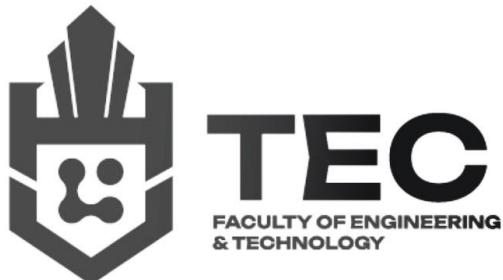


CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM  
Độc lập - Tự do - Hạnh phúc

**TRƯỜNG ĐẠI HỌC HÙNG VƯƠNG TP.HCM  
KHOA KỸ THUẬT CÔNG NGHỆ**



**CODE TRÍ TUỆ, KHÍ PHÁCH VỊ VUA**  
WRITING INTELLIGENT CODE, DISPLAYING KING'S METTLE

**TÀI LIỆU THỰC HÀNH**  
**MÔN: NHẬP MÔN TRÍ TUỆ NHÂN TẠO**  
**– LẬP TRÌNH PYTHON**

HỌ VÀ TÊN: .....  
LỚP: .....  
MSSV: .....

(LUU HÀNH NỘI BỘ)

## NỘI QUY PHÒNG MÁY TÍNH

1. Tuyệt đối không được hút thuốc, ăn, uống, xả rác gây mất vệ sinh trong phòng máy. Phải đóng cửa khi ra vào phòng máy có máy lạnh.
2. Tuyệt đối không được mang vào phòng máy các chất hay vật dụng có khả năng gây cháy, nổ, hung khí.
3. Đi đúng giờ, đúng ca quy định của mình. Khi vào phòng máy thực hành sinh viên phải đeo thẻ sinh viên (dùng dây mang trên cổ hoặc kẹp trên túi áo)
4. Không gây ồn ào mất trật tự, không được sử dụng các trang thiết bị của phòng máy vào các mục đích khác với việc học tập và nghiên cứu.
5. Cuối ca thực hành phải sắp xếp lại ghế, chuột, bàn phím theo đúng vị trí, tắt máy tính.
6. Không di chuyển các máy móc, thiết bị... khi không được sự đồng ý của người có thẩm quyền (bộ phận quản lý phòng máy).
7. **Chuẩn bị đầy đủ bài thực hành mà Giảng viên đã quy định. Bài chuẩn bị phải viết bằng tay và nộp cho giảng viên trước khi vào phòng máy tính.**

TUỲ THEO MỨC ĐỘ VI PHẠM, SẼ CÓ HÌNH THỨC XỬ LÝ KỶ LUẬT

# Demystifying Python Code



## Hướng dẫn cài đặt.

Cách cài đặt, sinh viên tham khảo trên Internet hoặc tại link:

<https://www.youtube.com/watch?v=lX0J8XbXs1Q>

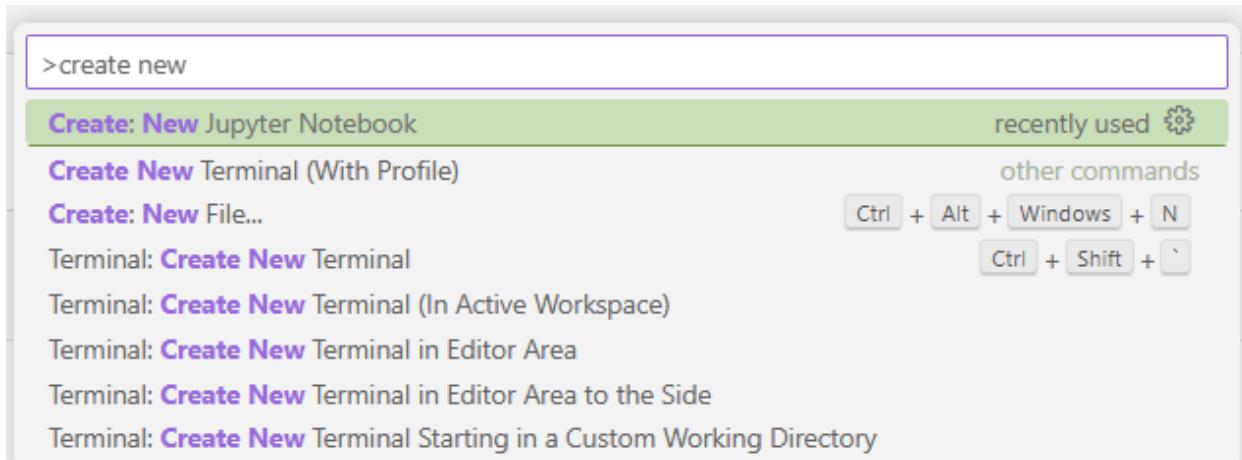
Cách tạo 1 file chương trình python như sau:

- 1) Tạo folder [tên]
- 2) Open folder trong Visual Studio Code
- 3)

Trong “Search” chọn “Show and Run Commands”



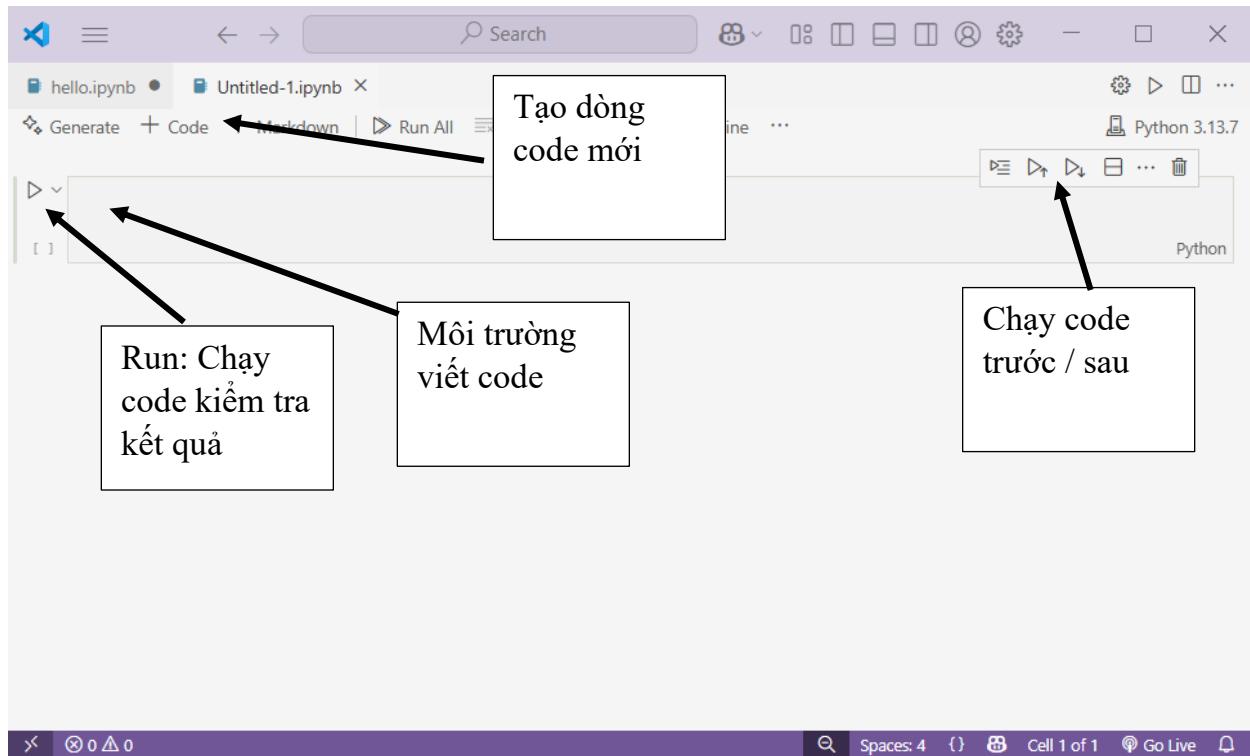
- 4) Chọn “Create new Jupyter Notebook”



- 5) Sau đó chọn File → Save as

Sau đó đặt tên [tên tự chọn] và chọn “Save”

