \dots + N!$. Viết chương trình để gọi thực hiện các hàm đã xây dựng.

2. Xây dựng các **hàm** sau:

- Nhập một list gồm n phần tử kiểu số nguyên.
- In một list gồm n phần tử kiểu số nguyên.
- Trả về list mới bằng cách xóa bỏ các phần tử trùng trong list đã có.
- Trả về list mới bằng cách trộn xen kẽ 2 list đã có.
- Trả về list mới gồm các phần tử là giao 2 list đã có.

Hãy viết chương trình để gọi thực hiện các hàm trên.

3. Xây dựng các **hàm** sau:

- Tìm kiếm tuần tự để tìm một phần tử trong một list bất kỳ bằng giải thuật không đệ quy.
- Tìm kiếm tuần tự để tìm một phần tử trong một list bất kỳ bằng giải thuật đệ quy.

- c. Tìm kiếm nhị phân để tìm một phần tử trong một list đã có thứ tự bằng giải thuật không đệ quy.
- d. Tìm kiếm nhị phân để tìm một phần tử trong một list đã có thứ tự bằng giải thuật đệ quy.

Hãy viết chương trình để gọi thực hiện các hàm trên.

4. Viết chương trình nhập danh sách các số nguyên, in ra danh sách đã nhập, lọc ra danh sách các số dương, danh sách các số chẵn, danh sách các số chính phương, danh sách các số nguyên tố bằng cách sử dụng hàm **lambda**, **filter** và các **hàm khác**.

5. Viết chương trình nhập danh sách các số thực (list_float), in ra danh sách đã nhập, tạo danh sách chứa các giá trị là phần nguyên (list_trunc) từ danh sách nhập, tạo list chứa giá trị bình phương của các số chẵn (list_square) từ list_trunc, tạo danh sách chứa giá trị là tổng 2 phần tử tương ứng (list_sum) từ list_trunc và list_square, khi list_square hết phần tử thì xem như giá trị tương ứng của phần tử là 0 bằng cách sử dụng hàm **lambda**, **filter**, **map** và các **hàm khác**.

6. Xây dựng các **hàm generator** sau:

- a. Tạo ra các số là bội số của 7 nằm trong $[1 .. n]$.
- b. Tạo ra các số là ước số của số n.
- c. Tạo ra các số chia hết cho 2 nhưng không chia hết cho 3 nằm trong $[1 .. n]$.

Viết chương trình sử dụng các hàm generator đã xây dựng để in các số tương ứng, các số này được cách nhau bởi dấu phẩy và n được nhập từ bàn phím.

7. Xây dựng các **hàm đệ quy** sau:

- a. Xây dựng hàm tính: $f(n) = f(n-1) + 1/n$ khi $n > 0$ và $f(n) = 0$ nếu $n = 0$.
- b. Xây dựng hàm tính: $f(n) = f(n-3) + 10$ khi $n > 0$ và $f(n) = 1$ nếu $n \leq 0$.
- c. Xây dựng hàm tính: $f(n) = f(n-1) + f(n-2)$ khi $n > 2$ và $f(1) = f(2) = 1$.

Viết chương trình sử dụng các hàm đã xây dựng và list comprehension để in các dãy số tương ứng, các số trong dãy được cách nhau bởi dấu phẩy và n được nhập từ bàn phím.

Hướng dẫn: Sử dụng list comprehension để tạo ra list các số và hàm join() để nối các số trong list thành chuỗi.

8. Xây dựng **module** gồm các hàm sau:

- a. Tính trung bình cộng của bốn số bất kỳ.
- b. Tính trung bình nhân của ba số bất kỳ.
- c. Kiểm tra một số nguyên là lẻ.
- d. Kiểm tra một số nguyên là số chính phương.
- e. Kiểm tra một số nguyên là số nguyên tố.
- f. Chuyển số nguyên thành chuỗi tương ứng.
- g. Tính tổng hai chuỗi số nguyên.
- h. Ghép 3 chuỗi lại với nhau.
- i. Xác định chuỗi dài hơn trong 2 chuỗi.

Hãy viết chương trình để gọi thực hiện các hàm có trong module trên.

9. Xây dựng **module** gồm các hàm sau:

- Tạo và in ra một dictionary chứa key là các số từ 1 đến n và các giá trị bình phương của chúng.
- Tạo và in ra một dictionary chứa các key là các số từ 1 đến n và các giá trị nghịch đảo của chúng. Hàm chỉ in các value của dictionary.
- Tạo và in ra một dictionary chứa key là các số từ 1 đến n và các giá trị nghịch đảo của chúng. Hàm chỉ in các key của dictionary.

Hãy viết chương trình để gọi thực hiện các hàm có trong module trên.

10. Xây dựng **module** gồm các hàm sau:

- Tạo và in ra list chứa các giá trị là căn bậc hai của các số từ 1 đến n.
- Tạo ra list chứa các giá trị là căn bậc ba của các số từ 1 đến n và in m phần tử đầu tiên trong list.
- Tạo ra tuple chứa các giá trị lập phương của các số từ 1 đến n và in m phần tử cuối cùng trong tuple.
- Tạo ra tuple chứa các giá trị bình phương của các số từ 1 đến n và in m phần tử tính từ vị trí k trong tuple.

Hãy viết chương trình để gọi thực hiện các hàm có trong module trên.

11. Xây dựng **module** có tên XACSUAT gồm các hàm tính hoán vị, tổ hợp và chỉnh hợp. Sau đó viết chương trình để gọi thực hiện các hàm có trong module XACSUAT.

12. Xây dựng **module** có tên HINHHOC gồm các hàm tính chu vi và diện tích của các hình: tam giác, vuông, chữ nhật, bình hành, thang, tròn. Sau đó viết chương trình để gọi thực hiện các hàm có trong module HINHHOC.

13. Xây dựng **module** có tên DAYSO gồm các hàm tính tổng các dãy số sau:

- Dãy số tự nhiên: $1 + 2 + 3 + \dots + n$
- Dãy số lẻ: $1 + 3 + 5 + \dots + (2n - 1)$
- Dãy bình phương số tự nhiên: $1^2 + 2^2 + 3^2 + \dots + n^2$
- Cấp số tự nhiên nhân dồn: $1.2 + 2.3 + 3.4 + \dots + n.(n+1)$
- Cấp số cộng: $a_1 + a_2 + a_3 + \dots + a_n$, với công sai là d
- Cấp số nhân: $ar^0 + ar^1 + ar^2 + \dots + ar^n$, với công bội là r

Hãy viết chương trình để gọi thực hiện các hàm có trong module DAYSO.

14. Đóng gói 3 module XACSUAT, HINHHOC và DAYSO thành một **package** có tên là TOANHOC. Sau đó viết chương trình để gọi thực hiện các hàm có trong các module nằm trong package TOANHOC.

15. Python có nhiều hàm đã được tích hợp sẵn, nếu chưa biết cách sử dụng các hàm này thì NLT có thể đọc tài liệu trực tuyến hoặc tìm vài cuốn sách. Tuy nhiên Python cũng có sẵn tài liệu về tất cả các hàm đã tích hợp. Viết chương trình để in tài liệu về một số hàm đã được tích hợp sẵn: input(), print(), int(), sqrt(), pow(), abs(), format(), encode(), compress(), decompress(), choice(), sample(), shuffle(), enumerate(), permutations() và sorted().

Chương 6: TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG



1. LẬP TRÌNH HƯỚNG CẤU TRÚC

Lập trình hướng cấu trúc là tổ chức chương trình thành các chương trình con. Trong một số ngôn ngữ như PASCAL có 2 loại chương trình con là thủ tục và hàm, còn trong C++ chỉ có một loại chương trình con là hàm.

Hàm là một đơn vị chương trình độc lập dùng để thực hiện một công việc cụ thể nào đó như: nhập dữ liệu, in kết quả hay thực hiện một chức năng nào đó như tính toán, kiểm tra, ... Hàm cần có đối số và các biến cục bộ dùng riêng cho nó.

Việc trao đổi dữ liệu giữa các hàm, thủ tục được thực hiện thông qua lời gọi hàm và truyền các đối số. Một chương trình hướng cấu trúc gồm các cấu trúc dữ liệu (như biến, mảng, bản ghi) và các hàm, thủ tục.

Nhiệm vụ chính của việc tổ chức và thiết kế chương trình hướng cấu trúc là tổ chức chương trình thành các hàm, thủ tục.

2. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

2.1. Giới thiệu

Hướng đối tượng (object orientation) cung cấp một cách thức mới để xây dựng phần mềm. Trong cách này, những đối tượng (object) và các lớp (class) là những khối cần xây dựng, còn các phương thức (method), thông điệp (message), và sự kế thừa (inheritance) cung cấp các cơ chế thực hiện.

Lập trình hướng đối tượng (OOP – Object Oriented Programming) là một cách tư duy mới, tiếp cận hướng đối tượng để giải quyết vấn đề bằng máy tính. Thuật ngữ OOP ngày càng trở nên thông dụng trong lĩnh vực công nghệ thông tin.

Lập trình hướng đối tượng (OOP) là một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp và đối tượng.

OOP là tập hợp các kỹ thuật quan trọng mà có thể dùng để làm cho việc triển khai chương trình hiệu quả hơn. Quá trình phát triển của OOP như sau:

- Lập trình tuyến tính
- Lập trình cấu trúc
- Sự trừu tượng hóa dữ liệu
- Lập trình hướng đối tượng

2.2. Trừu tượng hóa (Abstraction)

Trừu tượng hóa là một kỹ thuật chỉ trình bày các đặc điểm cần thiết của vấn đề mà không trình bày những chi tiết cụ thể hay những lời giải thích phức tạp của vấn đề đó. Hay nói cách khác, trừu tượng hóa là một kỹ thuật tập trung vào thứ cần thiết và phớt lờ (bỏ qua) những thứ không cần thiết.

Ví dụ, những thông tin sau đây là các đặc tính gắn kết với con người:

- Màu da
- Màu mắt
- Quốc tịch
- Tên
- Ngày sinh
- Địa chỉ
- Chiều cao
- ...

Giả sử ta cần phát triển ứng dụng khách hàng mua sắm hàng hóa thì những chi tiết thiết yếu là tên, địa chỉ, tuổi còn những chi tiết khác (màu da, màu mắt, chiều cao, ...) là không quan trọng đối với ứng dụng. Tuy nhiên, nếu chúng ta phát triển một ứng dụng hỗ trợ cho việc điều tra tội phạm thì những thông tin như màu da, màu mắt, quốc tịch và chiều cao là rất cần thiết.

Sự trừu tượng hóa đã không ngừng phát triển trong các ngôn ngữ lập trình, nhưng chỉ ở mức dữ liệu và thủ tục. Trong OOP, việc này được nâng lên ở mức cao hơn - mức đối tượng. Sự trừu tượng hóa được phân thành sự trừu tượng hóa dữ liệu và trừu tượng hóa chương trình.

Trừu tượng hóa dữ liệu (data abstraction) là tiến trình xác định và nhóm các thuộc tính, các hành động có liên quan đến một thực thể đặc thù trong ứng dụng đang phát triển.

Trừu tượng hóa chương trình (program abstraction) là một sự trừu tượng hóa dữ liệu để làm cho các dịch vụ thay đổi theo dữ liệu.

3. CÁC THÀNH PHẦN CƠ BẢN TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

3.1. Đối tượng (object)

Các đối tượng là chìa khóa để hiểu được kỹ thuật lập trình hướng đối tượng. Chúng ta có thể nhìn xung quanh và thấy được nhiều đối tượng trong thế giới thực như: con mèo, con chó, cái bàn, quyển vở, cây viết, tivi, xe hơi ...

Trong một hệ thống hướng đối tượng, mọi thứ đều có thể là đối tượng. Một bảng tính, một ô trong bảng tính, một biểu đồ, một bảng báo cáo, một con số hay một số điện thoại, một tập tin, một thư mục, một máy in, một câu hoặc một từ, thậm chí một ký tự, tất cả chúng là những ví dụ về một đối tượng.

Rõ ràng chúng ta viết một chương trình hướng đối tượng cũng có nghĩa là chúng ta đang xây dựng một mô hình của một vài bộ phận trong thế giới thực. Tuy nhiên, các đối tượng này có thể được biểu diễn hay mô hình hóa trên máy tính.

Một đối tượng thế giới thực là một thực thể cụ thể mà thông thường ta có thể sờ, nhìn thấy hay cảm nhận được. Tất cả các đối tượng trong thế giới thực đều có trạng thái (state) và hành vi (behaviour).

Ví dụ 1:

Đối tượng	Trạng thái	Hành vi
Xe đạp	<ul style="list-style-type: none"> Bánh răng Bàn đạp Dây xích 	<ul style="list-style-type: none"> Tăng tốc Giảm tốc Chuyển bánh răng

Các đối tượng phần mềm (software object) có thể được dùng để biểu diễn các đối tượng thế giới thực. Chúng được mô hình hóa sau khi các đối tượng thế giới thực có cả trạng thái và hành vi.

Giống như các đối tượng thế giới thực, các đối tượng phần mềm cũng có thể có trạng thái và hành vi. Một đối tượng phần mềm có trạng thái (state) mà thường được gọi là thuộc tính (attribute/property) để duy trì trạng thái và phương thức (method) để thực hiện các hành vi của nó.

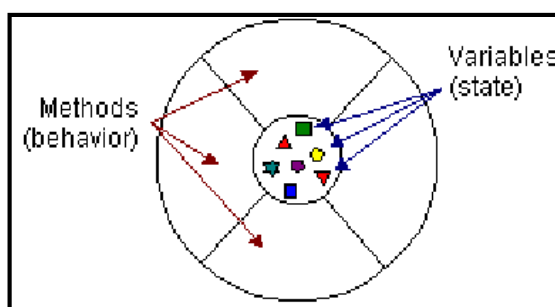
Thuộc tính là một hạng mục dữ liệu được đặt tên bởi một định danh (identifier) trong khi phương thức là một chức năng được kết hợp với đối tượng chứa nó.

Trong OOP thường sử dụng hai thuật ngữ là thuộc tính (attribute) và phương thức (method) để đặc tả tương ứng cho trạng thái (state) và hành vi (behavior) của đối tượng. Trong nhiều NNLT khác lại sử dụng hai thuật ngữ là dữ liệu thành viên (member data) và hàm thành viên (member function) thay cho các thuật ngữ trên.

Xét một cách đặc biệt, chỉ một đối tượng riêng lẻ thì chính nó không hữu dụng. Vì vậy một chương trình hướng đối tượng thường gồm có hai hay nhiều các đối tượng phần mềm tương tác lẫn nhau như là sự tương tác của các đối tượng trong thế giới thực.

Đối tượng (object) là một thực thể phần mềm bao bọc các thuộc tính và các phương thức có liên quan.

Kể từ đây, chúng ta sử dụng thuật ngữ đối tượng (object) để chỉ một đối tượng phần mềm. Hình 1 là một minh họa của một đối tượng phần mềm:



Hình 1 - Một đối tượng phần mềm

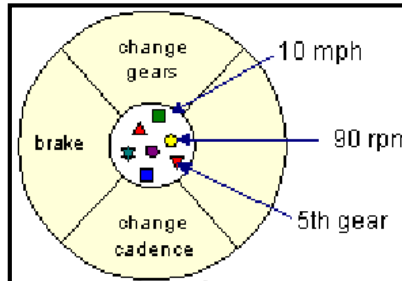
Mọi thứ mà đối tượng phần mềm biết (trạng thái) và có thể làm (hành vi) được thể hiện qua các thuộc tính và các phương thức. Một đối tượng phần mềm mô phỏng cho chiếc xe đạp sẽ có các thuộc tính để xác định các trạng thái của chiếc xe đạp như:

- Tốc độ của nó là 10 km/giờ
- Nhịp bàn đạp là 90 vòng/phút
- Bánh răng hiện tại là bánh răng thứ 5

Các thuộc tính này thông thường được xem như thuộc tính thể hiện (instance attribute) bởi vì chúng chứa đựng các trạng thái của một đối tượng xe đạp cụ thể. Trong kỹ thuật lập trình hướng đối tượng, một đối tượng cụ thể được gọi là một thể hiện (instance).

Thể hiện (instance) là một đối tượng cụ thể.

Hình 2 minh họa một xe đạp được mô hình như một đối tượng phần mềm:



Hình 2 - Đối tượng phần mềm xe đạp

Đối tượng xe đạp cũng có các phương thức để thắng lại, tăng nhịp đạp hay là chuyển đổi bánh răng. Nó không có phương thức để thay đổi tốc độ vì tốc độ của xe đạp có thể tính ra từ hai yếu tố số vòng quay và bánh răng hiện tại. Những phương thức này thông thường được biết như là các phương thức thể hiện (instance method) bởi vì chúng tác động làm thay đổi trạng thái của một đối tượng cụ thể.

3.2. Lớp

Trong thế giới thực thông thường có nhiều loại đối tượng cùng loại. Chẳng hạn chiếc xe đạp của bạn A chỉ là một trong hàng tỷ chiếc xe đạp trên thế giới. Tương tự, trong một chương trình hướng đối tượng có thể có nhiều đối tượng cùng loại và chia sẻ những đặc điểm chung.

Sử dụng thuật ngữ hướng đối tượng, chúng ta có thể nói rằng chiếc xe đạp của bạn A là một thể hiện của lớp xe đạp. Các chiếc xe đạp có một vài trạng thái chung (bánh răng hiện tại, số vòng quay hiện tại, hai bánh xe) và các hành động (chuyển bánh răng, giảm tốc). Tuy nhiên, trạng thái của mỗi chiếc xe đạp là độc lập và có thể khác với các trạng thái của các xe đạp khác.

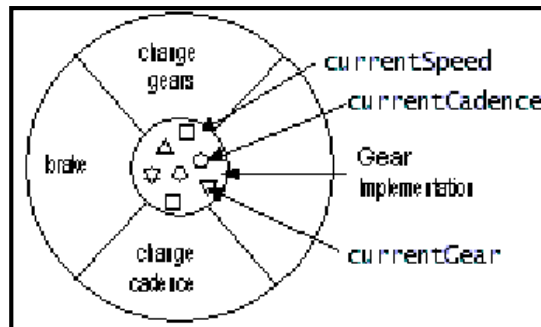
Trước khi tạo ra nhiều chiếc xe đạp, các nhà sản xuất thường tạo ra một bản thiết kế (blueprint) mô tả các đặc điểm và các yếu tố cơ bản của xe đạp. Sau đó, hàng loạt xe đạp sẽ được tạo ra từ bản thiết kế này. Thật không hiệu quả nếu như phải tạo ra một bản thiết kế mới cho mỗi chiếc xe đạp được sản xuất.

Trong phần mềm hướng đối tượng cũng có thể có nhiều đối tượng cùng loại chia sẻ những đặc điểm chung như là: các hình chữ nhật, các mẫu tin nhân viên, các đoạn phim, ... Giống như là các nhà sản xuất xe đạp, ta có thể tạo ra một bản thiết kế cho các đối tượng này. Một bản thiết kế phần mềm cho các đối tượng như vậy được gọi là lớp (class).

Lớp (class) là một bản thiết kế (blueprint) hay một mẫu ban đầu (prototype) định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó. Một đối tượng là một thể hiện cụ thể của một lớp.

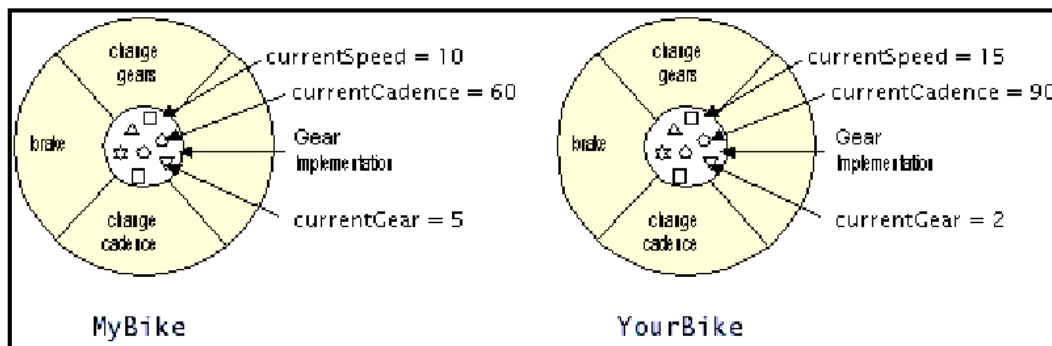
Trở lại ví dụ về xe đạp, chúng ta thấy rằng một lớp **Xedap** là một bản thiết kế cho hàng loạt các đối tượng xe đạp được tạo ra. Mỗi đối tượng xe đạp là một thể hiện của lớp Xedap và trạng thái của nó có thể khác với các đối tượng xe đạp khác.

Lớp Xedap sẽ khai báo các thuộc tính thể hiện cần thiết để lưu trữ giá trị bánh răng hiện tại, số vòng quay hiện tại, ... cho mỗi đối tượng xe đạp. Lớp Xedap cũng khai báo và cung cấp những hành động cho các phương thức thể hiện để cho phép người đi xe đạp chuyển đổi bánh răng, phanh lại, chuyển đổi số vòng quay, ... như hình 3.



Hình 3 - Khai báo cho lớp Xedap

Sau khi đã tạo ra lớp xe đạp, ta có thể tạo ra bất kỳ đối tượng xe đạp nào từ lớp này. Khi ta tạo ra một thể hiện của lớp, hệ thống cấp phát đủ bộ nhớ cho đối tượng và tất cả các thuộc tính thể hiện của nó. Mỗi thể hiện sẽ có vùng nhớ riêng cho các thuộc tính thể hiện của nó. Hình 4 minh họa hai đối tượng xe đạp khác nhau được tạo ra từ cùng lớp Xedap:



Hình 4 - Hai đối tượng của lớp Xedap

Ngoài các thuộc tính thể hiện, các lớp có thể định nghĩa các thuộc tính lớp (class attribute). Một thuộc tính lớp chứa đựng các thông tin được chia sẻ bởi tất cả các thể hiện của lớp. Ví dụ, tất cả xe đạp có cùng số lượng bánh răng. Trong trường hợp này, định nghĩa một thuộc tính thể hiện để giữ số lượng bánh răng là không hiệu quả bởi vì tất cả các vùng nhớ của các thuộc tính thể hiện này đều giữ cùng một giá trị. Trong những trường hợp như thế ta có thể định nghĩa một thuộc tính lớp để chứa đựng số lượng bánh răng của xe đạp. Khi đó, tất cả các thể hiện của lớp Xedap chia sẻ (có chung) thuộc tính này.

Một lớp cũng có thể khai báo các phương thức lớp (class methods). Ta có thể truy xuất một phương thức lớp trực tiếp từ lớp nhưng ngược lại ta phải truy xuất các phương thức thể hiện từ một thể hiện cụ thể nào đó.

Thuộc tính lớp (class attribute) là một hạng mục dữ liệu liên kết với một lớp cụ thể mà không liên kết với các thể hiện của lớp. Nó được định nghĩa bên trong lớp và được chia sẻ bởi tất cả các thể hiện của lớp.

Phương thức lớp (class method) là một phương thức được truy xuất mà không tham khảo đến một đối tượng thuộc lớp. Nó ảnh hưởng đến toàn bộ lớp chứ không ảnh hưởng đến một đối tượng thuộc lớp.

3.3. Thuộc tính

Các thuộc tính thể hiện trạng thái của đối tượng. Các thuộc tính chứa các giá trị dữ liệu trong một đối tượng, chúng xác định một đối tượng riêng biệt.

Thuộc tính (attribute) là dữ liệu thể hiện các đặc điểm về một đối tượng.

Một thuộc tính chỉ có thể được gán giá trị sau khi một đối tượng thuộc lớp được tạo ra. Các đối tượng của cùng một lớp thường có cùng các thuộc tính nhưng giá trị của các thuộc tính thì có thể khác nhau. Một thuộc tính của đối tượng có thể nhận các giá trị khác nhau tại những thời điểm khác nhau.

3.4 Phương thức

Các phương thức thực thi các hoạt động của đối tượng. Các phương thức là nhân tố làm thay đổi các thuộc tính của đối tượng.

Phương thức (method) có liên quan tới những công việc mà đối tượng có thể thực hiện. Một phương thức đáp ứng một chức năng nào đó tác động lên dữ liệu của đối tượng.

Các phương thức xác định cách thức hoạt động của một đối tượng và được thực thi khi đối tượng cụ thể được tạo ra. Ví dụ, các hoạt động chung của một đối tượng thuộc lớp Xedap là tăng tốc, giảm tốc, chuyển bánh răng. Tuy nhiên, chỉ khi một đối tượng cụ thể thuộc lớp Xedap được tạo ra thì các phương thức tăng tốc, giảm tốc, chuyển bánh răng mới được thực thi.

3.5 Thông điệp

Một chương trình hay ứng dụng lớn thường chứa nhiều đối tượng khác nhau. Các đối tượng phần mềm tương tác và giao tiếp với nhau bằng cách gửi các thông điệp (message). Khi đối tượng A muốn đối tượng B thực hiện các phương thức của đối tượng B thì đối tượng A gửi một thông điệp tới đối tượng B.

Ví dụ, đối tượng người đi xe đạp muốn đối tượng xe đạp thực hiện phương thức chuyển đổi bánh răng thì đối tượng người đi xe đạp cần phải gửi một thông điệp tới đối tượng xe đạp.

Đôi khi đối tượng nhận thông điệp cần thông tin nhiều hơn để biết chính xác thực hiện công việc gì. Ví dụ, khi ta chuyển bánh răng trên chiếc xe đạp thì ta phải chỉ rõ bánh răng nào mà ta muốn chuyển. Các thông tin này được truyền kèm theo thông điệp và được gọi là các tham số (parameter).

Một thông điệp gồm có:

- Đối tượng nhận thông điệp.

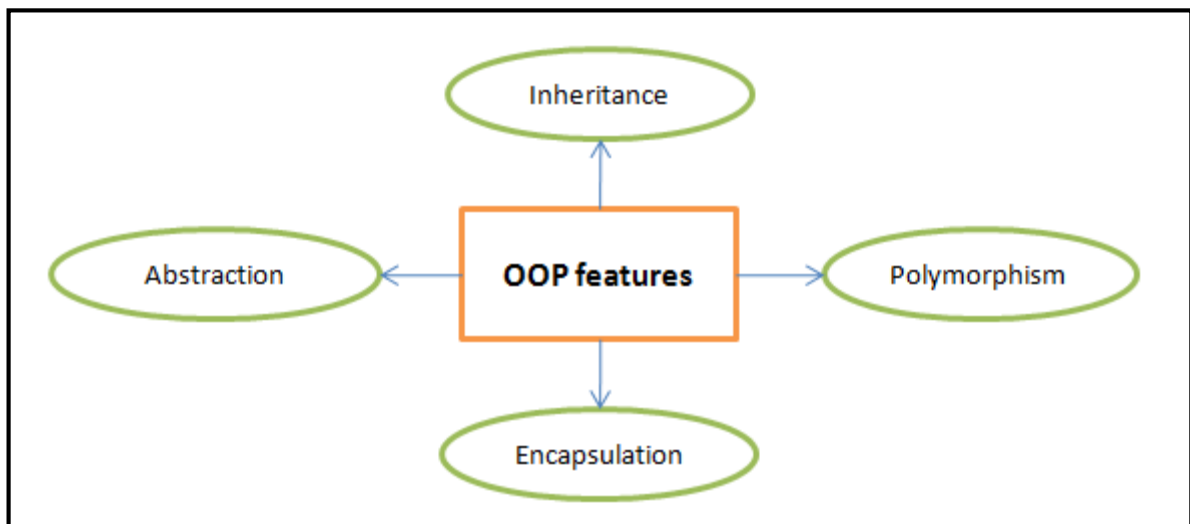
- Tên của phương thức cần thực thi.
- Các tham số mà phương thức cần biết để thực thi.

Thông điệp (message) là lời yêu cầu một hành vi/hoạt động. Một thông điệp được truyền khi một đối tượng muốn truy xuất một hay nhiều phương thức của đối tượng khác để yêu cầu thông tin.

Khi một đối tượng nhận được thông điệp thì nó sẽ thực hiện phương thức tương ứng. Ví dụ, khi đối tượng xe đạp nhận được thông điệp là chuyển đổi bánh răng thì nó sẽ thực hiện việc tìm kiếm phương thức chuyển đổi bánh răng tương ứng và thực hiện theo yêu cầu của thông điệp mà nó nhận được.

4. CÁC NGUYÊN LÝ CƠ BẢN TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Một ngôn ngữ lập trình hướng đối tượng nói chung đều thực hiện theo 4 nguyên lý như hình 5, trong đó kế thừa, đa hình và đóng gói là 3 nguyên lý không thể thiếu.



Hình 5 – Bốn nguyên lý cơ bản trong lập trình hướng đối tượng

4.1. Tính đóng gói

Trong đối tượng xe đạp, giá trị của các thuộc tính được chuyển đổi bởi các phương thức. Phương thức `changeGear()` chuyển đổi giá trị của thuộc tính `currentGear`. Thuộc tính `speed` được chuyển đổi bởi phương thức `changeGear()` hoặc `changRpm()`.

Trong OOP, các thuộc tính là trung tâm, là hạt nhân của đối tượng. Các phương thức bao quanh và che giấu đi hạt nhân của đối tượng từ các đối tượng khác trong chương trình. Việc bao bọc các thuộc tính của một đối tượng bên trong sự bảo vệ của các phương thức của nó được gọi là sự đóng gói (encapsulation) hay là đóng gói dữ liệu.

Nguyên lý đóng gói dữ liệu là ý tưởng của các nhà thiết kế các hệ thống hướng đối tượng. Tuy nhiên, việc áp dụng trong thực tế thì có thể không hoàn toàn như thế. Vì những lý do thực tế mà các đối tượng đôi khi cần phải trình bày ra một vài thuộc tính này và che giấu đi một số thuộc tính kia. Tùy thuộc vào từng NNLT hướng đối tượng mà chúng ta có các điều khiển cách truy xuất dữ liệu khác nhau.

Đóng gói (encapsulation) là tiến trình che giấu việc thực thi chi tiết của một đối tượng.

Một đối tượng có một giao diện chung cho các đối tượng khác sử dụng để giao tiếp với nó. Do đặc tính đóng gói mà các chi tiết như: các trạng thái được lưu trữ như thế nào hay các hành vi được thực thi ra sao có thể được che giấu từ các đối tượng khác.

Điều này có nghĩa là các chi tiết riêng của đối tượng có thể được thay đổi mà hoàn toàn không ảnh hưởng tới các đối tượng khác có liên hệ với nó. Ví dụ, một người đi xe đạp không cần biết chính xác cơ chế chuyển bánh răng trên xe đạp thực sự làm việc như thế nào nhưng vẫn có thể sử dụng nó. Điều này được gọi là che giấu thông tin.

Che giấu thông tin (information hiding) là việc ẩn đi các chi tiết của thiết kế hay thực thi từ các đối tượng khác.

4.2. Tính kế thừa

Lập trình hướng đối tượng cho phép các lớp được định nghĩa kế thừa từ các lớp khác. Ví dụ, lớp xe đạp leo núi và xe đạp đua là những lớp con (subclass) của lớp xe đạp. Như vậy ta có thể nói lớp xe đạp là lớp cha (superclass) của lớp xe đạp leo núi và xe đạp đua.

Kế thừa (inheritance) nghĩa là các thuộc tính và phương thức được định nghĩa trong một lớp có thể được thừa kế hoặc được sử dụng lại bởi lớp khác.

Lớp cha (superclass) là lớp có các thuộc tính hay phương thức được thừa hưởng bởi một hay nhiều lớp khác.

Lớp con (subclass) là lớp thừa hưởng các đặc tính chung của lớp cha và thêm vào những đặc tính riêng khác.

Các lớp con kế thừa thuộc tính và phương thức từ lớp cha của chúng. Ví dụ, một xe đạp leo núi không những có bánh răng, số vòng quay/phút và tốc độ giống như mọi xe đạp khác mà còn có thêm một vài loại bánh răng vì thế mà nó cần thêm một thuộc tính là gearRange (loại bánh răng).

Các lớp con có thể định nghĩa lại các phương thức được kế thừa để cung cấp các thực thi riêng biệt cho các phương thức này. Ví dụ, một xe đạp đua sẽ cần một phương thức đặc biệt để chuyển đổi bánh răng.

Các lớp con cung cấp các phiên bản đặc biệt của các lớp cha mà không cần phải định nghĩa mới hoàn toàn. Một lớp cha có thể được sử dụng lại nhiều lần.

4.3. Tính đa hình

Một khái niệm quan trọng khác có liên quan mật thiết với truyền thông điệp là đa hình (polymorphism). Với đa hình, nếu cùng một hành động (phương thức) áp dụng cho các đối tượng thuộc các lớp khác nhau thì có thể đưa đến những kết quả khác nhau.

Đa hình (polymorphism) nghĩa là “nhiều hình thức”, hành động cùng tên có thể được thực hiện khác nhau đối với các đối tượng hoặc các lớp khác nhau.

Chúng ta hãy xem xét các đối tượng Cửa Sổ và Cửa Cái. Cả hai đối tượng có một hành động chung có thể thực hiện là đóng. Nhưng một đối tượng Cửa Cái thực hiện hành động đóng có thể khác với cách mà một đối tượng Cửa Sổ thực hiện hành động đó. Cửa Cái khép cánh cửa lại trong khi Cửa Sổ hạ các thanh cửa xuống. Thật vậy, hành động đóng có thể thực hiện một trong hai hình thức khác nhau.

Một ví dụ khác là hành động hiển thị. Tùy thuộc vào đối tượng tác động, hành động hiển thị có thể xuất một chuỗi hay một số, hoặc vẽ một đường thẳng, hay in ra một hình chữ nhật.

Đa hình có sự liên quan tới việc truyền thông điệp. Đối tượng yêu cầu cần biết hành động nào để yêu cầu và yêu cầu từ đối tượng nào. Tuy nhiên đối tượng yêu cầu không cần lo lắng về một hành động được hoàn thành như thế nào.

4.4. Tính trừu tượng

Trừu tượng hóa (abstraction) là việc tập trung vào cốt lõi của đối tượng và bỏ qua những thứ không liên quan và không quan trọng.

Trừu tượng hóa dữ liệu và đóng gói thường được sử dụng như từ đồng nghĩa. Cả hai gần như đồng nghĩa với nhau vì sự trừu tượng hóa dữ liệu đạt được thông qua việc đóng gói.

5. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG PYTHON

Python là một ngôn ngữ lập trình hướng đối tượng mạnh. Vì vậy, việc tạo ra và sử dụng các đối tượng là hết sức dễ dàng.

Lập trình hướng đối tượng (OOP) là một kỹ thuật cho phép NLT trực tiếp làm việc với các đối tượng mà họ định nghĩa. Hiệu quả của kỹ thuật này giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm.

Khái niệm về OOP trong Python tập trung vào việc tạo code sử dụng lại. Khái niệm này còn được gọi là DRY (Don't Repeat Yourself).

Tương tự như các NLT khác, OOP trong Python cũng sử dụng các thành phần là *đối tượng*, *lớp*, *thuộc tính* và *phương thức* và cũng tuân theo 3 nguyên lý cơ bản là *tính đóng gói*, *tính kế thừa* và *tính đa hình*.

5.1. Lớp và đối tượng

Lớp và đối tượng là 2 khái niệm rất cơ bản trong lập trình hướng đối tượng.

5.1.1. Đối tượng

Đối tượng (object) là những thực thể tồn tại có hành vi.

Ví dụ đối tượng là một xe ô tô có tên hãng xe, màu sắc, loại nguyên liệu và các hành vi đi, dừng, đỗ, nổ máy, ...

5.1.2. Lớp

Lớp (class) là một kiểu dữ liệu đặc biệt do NLT định nghĩa, tập hợp nhiều thuộc tính đặc trưng cho mọi đối tượng được tạo ra từ lớp đó.

5.1.3. Thuộc tính (Attribute/Property)

Thuộc tính (attribute/property) là các thành phần dữ liệu của lớp hoặc đối tượng. Trong Python có 2 loại thuộc tính là thuộc tính của lớp và thuộc tính của đối tượng.

5.1.4. Phân biệt giữa đối tượng và lớp

Đối tượng (Object): Có trạng thái và hành vi.

Lớp (Class): Có thể được định nghĩa như là một template mô tả trạng thái và hành vi mà loại đối tượng của lớp hỗ trợ. Một đối tượng là một thực thể (instance) của một lớp.



5.1.5. Ví dụ về lớp và đối tượng

```
class Car:
    # thuộc tính lớp
    loaixe = "Ô tô"

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenvlieu):
        self.tenxe = tenxe
        self.mausac = mausac
        self.nguyenvlieu = nguyenvlieu

# Tạo các đối tượng thuộc lớp Car
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# Truy xuất thuộc tính lớp
print("Porsche là ", porsche.__class__.loaixe)
print("Toyota là ", toyota.__class__.loaixe)
print("Lamborghini cũng là ", lamborghini.__class__.loaixe)

# Truy xuất thuộc tính đối tượng
print("Xe", toyota.tenxe, "có màu", toyota.mausac,
      "và có nguyên liệu vận hành là", toyota.nguyenvlieu, ".")
```

```
print("Xe", lamborghini.tenxe, "có màu", lamborghini.mausac,  
      "và có nguyên liệu vận hành là", lamborghini.nguyenlieu, ".")  
print("Xe", porsche.tenxe, "có màu", porsche.mausac,  
      "và có nguyên liệu vận hành là", porsche.nguyenlieu, ".")
```

Kết quả hiển thị lên là:

Porsche là Ô tô

Toyota là Ô tô

Lamborghini cũng là Ô tô

Xe Toyota có màu Đỏ và có nguyên liệu vận hành là Điện.

Xe Lamborghini có màu Vàng và có nguyên liệu vận hành là Diesel.

Xe Porsche có màu Xanh và có nguyên liệu vận hành là Gas.

Trong ví dụ trên thì chương trình tạo một lớp *Car*, sau đó xác định các thuộc tính, đặc điểm của đối tượng.

Chúng ta truy cập thuộc tính của lớp bằng cách sử dụng `__class__.loaixe`. Các thuộc tính lớp được chia sẻ cho tất cả các cá thể của lớp hay là các đối tượng thuộc lớp.

Tương tự, chúng ta truy cập các thuộc tính của đối tượng, được gọi là instance bằng cách sử dụng `toyota.tenxe`, `toyota.mausac` và `toyota.nguyenlieu`.

Tuy nhiên, các thuộc tính instance là khác nhau cho mỗi đối tượng.

5.2. Phương thức

5.2.1. Phương thức

Phương thức (Method) là các hàm được định nghĩa bên trong phần thân của một lớp. Chúng được sử dụng để xác định các hành vi của một đối tượng.

5.2.2. Ví dụ về phương thức

```
class Car  
    # thuộc tính lớp  
    loaixe = "Ô tô"  
  
    # thuộc tính đối tượng  
    def __init__(self, tenxe, mausac, nguyenlieu):  
        self.tenxe = tenxe  
        self.mausac = mausac  
        self.nguyenlieu = nguyenlieu  
  
    # Các phương thức  
    def dungxe(self, mucdich):  
        return self.tenxe + " đang dùng xe để " + mucdich
```



```
def chayxe(self):
    return self.tenxe + " đang chạy trên đường"

def nomay(self):
    return self.tenxe + " đang nổ máy"

# Tạo các đối tượng thuộc lớp Car
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# Gọi các phương thức của các đối tượng
print(toyota.dungxe(" nạp điện" ))
print(lamborghini.chayxe())
print(porsche.nomay())

# Kết quả hiển thị là:
Toyota đang dùng xe để nạp điện
Lamborghini đang chạy trên đường
Porsche đang nổ máy
```

Ở ví dụ trên, có 3 phương thức là *dungxe()*, *chayxe()* và *nomay()*. Chúng được gọi là phương thức instance bởi vì chúng được gọi trên một đối tượng instance (*toyota*, *lamborghini*, *porsche*).

5.3. Tính kế thừa

5.3.1. Kế thừa

Kế thừa (inheritance) là cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa. Lớp đã có gọi là lớp cha/lớp cơ sở (parent class/superclass/base class), lớp mới phát sinh gọi là lớp con/lớp dẫn xuất (child class/subclass/derived class).

Lớp con kế thừa tất cả thành phần của lớp cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.

5.3.2. Ví dụ về kế thừa

```
class Car:
    # Constructor
    def __init__(self, hangxe, tenxe, mausac):
        # Lớp Car có 3 thuộc tính
        self.hangxe = hangxe
```

```
        self.tenxe = tenxe
        self.mausac = mausac
# Các phương thức
def chayxe(self):
    print(self.tenxe, "đang chạy trên đường")

def dungxe(self, mucdich):
    print(self.tenxe, "đang dừng xe để", mucdich)

# Lớp Toyota mở rộng từ lớp Car.
class Toyota(Car):

    # Gọi tới constructor của lớp cha (Car)
    # để gán giá trị vào thuộc tính của lớp cha
    def __init__(self, hangxe, tenxe, mausac, nguyenvlieu):
        super().__init__(hangxe, tenxe, mausac)
        self.nguyenvlieu = nguyenvlieu

    # Kế thừa phương thức cũ
    def chayxe(self):
        print(self.tenxe, "đang chạy trên đường")

    # Ghi đè (override) phương thức cùng tên của lớp cha
    def dungxe(self, mucdich):
        print(self.tenxe, "đang dừng xe để", mucdich)
        print(self.tenxe, "chạy bằng", self.nguyenvlieu)

    # Bổ sung thêm thành phần mới
    def nomay(self):
        print(self.tenxe, "đang nổ máy")

# Tạo các đối tượng thuộc lớp dẫn xuất Toyota
toyota1 = Toyota("Toyota", "Toyota Hilux", "Đỏ", "Electricity")
toyota2 = Toyota("Toyota", "Toyota Yaris", "Vàng", "Deisel")
toyota3 = Toyota("Toyota", "Toyota Vios", "Xanh", "Gas")

# Gọi các phương thức
toyota1.dungxe("nạp điện")
toyota2.chayxe()
```

```
toyota3.nomay()
# Kết quả hiển thị là:
Toyota Hilux đang dùng xe để nạp điện
Toyota Hilux chạy bằng Electricity
Toyota Yaris đang chạy trên đường
Toyota Vios đang nổ máy
```

Chương trình trên tạo 2 lớp: lớp cha (lớp cơ sở) là *Car* và lớp con (lớp dẫn xuất) là *Toyota*.

Khai báo constructor (phương thức khởi tạo) mới để gán giá trị vào thuộc tính của lớp cha. Hàm *super()* đứng trước phương thức `__init__` để gọi tới nội dung `__init__` của *Car*.

Lớp con *Toyota* kế thừa hàm *chayxe()* và *dungxe()* của lớp cha *Car* đồng thời sửa đổi một hành vi thể hiện ở phương thức *dungxe()*. Sau đó lớp con *Toyota* bổ sung thêm thành phần mới là phương thức *nomay()* để mở rộng kế thừa.