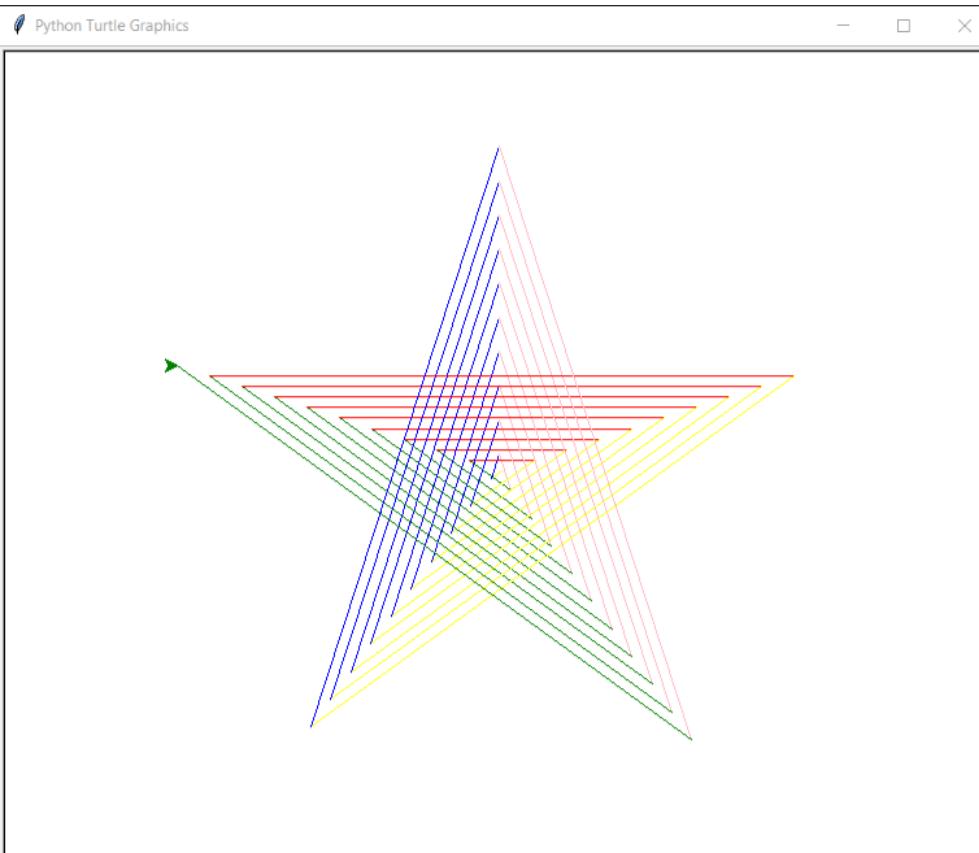
### 6) Giới thiệu về môi trường làm việc Jupyter trong VS



Test code sau:

```
import turtle  
  
t=turtle.Turtle()  
  
turtle.bgcolor("black")  
  
colors=["red","yellow","blue","pink","green"]  
  
for i in range(50):  
  
    t.color(colors[i%5])  
  
    t.forward(i*10)  
  
    t.right(144)
```

**turtle.done()**





## PHẦN 1

# TẠO CƠ BẢN

Đã đến lúc bắt đầu viết mã! Trong phần này, bạn sẽ tìm hiểu các thành phần cơ bản mà chúng ta sẽ sử dụng. Bạn sẽ tìm hiểu về chuỗi — tức là một kiểu dữ liệu chứa văn bản — và phép toán nối chuỗi, được sử dụng để kết hợp các chuỗi. Bạn cũng sẽ học cách đặt câu hỏi và cách in ra thông tin. Và quan trọng nhất, bạn sẽ học về biến. Hãy bắt đầu nào!

## I. Strings, input(), and print()

### 1. Viết văn bản: Chuỗi Strings

Trong lập trình, chúng ta sử dụng từ "chuỗi" để chỉ văn bản. Chúng ta có thể định nghĩa chuỗi như sau:

Chuỗi là văn bản nằm giữa hai dấu ngoặc kép ""

Hãy cùng xem hai ví dụ dưới đây.

```
▶ 
  "This is a string"
[1] ✓ 0.0s
```

.....

```
▶ 
  'Everything you write between quotes is a string'
[5] ✓ 0.0s
```

.....

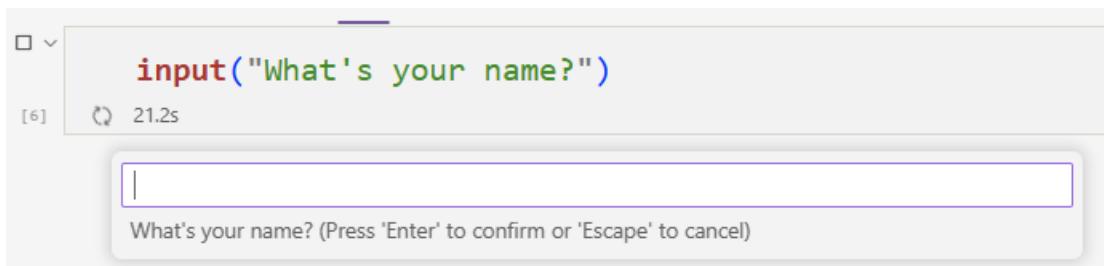
Sự khác biệt giữa 2 ví dụ trên là gì?

Hãy cùng phân tích chi tiết đoạn mã trên! Trong mỗi ô, có một chuỗi. Như chúng ta có thể thấy, một chuỗi chỉ là một đoạn văn bản nằm giữa các dấu ngoặc kép. Khi nói đến văn bản, chúng ta muốn nói đến bất kỳ ký tự nào chúng ta có thể nhập trên bàn phím: chữ cái, số, ký hiệu, và thậm chí cả dấu cách! Dấu ngoặc kép có thể là dấu ngoặc kép kép "", như trong ví dụ trên, hoặc dấu ngoặc đơn '', như trong ví dụ dưới. Dấu ngoặc kép bắt đầu một chuỗi được gọi là dấu ngoặc kép mở, trong khi dấu ngoặc kép kết thúc một chuỗi được gọi là dấu ngoặc kép đóng. Khi viết một chuỗi trong Python, chúng ta có thể sử dụng dấu ngoặc kép kép hoặc dấu ngoặc đơn; chỉ cần đảm bảo không nhầm lẫn chúng. Nói cách khác, nếu chúng ta bắt đầu viết một chuỗi bằng dấu ngoặc kép mở, chúng ta phải kết thúc chuỗi bằng dấu ngoặc kép đóng. Tương tự, nếu chúng ta bắt đầu viết một chuỗi bằng dấu ngoặc đơn mở, chúng ta phải kết thúc chuỗi bằng dấu ngoặc đơn đóng. Chuỗi là một kiểu

dữ liệu Python, có nghĩa là chúng là một trong những thành phần cốt lõi của ngôn ngữ Python.

## 2. Đặt câu hỏi: input()

Trong tất cả các ngôn ngữ lập trình, đều có cách để đặt câu hỏi cho một người, mà chúng ta thường gọi là người dùng. Đây là một tính năng rất quan trọng vì nó cho phép tương tác giữa máy tính và con người. Điều này có nghĩa là gì? Xét ví dụ sau:



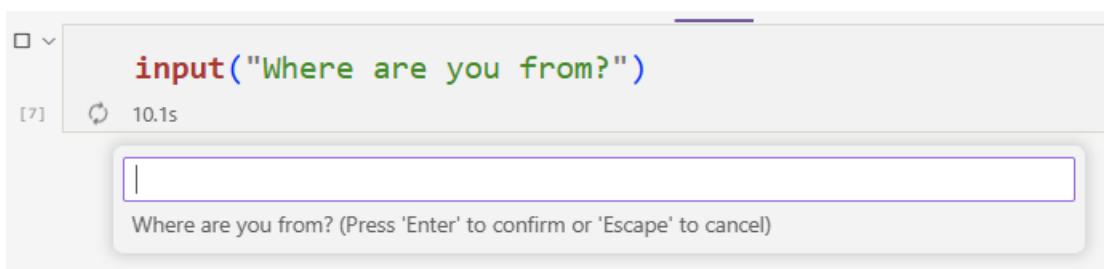
A screenshot of a Jupyter Notebook interface. A code cell at the top contains the Python command `input("What's your name?")`. Below it, a terminal window shows the question "What's your name? (Press 'Enter' to confirm or 'Escape' to cancel)" with a cursor. The entire interface is set against a light gray background.

Sau đó bạn đánh tên vào, ví dụ



A screenshot of a Jupyter Notebook interface. A code cell at the top contains the Python command `input("What's your name?")`. Below it, the output shows the user input "'Dung'" in green. The entire interface is set against a light gray background.

Ví dụ:



A screenshot of a Jupyter Notebook interface. A code cell at the top contains the Python command `input("Where are you from?")`. Below it, a terminal window shows the question "Where are you from? (Press 'Enter' to confirm or 'Escape' to cancel)" with a cursor. The entire interface is set against a light gray background.

Kết quả: .....

Để hoàn tất việc chạy ô và thực thi mã, chúng ta phải nhấn **return** hoặc **enter** sau khi nhập câu trả lời. Nếu việc chạy ô chưa hoàn tất, mã trong ô sẽ không được thực thi, và ngoài ra, chúng ta sẽ không thể chạy các ô tiếp theo.

Lưu ý:

Một hàm tích hợp là một lệnh thực hiện một nhiệm vụ cụ thể

Chúng ta có thể nhận biết một phần tử mã là một hàm dựng sẵn hay không thông qua hai đặc điểm. Thứ nhất, các hàm dựng sẵn luôn có màu đỏ như trên. Thứ hai, các hàm dựng sẵn luôn được theo sau bởi các hàm cha (). Trong cuốn sách này, thay vì cha, chúng ta sẽ gọi chúng là dấu ngoặc tròn, để phân biệt với các loại dấu ngoặc khác mà chúng ta sẽ gặp trong các chương tiếp theo. Giữa các dấu ngoặc tròn, chúng ta thường viết một đối số, đối với *input()*, đó là một chuỗi chứa câu hỏi chúng ta muốn hỏi. Các hàm dựng sẵn rất hữu ích, vì chúng chứa mã do những người tạo ra ngôn ngữ lập trình viết ra để giúp việc viết mã dễ dàng hơn.