5.4. Tính đóng gói

5.4.1. Đóng gói

Sử dụng OOP trong Python, chúng ta có thể hạn chế quyền truy cập vào trạng thái bên trong của đối tượng. Điều này ngăn chặn dữ liệu bị sửa đổi trực tiếp, được gọi là **đóng gói (encapsulation)**.

Để thực hiện điều này ta phải khai báo thuộc tính dữ liệu phải có thuộc tính truy cập là *private* hoặc *protected*. Trong Python các thuộc tính dữ liệu có thuộc tính truy cập mặc định là *public*.

5.4.2. Ví dụ về đóng gói

```
class Computer:
```

```
    # Thuộc tính private ngăn chặn việc sửa đổi trực tiếp
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Giá bán sản phẩm:", self.__maxprice)

    def setMaxPrice(self, price):
        self.__maxprice = price
```

```
c = Computer()
```

```
c.sell()
```

```
# Thay đổi giá bằng cách sửa trực tiếp giá trị của thuộc tính
```

```
c.__maxprice = 1000
```

```
c.sell()
```

```
# Thay đổi giá bằng cách sử dụng hàm setter
```

```
c.setMaxPrice(1200)
```

```
c.sell()
```

```
# Kết quả hiển thị là:
```

```
Giá bán sản phẩm: 900
```

```
Giá bán sản phẩm: 900 #Không thay đổi được giá trị của maxprice
```

```
Giá bán sản phẩm: 1200
```

Ví dụ trên chúng ta đã khởi tạo lớp Computer, sử dụng phương thức `__init__()` để lưu trữ giá bán tối đa của máy tính. Nhưng sau khi sử dụng, ta có nhu cầu sửa đổi giá, tuy nhiên không thể thay đổi theo cách bình thường bằng câu lệnh `c.__maxprice = 1000` vì Python đã coi thuộc tính `__maxprice` có thuộc tính truy cập là private. Vì vậy để thay đổi được giá trị của thuộc tính `__maxprice` thì ta phải gọi hàm setter `c.setMaxPrice(1200)`.

5.5. Tính đa hình

5.5.1. Đa hình

Đa hình (polymorphism) là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách khác nhau.

Giả sử, chúng ta cần tô màu một hình khối, có rất nhiều lựa chọn cho hình cần tô như hình tam giác, hình chữ nhật, hình tròn, ... Tuy nhiên, chúng ta có thể sử dụng cùng một phương pháp để tô màu bất kỳ hình dạng nào.

5.5.2. Ví dụ về đa hình

```
class Toyota:
```

```
    def dungxe(self):
        print("Toyota dùng xe để nạp điện")
```

```
    def nomay(self):
        print("Toyota nổ máy bằng hộp số tự động")
```

```
class Porsche:
```

```
    def dungxe(self):
        print("Porsche dùng xe để bơm xăng")
```

```
    def nomay(self):
        print("Porsche nổ máy bằng hộp số cơ")
```

```
# Hàm kiểm tra việc dùng xe đối với một chiếc xe bất kỳ
```

```
def kiểmtra_dungxe(car): car.dungxe()
```

```
toyota = Toyota()
```

```
porsche = Porsche()
kiemtra_dungxe(toyota)
kiemtra_dungxe(porsche)
# Kết quả hiển thị là:
Toyota dùng xe để nạp điện
Porsche dùng xe để bơm xăng
```

Ví dụ trên chúng ta vừa tạo 2 lớp *Toyota* và *Porsche*, cả hai lớp đều có phương thức *dungxe()*. Tuy nhiên hàm của chúng là khác nhau. Ta sử dụng tính đa hình để tạo hàm chung cho hai lớp, đó là hàm *kiemtra_dungxe()*. Tiếp theo, chúng ta truyền đối tượng *toyota* và *porsche* vào hàm *kiemtra_dungxe()* và được kết quả như trên.

Như vậy chúng ta vừa xét qua một cách tổng thể các thành phần và các nguyên lý cơ bản của OOP trong Python và những điểm nổi bật của nó. Từ đó chúng ta có thể rút ra một số nhận xét như sau:

- Lập trình trở nên dễ dàng và hiệu quả hơn với OOP.
- Lớp (class) có thể chia sẻ được nên code dễ dàng được sử dụng lại.
- Năng suất của chương trình sẽ tăng lên.
- Dữ liệu an toàn và bảo mật với trừu tượng hóa dữ liệu.

BÀI TẬP CHƯƠNG 6



1. Hãy nêu các đặc trưng cơ bản của lập trình hướng cấu trúc và lập trình hướng đối tượng?
2. Hãy nêu các quá trình tiến hóa của lập trình hướng đối tượng?
3. Trừu tượng hóa là gì?
4. Hãy nêu các thành phần cơ bản trong lập trình hướng đối tượng?
5. Hãy nêu các nguyên lý cơ bản trong lập trình hướng đối tượng?
6. Đối tượng, lớp, thuộc tính, phương thức, thông điệp là gì? Cho ví dụ cụ thể đối với từng thành phần này?
7. Đóng gói, kế thừa, đa hình, trừu tượng là gì? Cho ví dụ cụ thể đối với từng nguyên lý này?
8. Hãy liệt kê các ngôn ngữ lập trình hướng đối tượng mà bạn biết? Cho biết đặc trưng của từng ngôn ngữ này?
9. Xây dựng lớp có tên là **Circle** có thuộc tính là bán kính và có các phương thức là nhập thông tin, in thông tin, tính chu vi, tính diện tích. Viết chương trình kiểm tra đúng đúng đắn của lớp Circle.
10. Xây dựng lớp có tên là **Rectangle** có 2 thuộc tính là chiều dài, chiều rộng và có các phương thức là nhập thông tin, in thông tin, tính chu vi, tính diện tích. Viết chương trình kiểm tra đúng đúng đắn của lớp Rectangle.
11. Xây dựng lớp **String** gồm có 1 thuộc tính để chứa dữ liệu của chuỗi nhập và các phương thức: nhận một chuỗi do người dùng nhập vào từ bàn phím, in chuỗi lên màn hình. Xây dựng lớp **String_Standardized** kế thừa lớp Chuoi và có thêm các phương thức: đổi chuỗi sang dạng upper, đổi chuỗi sang dạng lower, đổi chuỗi sang dạng proper. Viết chương trình kiểm tra đúng đúng đắn của các lớp đã xây dựng.
12. Xây dựng lớp **Number** gồm có 1 thuộc tính để chứa dữ liệu của chuỗi số nhập và các phương thức: nhận một chuỗi số do người dùng nhập vào từ bàn phím, in chuỗi số lên màn hình, chuyển chuỗi số thành số nguyên tương ứng. Xây dựng lớp **Integer** kế thừa lớp Number và có thêm các phương thức: tính giá trị bình phương của một số, tính giá trị căn bậc ba của một số, kiểm tra một số xem có phải là số hoàn hảo không, kiểm tra một số xem có phải là số thuận nghịch không. Xây dựng lớp **Float** kế thừa lớp Number, viết đề phương thức chuyển chuỗi số thành số thực tương ứng và có thêm phương thức tính giá trị lập phương của một số. Viết chương trình kiểm tra đúng đúng đắn của các lớp đã xây dựng.

Chương 7: LỚP VÀ ĐỐI TƯỢNG



1. KHÁI NIỆM LỚP

Python là một ngôn ngữ lập trình hướng đối tượng. Không giống như lập trình hướng thủ tục nhấn mạnh vào các hàm, lập trình hướng đối tượng tập trung làm việc trên các đối tượng.

Đối tượng (Object) chỉ đơn giản là một tập hợp các dữ liệu (các biến) và các phương thức (các hàm) hoạt động trên các dữ liệu đó.

Lớp (class) là một kế hoạch chi tiết cho đối tượng.

Python coi lớp là sự trừu tượng hóa của các đối tượng, là một khuôn mẫu để biểu diễn các đối tượng thông qua các thuộc tính và các hành vi đặc trưng của đối tượng.

Như vậy, chúng ta có thể nghĩ về lớp như một bản phác thảo (nguyên mẫu) của một ngôi nhà. Nó chứa tất cả các chi tiết về sàn nhà, trần nhà, các bức tường, mái nhà, cửa cái, cửa sổ, ... Dựa trên những mô tả này, chúng ta sẽ xây dựng được những ngôi nhà. Và mỗi ngôi nhà mà chúng ta đã xây dựng nên chính là một đối tượng.



Vì nhiều ngôi nhà có thể được xây dựng từ một bản phác thảo (mô tả chung) nên chúng ta có thể tạo ra nhiều vật thể từ một lớp. Một đối tượng cũng được gọi là một **thể hiện (instance)** của một lớp và quá trình tạo đối tượng này được gọi là instantiation.

1.1. Khai báo lớp

Cú pháp:

```
class <Tên lớp>:
```

```
    """Docstring của lớp tương tự như chú thích"""
```

```
    Các câu lệnh để định nghĩa lớp
```

Trong đó:

- **class:** Tên từ khóa bắt buộc để định nghĩa một lớp trong Python.

- **Tên lớp:** NLT tự định nghĩa, tên lớp có tính chất như tên kiểu dữ liệu. Cách đặt tên lớp phải tuân thủ theo quy tắc đặt tên và nên thể hiện được tính chất chung của các đối tượng mà nó đại diện.
- **Docstring** thường là một mô tả ngắn gọn về lớp, nó không bắt buộc phải có nhưng được khuyến khích sử dụng. Để trả về nội dung của docstring ta sử dụng thuộc tính đặc biệt của lớp là `__doc__`.
- **Các câu lệnh** để định nghĩa lớp thường là các khai báo thuộc tính và phương thức của lớp. Trong trường hợp ta chưa muốn khai báo cụ thể thì ta có thể dùng lệnh `pass` để bỏ qua và sau này chúng ta sẽ khai báo sau.

Class tạo ra một local namespace mới và trở thành nơi để các thuộc tính và phương thức của lớp được khai báo.

Trong lớp còn có các thuộc tính đặc biệt, được bắt đầu và kết thúc bởi 2 dấu gạch dưới (`__`). Chẳng hạn, `__doc__` sẽ trả về chuỗi docstring của lớp, `__class__` để truy cập thuộc tính của lớp, `__name__` để trả về tên của lớp, ...

Ngay khi khai báo một lớp, một đối tượng trong lớp mới sẽ được tạo ra có cùng tên với lớp đó. Đối tượng lớp này cho phép chúng ta truy cập các thuộc tính khác nhau cũng như để khởi tạo các đối tượng mới của lớp đó.

Ví dụ 1:

```
class MyClass:
    """Đây là class MyClass vừa được khởi tạo"""

    a = 10

    def func(self):
        print('Xin chào')

print(MyClass.__name__)
print(MyClass.__doc__)
print(MyClass.a)
print(MyClass.func)
# => Kết quả hiển thị là:
MyClass
Đây là class MyClass vừa được khởi tạo
10
<function MyClass.func at 0x03414AD8>
```

Lưu ý:

- Từ khóa `class` là bắt buộc phải có để định nghĩa một lớp trong Python. Hơn nữa, Python phân biệt chữ hoa chữ thường cho nên chữ `class` phải toàn bộ chữ thường.

- Các thuộc tính của lớp phải thụt đầu hàng 1 cấp so với từ khóa class.
- Để phân biệt với tên biến thông thường, ta nên đặt tên lớp bắt đầu bằng một chữ hoa và các tên biến bắt đầu bằng một chữ thường.

1.2. Sử dụng lớp

Lớp được sử dụng khi ta khai báo các thể hiện (instance) của lớp đó. Một thể hiện của một lớp chính là một đối tượng (object) cụ thể của lớp đó.

Đối tượng của lớp có thể được sử dụng để truy cập các thuộc tính khác nhau và tạo các thể hiện mới của lớp đó. Cách để tạo một đối tượng tương tự như cách chúng ta gọi hàm. Việc khai báo một đối tượng thuộc lớp được thực hiện theo cú pháp:

<Tên đối tượng> = <Tên lớp>()

Chẳng hạn ta viết lệnh: `ob = MyClass()`

Lệnh trên sẽ tạo ra một đối tượng mới có tên là *ob* thuộc lớp *MyClass*.

Để sử dụng các thuộc tính của lớp ta thực hiện theo cú pháp:

<Tên đối tượng>.<Thuộc tính>

Chẳng hạn ta viết lệnh: `print(ob.a)`

Ví dụ 2:

```
class MyClass:
    """Đây là class MyClass vừa được khởi tạo"""

    a = 10

    def func(self):
        print('Xin chào')

ob = MyClass()
print(ob.a)
ob.func()
# => Kết quả hiển thị là:
10
Xin chào
```

2. CÁC THÀNH PHẦN CỦA LỚP

Các thành phần của lớp được chia làm hai loại:

- Các thành phần chỉ dữ liệu của lớp, được gọi là **thuộc tính**.
- Các thành phần chỉ hành vi của lớp, được gọi là **phương thức**.

Thứ tự khai báo 2 thành phần của lớp như sau:

class <Tên lớp>:

<Khai báo các thuộc tính>

<Khai báo các phương thức>

Các thuộc tính và phương thức của lớp có 3 **thuộc tính truy cập** (access attribute) là public, protected và private.

- **public:** Các thành phần chung/công cộng. Các đối tượng của các lớp khác đều có thể truy cập đến các thành phần chung này. Đây là thuộc tính truy cập mặc định của các thuộc tính và phương thức của lớp.
- **protected:** Các thành phần được bảo vệ. Trong Python các thành phần được bảo vệ cũng được truy cập tương tự như các thành phần chung. Tuy nhiên, nhiều NNLT khác thì các thành phần được bảo vệ chỉ được truy cập bên trong lớp và ở các lớp con/lớp dẫn xuất.
- **private:** Các thành phần riêng tư. Các thành phần có thuộc tính truy cập private chỉ được truy cập trong lớp có khai báo chúng mà thôi.

Ví dụ 3:

```
class Car:
    """Đây là lớp Car"""

    def __init__(self):
        self.color = "red"
        self._speed = 200
        self.__seat = 7

    def output(self):
        print(self.color)
        print(self._speed)
        print(self.__seat)

    def update_in(self):
        self._speed = 300
        self.__seat = 9

ob = Car()

def update_out():
    ob._speed = 250
    ob.__seat = 4

ob.output()

#=> Kết quả hiển thị là:
red
```

200

7

ob.update_in()

ob.output()

#=> Kết quả hiển thị là:

red

300**9**

update_out()

ob.output()

#=> Kết quả hiển thị là:

red

250

9

2.1. Thuộc tính

Thuộc tính trong lớp dùng để chứa dữ liệu cho đối tượng thuộc lớp. Trong Python có hai loại thuộc tính là thuộc tính của lớp và thuộc tính của đối tượng.

- Thuộc tính của lớp sẽ có dữ liệu giống nhau cho tất cả các đối tượng thuộc lớp đó và nó thường được khai báo ngay sau dòng lệnh khai báo tên lớp.
- Thuộc tính của đối tượng sẽ nhận dữ liệu khác nhau trên từng đối tượng thuộc lớp đó và thường được khai báo trong phương thức khởi tạo (**`__init__`**) của lớp.

2.1.1. Khai báo

```
class <Tên lớp>:
```

```
    [<Thuộc tính 1 của lớp>
```

```
    <Thuộc tính 2 của lớp>
```

```
    ...
```

```
    <Thuộc tính n của lớp>]
```

```
def __init__(self[, TSHT_1, TSHT_2, ..., TSHT_n])
```

```
    self.<Thuộc tính 1 của đối tượng> = TSHT_1
```

```
    self.<Thuộc tính 2 của đối tượng> = TSHT_2
```

```
    ...
```

```
    self.<Thuộc tính n của đối tượng> = TSHT_n
```

2.1.2. Sử dụng thuộc tính

- Nếu là thuộc tính của lớp thì ta có thể truy xuất bằng 1 trong 2 cách sau:

Cách 1: <Tên lớp>.<Thuộc tính của lớp>

Cách 2: <Tên đối tượng thuộc lớp>.<Thuộc tính của lớp>

- Nếu thuộc tính của đối tượng thì ta truy xuất bằng cách sau:

<Tên đối tượng thuộc lớp>.<Thuộc tính của đối tượng>

Trong các phương thức của một lớp luôn có tham số đầu tiên có kiểu chính là lớp đó và tham số này mặc định là **self**, tuy nhiên ta vẫn có thể đặt một tên khác.

2.1.3. Ví dụ về thuộc tính

```
class Car:
    """Đây là lớp Car"""
    brand = "Toyota"

    def __init__(self, color, speed, seat):
        self.color = color
        self.speed = speed
        self.seat = seat

    def output(self):
        print(Car.brand)
        print(self.color)
        print(self.speed)
        print(self.seat)

ct1 = Car("red", 200, 7)
ct2 = Car("blue", 250, 9)
ct3 = Car("white", 300, 4)
ct1.output()
#=> Kết quả hiển thị là:
Toyota
red
200
7
ct2.output()
#=> Kết quả hiển thị là:
Toyota
blue
```

250

9

ct3.output()

#=> Kết quả hiển thị là:

Toyota

white

300

4

Lưu ý: Phương thức output ở ví dụ trên có thể được viết bằng cách khác như sau:

```
def output(ob):
    print(ob.brand)
    print(ob.color)
    print(ob.speed)
    print(ob.seat)
```

2.2. Phương thức

Phương thức trong lớp tương tự như hàm mà chúng ta xây dựng trong chương trình. Tuy nhiên phương thức phải gắn liền với lớp khai báo nó và được gọi bởi các đối tượng thuộc lớp đó. Trong các phương thức của một lớp luôn có tham số đầu tiên có kiểu chính là lớp đó và tham số này có tên mặc định là **self**, tuy nhiên ta vẫn có thể đặt một tên khác. Như vậy phương thức trong lớp phải có ít nhất **một** tham số còn đối với hàm là không.

2.2.1. Khai báo

```
def <Tên_phương_thức> (self [, Danh sách các TSHT]):
    <khởi_lệnh>
    [return <biểu_thức>]
```

Trong đó:

- **def** là từ khóa dùng để khai báo một phương thức hay hàm.
- **Tên_phương_thức** sẽ do người dùng tự đặt tên, tuân theo quy tắc của Python.
- **Self** chính là tham số đầu tiên và bắt buộc phải có trong phương thức, nó có kiểu chính là lớp đó.
- **Danh sách các TSHT** là các tham số đầu vào của phương thức được biểu diễn bằng tên tương ứng, nếu có nhiều tham số thì chúng được phân cách bởi dấu phẩy.
- Nếu **phương thức có giá trị trả về** thì có lệnh return, ngược lại thì không cần.

2.2.2. Sử dụng phương thức

- Nếu phương thức được gọi bên trong lớp:
<Tên tham số đầu tiên>.<Tên phương thức>([Danh sách các TSTT])

- Nếu phương thức được gọi bên ngoài lớp:

<Tên đối tượng thuộc lớp>.<Tên phương thức>([Danh sách các TSTT])

2.2.3. Ví dụ về phương thức

```
class Car:
    """Đây là lớp Car"""
    brand = "Toyota"

    def __init__(self, color, speed, seat):
        self.color = color
        self.speed = speed
        self.seat = seat

    def output(self):
        print(Car.brand)
        print(self.color)
        print(self.speed)
        print(self.seat)

    def info(self):
        print("Thông tin về chiếc xe là:")
        self.output()
```

```
ct1 = Car("red", 200, 7)
```

```
ct1.info()
```

#=> Kết quả hiển thị là:

Thông tin về chiếc xe là:

Toyota

red

200

7

3. CÁC THAO TÁC CƠ BẢN

3.1. Tạo đối tượng thuộc lớp

3.1.1. Cách tạo

Khi đã xây dựng lớp, chúng ta có thể tạo ra nhiều đối tượng (instances) thuộc lớp đó. Để tạo đối tượng thuộc lớp ta thực hiện theo cú pháp:

<Tên đối tượng> = <Tên lớp>([Giá_trị_1, Giá_trị_2, ..., Giá_trị_n])

Tùy theo trong hàm tạo (`__init__`) có bao nhiêu tham số thì ta phải truyền bấy nhiêu giá trị cho đối tượng được tạo ra, ngoại trừ tham số đầu tiên (*self*) và các tham số mặc định. Cú pháp trên tương tự như lời gọi hàm với tên hàm bây giờ chính là tên lớp.

Ta thấy rằng khi định nghĩa các phương thức trong lớp, ta có tham số là **self**, nhưng khi gọi phương thức ta không cần truyền tham số này mà vẫn không bị báo lỗi. Bởi vì, bất cứ khi nào có một đối tượng gọi đến phương thức thì đối tượng này sẽ tự động **pass qua** tham số đầu tiên này.

3.1.2. Ví dụ về đối tượng

```
class Car:
    """Đây là lớp Car"""
    brand = "Toyota"
    def __init__(self, color, speed, seat=9):
        self.color = color
        self.speed = speed
        self.seat = seat

    def output(self):
        print(Car.brand)
        print(self.color)
        print(self.speed)
        print(self.seat)

    def info(self):
        print("Thông tin về chiếc xe là:")
        self.output()

ct1 = Car("red", 200, 7)
ct1.info()

#=> Kết quả hiển thị là:
Thông tin về chiếc xe là:
Toyota
red
200
7

ct2 = Car("blue", 250)
ct2.info()

#=> Kết quả hiển thị là:
Thông tin về chiếc xe là:
Toyota
blue
250
9
```

3.2. Xây dựng hàm tạo cho lớp

3.2.1. Cách xây dựng

Các phương thức/hàm trong lớp được bắt đầu và kết thúc bởi 2 dấu gạch dưới (__) là các hàm đặc biệt, mang các ý nghĩa đặc biệt. Và một trong số đó là hàm `__init__()`, nó chính là hàm tạo (constructor) của lớp. Hàm này được gọi bất cứ khi nào khởi tạo một đối tượng thuộc lớp.

Để xây dựng một hàm tạo cho lớp ta thực hiện theo cú pháp:

```
def __init__(self [, TSHT_1, TSHT_2, ..., TSHT_n]):  
    self.<Thuộc_tính_1> = TSHT_1  
    self.<Thuộc_tính_2> = TSHT_2  
    ...  
    self.<Thuộc_tính_n> = TSHT_n
```

3.2.2. Ví dụ về hàm tạo

```
class Car:  
    """Đây là lớp Car"""  
    brand = "Toyota"  
    def __init__(self, color, speed, seat=9):  
        self.color = color  
        self.speed = speed  
        self.seat = seat  
  
    def output(self):  
        print(Car.brand)  
        print(self.color)  
        print(self.speed)  
        print(self.seat)  
  
    def info(self):  
        print("Thông tin về chiếc xe là:")  
        self.output()  
  
ct1 = Car("red", 200, 7)  
ct1.info()  
  
ct2 = Car("blue", 250)  
ct2.info()
```

3.3. Thêm thuộc tính cho đối tượng

3.3.1. Cách thực hiện

Ta có thể bổ sung thêm thuộc tính mới cho một đối tượng bất kỳ khi nó đã được tạo ra. Tuy nhiên thuộc tính này chỉ là thuộc tính của riêng đối tượng cụ thể đó mà thôi.

Để bổ sung thêm thuộc tính cho đối tượng ta thực hiện theo cú pháp:

<Tên đối tượng>.<Tên thuộc tính> = <Giá trị>

3.3.2. Ví dụ về thêm thuộc tính cho đối tượng

Ở ví dụ 3.2.2, bây giờ ta bổ sung thêm 3 dòng sau:

```
ct2.year_of_manufacture = 2020
print(ct2.year_of_manufacture)
print(ct1.year_of_manufacture)
```

Khi đó thì đối tượng **ct2** sẽ có thêm thuộc tính **year_of_manufacture** có giá trị là **2020**. Nhưng câu lệnh `print(ct1.year_of_manufacture)` sẽ bị báo lỗi vì đối tượng **ct1 không có** thuộc tính **year_of_manufacture**.

3.4. Xóa thuộc tính, phương thức và đối tượng

3.4.1. Cách thực hiện

Trong quá trình làm việc nếu ta thấy không còn sử dụng thuộc tính, phương thức hay đối tượng nào đó thì ta có thể thực hiện thao tác xóa chúng.

- Để xóa thuộc tính của lớp ta thực hiện theo cú pháp:
del <Tên lớp>.<Tên thuộc tính>
- Để xóa thuộc tính của đối tượng ta thực hiện theo cú pháp:
del <Tên đối tượng>.<Tên thuộc tính>
- Để xóa phương thức của lớp ta thực hiện theo cú pháp:
del <Tên lớp>.<Tên phương thức>
- Để xóa đối tượng ta thực hiện theo cú pháp:
del <Tên đối tượng>

3.4.2 Ví dụ về xóa thuộc tính, phương thức và đối tượng

Ở ví dụ 3.2.2, bây giờ ta bổ sung thêm các dòng sau:

```
del Car.brand
ct3 = Car("white", 300, 4)
print(ct3.seat)
#=> 4
print(ct3.brand)
#=> Báo lỗi AttributeError: 'Car' object has no attribute 'brand'
del ct2.seat
print(ct2.seat)
#=> Báo lỗi AttributeError: 'Car' object has no attribute 'seat'
```

```
print(ct2.color)
#=> blue
del Car.info
ct1.info()
#=> AttributeError: 'Car' object has no attribute 'info'
del ct2
print(ct2.color)
#=> Báo lỗi NameError: name 'ct2' is not defined
```

Lưu ý:

Sau khi bị xóa bằng lệnh del thì đối tượng vẫn còn tồn tại trong bộ nhớ cho đến khi phương thức/hàm hủy (destructor) hay còn gọi là garbage collection được thực hiện thì đối tượng mới bị loại bỏ hoàn toàn các dữ liệu của chúng. Thông thường thì hàm hủy sẽ được thực hiện một cách tự động ngay sau khi lệnh xóa đối tượng được gọi.

4. CÁC CHƯƠNG TRÌNH ỨNG DỤNG

4.1. Chương trình 1

Xây dựng lớp **Tọa độ** gồm có 2 thuộc tính của đối tượng là x (hoành độ), y (tung độ) và 6 phương thức: khởi tạo đối tượng, nhập 2 thuộc tính cho đối tượng, xuất tọa độ của đối tượng theo dạng (*hoành độ*, *tung độ*), trả về giá trị hoành độ của đối tượng, trả về giá trị tung độ của đối tượng, tính khoảng cách giữa 2 đối tượng. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Tọa độ**, sử dụng các phương thức trên kiểm tra xem 3 điểm bất kỳ có tạo thành một tam giác hay không.

```
import math

class Toado:
    '''Lớp tọa độ gồm 2 thuộc tính và 6 phương thức'''
    def __init__(self, hd = 0, td = 0):
        self.x = hd
        self.y = td

    def Nhap(self):
        self.x = float(input("Nhập hoành độ: "))
        self.y = float(input("Nhập tung độ: "))

    def Xuat(self):
        print('(' , self.x , ',' , self.y , ')')

    def Hoanhdo(self):
        return self.x

    def Tungdo(self):
        return self.y
```

```
def Khoangcach(self, ob):
    kc = math.sqrt(pow(ob.x-self.x,2)+pow(ob.y-self.y,2))
    return kc

A = Toado(2, 3)
print('Điểm A có tọa độ là:')
A.Xuat()

B = Toado()
print('Điểm B có tọa độ là:')
B.Xuat()

print('Nhập tọa độ cho A:')
A.Nhap()
print('Điểm A có tọa độ là:')
A.Xuat()

print('Nhập tọa độ cho B:')
B.Nhap()
print('Điểm B có tọa độ là:')
B.Xuat()

C = Toado()
print('Nhập tọa độ cho C:')
C.Nhap()

print("Hoành độ của C là:",C.Hoanhdo())
print("Tung độ của C là:",C.Tungdo())

kcAB = A.Khoangcach(B)
print('Khoảng cách từ A đến B là:',kcAB)

kcAC = A.Khoangcach(C)
print('Khoảng cách từ A đến C là:',kcAC)

kcBC = B.Khoangcach(C)
print('Khoảng cách từ B đến C là:',kcBC)

if(kcAB+kcAC > kcBC and kcAB+kcBC > kcAC and kcAC+kcBC > kcAB):
    print('3 điểm A, B,C tạo thành một tam giác')
else:
    print('3 điểm A, B,C không tạo thành tam giác')
```

4.2. Chương trình 2

Xây dựng lớp **Con người** gồm có 2 thuộc tính lớp là đáng đi (thẳng), trí khôn (có), 5 thuộc tính của đối tượng là màu da, màu mắt, nhóm máu, quốc tịch, ngày sinh và các phương thức: khởi tạo đối tượng, nhập các thuộc tính cho đối tượng, xuất thông tin của đối tượng (đầy đủ 7 thuộc tính), trả về thông tin nhóm máu của đối tượng, trả về thông tin quốc tịch của đối tượng, trả về thông tin năm sinh của đối tượng, so sánh ngày sinh của 2 người bất kỳ. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Con người**, sử dụng các phương thức trên cho biết tuổi của một người nào đó.

```
from datetime import date

class Connguoi:
    '''Nội dung lớp con người'''

    dangdi = 'thẳng'
    trikhon = 'có'

    def __init__(self, md='vàng', mm='đen', nm='AB',
                  qt='Việt Nam', ns=date(1975, 6, 22)):
        self.mauda = md
        self.maumat = mm
        self.nhommau = nm
        self.quoctich = qt
        self.ngaysinh = ns

    def Nhap(self):
        self.mauda = input("Nhập màu da:")
        self.maumat = input("Nhập màu mắt:")
        self.nhommau = input("Nhập nhóm máu:")
        self.quoctich = input("Nhập quốc tịch:")
        ngay = int(input("Nhập ngày sinh:"))
        thang = int(input("Nhập tháng sinh:"))
        nam = int(input("Nhập năm sinh:"))
        self.ngaysinh = date(nam, thang, ngay)

    def Xuat(self):
        print('Đáng đi:', self.dangdi)
        print('Trí khôn:', self.trikhon)
        print('Màu da:', self.mauda)
        print('Màu mắt:', self.maumat)
        print('Nhóm máu:', self.nhommau)
        print('Quốc tịch:', self.quoctich)
        print('Ngày sinh:', self.ngaysinh.strftime("%d/%m/%Y"))
```

```
def Nhommau(self):
    return self.nhommau

def Quoctich(self):
    return self.quoctich

def Namsinh(self):
    return self.ngaysinh.year

def Sosanh(self, Person):
    if self.ngaysinh.year < Person.ngaysinh.year:
        return 1
    elif self.ngaysinh.year > Person.ngaysinh.year:
        return -1
    elif self.ngaysinh.month < Person.ngaysinh.month:
        return 1
    elif self.ngaysinh.month > Person.ngaysinh.month:
        return -1
    elif self.ngaysinh.day < Person.ngaysinh.day:
        return 1
    elif self.ngaysinh.day > Person.ngaysinh.day:
        return -1
    else:
        return 0

Person1 = Connguoii()
print('Thông tin về Person1:')
Person1.Xuat()

Person2 = Connguoii()
print('Nhập thông tin cho Person2:')
Person2.Nhap()
print('Thông tin về Person2:')
Person2.Xuat()

print('Nhóm máu của Person1 là:', Person1.Nhommau())
print('Quốc tịch của Person2 là:', Person2.Quoctich())
print('Tuổi của Person1 là:', date.today().year - Person1.Namsinh())

if Person1.Sosanh(Person2)==1:
    print('Person1 lớn hơn Person2')
elif Person1.Sosanh(Person2)==-1:
    print('Person1 nhỏ hơn Person2')
else:
    print('Person1 bằng Person2')
```

BÀI TẬP CHƯƠNG 7



1. Xây dựng lớp **Fraction** gồm có 2 thuộc tính của đối tượng là **tử số** và **mẫu số**, các phương thức: khởi tạo đối tượng, nhập 2 thuộc tính cho đối tượng, nếu mẫu số bằng 0 phải yêu cầu nhập lại, in phân số theo dạng *tử số/mẫu số* hoặc *-tử số/mẫu số* (không có dạng *tử số/-mẫu số*) và nếu tử số bằng 0 thì chỉ in 0 (không có dạng *0/mẫu số*), còn mẫu số bằng 1 thì chỉ in tử số (không có dạng *tử số/1*), trả về giá trị tử số của đối tượng, trả về giá trị mẫu số của đối tượng, trả về phân số nghịch đảo, tính giá trị thực của phân số (chẳng hạn phân số là $1/2$ thì giá trị thực là 0.5), tối giản một phân số (chẳng hạn phân số là $6/9$ được tối giản thành $2/3$), so sánh 2 phân số và các phương thức cộng, trừ, nhân, chia 2 phân số. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Fraction**.

2. Xây dựng lớp **Date** gồm có 3 thuộc tính của đối tượng là năm, tháng, ngày và các phương thức: khởi tạo đối tượng, nhập 3 thuộc tính cho đối tượng, nếu giá trị năm (1900 -> năm hiện tại), tháng (1 -> 12), ngày (1 -> 28/29/30/31) không hợp lệ phải yêu cầu nhập lại, in ngày theo dạng *dd/mm/yyyy*, trả về số năm của đối tượng, trả về số tháng của đối tượng, trả về số ngày của đối tượng, cộng thêm n ngày vào đối tượng và phải bảo đảm 3 thuộc tính năm, tháng, ngày luôn hợp lệ (chẳng hạn $22/10/2020 + 15 = 06/11/2020$), trừ ra n ngày khỏi đối tượng và bảo đảm 3 thuộc tính năm, tháng, ngày luôn hợp lệ (chẳng hạn $22/10/2020 - 25 = 27/09/2020$). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Date**.

3. Xây dựng lớp **Clock** gồm có 3 thuộc tính của đối tượng là giờ, phút, giây và các phương thức: khởi tạo đối tượng, nhập 3 thuộc tính cho đối tượng, nếu giá trị giờ (0 -> 23), phút (0 -> 59), giây (0 -> 59) không hợp lệ phải yêu cầu nhập lại, in thời gian theo dạng (*giờ : phút : giây*), trả về số giờ của đối tượng, trả về số phút của đối tượng, trả về số giây của đối tượng, cộng thêm 1 giây vào đối tượng và bảo đảm 3 thuộc tính giờ, phút, giây luôn hợp lệ (chẳng hạn hiện tại là (10 : 32 : 59) thì sau khi cộng là (10 : 33 : 00)), trừ ra 1 giây khỏi đối tượng và bảo đảm 3 thuộc tính giờ, phút, giây luôn hợp lệ (chẳng hạn hiện tại là (11 : 00 : 00) thì sau khi trừ là (10 : 59 : 59)). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Clock**.

4. Xây dựng lớp **Document** gồm có 1 thuộc tính lớp là trường ('ĐH.SPKT Vĩnh Long'), 4 thuộc tính của đối tượng là tên tài liệu, tên người chủ biên, năm xuất bản, số trang và các phương thức: khởi tạo đối tượng, nhập 4 thuộc tính cho đối tượng (nếu năm > năm hiện tại thì phải yêu cầu nhập lại, nếu số trang < 10 và > 500 thì phải yêu cầu nhập lại), xuất thông tin của đối tượng (đủ 5 thuộc tính), trả về tên của tài liệu, trả về tên người chủ biên, trả về năm xuất bản, trả về số trang của tài liệu, so sánh số trang của 2 tài liệu (nếu số trang tài liệu 1 > số trang tài liệu 2 thì trả về 1, nếu số trang tài liệu 1 < số trang tài liệu 2 thì trả về -1, ngược lại trả về 0). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên lớp **Document**. Tạo một danh sách gồm N tài liệu thuộc lớp **Document** ($N > 0$). In danh sách tài liệu vừa tạo. Đếm số tài liệu có số trang trên 300. Cho biết tên chủ biên có nhiều tài liệu nhất. Sắp xếp danh sách tài liệu tăng dần theo năm xuất bản và in ra danh sách sau khi sắp xếp.

Chương 8: KỸ THUẬT KẾ THỪA



1. GIỚI THIỆU CHUNG

1.1. Kế thừa là gì?

Kế thừa hay **thừa kế (inheritance)** là một trong các nguyên tắc cơ bản của lập trình hướng đối tượng. Đặc biệt đây là cơ sở cho việc nâng cao khả năng sử dụng lại code của chương trình.

Kế thừa cho phép một lớp (class) có thể thừa hưởng (sử dụng) các thuộc tính và phương thức từ các lớp khác đã được định nghĩa. Lớp đã có được gọi là lớp cha (parent class/super class) hay lớp cơ sở (base class), lớp mới phát sinh được gọi là lớp con (child class/sub class) hoặc lớp dẫn xuất (derived class).

Lớp con kế thừa tất cả thành phần (dữ liệu – thuộc tính, hàm – phương thức) của lớp cha và có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới, bao hàm cả việc làm “*tốt hơn*” hoặc làm lại những công việc mà trong lớp cơ sở chưa làm tốt hoặc không còn phù hợp với lớp dẫn xuất.

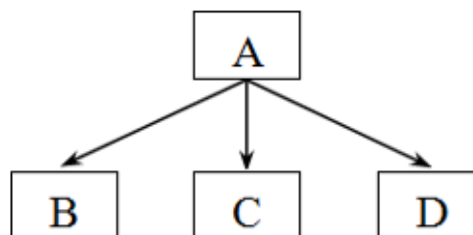
Lớp nào cũng có thể là một lớp cơ sở của lớp khác. Hơn thế nữa, một lớp có thể là cơ sở cho nhiều lớp dẫn xuất khác nhau. Bên cạnh đó, lớp dẫn xuất lại có thể dùng làm cơ sở để xây dựng các lớp dẫn xuất khác, được gọi là kế thừa đa cấp (multilevel inheritance). Ngoài ra, một lớp có thể kế thừa từ nhiều lớp cơ sở, được gọi là kế thừa bội hay đa kế thừa (multiple inheritance).

1.2. Một số dạng kế thừa thông dụng

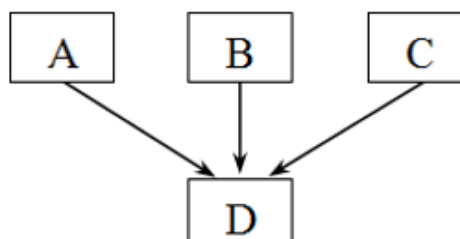
- **Dạng 1:** Lớp B dẫn xuất từ lớp A, lớp C dẫn xuất từ lớp B



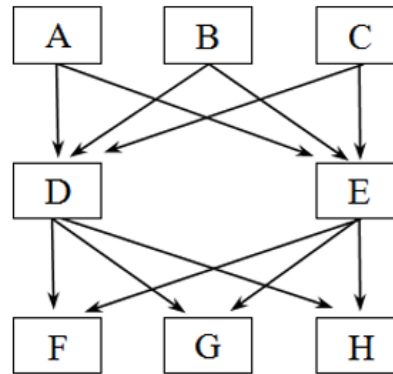
- **Dạng 2:** Lớp A là cơ sở của các lớp B, C và D



- **Dạng 3:** Lớp D dẫn xuất từ 3 lớp A, B, C



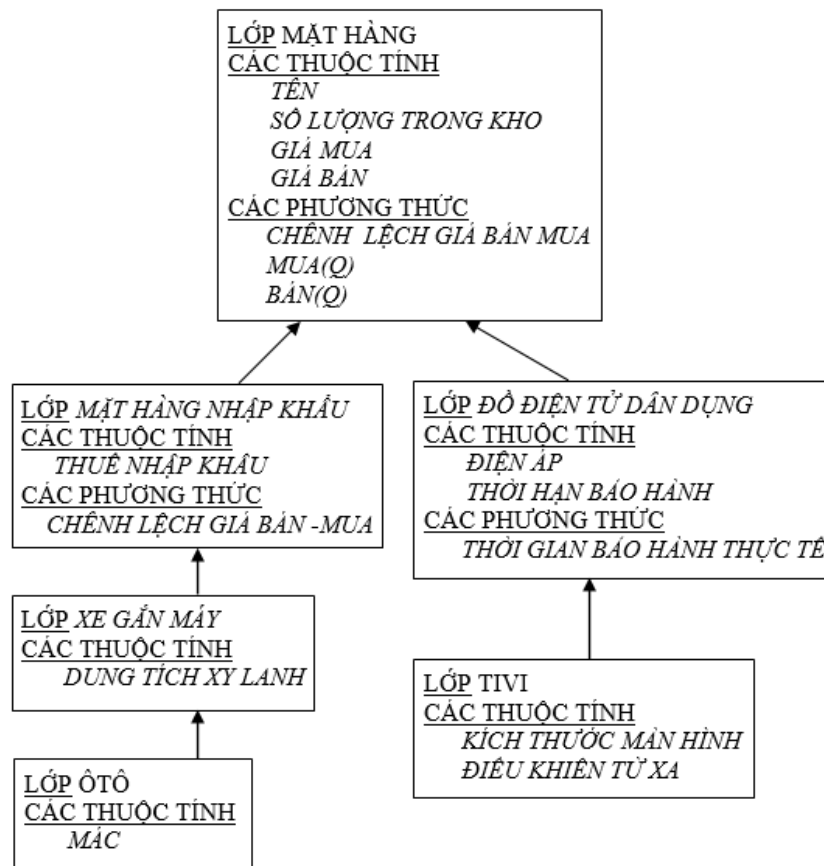
• **Dạng 4: Sơ đồ tổng quát**



Kế thừa cũng cho phép nhiều lớp có thể dẫn xuất từ cùng một lớp cơ sở, và không chỉ giới hạn ở một mức: một lớp dẫn xuất có thể là lớp cơ sở cho các lớp dẫn xuất khác. Ở đây ta thấy rằng khái niệm kế thừa giống như công cụ cho phép mô tả cụ thể các khái niệm theo nghĩa: lớp dẫn xuất là một cụ thể hóa hơn nữa của lớp cơ sở và nếu bỏ đi các dị biệt trong các lớp dẫn xuất sẽ chỉ còn các đặc điểm chung nằm trong lớp cơ sở.

Các NNLT hướng đối tượng đều cho phép đa kế thừa, theo đó một lớp có thể là dẫn xuất của nhiều lớp khác. Do vậy dẫn tới khả năng một lớp cơ sở có thể được kế thừa nhiều lần trong một lớp dẫn xuất khác, ta gọi đó là sự xung đột kế thừa. Điều này hoàn toàn không hay, cần phải tránh. Từng NNLT sẽ có những giải pháp cho riêng mình, Python xử lý theo cách đơn giản là sẽ ưu tiên kế thừa các thuộc tính và phương thức của lớp xuất hiện trước trong danh sách kế thừa. Trong chương này chúng ta sẽ nghiên cứu đến các khả năng của Python để thể hiện nguyên tắc kế thừa khi viết chương trình.