### 3. ASCII art: print()

Giờ chúng ta đã biết cách đặt câu hỏi cho người dùng, nhưng làm thế nào để cung cấp cho họ một thông tin? Chúng ta Sử dụng hàm *print()* tích hợp! Có một số cách để tìm hiểu về *print()*, và cách sau đây thực sự rất thú vị. Nó liên quan đến một loại hình nghệ thuật kỹ thuật số gọi là nghệ thuật ASCII, trong đó hình ảnh có thể được tạo ra bằng cách sử dụng các ký hiệu trên bàn phím. Hãy cùng xem xét ô sau:

Ví dụ:



```
print("/_\/\ ")
print(">^.^< ")
print(" / \ ")
print("(____)___")
```

[8]

Kết quả:

.....

.....

.....

.....  
.....  
.....  
.....

Nếu chúng ta xóa hàm `print()` khỏi mã ở ô 5, nó sẽ chỉ hiển thị chuỗi cuối cùng:

```
▶ ▾
  "/\_\_/"  
">^.^<"  
"/ \ "  
"(\_)_"  
[8] ✓ 0.0s
```

Kết quả:

.....  
.....  
.....

**Thực hành: Sinh viên vẽ các hình sau:**

/ /  
/ . . \  
/ \ \ /  
/ \ \ /  
/ \ \ /  
/ \ \ /

\_\_\_\_\_ / \\_\\_/\  
| Vvvv ^^\\_ ) vvvV

-	( -			
	\_	-		
	- \	-		

@@@@ @@@ ( - )  
@ @ ( ) @ @ w W W W W ( - ) \\\  
@ @ @ @ ( \_\_ ) Y \ / /  
| / \ / / / \ \ \ | / /

\ || | /  
( o o )  
| ~~~oo0~~( \_ )~~~~~ |  
| Write  
| your favorite  
| sentence here  
| ~~~~~~0oo~~ |  
| \_ | \_ |  
| | | |  
oo0 0oo

( - \ ) / -  
- \ - . / - - -

## 2. Sự kiện và mục ưa thích

Biến, phép gán và nối chuỗi

*Variables, assignment, and string concatenation*

### 1. Tổ chức sự kiện

PHIẾU ĐĂNG KÝ
Họ: .....
Tên:.....

- Người tham gia đầu tiên đến và bạn điền vào mẫu đơn:

```
▷ ▾
  Ho="Nguyen"
  ten="Dung"
[9] ✓ 0.0s
```

Sau đó, bạn in ra những thông tin bạn đã nhập vào mẫu đăng ký:

```
▷ ▾
  print(Ho)
  print(ten)
[10] ✓ 0.0s
...
  Nguyen
  Dung
```

Sinh viên tự điền tên mình, kết quả:

.....  
.....  
.....

Biến là một nhãn được liên kết với một giá trị.

Trong Python, các biến được viết thường. Khi được tạo thành từ nhiều từ, chúng được nối với nhau bằng dấu gạch dưới, như trong họ. Trong Jupyter Notebook, các biến được tô màu đen. **Ký hiệu =** **được gọi là toán tử gán**. Điều này không liên quan gì đến toán tử bằng mà chúng ta đã học trong toán! Trong mã hóa, chúng ta sử dụng ký hiệu = để gán giá trị cho một biến và chúng ta phát âm nó là được gán.

## 2. Kết hợp

Đã đến lúc tổng hợp những gì chúng ta đã học! Hãy cùng đọc đoạn mã sau:

```
[11]    ten=input("Ban ten la gi? ")
          food=input("mon an yeu thich cua ban la gi? ")
          ✓ 6.2s
```

Sau đó nhập vào, ví dụ kết quả là:

```
[12]    print("Ban ten la: " + ten)
          print("Mon an yeu thich cua ban la: " + food)
          ✓ 0.0s
...
... Ban ten la: Dung
      Mon an yeu thich cua ban la: Pho
```

Sinh viên ghi kết quả của mình:

.....  
.....

```
[13]    print("Toi la " + ten + ". Mon an yeu thich cua toi la " + food)
          ✓ 0.0s
...
... Toi la Dung. Mon an yeu thich cua toi la Pho
```

Kết quả của sinh viên

.....



## PHẦN 2

# GIỚI THIỆU VỀ DANH SÁCH VÀ CẤU TRÚC

## IF/ELSE

Trong phần này, bạn sẽ tìm hiểu về danh sách, vốn đơn giản là chuỗi các phần tử thuộc nhiều kiểu khác nhau—ví dụ: chuỗi. Bạn cũng sẽ học cách thao tác với chúng, tức là cách thêm, xóa hoặc thay thế một hoặc nhiều phần tử. Và cuối cùng, bạn sẽ học các cấu trúc *if/else*, cho phép thực thi mã dựa trên các điều kiện.

## 1. Xét ví dụ trong nhà sách

- Bạn là chủ một hiệu sách. Trên kệ sách lập trình có:

```
sach=[ "Learn Python", "Python for all", "Intro to Python"]
print(sach)
```

Kết quả: .....

- Một khách hàng mới đến và bạn hỏi cô ấy muốn mua cuốn sách nào:

```
> ~
  muasach=input("Ban muon mua sach ten gi? ")
  print(muasach)
[ ]
```

Kết quả: .....

- Bạn kiểm tra xem mình có cuốn sách đó không và trả lời tương ứng:

```
> ~
  if muasach in sach:
    print("Vang! chung toi ban no.")
  else:
    print("Xin loi, chung toi khong ban no.")
[ ]
```

Kết quả: .....

Bây giờ chúng ta chạy lại phần mua sách và chạy lại hàm *if ... in/else ...*

```
muasach=input("Ban muon mua sach ten gi? ")
print(muasach)
[19]   ✓ 11.4s
...
learn Python

▷ ▾
if muasach in sach:
    print("Vang! chung toi ban no.")
else:
    print("Xin loi, chung toi khong ban no.")
[20]   ✓ 0.0s
...
Xin loi, chung toi khong ban no.
```

Rõ ràng, sai 1 ký tự hoa và thương, đáp án cũng tương ứng theo.

Danh sách được định nghĩa như sau:

**Danh sách** là một chuỗi các phần tử được phân tách bằng dấu phẩy,  
và nằm giữa hai dấu ngoặc vuông [].

chúng ta có thể định nghĩa cấu trúc if/else như sau:

Cấu trúc **if/else** kiểm tra xem một điều kiện là đúng hay sai,

và thực thi mã tương ứng:

nếu điều kiện **được** đáp ứng, mã **trong** điều kiện **if** được thực thi;

nếu điều kiện **không** được đáp ứng, mã trong **else** được thực thi

## **Bài tập thực hành:**

- Trong một phòng trưng bày nghệ thuật. Bạn là chủ sở hữu của một phòng trưng bày nghệ thuật. Hãy viết một danh sách các bức tranh bạn đang bán. Một khách hàng mới đến, và bạn hỏi cô ấy muốn mua bức tranh nào. Bạn kiểm tra xem bạn có bức tranh đó không và trả lời tương ứng.

list=[“hoa”, “ong mat troi”, “sao”, “diem vuong”, “trai dat”]

2. Trong một công ty du lịch. Bạn là chủ sở hữu của một công ty du lịch. Hãy viết một danh sách các điểm đến du lịch mà bạn đang bán vé. Một khách hàng mới đến, và bạn hỏi cô ấy muốn đi đâu. Bạn kiểm tra xem bạn có cung cấp điểm đến du lịch đó không và trả lời tương ứng.

List=[“da lat”, “da nang”, “ha noi”, “nha trang”, “ben tre”]

3. Trong một phòng thí nghiệm hóa học. Bạn là quản lý của một phòng thí nghiệm. Trên kệ có một số lọ đựng hóa chất. Viết một danh sách tên các loại hóa chất. Một thành viên trong phòng thí nghiệm đến gặp bạn và bạn hỏi cô ấy muốn mua loại hóa chất nào. Bạn kiểm tra trong hệ thống xem bạn có loại hóa chất đó không và trả lời tương ứng.

List=[“co2”, “phot pho”, “luu huynh”, “oxi”, “hydro”]

4. Trong một phòng trà. Bạn là chủ sở hữu của một phòng trà. Hãy viết một danh sách các loại trà bạn đang cung cấp. Một khách hàng mới đến, bạn hỏi anh ấy muốn uống loại trà nào. Bạn kiểm tra trên thực đơn xem mình có phục vụ loại trà đó không và trả lời theo đúng yêu cầu.