1.3. Ví dụ minh họa



2. CÁC LOẠI KẾ THỪA TRONG PYTHON

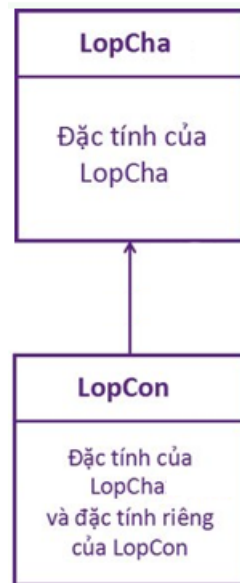
2.1. Kế thừa đơn

2.1.1. Cấu trúc chung

```
class LopCha:
    Body of parent class

class LopCon(LopCha):
    Body of child class
```

2.1.2. Hình ảnh minh họa



2.1.3. Ví dụ về kế thừa đơn

Đa giác là một hình khép kín có 3 cạnh trở lên. Chúng ta xây dựng lớp *DaGiac* gồm 2 thuộc tính: số cạnh (n) và list chứa giá trị các cạnh của một đa giác và các phương thức: khởi tạo, nhập giá trị các cạnh và hiển thị giá trị các cạnh.

```
class DaGiac:
    '''Định nghĩa lớp Đa giác'''

    def __init__(self, socanh):
        self.n = socanh
        self.canh = [0 for i in range(socanh)]

    def nhapcanh(self):
        self.canh = [float(input("Bạn hãy nhập giá trị cạnh "\
                                +str(i+1)+":")) for i in range(self.n)]
```

```
def hienthicanh(self):  
    for i in range(self.n):  
        print("Giá trị cạnh", i+1, "là", self.canh[i])
```

Để kiểm tra tính đúng đắn của lớp *DaGiac* vừa xây dựng ta viết các câu lệnh sau:

```
dg = DaGiac(4)  
dg.nhapcanh()  
dg.hienthicanh()
```

Ta có thể thực thi đoạn chương trình trên và có kết quả là:

```
Bạn hãy nhập giá trị cạnh 1 : 3  
Bạn hãy nhập giá trị cạnh 2 : 6  
Bạn hãy nhập giá trị cạnh 3 : 4  
Bạn hãy nhập giá trị cạnh 4 : 5  
Giá trị cạnh 1 là 3.0  
Giá trị cạnh 2 là 6.0  
Giá trị cạnh 3 là 4.0  
Giá trị cạnh 4 là 5.0
```

Tam giác là một đa giác có 3 cạnh, vì vậy ta sẽ tạo một lớp *TamGiac* kế thừa từ lớp *DaGiac*. Lớp *TamGiac* này sẽ kế thừa tất cả các thuộc tính sẵn có trong lớp *DaGiac* nên ta sẽ không cần khai báo lại (khả năng sử dụng lại code). Như vậy lớp *TamGiac* được định nghĩa như sau:

```
class TamGiac(DaGiac):  
    '''Lớp Tam giác kế thừa lớp Đa giác'''  
  
    def __init__(self):  
        DaGiac.__init__(self, 3)  
  
    def dientich(self):  
        a, b, c = self.canh  
        # Tính nửa chu vi  
        p = (a + b + c) / 2  
        dt = (p * (p - a) * (p - b) * (p - c)) ** 0.5  
        print('Diện tích của hình tam giác là', dt)
```

Lớp *TamGiac* không chỉ kế thừa lớp *DaGiac* mà còn định nghĩa thêm một phương thức mới để tính *dientich* tam giác.

Để kiểm tra tính đúng đắn của lớp *TamGiac* vừa xây dựng ta viết các câu lệnh sau:

```
tg = TamGiac()  
tg.nhapcanh()
```

```
tg.hienthicanh()
```

```
tg.dientich()
```

Ta có thể thực thi đoạn chương trình trên và có kết quả là:

Bạn hãy nhập giá trị cạnh 1 : 4

Bạn hãy nhập giá trị cạnh 2 : 3

Bạn hãy nhập giá trị cạnh 3 : 5

Giá trị cạnh 1 là 4.0

Giá trị cạnh 2 là 3.0

Giá trị cạnh 3 là 5.0

Diện tích của hình tam giác là 6.0

Ta có thể thấy, các hàm *nhapcanh()*, *hienthicanh()* đều không có trong lớp *TamGiac*, nhưng chúng ta vẫn sử dụng được bởi vì chúng đã được định nghĩa trong lớp cha của nó chính là lớp *DaGiac*.

2.2. Kế thừa bội

Giống như nhiều NNLT khác, trong Python một lớp con có thể kế thừa từ nhiều lớp cha. Điều này được gọi là **kế thừa bội** hay **đa kế thừa** (multiple inheritance). Kế thừa bội cũng là kế thừa nhưng ở mức độ sâu sắc hơn và phức tạp hơn. Trong kế thừa bội ta cần phải lưu ý đến thứ tự kế thừa các lớp cha và thứ tự truy xuất phương thức của các lớp cha.

2.2.1. Cấu trúc chung

```
class LopCha_1:
    Body of parent class 1

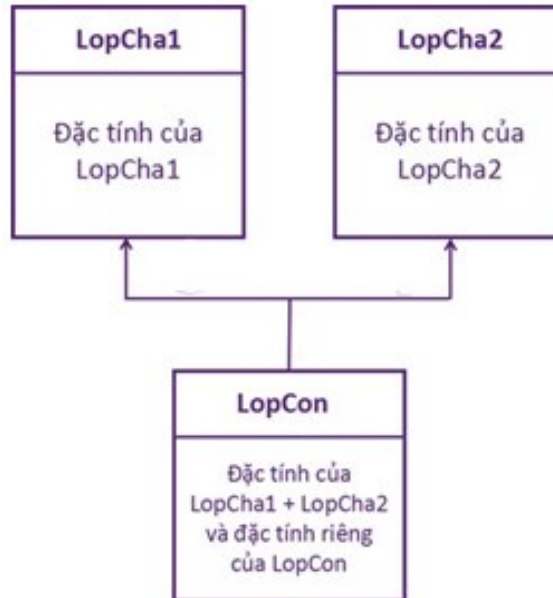
class LopCha_2:
    Body of parent class 2

...

class LopCha_n:
    Body of parent class n

class LopCon(LopCha_1, LopCha_2, ..., LopCha_n):
    Body of child class
```

Theo cấu trúc chung thì lớp con có thể được định nghĩa từ nhiều lớp cha và kế thừa đặc tính của tất cả các lớp này. Các lớp cha có thể có các thuộc tính và các phương thức giống nhau. Khi đó lớp con sẽ ưu tiên kế thừa thuộc tính cũng như phương thức của lớp cha nào được liệt kê trước trong danh sách thừa kế của lớp con.

2.2.2. Hình ảnh minh họa**2.2.3. Ví dụ về kế thừa bội**

Xây dựng 2 lớp Phone và Camera gồm các tính năng cơ bản tương ứng của chúng. Sau đó xây dựng lớp SmartPhone kế thừa cả 2 lớp Phone và Camera đã có. Có một số giá trị của các thuộc tính cần phải thay đổi từ lớp cha để phù hợp có lớp con, bổ sung thêm một số phương thức mới và viết đè một số phương thức đã có từ 2 lớp cha. Chương trình được cài đặt cụ thể như sau:

```

class Phone:
    def __init__(self):
        self.brand = 'Nokia'
        self.version = '105 Single'
        self.price = 370
        self.ram = 4
        self.screen = 1.77
        self.pin = 800
        self.sim = 1

    def use(self):
        print('call, listen, save number, message')

    def sell(self):
        print('Thế giới di động, Điện máy xanh, Cửa hàng điện thoại')

class Camera:
    def __init__(self):
        self.brand = 'Samsung'
  
```

```
        self.version = 'Ultra'
        self.price = 2500
        self.megapixel = 108
        self.size = 1.33
        self.aperture = 'F/1.8'

    def use(self):
        print('take photograph, record')

    def sell(self):
        print('Thế giới di động, Điện máy xanh, Cửa hàng điện tử')

class SmartPhone(Phone, Camera):
    def __init__(self):
        Phone.__init__(self)
        Camera.__init__(self)

    def info_in(self):
        self.price = float(input('Nhập giá:'))
        self.screen = float(input('Nhập kích thước màn hình:'))
        self.sim = int(input('Nhập số sim:'))
        self.megapixel = float(input('Nhập độ phân giải:'))

    def info_out(self):
        print('Thông tin về Smartphone:')
        print('Brand:', self.brand)
        print('Version:', self.version)
        print('Price:', self.price)
        print('Ram:', self.ram)
        print('Screen:', self.screen)
        print('Pin:', self.pin)
        print('Sim:', self.sim)
        print('Megapixel:', self.megapixel)
        print('Size:', self.size)
        print('Aperture:', self.aperture)

    def use_sm(self):
        print('Chức năng chính của Smartphone:')
        Phone.use(self)
        Camera.use(self)
        print('internet, game, ...')
```

```

def sell_sm(self):
    print('Smartphone bán tại:')
    Phone.sell(self)
    print('Cửa hàng điện tử, siêu thị, ...')

sm = SmartPhone()
sm.info_in()
sm.info_out()
sm.use_sm()
sm.sell_sm()

```

2.3. Kế thừa đa cấp

Trong Python, ngoài việc một lớp con có thể kế thừa từ nhiều lớp cha (đa kế thừa), ta còn có thể tạo lớp con mới kế thừa từ các lớp con trước đó. Đây được gọi là **kế thừa đa cấp** (multilevel inheritance).

Trong trường hợp này, các đặc tính của lớp cha và lớp con trước đó đều sẽ được lớp con mới (có thể gọi là lớp cháu) kế thừa.

2.3.1. Cấu trúc chung

```

class LopCha:
    Body of parent class

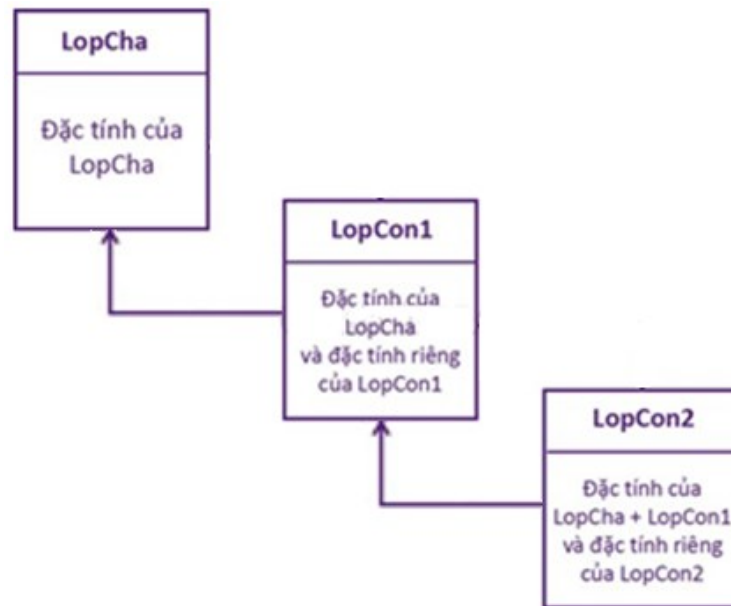
class LopCon_1(LopCha):
    Body of child class 1

class LopCon_2(LopCon_1):
    Body of child class 2
...
class LopCon_n(LopCon_n-1):
    Body of child class n

```

Theo cấu trúc chung thì một lớp có thể là lớp con của lớp này nhưng lại là lớp cha của lớp con khác. Chẳng hạn theo cấu trúc trên thì LopCon_1 là lớp con của LopCha và là lớp cha của LopCon_2. Hay nói cách khác LopCon_1 là lớp dẫn xuất từ LopCha nhưng lại là lớp cơ sở của LopCon_2.

2.3.2. Hình ảnh minh họa



2.3.3. Ví dụ về kế thừa đa cấp

Xây dựng lớp Cat gồm 3 thuộc tính: `mass_in_kg`, `lifespan_in_years`, `speed_in_kph` và các phương thức: `__init__`, `eat`, `vocalize`, `__str__`, `print_cat`. Sau đó xây dựng lớp Tiger kế thừa lớp Cat có bổ sung thêm thuộc tính `coat_pattern` và 2 phương thức `swim`, `print_tiger` và viết đè 2 phương thức `__init__`, `vocalize`. Cuối cùng xây dựng lớp Liger kế thừa lớp Tiger có bổ sung thêm thuộc tính `is_social` và phương thức `print_liger` và viết đè phương thức `__init__`.

```

class Cat:
    '''Lớp Cat là lớp cơ sở'''
    def __init__(self, mass, lifespan, speed):
        self.mass_in_kg = mass
        self.lifespan_in_years = lifespan
        self.speed_in_kph = speed

    def eat(self):
        return 'meat'

    def vocalize(self):
        return 'Chuff'

    def __str__(self):
        st = 'The ' + type(self).__name__.lower() + ' weighs '\
            + str(self.mass_in_kg) + ' kg,'\
            + ' has a lifespan of ' + str(self.lifespan_in_years)\
            + ' years and can run at a maximum speed of '\
            + str(self.speed_in_kph) + ' kph.'
        return st
  
```

```
def print_cat(self):
    print('Infomation of', type(self).__name__.lower())
    print(self.__str__())
    print('The', type(self).__name__.lower(), 'eat', self.eat(),\
          'and has vocalization is', self.vocalize(), '.')

class Tiger(Cat):
    '''Lớp Tiger kế thừa lớp Cat và có mở rộng'''
    def __init__(self, mass=310, lifespan=26, speed=65):
        super().__init__(mass, lifespan, speed)
        self.coat_pattern = 'striped'

    def vocalize(self):
        return 'Roar'

    def swim(self):
        return 'Splash splash'
    def print_tiger(self):
        self.print_cat()
        print('The', type(self).__name__.lower(),\
              'has coat pattern is', self.coat_pattern,\
              'and can swim with', self.swim(), '.')

class Liger(Tiger):
    '''Lớp Liger kế thừa lớp Tiger và có mở rộng'''
    def __init__(self, mass=190, lifespan=14, speed=80):
        super().__init__(mass, lifespan, speed)
        self.is_social = True

    def print_liger(self):
        self.print_tiger()
        print('And social custom of',\
              type(self).__name__.lower(),\
              'is', self.is_social, '.')

c = Cat(4, 18, 48)
c.print_cat()

t = Tiger()
t.print_tiger()

l = Liger()
l.print_liger()
```


3. CÁC THAO TÁC CÓ LIÊN QUAN

3.1. Ghi đè phương thức

3.1.1. Cách thực hiện

Python cho phép NLT ghi đè (overriding) lên các phương thức của lớp cha. Ta có thể thực hiện việc ghi đè phương thức của lớp cha nếu như ta muốn phương thức ở lớp con có tính năng khác biệt hơn hoặc đặc biệt hơn so với chức năng của phương thức này ở lớp cha.

- Ghi đè **phương thức tạo** đối tượng ta thực hiện theo cú pháp:

```
super().__init__([Giá trị_1, Giá trị_2, ... Giá trị_n])
[Các câu lệnh]
```

- Ghi đè **phương thức thường** ta thực hiện theo cú pháp:

```
def <Tên phương thức của lớp cha> ([Danh sách các TSHT]):
    <Các câu lệnh>
    [return <biểu_thức>]
```

3.1.2. Ví dụ ghi đè phương thức

Xây dựng lớp Hình Chữ Nhật có các thuộc tính: số cạnh, danh sách chứa giá trị các cạnh và các phương thức: tạo, nhập cạnh, xuất cạnh và tính diện tích. Sau đó xây dựng lớp Hình Vuông kế thừa lớp Hình Chữ Nhật và ghi đè các phương thức tạo và tính diện tích tương ứng.

```
class HìnhChuNhat:
    '''Định nghĩa lớp Hình chữ nhật'''
    def __init__(self, sc):
        self.n = sc
        self.canh = [0 for i in range(sc)]

    def nhap(self):
        self.canh = [float(input("Nhập giá trị cạnh " \
                                +str(i+1)+":")) for i in range(self.n)]

    def xuất(self):
        for i in range(self.n):
            print("Giá trị cạnh", i+1, "là", self.canh[i])

    def dientich(self):
        cd = self.canh[0]
        cr = self.canh[1]
        dt = cd * cr
        return dt
```

```

class HìnhVuong(HìnhChuNhat):
    '''Lớp Hình vuông kế thừa lớp Hình chữ nhật
    và ghi đè phương thức tạo và tính diện tích'''
    def __init__(self):
        super().__init__(1)

    def dientich(self):
        a = self.canh[0]
        return a * a

print("Làm việc với hình chữ nhật:")
hcn = HìnhChuNhat(2)
hcn.nhap()
hcn.xuat()
print("Diện tích = ", hcn.dientich())
print("Làm việc với hình vuông:")
hv = HìnhVuong()
hv.nhap()
hv.xuat()
print("Diện tích = ", hv.dientich())

```

3.2. Các hàm kiểm tra lớp cơ sở và lớp dẫn xuất

3.2.1. Hàm *issubclass*

Cú pháp:

```
issubclass(Lớp A, Lớp B)
```

Chức năng:

Kiểm tra xem Lớp A có phải là lớp dẫn xuất/lớp con của Lớp B hay không. Nếu có thì hàm trả về True và ngược lại.

Ví dụ:

```

issubclass(TamGiac, DaGiac) # => True
issubclass(DaGiac, TamGiac) # => False
issubclass(HìnhChuNhat, DaGiac) # => False
issubclass(HìnhVuong, HìnhChuNhat) # => True
issubclass(TamGiac, HìnhChuNhat) # => Báo lỗi
issubclass(HìnhVuong, DaGiac) # => Báo lỗi

```

3.2.2. Hàm *isinstance*

Cú pháp:

`isinstance(Đối tượng, Lớp)`

Chức năng:

Kiểm tra xem Đối tượng có phải là một thể hiện (instance) của Lớp hoặc của lớp con của Lớp hay không. Nếu có thì hàm trả về True và ngược lại.

Ví dụ:

```
isinstance(dg, DaGiac) # => True
isinstance(tg, TamGiac) # => True
isinstance(tg, DaGiac) # => True
isinstance(dg, TamGiac) # => False
isinstance(hv, HìnhChuNhat) # => True
isinstance(hcn, HìnhVuong) # => False
isinstance(hcn, TamGiac) # => Báo lỗi
isinstance(hv, DaGiac) # => Báo lỗi
```

3.3. Thứ tự truy xuất phương thức

Lớp được bắt nguồn từ đối tượng. Trong kịch bản đa kế thừa, bất kỳ thuộc tính cần được truy xuất nào thì đầu tiên nó sẽ được tìm kiếm trong lớp hiện tại. Nếu không tìm thấy thì sẽ được tìm kiếm tiếp vào các lớp cha theo thứ tự liệt kê từ trái sang phải.

Như vậy thứ tự truy xuất sẽ là: [*Lớp con (Lớp hiện tại), Lớp cha 1, Lớp cha 2, Đối tượng*]

Thứ tự này còn được gọi là **tuyến tính hóa** của **Lớp con** và tập hợp các quy tắc được sử dụng để tìm thứ tự này được gọi là thứ tự truy xuất phương thức (Method Resolution Order - MRO).

Nói cách khác, MRO dùng để hiển thị list/tuple các lớp cha của một lớp nào đó.

MRO có thể được sử dụng theo hai cách:

- **`mro()`**: trả về một danh sách
- **`__mro__`**: trả về một tuple

Ví dụ:

```
class Cat:
    '''Lớp Cat là lớp cơ sở'''
    def __init__(self, mass, lifespan, speed):
        self.mass_in_kg = mass
        self.lifespan_in_years = lifespan
        self.speed_in_kph = speed
```

```
class Tiger(Cat):
    '''Lớp Tiger kế thừa lớp Cat và có mở rộng'''
    def __init__(self, mass=310, lifespan=26, speed=65):
        super().__init__(mass, lifespan, speed)
        self.coat_pattern = 'striped'

class Liger(Tiger):
    '''Lớp Liger kế thừa lớp Tiger và có mở rộng'''
    def __init__(self, mass=190, lifespan=14, speed=80):
        super().__init__(mass, lifespan, speed)
        self.is_social = True
```

```
print(Liger.mro())
```

#=> Kết quả hiển thị là:

```
[<class '__main__.Liger'>, <class '__main__.Tiger'>,
<class '__main__.Cat'>, <class 'object'>]
```

```
print(Liger.__mro__)
```

#=> Kết quả hiển thị là:

```
(<class '__main__.Liger'>, <class '__main__.Tiger'>,
<class '__main__.Cat'>, <class 'object'>)
```

BÀI TẬP CHƯƠNG 8



1. Xây dựng lớp **Tọa độ** gồm có 2 thuộc tính của đối tượng là x (hoành độ), y (tung độ) và 6 phương thức: khởi tạo đối tượng, nhập 2 thuộc tính cho đối tượng, xuất tọa độ của đối tượng theo dạng (*hoành độ*, *tung độ*), trả về giá trị hoành độ, trả về giá trị tung độ, tính khoảng cách giữa 2 đối tượng. Xây dựng lớp **Tọa độ màu** kế thừa lớp **Tọa độ** và có thêm thuộc tính đối tượng là màu sắc và phương thức trả về giá trị màu cho tọa độ tương ứng, viết đè các phương thức nhập và xuất thông tin. Xây dựng lớp **Hình tròn** kế thừa lớp **Tọa độ màu** với 2 thuộc tính x, y là tọa độ tâm hình tròn và thuộc tính màu sắc là màu của hình tròn, lớp **Hình tròn** có thêm thuộc tính r (bán kính) và phương thức kiểm tra 1 điểm bất kỳ có thuộc hình tròn hay không, viết đè các phương thức nhập và xuất thông tin. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên các lớp.