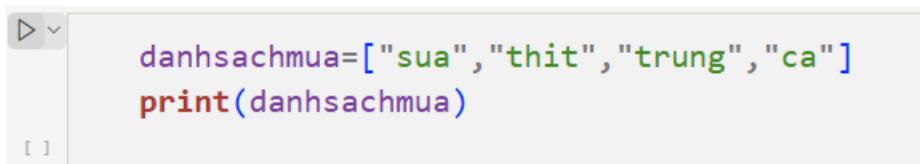
List=[“oolong”, “tam coc”, “tra dao”, “tra gung”, “tra da”, “tra cuc”, “tra sen”]

#### 4. Mua sắm hàng tạp hóa

Liệt kê các phương thức: `.append()` và `.remove()`

Các phương thức này là gì? Và `.append()` và `.remove()` làm gì? Để trả lời những câu hỏi này. Hãy bắt đầu với ví dụ sau:

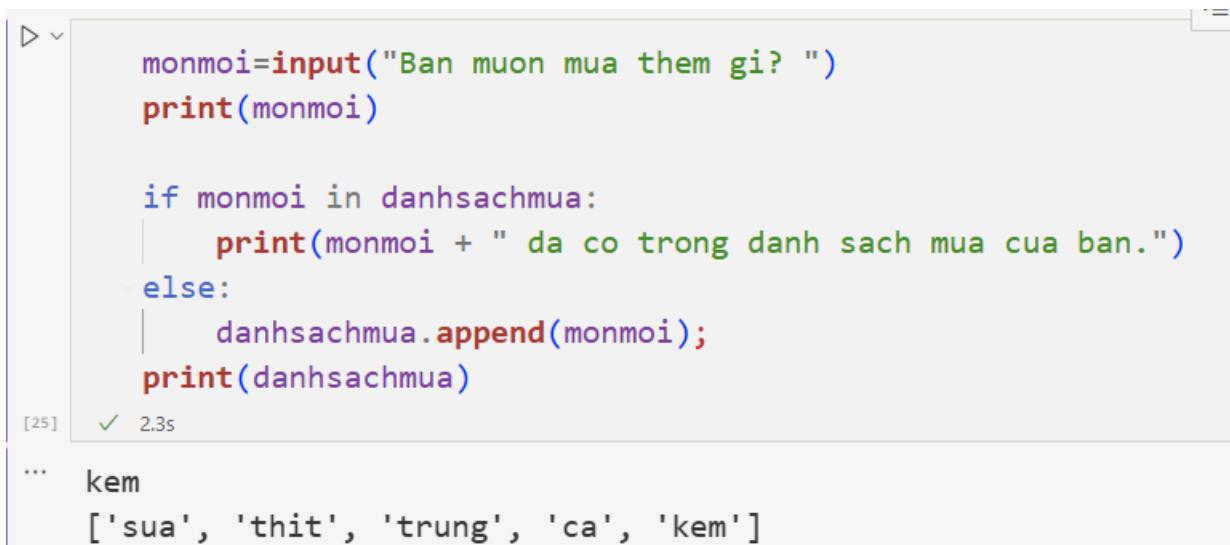
- Bạn đang đến một cửa hàng tạp hóa và cần mua một ít thực phẩm:



```
danh sach mua = ["sua", "thit", "trung", "ca"]
print(danh sach mua)
```

Kết quả: .....

- Ngay trước khi rời khỏi nhà, bạn tự hỏi liệu mình có cần mua thêm thứ gì không. Nếu món đồ đó không có trong danh sách, bạn hãy thêm nó vào:



```
mon moi = input("Ban muon mua them gi? ")
print(mon moi)

if mon moi in danh sach mua:
    print(mon moi + " da co trong danh sach mua cua ban.")
else:
    danh sach mua.append(mon moi);
print(danh sach mua)
[25]   ✓  2.3s
...
kem
['sua', 'thit', 'trung', 'ca', 'kem']
```

Kết quả: .....

Chạy thử kết quả lại trên.

- Cuối cùng, bạn tự hỏi liệu mình có cần xóa một mục nào đó không. Nếu có, bạn xóa mục đó khỏi danh sách:

The screenshot shows a Python code editor window. The code is as follows:

```
mon_xoa=input("Ban muon xoa mon nao? ")
print(mon_xoa)

if mon_xoa in danh sachmua:
    danh sachmua.remove(mon_xoa)

    print(mon_xoa + " da duoc xoa khoi danh sach.")
else:
    print(mon_xoa + " Khong co trong danh sach mua")
print(danh sachmua)
```

At the bottom left, there is a status bar showing [27] and 14.1s. On the right, it says Python. A message box is displayed at the bottom center with the text: "Ban muon xoa mon nao? (Press 'Enter' to confirm or 'Escape' to cancel)".

Kết quả: .....

Phương thức (method) là gì? Định nghĩa sơ bộ—chúng ta sẽ định nghĩa lại khi nói về lập trình hướng đối tượng, trong phần cuối của cuốn sách—như sau:

Phương thức (method) là một hàm tích hợp cho một kiểu (type) biến cụ thể.

- Phương thức .append() thêm một phần tử vào cuối danh sách.
- Phương thức .remove() xóa một phần tử khỏi danh sách.

### **Bài tập áp dụng:**

Đối với mỗi tình huống sau, hãy tạo mã tương tự như mã được trình bày trong chương này.

- a. Tổ chức một sự kiện. Bạn đang tổ chức một sự kiện. Hãy viết một danh sách những thứ bạn cần mua. Sau đó, hãy hỏi người đồng tổ chức xem bạn cần mua thêm gì nữa. Nếu món đồ đó không có trong danh sách, hãy thêm nó vào. Cuối cùng, hãy hỏi người đồng tổ chức xem có thứ gì bạn cần loại bỏ khỏi danh sách không. Nếu có, hãy loại bỏ món đồ đó khỏi danh sách.

b. Các thành phố yêu thích. Hãy viết một danh sách gồm tên các thành phố. Hỏi một người bạn về thành phố yêu thích của họ. Nếu thành phố đó không có trong danh sách, hãy thêm nó vào. Sau đó, hãy hỏi bạn bè xem họ có thích một trong những thành phố bạn đã liệt kê không. Nếu thích, hãy xóa thành phố đó khỏi danh sách của bạn.

## 5. Tùy chỉnh menu

Liệt kê các phương thức: `.index()`, `.pop()` và `.insert()`

- Bạn đang ở một khu ẩm thực, sẵn sàng gọi món. Thực đơn hôm nay bao gồm một chiếc burger, một món ăn kèm và một đồ uống:



```
▶ 
menu=["burger", "coca", "salad"]
print(menu)
[28] ✓ 0.0s
```

Kết quả: .....

*Burger* sẽ có vị trí 0, *coca* có vị trí 1, *salad* có vị trí 2

- Bạn hài lòng với *burger* và *coca*, nhưng muốn đổi món ăn kèm từ *salad* sang *khoai tây chiên*. Để làm vậy, bạn:

1. Xem vị trí của món ăn kèm salad trong thực đơn:



```
▶ 
sothutu=menu.index("salad")
print(sothutu)
[29] ✓ 0.0s
```

Kết quả: .....

2. Bỏ salad ra khỏi vị trí món ăn kèm:

```
▷ ▾  
    menu.pop(sothutu)  
    print(menu)  
[30] ✓ 0.0s
```

Kết quả: .....

3. Thêm *khoai tây chiên* vào vị trí *món ăn kèm*:

```
▷ ▾  
    menu.insert(sothutu, "khoai tay chiem")  
    print(menu)  
[✓ 0.0s
```

Kết quả: .....

4. Chèn *kem* vào vị trí 1 như sau:

```
▷ ▾  
    menu.insert(1, "kem")  
    print(menu)  
[32] ✓ 0.0s
```

Kết quả: .....

- Phương thức *.index()* trả về vị trí của một phần tử trong danh sách.
- Phương thức *.pop()* xóa một phần tử ở một vị trí nhất định khỏi danh sách.
- Phương thức *.insert()* thêm một phần tử ở một vị trí nhất định vào danh sách.
- Chỉ số (hoặc **vị trí**) của các phần tử là các số nguyên bắt đầu từ **0** và tăng dần theo từng đơn vị; chúng có kiểu là **số nguyên**.

### **Bài tập áp dụng:**

1. Steve Jobs. Cho danh sách sau:

```
steve_jobs = ["somebody", "learn", "use", "a computer", "it teaches us"]
```

Tìm một câu nói nổi tiếng của Steve Jobs bằng cách làm như sau:

- a. Thêm chuỗi mới "think" vào cuối danh sách.
  - b. Thêm "should" vào vị trí 1.
  - c. Thêm "how to" vào vị trí 3. Sau đó, thêm nó vào vị trí 7.
  - d. Thay "use" bằng "program".
  - e. Thêm "because" sau "a computer".
  - f. Thay "somebody" bằng "everybody".
  - g. Thêm "-Steve Jobs" vào cuối danh sách.
2. Grace Hopper. Bạn có biết tại sao chúng ta lại nói đến gỡ lỗi trong lập trình không?  
Hãy cùng tìm hiểu! Cho danh sách sau:

```
grace_hopper = ["In 1946", "a moth", "caused", "a malfunction", "in an early",
"electromechanical", "computer"]
```

Hãy sửa lại bằng cách làm như sau:

- a. Replace "In 1946" with "From then on".
- b. Add "we said" after "computer".
- c. Remove the string in position 5 (6th element) and add "with a" in the same position.
- d. Remove the string in position 3 (4th element).
- e. Substitute (or replace) "a moth" with "when anything".
- f. Remove "in an early".
- g. Add "it had bugs in it" at the end of the list.
- h. Substitute "caused" with "went wrong".
- i. Add " - Grace Hopper" at the end of the list.

## 6. Du lịch vòng quanh thế giới

Cắt nhỏ danh sách *List slicing*

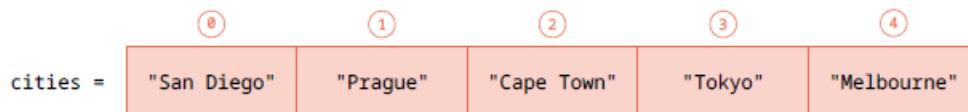
Cắt lát *slicing* có nghĩa là truy cập các phần tử danh sách thông qua chỉ mục của chúng

- Chúng ta hãy cùng xem danh sách chúng ta sẽ cắt ra:

```
cities = ["San Diego", "Prague", "Cape Town", "Tokyo", "Melbourne"]
print(cities)
```

Python

Kết quả: .....



1. Cắt "Prague":

```
[35]: print(cities[1])
```

0.0s

Kết quả: .....

2. Cắt các thành phố từ "Prague" đến "Tokyo":

```
[36]: print(cities[1:4])
```

0.0s

Kết quả: .....

Chú ý không phải là 1:3?

Trong ví dụ này, vị trí của phần tử cuối cùng ("Tokyo") là 3, chúng ta phải cộng thêm 1 vào đó vì quy tắc cộng một, do đó điểm dừng là 4. Chúng ta có thể tóm tắt cú pháp để cắt

các phần tử liên tiếp thành **list\_name[start:stop]**, và chúng ta có thể đọc nó như tên danh sách theo các vị trí từ đầu đến cuối.

3. Cắt “*Prague*” và “*Tokyo*”:



Kết quả: .....

**list\_name[start:stop:step]**, bạn có thể đọc là tên danh sách *từ start đến stop với step*.

Chúng ta có thể gọi nó là quy tắc ba chữ số, trong đó ba chữ số s lần lượt là chữ cái đầu của *start*, *stop* và *step*.

**Sinh viên thực hiện yêu cầu và viết lại code yêu cầu sau:**

1) print cities in positions from one to four with a step of one

Lệnh: .....

Kết quả: .....

2) print cities in positions from zero to three

Lệnh: .....

Kết quả: .....

3) print cities from the beginning of the list to position three

Lệnh: .....

Kết quả: .....

Cách viết rút gọn: `print(cities[:3])`

4) print cities in positions from two to five

Lệnh: .....

Kết quả: .....

5) print cities from position two to the end of the list

Lệnh: .....

Kết quả: .....

Cách viết rút gọn: print(cities[2:])

6) print cities in positions from zero to five with a step of two

Lệnh: .....

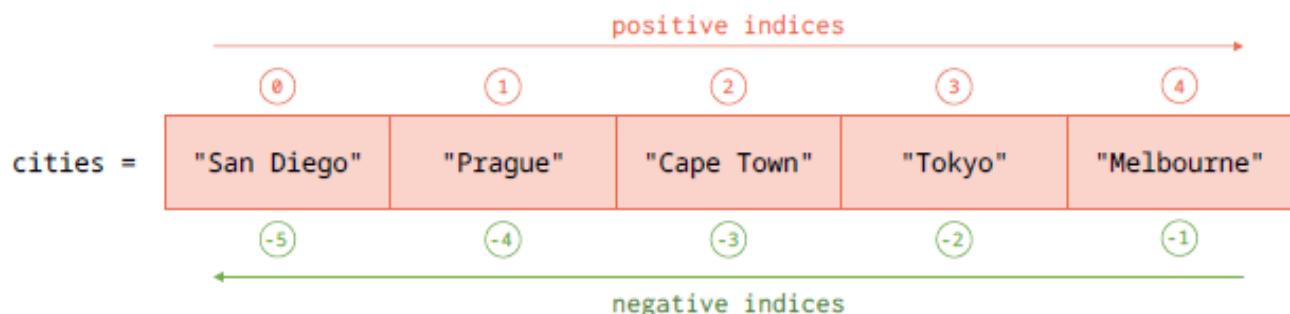
Kết quả: .....

7) print cities from the beginning to the end of the list with a step of two

Lệnh: .....

Kết quả: .....

Cách viết rút gọn: print(cities[::2])



1) Cắt "Melbourne":

```
▶ 
print(cities[4])
[38]   ✓  0.0s
```

Kết quả: .....

```
[49] ⌄ print(cities[-1])  
[49] ✓ 0.0s
```

Kết quả: .....

2) Cắt tất cả các thành phố từ "Prague" đến "Tokyo" bằng cách sử dụng chỉ số âm:

```
[50] ⌄ print(cities[-4:-1])  
[50] ✓ 0.0s
```

Kết quả: .....

3) Cắt tất cả các thành phố từ "Tokyo" đến "Prague" theo chỉ số dương (thứ tự ngược lại):

```
[51] ⌄ print(cities[3:0:-1])  
[51] ✓ 0.0s
```

Kết quả: .....

4) Cắt tất cả các thành phố từ "Tokyo" đến "Prague" bằng cách sử dụng chỉ số **âm** (thứ tự ngược lại):

```
[52] ⌄ print(cities[-2:-5:-1])  
[52] ✓ 0.0s
```

Kết quả: .....

5) Cắt tất cả các thành phố (**theo thứ tự ngược lại**):

```
[53] ⌄ print(cities[::-1])  
[53] ✓ 0.0s
```

Kết quả: .....

- Để cắt một phần tử, chúng ta sử dụng quy tắc: **list\_name[element\_position]**.
- Để cắt nhiều phần tử, chúng ta sử dụng quy tắc ba chữ số:  
**list\_name[start:stop:step]**, trong đó:
  - Chúng ta có thể bỏ qua start khi cắt từ phần tử đầu tiên của danh sách, stop khi cắt đến phần tử cuối cùng của danh sách và step khi cắt các phần tử liên tiếp của danh sách.
  - Stop tuân theo **quy tắc cộng** một khi cắt từ trái sang phải (thứ tự trực tiếp) và **quy tắc trừ** một khi cắt từ phải sang trái (thứ tự ngược lại).
    - Các giá trị của element\_position, start, stop và step có thể là:
      - **Dương:** khi xem xét các phần tử từ trái sang phải (thứ tự trực tiếp).
      - **Âm:** khi xem xét các phần tử từ phải sang trái (thứ tự ngược lại).
    - Các bước âm được sử dụng để đảo ngược danh sách.

### Bài tập áp dụng:

#### 1. *Fruits and veggies.*

Traicay\_raucu= ["ót chuông", "mơ", "cà rốt", "táo", "bí ngòi", "nho", "bắp cải", "cam", "măng tây", "lê"]

Dùng dao cắt lát để lấy:

- Sản phẩm giữa táo và nho (bao gồm);
- Tất cả các loại rau củ;
- Tất cả các loại trái cây;
- Các loại rau củ giữa cà rốt và măng tây (bao gồm);

e. Các loại trái cây giữa táo và cam (bao gồm).

2. *Clothes, stationery, and electronics.* Given the following list:

```
objects = ["mobile", "t_shirt", "pencil", "laptop", "hat", "ruler", "tv", "pants", "pen"]
```

Use slicing to extract:

a. All the clothes;

b. All the stationery;

c. All the electronics;

d. The second and the last stationery items;

e. The first and the last electronics items;

f. The first and the second clothing items.

3. *Interior design.* Given the following list: `interior_design = ["sofa", "curtain", "lamp", "table", "carpet", "plant", "armchair", "blanket", "vase"]`

Use slicing to extract the following elements in *direct* order (from left to right), once using *positive*

indices and once using *negative* indices:

a. All furniture;

b. All textiles;

c. All decorative elements;

d. The pieces composed of 5 letters (count them by hand, no coding required).

4. *Botanic garden.* Given the following list:

```
botanic_garden = ["tulip", "pine", "poppy", "palm", "rose", "oak", "daisy", "eucalyptus"]
```

Use slicing to extract the following elements, once in *direct* order (from left to right) and once in

*inverse* order (from right to left):

- a. All flowers;
- b. All trees;
- c. All flowers and trees starting with *p* (find them by hand, no coding required);
- d. "pine", "rose", and

## 7. Giác quan, hành tinh và ngôi nhà

Thay đổi, thêm và xóa các phần tử danh sách bằng cách sử dụng cắt lát

*Changing, adding, and removing list elements using slicing*

### 1. Giác quan

Trước tiên, chúng ta hãy tìm hiểu cách thay đổi các phần tử danh sách bằng cách sử dụng cắt và gán.