2. Xây dựng lớp **Document** gồm có 1 thuộc tính lớp là *trường* ('ĐH.SPKT Vĩnh Long'), 4 thuộc tính của đối tượng là tên tài liệu, tên người chủ biên, năm xuất bản, số trang và các phương thức: khởi tạo đối tượng, nhập 4 thuộc tính cho đối tượng (nếu năm > năm hiện tại thì phải yêu cầu nhập lại, nếu số trang < 10 và > 500 thì phải yêu cầu nhập lại), Xuất thông tin của đối tượng (đủ 5 thuộc tính), trả về tên của tài liệu, trả về tên người chủ biên, trả về năm xuất bản, trả về số trang của tài liệu, so sánh số trang của 2 tài liệu (nếu số trang tài liệu 1 > số trang tài liệu 2 thì trả về 1, nếu số trang tài liệu 1 < số trang tài liệu 2 thì trả về -1, ngược lại trả về 0). Xây dựng lớp **Book** kế thừa lớp **Document** và có thêm các thuộc tính chuyên ngành, tên nhà xuất bản, thêm các phương thức trả về chuyên ngành, trả về tên nhà xuất bản, tìm quyền sách dày hơn trong 2 quyền sách bất kỳ. Xây dựng lớp **Magazine** kế thừa lớp **Document** và có thêm các thuộc tính lĩnh vực, ngôn ngữ, thêm các phương thức trả về lĩnh vực, trả về ngôn ngữ, tìm quyền tạp chí mới nhất trong 3 quyền tạp chí bất kỳ. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên các lớp. Tạo một danh sách gồm N quyền sách thuộc lớp **Book** ($0 < N < 100$). In các quyền sách đã nhập. Đếm số sách được xuất bản bởi nhà xuất bản X. Cho biết các quyền sách đã nhập thuộc bao nhiêu chuyên ngành khác nhau. Sắp xếp các quyền sách tăng dần theo số trang và in ra danh sách sau khi sắp xếp.

3. Xây dựng lớp **Animal** gồm có các thuộc tính đối tượng là cân nặng, tuổi thọ, số chân, nơi sinh sống, tiếng kêu, phương thức nhập thông tin, phương thức in thông tin, các phương thức trả về giá trị các thuộc tính tương ứng. Xây dựng lớp **Predator** kế thừa lớp **Animal** và có thêm thuộc tính tốc độ chạy, phương thức trả về tốc độ chạy, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Lion** kế thừa lớp **Predator** và có thêm thuộc tính có bờm, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Tiger** kế thừa lớp **Predator** và có thêm thuộc tính mầu lông, phương thức trả về mầu lông, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Tigon** kế thừa cả 2 lớp **Tiger** và **Lion** và có thêm thuộc tính biết bơi, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Herbivore** kế thừa lớp **Animal** và có thêm 2 thuộc tính là có nhai lại, lượng cỏ ăn mỗi ngày, phương thức trả về lượng cỏ ăn mỗi ngày, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Horse** kế thừa lớp **Herbivore** và có thêm thuộc tính mầu

lông, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Donkey** kế thừa lớp **Herbivore** và có thêm thuộc tính tốc độ di chuyển, phương thức trả về tốc độ di chuyển, ghi đè các phương thức nhập và in thông tin. Xây dựng lớp **Mule** kế thừa cả 2 lớp **Horse** và **Donkey** và có thêm 2 thuộc tính lớp là *số nhiễm sắc thể* (63) và *khả năng sinh con* (False), thêm thuộc tính đối tượng độ nhạy cảm của da, ghi đè các phương thức nhập và in thông tin. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên các lớp.

4. Xây dựng lớp **Person** gồm có 1 thuộc tính lớp là quốc tịch = “Việt Nam”, 4 thuộc tính của đối tượng là họ tên, ngày sinh, giới tính, quê quán và các phương thức: khởi tạo đối tượng, nhập các thuộc tính cho đối tượng, xuất thông tin của đối tượng (đầy đủ 5 thuộc tính), trả về thông tin họ tên, trả về thông tin ngày sinh, trả về thông tin giới tính, trả về thông tin quê quán. Xây dựng lớp **Staff** kế thừa lớp **Person** và có thêm các thuộc tính ngày bắt đầu làm việc, chuyên môn, hệ số lương, loại nhân viên, thêm các phương thức trả về thông tin ngày bắt đầu làm việc, trả về thông tin hệ số lương, viết đè các phương thức nhập và xuất thông tin. Xây dựng lớp **Lecturer** kế thừa lớp **Person** và có thêm các thuộc tính ngày bắt đầu làm việc, chuyên môn, trình độ, hệ số lương, hạng giảng viên, thêm các phương thức trả về thông tin ngày bắt đầu làm việc, trả về thông tin trình độ, trả về thông tin hệ số lương, viết đè các phương thức nhập và xuất thông tin. Xây dựng lớp **IT_Officers** kế thừa cả 2 lớp **Staff** và **Lecturer** có thêm thuộc tính lớp là *khoa* = ‘CNTT’ và thuộc tính đối tượng là loại cán bộ, viết đè phương thức để nhập và xuất thông tin tương ứng cho cán bộ là Staff hoặc Lecturer, thêm các phương thức tính số tuổi cho cán bộ, tính số năm công tác cho cán bộ, tính lương cho cán bộ theo công thức: $\text{lương} = \text{hệ số lương} * 1.490.000$. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo trên các lớp. Tạo một danh sách gồm N nhân viên thuộc lớp **IT_Officers** (N không âm và không quá 50). In danh sách nhân viên vừa tạo. Đếm số nhân viên có thâm niên trên 10 năm. Cho biết họ, tên của nhân viên có tuổi đời cao nhất. Sắp xếp danh sách nhân viên giảm dần theo lương và in ra danh sách nhân viên sau khi sắp xếp.

Chương 9: QUÁ TẢI TOÁN TỬ



1. GIỚI THIỆU CHUNG

1.1. Quá tải toán tử là gì?

Quá tải toán tử (Operator overload) hay còn được gọi là **nạp chồng toán tử**, nhằm mục đích thay đổi ý nghĩa của toán tử tùy thuộc vào toán hạng được sử dụng.

Trong Python, toán tử làm việc bằng các hàm đã được dựng sẵn, nhưng một toán tử có thể được sử dụng để thực hiện nhiều hoạt động khác nhau. Ví dụ với toán tử '+', ta có thể cộng số học hai số với nhau, có thể kết hợp hai danh sách, hoặc nối hai chuỗi khác nhau lại, ...

Bây giờ ta lại muốn toán tử '+' thực hiện trên 2 toán hạng kiểu phân số hay tọa độ, ... Tính năng này trong Python gọi là quá tải toán tử hay nạp chồng toán tử, nó cho phép cùng một toán tử được sử dụng trên nhiều kiểu toán hạng khác nhau tùy vào từng ngữ cảnh.

1.2. Một số loại toán tử thông dụng

Khi xây dựng một lớp, để các toán tử có thể thực hiện trên các đối tượng thuộc lớp đó thì NLT cần phải quá tải các toán tử này. Có 3 loại toán tử thường được quá tải đó là: toán tử số học (+, -, *, /, ...), toán tử luận lý (and, or, not, ...) và toán tử so sánh (>, >=, ==, !=, ...).

Phương pháp chung để quá tải một toán tử là ta định nghĩa lại tên hàm tương ứng cho toán tử cần quá tải trong lớp mà ta xây dựng.

Trong Python mỗi toán tử là một hàm đặc biệt, được bắt đầu và kết thúc bởi 2 dấu gạch dưới (__) và có một tên tương ứng. Chẳng hạn toán tử '+' có tên hàm tương ứng là `__add__`, toán tử 'and' có tên hàm tương ứng là `__and__` và toán tử '>' có tên hàm tương ứng là `__gt__`.

1.3. Tên hàm tương ứng của các toán tử trong Python

1.3.1. Toán tử số học

TOÁN TỬ	BIỂU DIỄN	TÊN HÀM	HOẠT ĐỘNG
Cộng	ob1 + ob2	<code>__add__</code>	ob1. <code>__add__</code> (ob2)
Trừ	ob1 - ob2	<code>__sub__</code>	ob1. <code>__sub__</code> (ob2)
Nhân	ob1 * ob2	<code>__mul__</code>	ob1. <code>__mul__</code> (ob2)
Lũy thừa	ob1 ** ob2	<code>__pow__</code>	ob1. <code>__pow__</code> (ob2)
Chia	ob1 / ob2	<code>__truediv__</code>	ob1. <code>__truediv__</code> (ob2)

Chia lấy phần nguyên (Floor Division)	ob1 // ob2	<code>__floordiv__</code>	ob1. <code>__floordiv__</code> (ob2)
Chia lấy phần dư (Modulo)	ob1 % ob2	<code>__mod__</code>	ob1. <code>__mod__</code> (ob2)

1.3.2. Toán tử luận lý

TOÁN TỬ	BIỂU DIỄN	TÊN HÀM	HOẠT ĐỘNG
Và	ob1 and ob2	<code>__and__</code>	ob1. <code>__and__</code> (ob2)
Hoặc	ob1 or ob2	<code>__or__</code>	ob1. <code>__or__</code> (ob2)
Hoặc loại trừ	ob1 xor ob2	<code>__xor__</code>	ob1. <code>__xor__</code> (ob2)
Phủ định	not ob1	<code>__invert__</code>	ob1. <code>__invert__</code> ()

1.3.3. Toán tử so sánh

TOÁN TỬ	BIỂU DIỄN	TÊN HÀM	HOẠT ĐỘNG
Nhỏ hơn	ob1 < ob2	<code>__lt__</code>	ob1. <code>__lt__</code> (ob2)
Nhỏ hơn hoặc bằng	ob1 <= ob2	<code>__le__</code>	ob1. <code>__le__</code> (ob2)
Bằng	ob1 == ob2	<code>__eq__</code>	ob1. <code>__eq__</code> (ob2)
Khác	ob1 != ob2	<code>__ne__</code>	ob1. <code>__ne__</code> (ob2)
Lớn hơn	ob1 > ob2	<code>__gt__</code>	ob1. <code>__gt__</code> (ob2)
Lớn hơn hoặc bằng	ob1 >= ob2	<code>__ge__</code>	ob1. <code>__ge__</code> (ob2)

2. CẤU TRÚC CHUNG CỦA MỘT HÀM QUÁ TẢI TOÁN TỬ

2.1. Đối với các toán tử hai ngôi

- **Dạng 1:**

```
def <Tên hàm>(self, ob2):
```

```
    Các câu lệnh
```

```
    return <Giá trị trả về>
```


- **Dạng 2:**

```
def <Tên hàm>(self, TSHT):
    Các câu lệnh
    return <Giá trị trả về>
```

Cả 2 dạng trên thì **self** là **đối số thứ nhất** của hàm và chính là đối tượng thuộc lớp, đóng vai trò là toán hạng thứ nhất của toán tử cần quá tải. Và **đối số thứ hai** của hàm có thể là **một đối tượng khác** thuộc lớp hoặc có thể là một **tham số** bất kỳ, đóng vai trò là toán hạng thứ hai của toán tử cần quá tải.

Giá trị trả về của hàm có thể là **một đối tượng** thuộc lớp hoặc cũng có thể là một **giá trị bất kỳ** nào khác, tùy vào ngữ cảnh cụ thể.

2.2. Đối với các toán tử một ngôi

```
def <Tên hàm>(self):
    Các câu lệnh
    return <Giá trị trả về>
```

Đối với toán tử 1 ngôi thì chỉ có 1 dạng duy nhất và đối số **self** của hàm chính là đối tượng thuộc lớp, đóng vai trò là toán hạng của toán tử cần quá tải. **Giá trị trả về** của hàm có thể là **một đối tượng** thuộc lớp hoặc cũng có thể là một **giá trị bất kỳ** nào khác, tùy vào ngữ cảnh cụ thể.

3. CÁC VÍ DỤ MINH HỌA

3.1. Ví dụ về quá tải toán tử số học

Xây dựng lớp **Tọa độ** gồm có 2 thuộc tính của đối tượng là x (hoành độ), y (tung độ) và các phương thức: khởi tạo, nhập thông tin, xuất thông tin theo dạng (*hoành độ, tung độ*) và mỗi giá trị có 2 chữ số lẻ. Quá tải các toán tử cộng, lũy thừa, chia và chia lấy phần dư. Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Tọa độ**.

```
class Toado:
    '''Lớp tọa độ gồm 2 thuộc tính, 4 phương thức'''

    def __init__(self, hd = 0, td = 0):
        self.x = hd
        self.y = td

    def Nhap(self):
        self.x = float(input("Nhập hoành độ: "))
        self.y = float(input("Nhập tung độ: "))

    def Xuat(self):
        print('(', format(self.x, '.2f'), ', ', \
              format(self.y, '.2f'), ')')
```

```
def __add__(self, ob2):
    x = self.x + ob2.x
    y = self.y + ob2.y
    return Toado(x, y)

def __pow__(self, n):
    x = pow(self.x, n)
    y = pow(self.y, n)
    return Toado(x, y)

def __truediv__(self, ob2):
    x = self.x / ob2.x
    y = self.y / ob2.y
    return Toado(x, y)

def __mod__(self, m):
    x = self.x % m
    y = self.y % m
    return Toado(x, y)

A = Toado()
print('Nhập tọa độ cho A:')
A.Nhap()
print('Điểm A có tọa độ là:')
A.Xuat()

B = Toado()
print('Nhập tọa độ cho B:')
B.Nhap()
print('Điểm B có tọa độ là:')
B.Xuat()

C = A.__add__(B)
#C = A + B
print('Điểm C = A + B có tọa độ là:')
C.Xuat()

n = int(input('Nhập giá trị n = '))
D = A.__pow__(n)
#D = A ** n
print('Điểm D = A ** n có tọa độ là:')
D.Xuat()

E = A.__truediv__(B)
```

```
#E = A / B
print('Điểm E = A / B có tọa độ là:')
E.Xuat()
m = int(input('Nhập giá trị m = '))
F = B.__mod__(m)
#F = B % m
print('Điểm F = B % m có tọa độ là:')
F.Xuat()
```

Đối với các toán tử số học thì ta có thể gọi các hàm quá tải với các đại diện là các phép toán tương ứng của nó. Vì thế câu lệnh `C = A.__add__(B)` ta có thể thay thế bởi câu lệnh `C = A + B`, và tương tự cho các toán tử khác.

3.2. Ví dụ về quá tải toán tử luận lý

Xây dựng lớp **Con người** gồm có 2 thuộc tính lớp là đáng đi (thằng), trí khôn (có), 5 thuộc tính của đối tượng là màu da, màu mắt, nhóm máu, quốc tịch, ngày sinh và các phương thức: khởi tạo đối tượng, nhập thông tin, xuất thông tin, trả về thông tin nhóm máu, trả về thông tin quốc tịch, trả về thông tin năm sinh. Quá tải các toán tử and (trả về True nếu 2 người có cùng quốc tịch và ngược lại), or (trả về True nếu có ít nhất 1 người có nhóm máu AB và ngược lại), not (trả về True nếu người này không phải 19 tuổi và ngược lại). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Con người**.

```
from datetime import date

class Connguoi:
    '''Lớp con người gồm 7 thuộc tính, 6 phương thức
    và 3 hàm quá tải'''
    dangdi = 'thằng'
    trikhon = 'có'

    def __init__(self, md='vàng', mm='đen', nm='AB',
                  qt='Việt Nam', ns=date(1975,6,22)):
        self.mauda = md
        self.maumat = mm
        self.nhommau = nm
        self.quoctich = qt
        self.ngaysinh = ns

    def Nhap(self):
        self.mauda = input("Nhập màu da:")
        self.maumat = input("Nhập màu mắt:")
        self.nhommau = input("Nhập nhóm máu:")
        self.quoctich = input("Nhập quốc tịch:")
```

```
        ngay = int(input("Nhập ngày sinh:"))
        thang = int(input("Nhập tháng sinh:"))
        nam = int(input("Nhập năm sinh:"))
        self.ngaysinh = date(nam, thang, ngay)

    def Xuat(self):
        print('Dáng đi:', self.dangdi)
        print('Trí khôn:', self.trikhon)
        print('Màu da:', self.mauda)
        print('Màu mắt:', self.maumat)
        print('Nhóm máu:', self.nhommau)
        print('Quốc tịch:', self.quoctich)
        print('Ngày sinh:', self.ngaysinh.strftime("%d/%m/%Y"))

    def Nhommau(self):
        return self.nhommau

    def Quoctich(self):
        return self.quoctich

    def Namsinh(self):
        return self.ngaysinh.year

    def __and__(self, ob2):
        if(self.Quoctich() == ob2.Quoctich()):
            return True
        else:
            return False

    def __or__(self, ob2):
        if(self.Nhommau() == 'AB' or ob2.Nhommau() == 'AB'):
            return True
        else:
            return False

    def __invert__(self):
        if(date.today().year - self.Namsinh() != 19):
            return True
        else:
            return False
```

```

P1 = Connguoi()
P1.Nhap()
print('Thông tin về P1:')
P1.Xuat()

P2 = Connguoi()
P2.Nhap()
print('Thông tin về P2:')
P2.Xuat()

if(P1.__and__(P2) == True):
    print('Hai người cùng quốc tịch')
else:
    print('Hai người KHÔNG cùng quốc tịch')

if(P1.__or__(P2) == True):
    print('Có ít nhất 1 người có nhóm máu AB')
else:
    print('Cả 2 người đều KHÔNG có nhóm máu AB')

if(P1.__invert__() == True):
    print('Người này KHÔNG PHẢI 19 tuổi')
else:
    print('Người này 19 tuổi')

```

Ta cũng có thể gọi các hàm quá tải các toán tử luận lý với các đại diện là các phép toán tương tự như các toán tử số học.

3.3. Ví dụ về quá tải toán tử so sánh

Xây dựng lớp **Fraction** gồm có 2 thuộc tính của đối tượng là tử số, mẫu số và các phương thức: khởi tạo đối tượng, nhập thông tin, nếu mẫu số bằng 0 phải yêu cầu nhập lại, xuất thông tin theo dạng *tử số/mẫu số* hoặc *-tử số/mẫu số* (không có dạng *tử số/-mẫu số*) và nếu tử số bằng 0 thì chỉ in 0 (không có dạng *0/mẫu số*), còn nếu mẫu số bằng 1 thì chỉ in tử số (không có dạng *tử số/1*), tối giản một phân số. Quá tải các toán tử > (trả về True nếu phân số 1 lớn hơn phân số 2 và ngược lại), <= (trả về True nếu phân số 1 nhỏ hơn hoặc bằng phân số 2 và ngược lại), == (trả về True nếu 2 phân số bằng nhau và ngược lại). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Fraction**.

```

class Fraction:
    '''Lớp Fraction gồm 2 thuộc tính, 4 phương thức
    và 3 hàm quá tải'''

    def __init__(self, tu = 0, mau = 1):
        self.ts = tu
        self.ms = mau

```

```
def Nhap(self):
    tu = int(input("Nhập tử số:"))
    self.ts = tu
    mau = int(input("Nhập mẫu số:"))
    while mau == 0:
        mau = int(input("Nhập mẫu số:"))
    self.ms = mau

def Xuat(self):
    if self.ts == 0:
        print(0)
    elif self.ms == 1:
        print(self.ts)
    elif self.ts < 0 and self.ms < 0:
        print(abs(self.ts), '/', abs(self.ms))
    elif self.ts > 0 and self.ms < 0:
        print(-self.ts, '/', abs(self.ms))
    else:
        print(self.ts, '/', self.ms)

def Toigian(self):
    if self.ts < self.ms:
        i = self.ts
    else:
        i = self.ms
    while self.ts % i != 0 or self.ms % i != 0:
        i -= 1
    tu = self.ts // i
    mau = self.ms // i
    return Fraction(tu, mau)

def __gt__(self, ob2):
    tu1 = self.ts * ob2.ms
    tu2 = ob2.ts * self.ms
    if tu1 > tu2:
        return True
    else:
        return False

def __le__(self, ob2):
    tu1 = self.ts * ob2.ms
    tu2 = ob2.ts * self.ms
```

```
        if tu1 <= tu2:
            return True
        else:
            return False

    def __eq__(self, ob2):
        tu1 = self.ts * ob2.ms
        tu2 = ob2.ts * self.ms
        if tu1 == tu2:
            return True
        else:
            return False

ps1 = Fraction()
ps1.Nhap()
ps1.Xuat()

pstg = ps1.Toigian()
pstg.Xuat()

ps2 = Fraction()
ps2.Nhap()
ps2.Xuat()

if ps1 > ps2:
    print('Phân số 1 lớn hơn phân số 2')
else:
    print('Phân số 1 KHÔNG lớn hơn phân số 2')

if ps1 <= ps2:
    print('Phân số 1 nhỏ hơn hoặc bằng hơn phân số 2')
else:
    print('Phân số 1 KHÔNG nhỏ hơn hoặc bằng phân số 2')

if ps1 == ps2:
    print('Phân số 1 bằng với phân số 2')
else:
    print('Phân số 1 KHÔNG bằng phân số 2')
```

Ta cũng có thể gọi các hàm quá tải các toán tử so sánh với các đại diện là các phép toán tương tự như các toán tử số học.