- Cho danh sách sau:

```
senses = ["eyes", "nose", "ears", "tongue", "skin"]
print(senses)
```

[44] 0.0s

Kết quả: .....

- Thay “*nose*” bằng “*smell*”: *nose* đang ở vị trí (index) là 1

```
senses[1] = "smell"
print(senses)
```

[45] 0.0s

Kết quả: .....

Trong trường hợp chúng ta không biết vị trí của từ cần thay, chúng ta có thể đi tìm vị trí và xoá nó đi, sau đó chèn từ mới vào vị trí đó.

Bước 1: Chạy lại danh sách

Bước 2:

```
timvitri = senses.index("nose")
#xoa
senses.pop(timvitri)
#them
senses.insert(timvitri,"smell")

print(senses)
```

[48] ✓ 0.0s

Kết quả: .....

- Thay "tongue" và "skin" với "taste" và "touch":

Nhớ lại cấu trúc `list_name[start:stop]`

```
senses[3:5] = ["taste", "touch"]

print(senses)
```

[49] ✓ 0.0s

Kết quả: .....

- thay "eyes" và "ears" với "sight" và "hearing": các giác quan "eyes" và "ears" ở các vị trí từ 0 đến 3 với bước nhảy hai

```
senses[0:3:2] = ["sight", "hearing"]
print(senses)
[50]    ✓ 0.0s
```

Kết quả: .....

## 2. Hành tinh

Để thêm phần tử mới vào danh sách, chúng ta có thể sử dụng phép cắt kết hợp với phép nối và phép gán danh sách.

Bằng cách nào? Hãy cùng xem các ví dụ sau!

- Cho danh sách sau:

```
planets = ["Mercury", "Mars", "Earth", "Neptune"]
print(planets)
[51]    ✓ 0.0s
```

Kết quả: .....

- Thêm “*Jupiter*” vào cuối danh sách:

```
planets = planets + ["Jupiter"]
print(planets)
[52]    ✓ 0.0s
```

Kết quả: .....

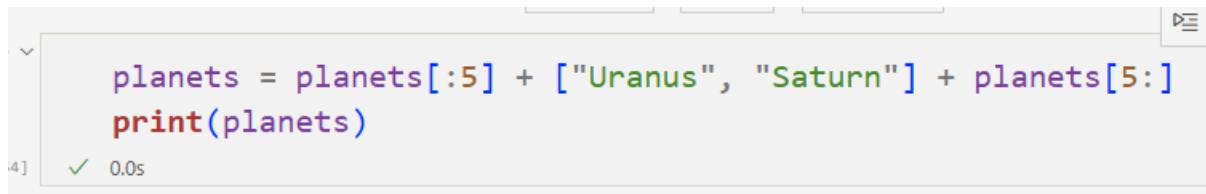
- Thêm “*Venus*” vào giữa “*Mars*” và “*Earth*”:

```
planets = planets[0:2] + ["Venus"] + planets[2:5]
print(planets)
[53]    ✓ 0.0s
```

Kết quả: .....

- Thêm "*Uranus*" và "*Saturn*" vào giữa "*Neptune*" and "*Jupiter*":

Các hành tinh được gán từ đầu danh sách đến vị trí thứ năm nối với "*Uranus*" and "*Saturn*" nối với các hành tinh từ vị trí thứ năm đến cuối danh sách



```
planets = planets[:5] + ["Uranus", "Saturn"] + planets[5:]  
print(planets)
```

[4] ✓ 0.0s

Kết quả: .....

### 3. Một ngôi nhà

Để xóa các phần tử trong danh sách, chúng ta có thể sử dụng từ khóa `del` kết hợp với phép cắt danh sách. Việc này rất dễ.

Hãy cùng xem nhé!

- Cho danh sách sau:



```
house = ["kitchen", "dining room", "living room",  
        "bedroom", "bathroom", "garden", "balcony",  
        "terrace"]  
print(house)
```

[55] ✓ 0.0s

Kết quả: .....

- Xóa "dining room":



```
del house[1]  
print(house)
```

[56] ✓ 0.0s

Kết quả: .....

- Xoá "garden" và "balcony": Vị trí từ 4 tới 6

```
del house[4:6]
print(house)
```

[57] ✓ 0.0s

Kết quả: .....

- Xoá "kitchen", "bedroom" và "terrace": xoá từ đầu đến cuối bước nhảy là 2

```
del house[::2]
print(house)
```

[58] ✓ 0.0s

Kết quả: .....

- Xoá list "house":

```
del house
print(house)
```

[59] ✘ 0.0s

...

NameError Traceback (most recent call last)  
Cell In[59], line 2  
 1 del house  
----> 2 print(house)

NameError: name 'house' is not defined

- Để thay đổi các phần tử danh sách, chúng ta có thể sử dụng *lệnh cắt* và *lệnh gán*.
- Để thêm các phần tử danh sách, chúng ta có thể kết hợp lệnh cắt, lệnh nối và lệnh gán.
- Để xóa các phần tử danh sách, chúng ta có thể sử dụng từ *khóa del* và *lệnh cắt*.
- Lệnh nối danh sách được thực hiện bằng ký hiệu + và hoạt động tương tự như lệnh nối chuỗi.

## **Bài tập áp dụng**

1. Stephanie Shirley. Do you know the story of Stephanie Shirley? Let's see what she did! Given the following list:

```
stefanie_shirley = ["In 1962", "Stephanie Shirley", "founded", "a software company",  
"employing", "only women", "working from home"]
```

Do the following using list slicing:

- Replace "founded" with "thrived".
- Remove the element in position 0 (first element).
- Replace "employing" with "transferred ownership".
- Add "and over the years" between "thrived" and "a software company".
- Replace "only women" with "to her staff".
- Insert "gradually" in position 4 (fifth element).
- Replace "a software company" with "she".
- Add "70 millionaires" at the end of the list.
- Remove "Stephanie Shirley".

- j. Replace "working from home" with creating".
- k. Insert "The business" at the beginning of the list.

Then, redo the same using list methods.

2. Tim Berners-Lee. What did Tim Berners-Lee invent? Let's find it out! Given the following list:

```
tim_bernarslee = ["Tim Berners-Lee", "invented", "the World Wide Web", "in 1989",  
"at CERN in Geneva", "info.cern.ch", "was", "the address of",  
"the world's first website and Web server"]
```

Do the following using list slicing:

- a. Remove "info.cern.ch".
- b. Replace "was" with "consists of".
- c. Remove the element in position 1 (second element).
- d. Add "all over the world" at the end of the list.
- e. Replace "the world's first website and Web server" with "about 75 million servers".
- f. Remove the element in position 0 (first element).
- g. Replace "in 1989" with "Nowadays".
- h. Remove the element in position 0 (first element).
- i. Replace "at CERN in Geneva" with "it is estimated that".
- j. Add "the internet" in position 2 (third element).
- k. Remove the element in position 4 (fifth element).

Then, redo the same using *list methods*.

3. *Alan Turing*. What happened thanks to Alan Turing's contributions? Let's discover it!

Given the following list:

```
alan_turing = ["Turing", "created", "an electromechanical machine", "to crack",  
"the Nazi Navy's", "Enigma Code"]
```

Do the following using *list slicing*:

- a. Replace "the Nazi Navy's" with "shortened the war".
- b. Insert "by two years" in position 5 (sixth element).
- c. Replace "an electromechanical machine" with "his contribution".
- d. Add "saving millions of lives" to the end.
- e. Replace "created" with "that".
- f. Remove "to crack".
- g. Replace "Turing" with "It is estimated".
- h. Remove the element in position 5 (sixth element).

Then, redo the same using *list methods*.

## PHẦN 3

### GIỚI THIỆU VỀ

### VÒNG LẶP *FOR*

Trong phần này, bạn sẽ tìm hiểu về vòng lặp *for*, một trong hai vòng lặp trong lập trình—vòng lặp còn lại là vòng lặp *while*. Bạn sẽ tìm hiểu cú pháp của vòng lặp này và cách sử dụng nó để tìm kiếm các phần tử trong danh sách, chỉnh sửa danh sách và tự động tạo danh sách mới.



## 8. Món ăn ưa thích của bạn tôi

*for...in range()*

Vòng lặp *for* là một trong những cấu trúc quan trọng nhất trong lập trình vì nó cho phép chúng ta thực hiện lặp đi lặp lại các lệnh. Điều này có nghĩa là gì và nó hoạt động như thế nào? và trả lời những câu hỏi này!

- Dưới đây là danh sách bạn bè của tôi và danh sách các món ăn yêu thích của họ:

```
[60] friends = ["An", "Binh", "Lan", "Hung"]
       monan=["banh mi","xuc xich","pho","bun bo"]
       ✓ 0.0s
```

Kết quả: .....

- Đây là tất cả bạn bè của tôi:

```
▷ 
      print("Ten nhung dua ban toi :")
      print(friends)
[61] ✓ 0.0s
```

Kết quả: .....

- Đây là những người bạn của tôi từng người một:

```
▷ 
      for stt in range(0,4):
          print("stt: " + str(stt))
          print("Ban:" + friends[stt])
[63] ✓ 0.0s
```

Kết quả: .....

- Đây là tất cả các món ăn yêu thích của họ:

```
print("Tat ca mon yeu thich:")
print(monan)
```

[64] ✓ 0.0s

Kết quả: .....

- Sau đây là những món ăn ưa thích của từng người một::

```
for stt in range(0,4):
    print("stt: " + str(stt) )
    print("Mon an: " + monan[stt])
```

[66] ✓ 0.0s

Kết quả: .....

- Đây là những người bạn của tôi, cùng với những món ăn yêu thích của họ:

```
for stt in range(0,4):
    print("Ten ban toi la " + friends[stt] +
          " va mon yeu thich la " + monan[stt])
```

[68] ✓ 0.0s

Kết quả: .....

**for...in range()** nó bao gồm năm thành phần:

- **for**: Từ khóa bắt đầu một vòng lặp *for*. Giống như tất cả các từ khóa khác, nó được in đậm màu xanh lá cây trong Jupyter Notebook.
- **index**: Một biến được gán một giá trị khác nhau ở mỗi lần lặp lại vòng lặp (chúng ta sẽ nói thêm về điều này sau).
- **in**: Một toán tử thành viên, tương tự như bạn đã học trong cấu trúc **if...in/else**

- **range()**: Một hàm Python tích hợp. Bạn có thể nhận ra đây là một hàm vì nó được theo sau bởi dấu ngoặc tròn và được tô màu xanh lá cây trong Jupyter Notebook—giống như **input()** và **print()**. Chúng ta cũng sẽ nói thêm về **range()** sau.
- : tức là dấu hai chấm.

Vòng lặp **for** là sự lặp lại của một nhóm lệnh  
với **một số lần** xác định.

Định nghĩa này tóm tắt hai đặc điểm chính của vòng lặp **for**.

1. Chúng ta thực thi các dòng mã nằm trong thân vòng lặp **for** nhiều lần.
2. Số lần lặp được biết trước và được xác định bởi một chuỗi số được tạo bởi hàm dựng sẵn **range()**
  - Vòng lặp **for** là sự lặp lại các lệnh với số lần xác định.
  - Khi vòng lặp **for** được sử dụng để cắt một danh sách, số lần lặp trùng với độ dài của danh sách.
  - Cú pháp chung của tiêu đề vòng lặp **for** là: **for index in range(start, stop, step):**.
  - Phần thân của vòng lặp **for** được **thụt lề** và có thể chứa bao nhiêu dòng mã tùy ý.
  - **range()** là một hàm Python tích hợp sẵn, tạo ra một chuỗi các số nguyên trải dài từ điểm **bắt đầu (included)** đến **điểm kết thúc (excluded)**.
  - **str()** là một hàm Python tích hợp sẵn, chuyển đổi một biến thành một chuỗi.

## **Bài tập áp dụng**

1. *Mountains and rivers*. Given the following list:

```
mountains_rivers = ["everest", "mississippi", "yosemite", "nile", "mont blanc",
"amazon"]
```

Print:

- a. All elements as a list;
- b. All elements one by one using a *for* loop;
- c. Mountains using slicing;
- d. Mountains one by one using a *for* loop (tip: remember that range() can have three arguments: start, stop, step);
- e. Rivers using slicing;
- f. Rivers one by one using a *for* loop (what start do you use?);
- g. All elements in reverse order using slicing;
- h. All elements in reverse order, one by one, using a *for* loop (what start, stop, and step do you use?).

2. *Wild animals.* Given the following list:

```
wild_animals = ["eagle", "bear", "parrot", "tiger", "pelican", "coyote"]
```

Print:

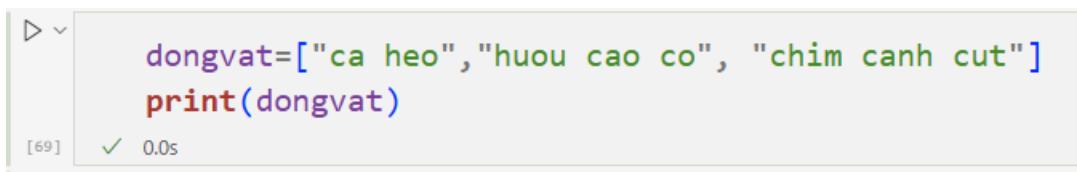
- a. All animals as a list;
- b. All animals one by one using a *for* loop;
- c. Mammals using slicing;
- d. Mammals one by one using a *for* loop;
- e. Birds using slicing;
- f. Birds one by one using a *for* loop (what start do you use?);
- g. All animals in reverse order using slicing;
- h. All animals, one by one, in reverse order using a *for* loop.

## 9. Ở sở thú

Vòng lặp *For* với *if... ==... / else...*

Chúng ta có thể kết hợp vòng lặp *for* và cấu trúc *if/else* không?

- Bạn đang ở sở thú và viết ra danh sách một số loài động vật bạn nhìn thấy:



```
dongvat=["ca heo", "huou cao co", "chim canh cut"]
print(dongvat)
[69] ✓ 0.0s
```

Kết quả: .....

- Sau đó, bạn in từng con vật một:



```
for stt in range(0,3):
    print("Bat dau vong lap ")
    print("Con vat co stt " + str(stt) + " la " + dongvat[stt])
[70] ✓ 0.0s
```

Kết quả: .....

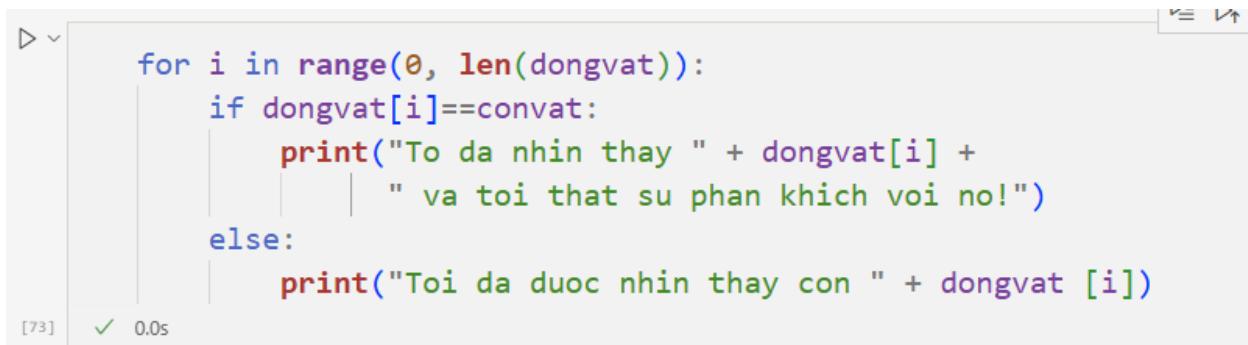
- Bạn thực sự muốn thấy một chú chim cánh cụt:



```
convat="chim canh cut"
[71] ✓ 0.0s
```

Kết quả: .....

- Khi về nhà, bạn kể cho bạn mình nghe về những con vật bạn đã nhìn thấy, nêu rõ con vật nào bạn thực sự muốn nhìn thấy:



```
for i in range(0, len(dongvat)):
    if dongvat[i]==convat:
        print("To da nhin thay " + dongvat[i] +
              " va toi that su phan khich voi no!")
    else:
        print("Toi da duoc nhin thay con " + dongvat [i])
```

Kết quả: .....

Bây giờ chúng ta làm quen với chú thích, nó bắt đầu là dấu “#”

Bình luận là mô tả hoặc giải thích về mã.

Bắt đầu là dấu #

- Trong vòng lặp *for*, biến *index* thường được viết tắt là *i*.
- Hàm *len()* tích hợp trả về độ dài của một biến.
- Chúng ta có thể sử dụng cấu trúc *if/else* trong vòng lặp *for*.
- Chúng ta có thể sử dụng toán tử so sánh == (*bằng hoặc bằng*) trong điều kiện *if*.
- Chú thích bắt đầu bằng ký hiệu # và chúng là mô tả hoặc giải thích.