BÀI TẬP CHƯƠNG 9



1. Xây dựng lớp **Tọa độ** gồm có 2 thuộc tính của đối tượng là x (hoành độ), y (tung độ) và các phương thức: khởi tạo, nhập thông tin, xuất thông tin theo dạng *{hoành độ, tung độ}* và mỗi giá trị có 4 chữ số lẻ, tính khoảng cách giữa 2 điểm bất kỳ. Quá tải các toán tử số học: trừ, nhân, chia lấy phần nguyên. Quá tải các toán tử luận lý: and (trả về True nếu cả 2 điểm cùng thuộc góc phần tư thứ nhất và ngược lại), or (trả về False nếu cả 2 điểm cùng thuộc góc phần tư thứ ba và ngược lại), xor (trả về True nếu cả 2 điểm cùng thuộc góc phần tư thứ hai hoặc tư và ngược lại), not (trả về True nếu điểm đó có cả hoành độ và tung độ đều âm). Quá tải các toán tử so sánh: > (trả về True nếu điểm thứ nhất xa gốc tọa độ hơn điểm thứ hai và ngược lại), >= (tương tự >), < (trả về True nếu hoành độ điểm thứ nhất nhỏ hơn hoành độ điểm thứ hai và ngược lại), <= (tương tự <), == (trả về True nếu trị tuyệt đối của hoành độ 2 điểm bằng nhau và trị tuyệt đối của tung độ 2 điểm bằng nhau và ngược lại), != (trả về True nếu trị tuyệt đối của hoành độ 2 điểm khác nhau hoặc trị tuyệt đối của tung độ 2 điểm khác nhau và ngược lại). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Tọa độ**.

2. Xây dựng lớp **Con người** gồm có 2 thuộc tính lớp là đáng đi (thăng), trí khôn (có), 4 thuộc tính của đối tượng là màu da, màu mắt, nhóm máu, quốc tịch, ngày sinh và các phương thức: khởi tạo đối tượng, nhập thông tin, xuất thông tin, trả về thông tin màu da, trả về thông tin màu mắt, trả về thông tin ngày sinh (dạng date có cả năm, tháng, ngày). Quá tải các toán tử số học: + (cộng giá trị n vào ngày sinh của một người và trả về một người với thông tin ngày sinh đã được cộng hợp lệ), - (trừ giá trị n khỏi ngày sinh của một người và trả về một người với thông tin ngày sinh đã được trừ hợp lệ). Quá tải các toán tử luận lý: and (trả về True nếu 2 người có cùng sinh nhật và ngược lại), or (trả về True nếu có ít nhất 1 người có mắt màu xanh và ngược lại), xor (trả về True nếu có 1 người có da màu trắng và 1 người có da màu vàng và ngược lại), not (trả về True nếu người này không trùng ngày sinh với bạn và ngược lại). Quá tải các toán tử so sánh: > (trả về True nếu tuổi người 1 lớn hơn tuổi người 2 và ngược lại), >= (tương tự >), < (trả về True nếu tháng sinh người 1 nhỏ hơn tháng sinh người 2 và ngược lại), <= (tương tự <), == (trả về True nếu ngày sinh (cả năm, tháng, ngày) người 1 bằng ngày sinh người 2 và ngược lại), != (tương tự ==). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Con người**.

3. Xây dựng lớp **Fraction** gồm có 2 thuộc tính của đối tượng là tử số, mẫu số và các phương thức: khởi tạo đối tượng, nhập thông tin, nếu mẫu số bằng 0 phải yêu cầu nhập lại, xuất thông tin theo dạng *tử số/mẫu số* hoặc *-tử số/mẫu số* (không có dạng *tử số/-mẫu số*) và nếu tử số bằng 0 thì chỉ in 0 (không có dạng *0/mẫu số*), còn nếu mẫu số bằng 1 thì chỉ in tử số (không có dạng *tử số/1*), tối giản một phân số. Quá tải các toán tử số học: +, -, *, **, /, // (trả về phân số có tử số = tử số của phân số 1 // tử số của phân số 2 và mẫu số = mẫu số của phân số 1 // mẫu số của phân số 2), % (trả về phân số có tử số = tử số của phân số 1 % tử số của phân số 2 và mẫu số = mẫu số của phân số 1 % mẫu số của phân số 2). Quá tải các toán tử luận lý: and (trả về True nếu cả 2 phân số dương và ngược lại), or (trả về True nếu có ít nhất 1 phân số dương và ngược lại), xor

(trả về True nếu chỉ có 1 phân số dương và ngược lại), not (trả về True nếu phân số âm và ngược lại). Quá tải các toán tử so sánh: < (trả về True nếu phân số 1 nhỏ hơn phân số 2 và ngược lại), >= (trả về True nếu phân số 1 lớn hơn hoặc bằng phân số 2 và ngược lại), != (trả về True nếu 2 phân số khác nhau và ngược lại). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Fraction**.

4. Xây dựng lớp **Clock** gồm có 3 thuộc tính của đối tượng là giờ, phút, giây và các phương thức: khởi tạo đối tượng, nhập thông tin, nếu giá trị giờ (0 -> 23), phút (0 -> 59), giây (0 -> 59) không hợp lệ phải yêu cầu nhập lại, xuất thông tin theo dạng [giờ - phút - giây]. Quá tải các toán tử số học: + (cộng giá trị n giây vào đối tượng thuộc lớp Clock và trả về một clock với thông tin hợp lệ), - (trừ giá trị n giây khỏi đối tượng thuộc lớp Clock và trả về một clock với thông tin hợp lệ). Quá tải các toán tử luận lý: and (trả về True nếu cả 2 clock có số giờ đều >= 12 và ngược lại), or (trả về True nếu có ít nhất 1 clock có số phút >= 30 và ngược lại), xor (trả về True nếu chỉ có 1 clock có số giây >= 30 và ngược lại), not (trả về True nếu clock có số giờ < 12 và số phút < 30 và số giây < 30 và ngược lại). Quá tải các toán tử so sánh: < (trả về True nếu clock 1 nhỏ hơn clock 2 và ngược lại), <= (tương tự <), > (trả về True nếu clock 1 lớn hơn clock 2 và ngược lại), >= (tương tự >), == (trả về True nếu clock 1 bằng với clock 2 và ngược lại), != (tương tự ==). Viết chương trình kiểm tra tính đúng đắn của các phương thức đã tạo và các toán tử đã quá tải trên lớp **Clock**.

5. Viết chương trình thực hiện các công việc sau:

- Xây dựng lớp SINH VIÊN gồm có 1 thuộc tính lớp là trường = 'VLUTE' và các thuộc tính đối tượng: mã số sinh viên, họ tên, ngày sinh (có đủ ngày, tháng, năm), điểm trung bình; các phương thức: khởi tạo, nhập thông tin, trả về chuỗi con (là từ đầu tiên trong họ tên chính là họ của sinh viên), trả về tuổi, xuất thông tin gồm: trường, mã số sinh viên, họ, tuổi và điểm trung bình. Tạo một đối tượng thuộc lớp SINH VIÊN và kiểm tra tính đúng đắn của các phương thức đã tạo.
- Quá tải toán tử cộng (+) để thực hiện chức năng cộng vào điểm trung bình của sinh viên với một số thực bất kỳ. Kiểm tra tính đúng đắn của toán tử cộng đã quá tải.
- Xây dựng lớp SINH VIÊN_IT kế thừa lớp SINH VIÊN và có thêm 1 thuộc tính lớp là khoa = 'FIT', 1 thuộc tính đối tượng là quê quán; viết đề các phương thức: nhập thông tin, xuất thông tin sao cho đủ thông tin, trả về chuỗi con (là từ cuối cùng trong họ tên chính là tên của sinh viên). Tạo một đối tượng thuộc lớp SINH VIÊN_IT và kiểm tra tính đúng đắn của các phương thức được kế thừa và viết đề.
- Quá tải toán tử nhỏ hơn (<) để thực hiện chức năng so sánh điểm trung bình của 2 SINHVIEN_IT, nếu điểm trung bình của sinh viên 1 < điểm trung bình của sinh viên 2 thì trả về True và ngược lại. Kiểm tra tính đúng đắn của toán tử nhỏ hơn đã quá tải.
- Tạo một danh sách gồm N sinh viên thuộc lớp SINHVIEN_IT (N không âm và không quá 99). In danh sách sinh viên vừa tạo. Đếm số sinh viên có quê ở Vĩnh Long. Cho biết tên của các sinh viên có điểm trung bình cao nhất. Sắp xếp danh sách sinh viên giảm dần theo điểm trung bình và in ra danh sách nhân viên sau khi sắp xếp.

Chương 10: LÀM VIỆC VỚI TẬP TIN VÀ THƯ MỤC



1. TẬP TIN

Tập tin hay tệp (file) là tập hợp của các thông tin được đặt tên và được lưu trữ trong bộ nhớ máy tính như đĩa cứng, đĩa mềm, CD, ... Hiểu theo một cách khác thì tập tin chính là một dãy bit có tên và được lưu trữ trên các thiết bị bộ nhớ của máy tính.

Khi làm việc với tập tin thì NLT sẽ thực hiện 3 bước theo thứ tự: mở tập tin, thao tác trên tập tin (thường là đọc và ghi), đóng tập tin.

Có rất nhiều loại tập tin khác nhau có thể được lưu trữ trong bộ nhớ của máy tính, trong đó loại phổ biến nhất chính là tập tin văn bản. Chính vì thế trong tài liệu này chúng ta tập trung tìm hiểu và nghiên cứu về tập tin văn bản trong ngôn ngữ lập trình Python.

1.1. Mở tập tin

Để đọc và ghi một tập tin nào đó thì công việc đầu tiên ta cần phải thực hiện là mở tập tin đó. Python cho phép chúng ta thực hiện công việc mở file bằng phương thức **open** với cú pháp như sau:

<ob> = open(filepath, mode)

Trong đó, **ob** là đối tượng gắn với file, **filepath** là đường dẫn đến tập tin cần mở và **mode** là chế độ để mở tập tin. Có một số chế độ là:

Chế độ	Cách thức làm việc
'r'	Mở file chỉ để đọc. Nếu file chưa tồn tại thì chương trình sẽ báo lỗi.
'r+'	Mở file để đọc và ghi.
'w'	Mở file chỉ để ghi. Nếu file chưa tồn tại thì sẽ tạo file mới, nếu file đã tồn tại thì sẽ xóa bỏ nội dung cũ để ghi nội dung mới.
'w+'	Mở file để đọc và ghi. Nếu file chưa tồn tại thì sẽ tạo file mới, nếu file đã tồn tại thì sẽ xóa bỏ nội dung cũ để ghi nội dung mới.
'a'	Mở file để ghi tiếp. Nếu file đã tồn tại thì sẽ ghi tiếp nội dung vào cuối file, nếu file chưa tồn tại thì tạo một file mới và ghi nội dung vào đó.
'a+'	Mở file để đọc và ghi tiếp. Nếu file đã tồn tại thì 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 để 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 để đọc và ghi. 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).

Trong Python, khi mở file thì nó sẽ mặc định là file văn bản (text), nếu NLT muốn mở file ở dạng nhị phân (binary) thì phải thêm ký tự 'b' vào các chế độ trên, chẳng hạn như: rb, wb, ab, rb+/r+b, ...

Ví dụ 1: Giả sử thư mục hiện hành là *D:\DriveD\Bai tap thuc hanh\LTHDT by Python\Vidu\Chuong10*

```
f1 = open('Test.txt', 'r')
```

Câu lệnh trên sẽ mở tập tin Test.txt để đọc nội dung ở dạng văn bản nếu có tập tin Test.txt trong thư mục hiện hành, ngược lại sẽ bị báo lỗi FileNotFoundError.

```
f2 = open('D:\DriveD\Bai tap thuc hanh\LTHDT by  
Python\Vidu\Chuong10\Quehuong.txt', 'a+b')
```

Câu lệnh trên sẽ mở tập tin Quehuong.txt để đọc và ghi tiếp nội dung nếu có tập tin Quehuong.txt trong thư mục hiện hành, ngược lại tập tin Quehuong.txt sẽ được tạo ra để chứa nội dung.

Lưu ý:

- Sau khi mở tập tin và gán với đối tượng ob thì sau đó ob trở thành tên đại diện cho file đó trong các câu lệnh của chương trình.
- Khi làm việc với tập tin văn bản thì ta nên chỉ định thêm loại mã hóa bởi tham số thứ ba là encoding trong hàm open.

1.2. Đọc nội dung từ tập tin

Ngôn ngữ lập trình Python hỗ trợ một số phương thức để lấy được nội dung của tập tin thông qua các phương thức đọc tập tin.

1.2.1 Phương thức read

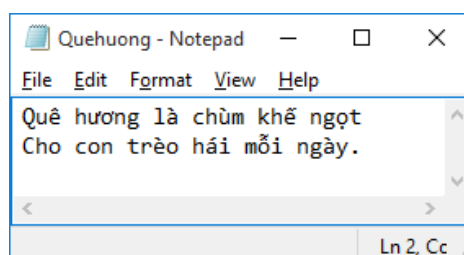
Cú pháp:

```
<ob>.read([num])
```

Phương thức này cho phép ta đọc một lượng dữ liệu tương ứng với num là số byte/ký tự cần đọc và trả về một chuỗi. Giá trị mặc định của num là -1.

- Nếu không đọc được gì thì chuỗi rỗng (có độ dài bằng 0) sẽ được trả về.
- Khi size bị bỏ qua thì toàn bộ nội dung của tập tin sẽ được đọc.
- Nếu file quá lớn thì nó sẽ đọc cho đến khi hết bộ nhớ cho phép.

Ví dụ 2: Giả sử đã có tập tin Quehuong.txt lưu trong thư mục hiện hành với nội dung như sau:



Đoạn chương trình sau sẽ in ra toàn bộ nội dung của tập tin Quehuong.txt.

```
f1 = open('Quehuong.txt', 'r', encoding = 'utf-8')
nd = f1.read() #{2}
print(nd)
```

Nếu câu lệnh {2} được thay thế bởi câu lệnh `nd = f1.read(10)` thì biến chuỗi `nd` chứa chuỗi 'Quê hương'.

1.2.2. Phương thức *readline*

Cú pháp:

`<ob>.readline([num])`

Tương tự như phương thức `read`, tuy nhiên mỗi lần đọc thì nó chỉ đọc 1 dòng, tức là đọc tới khi gặp newline hoặc hết tập tin. Con trỏ tập tin sẽ di chuyển từ dòng này qua dòng khác. Kết quả trả về dưới dạng một chuỗi và một chuỗi rỗng (có độ dài bằng 0) được trả về nếu phương thức không đọc được gì.

Ví dụ 3: Đọc nội dung tập tin Quehuong.txt theo từng dòng.

```
f1 = open('Quehuong.txt', 'r', encoding = 'utf-8')
nd_l1 = f1.readline()
print('Nội dung dòng 1:', nd_l1)
nd_l2 = f1.readline()
print('Nội dung dòng 2:', nd_l2)
nd_l3 = f1.readline()
print('Nội dung dòng 3:', nd_l3)
```

1.2.3. Phương thức *readlines*

Cú pháp:

`<ob>.readlines([num])`

Tương tự như phương thức `read` và `readline`, tuy nhiên phương thức này sẽ đọc toàn bộ các dòng trong tập tin kể từ vị trí con trỏ tập tin. Nếu con trỏ tập tin ở đầu tập tin thì phương này sẽ đọc toàn bộ tập tin. Kết quả của phương thức này là một list với mỗi phần tử là một dòng dữ liệu trong tập tin. Khi đọc xong thì con trỏ tập tin sẽ được đưa đến cuối tập tin và khi đó nếu tiếp tục dùng `readlines` thì kết quả trả về sẽ là một list rỗng. Đối với phương thức này thì ký tự xuống dòng (`\n`) vẫn được đọc và nếu con trỏ tập tin ở đầu tập tin thì ký tự đầu tiên của tập tin cũng được đọc.

Ví dụ 4: Đọc nội dung tập tin Quehuong.txt bằng phương thức `readlines`.

```
f1 = open('Quehuong.txt', 'r', encoding = 'utf-8')
nd_ls = f1.readlines()
print(nd_ls)
```

Kết quả sẽ là:

```
['\ufeffQuê hương là chùm khế ngọt\n', 'Cho con trèo hái  
mỗi ngày.']
```

1.3. Ghi nội dung vào tập tin

Tương tự như đọc tập tin, để ghi nội dung vào một tập tin thì ta cần mở tập tin với chế độ phù hợp và sau đó sử dụng phương thức **write** để ghi.

Cú pháp:

```
<ob>.write(string)
```

Phương thức này cho phép ghi một chuỗi có nội dung là string vào vị trí của con trỏ trong tập tin.

Ví dụ 5: Mở tập tin mới có tên là Lthdt.txt và ghi nội dung vào tập tin.

```
f2 = open('Lthdt.txt', 'w', encoding = 'utf-8')  
f2.write('Học phần: LẬP TRÌNH HDT\n')  
f2.write('Số tín chỉ: 3\n')  
f2.write('Thực hành trên NNLT: Python')
```

1.4. Xác định vị trí hiện hành của con trỏ tập tin

Cú pháp:

```
<ob>.tell()
```

Phương thức này dùng để xác định vị trí hiện hành của con trỏ tập tin. Nói cách khác, việc đọc và ghi tiếp theo sẽ diễn ra tại vị trí này.

Ví dụ 6:

```
f2 = open('Lthdt.txt', 'r', encoding = 'utf-8')  
print(f2.tell()) #=> 0  
nd = f2.readline()  
print(nd)  
print(f2.tell()) #=> 33  
nd = f2.readlines()  
print(nd)  
print(f2.tell()) #=> 84  
f2.close()
```

1.5. Di chuyển con trỏ tập tin

Cú pháp:

```
<ob>.seek(num [, from])
```

Phương thức này dùng để di chuyển con trỏ tập tin đến vị trí num tính từ vị trí đầu tập tin nếu không có from hoặc from là 0 (mặc định). Nếu from là 1 (tính từ vị trí hiện hành) hoặc from là 2 (tính từ vị trí cuối tập tin) thì num phải có giá trị là 0.

Ví dụ 7:

```
f2 = open('Lthdt.txt', 'r', encoding = 'utf-8')
print(f2.tell()) #=> 0
nd = f2.readline()
print(f2.tell()) #=> 33
f2.seek(0, 1)
print(f2.tell()) #=> 33
f2.seek(0, 0)
print(f2.tell()) #=> 0
f2.seek(0, 2)
print(f2.tell()) #=> 84
```

1.6. Đóng tập tin đã mở

Để làm việc an toàn với tập tin thì sau khi hoàn tất các thao tác đọc, ghi tập tin thì ta nên gọi phương thức **close()** để đóng tập tin đã mở.

Cú pháp:

<ob>.close()

Ví dụ 8: Mở tập tin mới có tên là Lthdt.txt và ghi nội dung vào tập tin. Sau đó mở tập tin Lthdt.txt để đọc nội dung và in lên màn hình.

```
f2 = open('Lthdt.txt', 'w', encoding = 'utf-8')
f2.write('Học phần: LẬP TRÌNH HDT\n')
f2.write('Số tín chỉ: 3\n')
f2.write('Thực hành trên NNLT: Python')
f2.close()
f2 = open('Lthdt.txt', 'r', encoding = 'utf-8')
nd = f2.read()
print(nd)
f2.close()
```

1.7. Đổi tên tập tin

Cú pháp:

os.rename(old_name, new_name)

Phương thức rename dùng để đổi tên một tập tin từ old_name thành new_name.

Ví dụ 9:

```
import os
os.rename('Lthdt.txt', 'LtPython.txt')
os.rename('Quehuong2.txt', 'D:\DriveD\Qh2.txt')
```

Lưu ý: Nếu tên old_name không ở thư mục hiện hành thì ta phải chỉ ra đường dẫn đến old_name. Và nếu ta muốn new_name nằm ở thư mục khác thì ta cũng phải chỉ ra đường dẫn cụ thể.

1.8. Xóa tập tin**Cú pháp:**

```
os.remove(file_name)
```

Phương thức remove dùng để xóa một tập tin trong thư mục hiện hành. Nếu tập tin không nằm trong thư mục hiện hành thì ta phải chỉ ra đường dẫn đến tập tin đó.

Ví dụ 10:

```
import os
os.remove('Test.txt')
os.remove('D:\DriveD\Qh2.txt')
```

1.9. Một số phương thức khác

Ngoài các phương thức thường được sử dụng đã được trình bày. Khi làm việc với file ta có thể cần thêm các phương thức được cho trong bảng sau:

PHƯƠNG THỨC	CHỨC NĂNG
fileno()	Trả về một số nguyên mô tả file (file descriptor).
flush()	Xóa sạch bộ nhớ đệm của luồng file.
readable()	Trả về True nếu file có thể đọc được.
seekable()	Trả về True nếu luồng hỗ trợ truy cập ngẫu nhiên.
truncate(size=None)	Cắt gọn kích cỡ file thành kích cỡ tham số size.
writable()	Trả về True nếu file có thể ghi được.
writelines(lines)	Ghi một danh sách các dòng vào file.

2. THƯ MỤC

Thư mục là nơi chứa các tập tin và các thư mục con của nó. Khi làm việc với máy tính nói chung và lập trình nói riêng thì việc truy xuất hay tham khảo đến các thư mục là công việc thường xảy ra. Chính vì thế NNLT Python đã cung cấp nhiều phương thức cho NLT xử lý các hoạt động đa dạng liên quan tới thư mục thông qua module os.

2.1. Tạo thư mục

Cú pháp:

os.mkdir(dir_name)

Phương thức mkdir dùng để tạo ra một thư mục mới có tên là dir_name. Nếu thư mục cần tạo không nằm trong thư mục hiện hành thì ta cần chỉ thêm đường dẫn đến nơi chứa thư mục mới này.

Ví dụ 11:

```
import os
os.mkdir('TT_TM')
os.mkdir('D:\DriveD\Chuong10')
```

2.2. Đọc nội dung thư mục

Cú pháp:

os.listdir([dir_name])

Phương thức listdir dùng để trả về một danh sách các tập tin và thư mục con của thư mục dir_name. Nếu không có dir_name thì thư mục hiện hành sẽ được liệt kê.

Ví dụ 12:

```
import os
print(os.listdir())
print(os.listdir("D:\\"))
ndTM = os.listdir('D:\DriveD\Bai giang\Bai giang LTHDT by Python')
print(ndTM)
```

2.3. Xóa thư mục

Cú pháp:

os.rmdir(dir_name)

Phương thức rmdir dùng để xóa một thư mục rỗng có tên là dir_name.

Ví dụ 13:

```
import os
os.rmdir('D:\DriveD\Chuong10')
```

Lưu ý: Để xóa được một thư mục không rỗng, chúng ta có thể sử dụng phương thức rmtree bên trong module shutil.

Ví dụ 14:

```
import os
os.mkdir('D:\DriveD\Chuong10')
os.mkdir('D:\DriveD\Chuong10\Ly thuyet')
```



```

os.mkdir('D:\DriveD\Chuong10\Bai tap')
print(os.listdir('D:\DriveD\Chuong10'))
#=> ['Bai tap', 'Ly thuyet']
os.rmdir('D:\DriveD\Chuong10')
#=> OSError: [WinError 145] The directory is not empty: ...
import shutil
shutil.rmtree('D:\DriveD\Chuong10')
print(os.listdir('D:\DriveD\Chuong10'))
#=> FileNotFoundError: [WinError 3] The system cannot find
the path specified: 'D:\\DriveD\\Chuong10'

```

3. CÁC MODULE CÓ LIÊN QUAN

3.1. Giới thiệu

Ở các phần trên chúng ta cũng đã sử dụng một số phương thức của các module để thao tác với tập tin và thư mục.

Trong NNLT Python, **os** và **shutil** là 2 module có sẵn và được sử dụng rất phổ biến cho các công việc liên quan đến việc xử lý tập tin và thư mục.

Module **os** có thể coi là một module bao đóng các lệnh gọi POSIX (Portable Operating System Interface) hoặc mô phỏng chúng trên một số nền tảng mà không có POSIX. Ta thấy các phương thức gần như được đặt tên giống hệt như trong ngôn ngữ lập trình C/C++, và các ngôn ngữ khác cũng thực hiện tương tự.

Module **shutil** chứa các phương thức đặc trưng của Python, cho phép NLT sử dụng nhiều thao tác phức tạp hơn so với **os**. Các phương thức trong **shutil** thường gọi lại các phương thức trong **os**.

Và để sử dụng các phương thức trong module nào thì điều đầu tiên ta cần phải thực hiện là import module đó vào chương trình.

3.2. Một số phương thức thường dùng

3.2.1 Phương thức *getcwd*

Cú pháp:

```
os.getcwd()
```

Phương thức *getcwd* dùng để trả về thư mục làm việc hiện hành, kết quả trả về dưới dạng một chuỗi.

Ví dụ 15:

```

import os
TM_HH = os.getcwd()
print(TM_HH)
#=> D:\DriveD\Bai tap thuc hanh\LTHDT by Python\Vidu\Chuong10

```

3.2.2 Phương thức walk

Cú pháp:

os.walk(dir_name [, topdown])

Phương thức walk dùng để liệt kê tất cả nội dung có trong cây thư mục có tên là dir_name bằng việc duyệt qua từng nút của cây thư mục đó từ trên xuống hoặc từ dưới lên tùy thuộc vào topdown có giá trị là True (mặc định) hay False.

Đối với mỗi thư mục, phương thức walk sẽ liệt gồm một bộ 3 thành phần là (dir_path, dir_names, name_files).

Trong đó:

- **dir_path:** đường dẫn đến thư mục đang được duyệt.
- **dir_name:** danh sách các thư mục con của thư mục đang được duyệt.
- **name_files:** danh sách các tập tin của thư mục đang được duyệt.

Ví dụ 16:

```
import os
print('Nội dung của thư mục gốc trong cây cần duyệt:')
for (dir_path, dir_names, name_files) in os.walk('D:\
DriveD\Bai giang\Bai giang LTHDT by Python', topdown=True):
    print(dir_path)
    print(dir_names)
    print(name_files)
    print('Nội dung của thư mục con kế tiếp được duyệt:')
```

Kết quả hiển thị của đoạn chương trình là:

Nội dung của thư mục gốc trong cây cần duyệt:

D:\DriveD\Bai giang\Bai giang LTHDT by Python

['De kiểm tra', 'Giao trình', 'Source Python and PyCharm',
'Tai lieu tham khao', 'ĐỀ THI']

['Demo_Sử dụng date và Danh sách đối tượng.docx', 'Source
Python and PyCharm.rar', 'Tài liệu LTHDT bang
Python.docx', '~\$i giang LTHDT bang Python - Chuong
12345.docx']

Nội dung của thư mục con kế tiếp được duyệt:

D:\DriveD\Bai giang\Bai giang LTHDT by Python\De kiểm tra

['CÂU HỎI TN PHẦN 1', 'CÂU HỎI TN PHẦN 2']

['Bài kiểm tra trắc nghiệm về Python.docx', 'De kiểm tra
THUC HANH - LTHDT - De 1 & 2.docx', 'Kiểm tra LTHDT-Lan 1
- Dot 1.pptx', 'Kiểm tra LTHDT-Lan 1 - Dot 2.pptx']

Nội dung của thư mục con kế tiếp được duyệt:

```
D:\DriveD\Bai giang\Bai giang LTHDT by Python\Giao trinh  
[]
```

```
['1.Bia - Bai giang LTHDT bang Python - 2021.docx', '2.Loì  
noi dau - Bai giang LTHDT bang Python - 2021.docx', '3.Noi  
dung - Bai giang LTHDT bang Python - 2021.docx', 'Bai  
giang LTHDT bang Python.pdf', 'Gioi thieu chung - Cac qui  
dinh - Cach danh gia hoc phan LTHDT.pptx']
```

Nội dung của thư mục con kế tiếp được duyệt:

```
D:\DriveD\Bai giang\Bai giang LTHDT by Python\Source  
Python and PyCharm  
[]
```

```
['pycharm-community-2020.1.3.exe', 'python-3.8.5.exe']
```

Nội dung của thư mục con kế tiếp được duyệt:

```
D:\DriveD\Bai giang\Bai giang LTHDT by Python\Tai lieu  
tham khao  
[]
```

```
['Bài tập Python thực hành với code mẫu-3 cấp độ.docx',  
'Lập trình hướng đối tượng trong Python-QuanTriMang.docx',  
'python-Đỗ Thanh Nghị-2016.pdf']
```

3.2.3. Phương thức chdir**Cú pháp:**

os.chdir(dir_name)

Phương thức chdir dùng để chuyển đổi thư mục hiện hành sang một thư mục khác.

Ví dụ 17:

```
import os  
print(os.getcwd())  
os.chdir('D:\DriveD\Bai giang')  
print(os.getcwd())
```

3.2.4. Các phương thức trong module os.path

Module os.path hỗ trợ các phương thức giúp cho việc thao tác trên đường dẫn nhanh chóng và thuận tiện hơn.

Một số phương thức thông dụng là:

- **os.path.exists(path):** Kiểm tra path có tồn tại hay không.
- **os.path.getsize(path):** Trả về kích thước (byte) của tập tin hay thư mục cuối path.

- **`os.path.isfile(path)`**: Kiểm tra xem cuối path có phải là một tập tin hay không.
- **`os.path.isdir(path)`**: Kiểm tra xem cuối path có phải là một thư mục hay không.
- **`os.path.dirname(path)`**: Trả về đường dẫn đến thư mục cha của path.
- **`os.path.getatime(path)`**: Trả về thời gian (giây, tính từ 1/1/1970) truy cập mới nhất.
- **`os.path.getmtime(path)`**: Trả về thời gian chỉnh sửa cuối cùng.
- **`os.path.getctime(path)`**: Trả về thời gian chỉnh sửa cuối cùng của metadata trên một số hệ thống hoặc trả về thời gian tạo ra tập tin trên Windows.

Ví dụ 18:

```
import os
print(os.path.exists('D:\DriveD\Chuong10'))
#=> True
print(os.path.exists('D:\DriveD\Chuong10\Python'))
#=> False
print(os.path.exists('D:\DriveD\Chuong10\Quehuong.txt'))
#=> True
print(os.path.getsize('D:\DriveD\Chuong10'))
#=> 0
print(os.path.getsize('D:\DriveD\Chuong10\Quehuong.txt'))
#=> 71
print(os.path.isfile('D:\DriveD\Chuong10'))
#=> False
print(os.path.isfile('D:\DriveD\Chuong10\Quehuong.txt'))
#=> True
print(os.path.isdir('D:\DriveD\Chuong10'))
#=> True
print(os.path.isdir('D:\DriveD\Chuong10\Quehuong.txt'))
#=> False
print(os.path.dirname('D:\DriveD\Chuong10'))
#=> D:\DriveD
print(os.path.dirname('D:\DriveD\Chuong10\Quehuong.txt'))
#=> D:\DriveD\Chuong10
print(os.path.getatime('D:\DriveD\Chuong10'))
#=> 1629860014.6890566
print(os.path.getatime('D:\DriveD\Chuong10\Quehuong.txt'))
#=> 1629860014.6860554
print(os.path.getmtime('D:\DriveD\Chuong10'))
#=> 1629860014.6890566
```

```
print(os.path.getmtime('D:\DriveD\Chuong10\Quehuong.txt'))  
#=> 1629622027.0592413  
  
print(os.path.getctime('D:\DriveD\Chuong10'))  
#=> 1629857701.3248053  
  
print(os.path.getctime('D:\DriveD\Chuong10\Quehuong.txt'))  
#=> 1629860014.6860554
```

3.2.5. Phương thức copy2

Cú pháp:

shutil.copy2(source_file, destination)

Phương thức copy2 dùng để sao chép tập tin source_file sang thư mục destination.

Ví dụ 19:

```
import shutil  
  
shutil.copy2('D:\DriveD\Chuong10\Quehuong.txt', 'D:\DriveD\Bai_giang')  
shutil.copy2('D:\DriveD\Chuong10\Quehuong.txt', 'D:\DriveD\Qh_copy.txt')  
shutil.copy2('D:\DriveD\Chuong10', 'D:\DriveD\Bai_giang')  
#=> PermissionError: [Errno 13] Permission denied: 'D:\\DriveD\\Chuong10'
```

3.2.6 Phương thức move

Cú pháp:

shutil.move(source_file, destination)

Phương thức move dùng để di chuyển tập tin source_file sang thư mục destination. Phương thức này tương tự như phương thức rename trong os.

Ví dụ 20:

```
import shutil  
  
shutil.move('D:\DriveD\Chuong10\Quehuong.txt', 'D:\DriveD\Bai_giang')  
shutil.move('D:\DriveD\Bai_giang\Quehuong.txt', 'D:\DriveD\Qh_new.txt')  
shutil.move('D:\DriveD\Chuong10', 'D:\DriveD\Bai_giang')
```

Lưu ý:

- Thư mục chứa tập tin mới phải tồn tại nếu không phương thức sẽ thực hiện không đúng ý đồ của NLT, và trên môi trường Windows thì trong cùng một thư mục thì không được có tập tin và thư mục con trùng tên nhau.
- Thường thì phương thức move sẽ gọi thực hiện phương thức rename trong hầu hết các trường hợp, tuy nhiên nếu destination nằm trên một đĩa khác với source_file thì hàm này sẽ không gọi phương thức rename mà sẽ sao chép tập tin trước bằng phương thức copy2 rồi sau đó sẽ xóa tập tin nguồn.

BÀI TẬP CHƯƠNG 10

1. Viết chương trình đọc và in nội dung của một file bất kỳ với tên file được nhập từ bàn phím.
2. Viết chương trình đọc và in n dòng đầu tiên của một file bất kỳ với tên file và số dòng được nhập từ bàn phím.
3. Viết chương trình đọc và in n dòng cuối cùng của một file bất kỳ với tên file và số dòng được nhập từ bàn phím.
4. Viết chương trình nhập vào từng câu và ghi vào một file có tên được nhập từ bàn phím, quá trình ghi dừng khi nhập một câu rỗng. Hãy in nội dung file vừa ghi lên màn hình.
5. Viết chương trình đọc một file bất kỳ với tên file được nhập từ bàn phím, in ra các từ dài nhất có trong file.
6. Viết chương trình đọc một file bất kỳ với tên file được nhập từ bàn phím, cho biết trong file có bao nhiêu từ.
7. Viết chương trình đọc một file bất kỳ với tên file được nhập từ bàn phím, cho biết trong file có bao nhiêu dòng.
8. Viết chương trình đọc, in và lưu vào list nội dung từng dòng của một file bất kỳ với tên file được nhập từ bàn phím.
9. Viết chương trình mô phỏng việc copy một file bằng 3 cách khác nhau.
10. Viết chương trình đọc lần lượt từng dòng trên 2 file bất kỳ, sau đó ghi vào file mới, với các tên file được nhập từ bàn phím. Nếu 1 trong 2 file hết thì đọc nội dung file còn lại và ghi vào cuối file mới. Đọc và in nội dung file mới lên màn hình.
11. Viết chương trình đọc một file bất kỳ với tên file được nhập từ bàn phím, in ra một dòng ngẫu nhiên trong file đó.
12. Xây dựng một module gồm các hàm sau:
 - a. Đếm số dòng không bắt đầu bởi ký tự character có trong file_name, với character và file_name là 2 tham số đầu vào của hàm.
 - b. Đếm số từ là word có trong file_name, với word và file_name là 2 tham số đầu vào của hàm.

Hãy viết chương trình để gọi thực hiện các hàm có trong module trên.

13. Xây dựng một module gồm các hàm sau:

- a. Đếm số từ có ít nhất k ký tự có trong file_name, với k và file_name là 2 tham số đầu vào của hàm.
- b. Đếm số từ có ký tự cuối là kt có trong file_name, với kt và file_name là 2 tham số đầu vào của hàm.

Hãy viết chương trình để gọi thực hiện các hàm có trong module trên.

14. Đóng gói 2 module trên thành một package. Sau đó viết chương trình để gọi thực hiện các hàm có trong các module nằm trong package.

15. Xây dựng hàm kiểm tra nội dung trong file_name sau khi đã thay thế tất cả các ký tự kt1 bởi kt2, với file_name, kt1 và kt2 là 3 tham số đầu vào của hàm. Viết chương trình kiểm tra tính đúng đắn của hàm trên.

16. Giả sử cần có một file nhị phân Sach.dat có cấu trúc là list gồm [Mã số, tựa sách, tên tác giả, nhà xuất bản, năm xuất bản]. Xây dựng các hàm thực hiện các chức năng sau:

- a. Nhập dữ liệu vào file Sach.dat theo đúng cấu trúc đã cho.
- b. Đếm số sách được viết bởi tác giả X, với X là tham số đầu vào của hàm.
- c. Cho biết các tựa sách được in bởi nhà xuất bản Z, với Z là tham số đầu vào của hàm.
- d. Cho biết nhà xuất bản nào có số sách nhiều nhất.
- e. Tạo ra file mới với tên là Sach_moi.dat gồm các quyển sách được xuất bản sau năm 2020.

Hãy viết chương trình để kiểm tra tính đúng đắn của các hàm trên.

17. Giả sử cần có một file nhị phân Sinhvien.dat có cấu trúc là tuple gồm (Mã số, họ tên, năm sinh, điểm trung bình). Xây dựng các hàm thực hiện các chức năng sau:

- a. Nhập dữ liệu vào file Sinhvien.dat theo đúng cấu trúc đã cho.
- b. In ra những sinh viên có năm sinh là Y, với Y là tham số đầu vào của hàm.
- c. Cho biết có bao nhiêu sinh viên có điểm trung bình lớn hơn hoặc bằng 3.2.
- d. Hiển thị thông tin các sinh viên có họ tên bắt đầu là ký tự 'N'.

Hãy viết chương trình để kiểm tra tính đúng đắn của các hàm trên.

18. Giả sử cần có một file nhị phân Nhanvien.dat có cấu trúc là dict gồm các key: Mã số, họ tên, số điện thoại, lương. Xây dựng các hàm thực hiện các chức năng sau:

- a. Nhập dữ liệu vào file Nhanvien.dat theo đúng cấu trúc đã cho.
- b. Hiển thị tất cả nhân viên có lương lớn hơn M, với M là tham số đầu vào của hàm.
- c. Cho biết họ tên của các nhân viên có số điện thoại bắt đầu là 0907.
- d. Tính tổng lương của tất cả nhân viên.

Hãy viết chương trình để kiểm tra tính đúng đắn của các hàm trên.

TÀI LIỆU THAM KHẢO



- [1] Mike McGrath, *Python in easy steps – 2nd edition*, 2018
- [2] Ryan Marvin, Mark Ng'ang'a, and Amos Omondi, *Python Fundamentals*, 2018.
- [3] Clive Campbell, *Python for beginner – A step to step guide to learn Python from zero in just 5 days*, 2019.
- [4] Herbert Schildt - Trường Đại học FPT biên dịch, *Lập trình hướng đối tượng – Java: A Beginner's Guide*, Nhà xuất bản Bách khoa Hà Nội, 2019.
- [5] GS. Phạm Văn Ất, *C++ và lập trình hướng đối tượng*, Nhà xuất bản Khoa học và Kỹ thuật Hà Nội, 1999.
- [6] ThS. Nguyễn Vạn Năng, *Giáo trình Lập trình hướng đối tượng bằng C++*, Khoa Công nghệ thông tin – Trường Đại học Sư Phạm Kỹ thuật Vĩnh Long, 2017.
- [7] Các trang Web có liên quan đến môn học.

MỤC LỤC



Chương 1: TỔNG QUAN VỀ NGÔN NGỮ PYTHON	Trang 1
1. GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH PYTHON.....	Trang 1
2. CÁC THÀNH PHẦN CƠ BẢN TRONG PYTHON.....	Trang 3
3. CÁC BƯỚC ĐỀ TẠO VÀ THỰC THI MỘT CHƯƠNG TRÌNH.....	Trang 4
4. NHẬP, XUẤT TRONG PYTHON	Trang 9
5. CÁC KIỂU DỮ LIỆU CƠ BẢN	Trang 10
6. BIẾN.....	Trang 11
7. CÁC TOÁN TỬ TRONG PYTHON	Trang 12
8. CÁC HÀM THƯỜNG DÙNG TRONG PYTHON.....	Trang 13
BÀI TẬP CHƯƠNG 1	Trang 14
Chương 2: CÁC CẤU TRÚC ĐIỀU KHIỂN.....	Trang 16
1. CẤU TRÚC Rẽ NHÁNH	Trang 16
2. CẤU TRÚC LẶP	Trang 18
3. LỖI, NGOẠI LỆ VÀ XỬ LÝ NGOẠI LỆ.....	Trang 22
BÀI TẬP CHƯƠNG 2	Trang 28
Chương 3: DỮ LIỆU KIỂU CHUỖI	Trang 30
1. CHUỖI TRONG PYTHON	Trang 30
2. THAO TÁC TRÊN CHUỖI.....	Trang 32
3. CÁC CHƯƠNG TRÌNH ỨNG DỤNG.....	Trang 37
BÀI TẬP CHƯƠNG 3	Trang 39
Chương 4: DỮ LIỆU KIỂU TẬP HỢP	Trang 42
1. DỮ LIỆU KIỂU LIST	Trang 42
2. DỮ LIỆU KIỂU TUPLE	Trang 51
3. DỮ LIỆU KIỂU SET	Trang 53
4. DỮ LIỆU KIỂU DICT	Trang 54
BÀI TẬP CHƯƠNG 4	Trang 61
Chương 5: HÀM VÀ MODULE	Trang 65
1. HÀM.....	Trang 65
2. MODULE.....	Trang 77
BÀI TẬP CHƯƠNG 5	Trang 81

Chương 6: TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	Trang 84
1. LẬP TRÌNH HƯỚNG CẤU TRÚC.....	Trang 84
2. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	Trang 84
3. CÁC THÀNH PHẦN CƠ BẢN TRONG LẬP TRÌNH HĐT	Trang 85
4. CÁC NGUYÊN LÝ CƠ BẢN TRONG LẬP TRÌNH HĐT.....	Trang 90
5. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG PYTHON	Trang 92
BÀI TẬP CHƯƠNG 6	Trang 100
Chương 7: LỚP VÀ ĐỐI TƯỢNG	Trang 101
1. KHÁI NIỆM LỚP	Trang 101
2. CÁC THÀNH PHẦN CỦA LỚP	Trang 103
3. CÁC THAO TÁC CƠ BẢN.....	Trang 108
4. CÁC CHƯƠNG TRÌNH ỨNG DỤNG.....	Trang 112
BÀI TẬP CHƯƠNG 7	Trang 116
Chương 8: TÍNH KẾ THỪA.....	Trang 117
1. GIỚI THIỆU CHUNG	Trang 117
2. CÁC LOẠI KẾ THỪA TRONG PYTHON.....	Trang 119
3. CÁC THAO TÁC CÓ LIÊN QUAN.....	Trang 127
BÀI TẬP CHƯƠNG 8	Trang 131
Chương 9: QUÁ TẢI TOÁN TỬ'	Trang 133
1. GIỚI THIỆU CHUNG	Trang 133
2. CẤU TRÚC CHUNG CỦA MỘT HÀM QUÁ TẢI TOÁN TỬ'.....	Trang 134
3. CÁC VÍ DỤ MINH HỌA.....	Trang 135
BÀI TẬP CHƯƠNG 9	Trang 142
Chương 10: LÀM VIỆC VỚI TẬP TIN VÀ THƯ MỤC	Trang 144
1. TẬP TIN.....	Trang 144
2. THƯ MỤC.....	Trang 149
3. CÁC MODULE CÓ LIÊN QUAN.....	Trang 151
BÀI TẬP CHƯƠNG 10	Trang 156
TÀI LIỆU THAM KHẢO.....	Trang 158