## Bài tập áp dụng:

2. Tháng. Cho danh sách sau:

```
months = ["Tháng Hai", "Tháng Bảy", "Tháng Một", "Tháng Tám", "Tháng Mười Hai",
          "Tháng Sáu"]
```

In ra tên các tháng mùa đông bằng vòng lặp *for*. Sau đó, in ra tên các tháng mùa hè bằng vòng lặp *for*. Chọn một tháng bạn thích và gán nó vào một biến. In ra tất cả các tháng, từng một, cho biết tháng hiện tại có phải là tháng bạn yêu thích hay không. Cuối cùng,

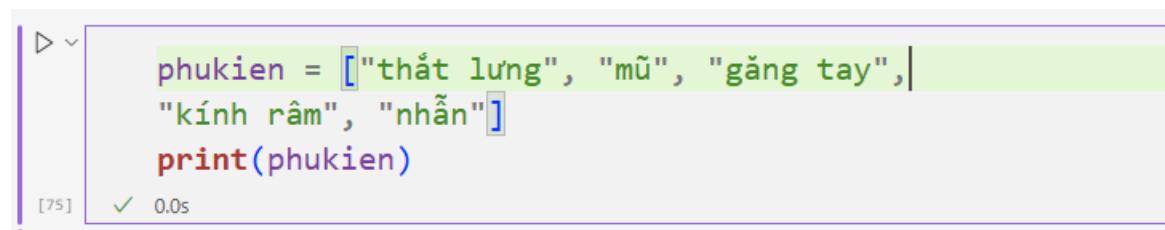
bạn có thể sử dụng cách nào khác để kiểm tra xem tháng bạn yêu thích có trong danh sách hay không?

## 10. Găng tay của tôi đâu rồi?

Vòng lặp *For* để tìm kiếm

Khi kết hợp với danh sách, vòng lặp *for* thường được sử dụng cho ít nhất ba thao tác: tìm kiếm phần tử, thay đổi phần tử và tạo danh sách mới, như bạn sẽ tìm hiểu trong ba chương tiếp theo. Trong chương này, chúng ta sẽ bắt đầu bằng cách học cách sử dụng vòng lặp *for* để tìm kiếm phần tử trong danh sách. Bạn đã sẵn sàng chưa?

- Ai mà chẳng có một ngăn kéo bừa bộn chứ? Đây là ngăn kéo của chúng tôi! Nó chứa một số phụ kiện:



```
phukien = ["thắt lưng", "mũ", "găng tay",  
          "kinh râm", "nhẫn"]  
print(phukien)
```

[75] ✓ 0.0s

Kết quả: .....

- In ra tất cả các phụ kiện, từng cái một, cũng như vị trí của chúng trong danh sách. Sử dụng câu như sau: Phần tử *x* nằm ở vị trí *y*:



```
for i in range(len(phukien)):  
    print("Thanh phan phu kien "+ phukien[i] +  
          " o vi tri so " + str(i))
```

[76] ✓ 0.0s

Kết quả: .....

1. In ra phụ kiện có tên gồm 8 ký tự và vị trí của nó trong danh sách. Sử dụng câu như sau: Phần tử *x* nằm ở vị trí *y* và có *n* ký tự:

```
for i in range(len(phukien)):
    if len(phukien[i]) == 8:
        print("Thanh phan phu kien "+phukien[i] +
              " o vi tri so " + str(i) + " va co 8 ky tu")
```

Kết quả: .....

2. In ra các phụ kiện có tên có độ dài *dưới* 8 ký tự:

Sinh viên tự làm

Kết quả: .....

3. In ra các phụ kiện có tên *dài hơn* 8 ký tự. Đồng thời, gán 8 cho một biến:

```
bien=8
for i in range(len(phukien)):
    if len(phukien[i]) > bien:
        print("Thanh phan phu kien "+phukien[i] +
              " o vi tri so " + str(i) + " va co " + str(bien) + " ky tu")
```

Kết quả: .....

4. In ra các phụ kiện có tên được tạo thành từ *một số ký tự khác* 8:

Sinh viên tự làm

Kết quả: .....

5. In ra các phụ kiện có *vị trí nhỏ hơn hoặc bằng* 2:

```
:ri=2
i in range(len(phukien)):
    if i<= vitri:
        print("Thanh phan phu kien "+phukien[i] +
              " o vi tri so " + str(i) + " nho hon hoac bang " + str(vitri))
```

Kết quả: .....

6. In các phụ kiện có *vị trí ít nhất là 2*:

Sinh viên tự làm

Kết quả: .....

### **Tóm tắt**

- Chúng ta có thể sử dụng vòng lặp *for* kết hợp với cấu trúc *if/else* để tìm kiếm các phần tử trong danh sách.
- Nên tạo biến thay vì các giá trị được mã hóa cứng trong một khối mã để giảm khả năng xảy ra lỗi. Biến thường nằm ở đầu khối mã.
- Trong Python, có sáu toán tử so sánh: ==, !=, >, >=, <, <=.

### **Bài tập áp dụng**

1. Các mùa. Cho danh sách sau:

```
seasons = ["xuân", "hạ", "thu", "đông"]
```

In:

- Tất cả các mùa có tên được tạo thành từ ít nhất 5 ký tự;
- Tất cả các mùa có tên được tạo thành từ số ký tự bằng hoặc nhỏ hơn 4;
- Tất cả các mùa có vị trí nhỏ hơn 2;
- Tất cả các mùa có vị trí ít nhất 2.

2. Tìm từ. Bạn đang làm việc cho một tạp chí và vừa tạo một trò chơi tìm từ mới cho độc giả. Dưới đây là các từ ẩn trong trò chơi:

```
words = ["cards", "park", "pets", "football", "golf", "crosswords", "toys",
```

```
"exercise", "hobbies", "riding", "biking", "games", "reading", "movies",
```

"walking", "concerts"]

Sau khi hoàn thành lưới ô vuông:

- a. Tạo một biến có tên là tiêu đề chứa số lượng từ cần tìm, sau đó in ra (ví dụ: tìm từ với 16 từ).
- b. Tìm các từ gồm 5 chữ cái. Cụ thể hơn, hãy in ra một tiêu đề, tiêu đề này phải chứa số lượng chữ cái của nhóm từ này và các từ.
- c. Có từ nào ít hơn 5 ký tự không? Nếu có, với mỗi từ, hãy in ra một câu chứa chính từ đó, vị trí của nó trong danh sách và số lượng ký tự của nó.
- d. Tương tự, có từ nào nhiều hơn 8 ký tự không? Nếu có, với mỗi từ, hãy in ra một câu chứa chính từ đó, vị trí của nó trong danh sách và số lượng ký tự của nó.
- e. Những từ nào trong phần thứ hai của danh sách có số lượng ký tự khác 7? Vị trí của chúng là gì? Và số lượng ký tự của chúng là bao nhiêu?
- f. Cuối cùng, những từ nào trong phần thứ tư đầu tiên của danh sách được tạo thành từ 4 ký tự? Vị trí của chúng là gì?

3. Cuộc thi đánh vần. Dưới đây là một số từ thuộc nhóm cơ xương khớp (msk) mà bạn cần ghi nhớ cho cuộc thi đánh vần tiếp theo:

```
msk_words = ["ankle", "patella", "rib", "femur", "sternocleidomastoid", "tendon",  
"sternum", "abdominal external oblique", "muscle", "scapula", "radius", "bone",  
"vertebra", "ligament", "ulna", "skull", "clavicle"]
```

- a. Bạn cần học bao nhiêu từ? Hãy tính toán và in ra.
- b. Độ dài của mỗi từ là bao nhiêu? (bao gồm cả khoảng trắng nếu có).
- c. Bây giờ, hãy nhóm các từ dựa trên độ dài của chúng. Dưới đây là danh sách các từ ngắn:

```
short = ["leg"]
```

Thêm tất cả các từ có 6 ký tự trở xuống vào danh sách và in ra kết quả. Có bao nhiêu từ trong danh sách?

d. Dưới đây là danh sách các từ có độ dài trung bình:

```
intermediate = ["cartilage"]
```

Thêm tất cả các từ có 7, 8 và 9 ký tự. Sau đó in ra kết quả. Có bao nhiêu từ trong danh sách?

e. Và cuối cùng, đây là danh sách các từ dài:

```
long = ["pectoralis major"]
```

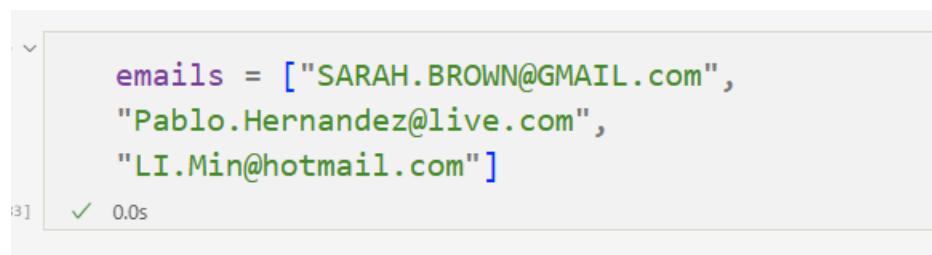
Thêm tất cả các từ còn lại và in ra kết quả. Có bao nhiêu từ trong danh sách?

## 11. Đọn dẹp danh sách gửi thư

Vòng lặp *For* để thay đổi các phần tử danh sách

Đã đến lúc học cách sử dụng vòng lặp *for* để thay đổi các phần tử danh sách!

- Bạn chịu trách nhiệm cho một bản tin và bạn phải gửi email đến các địa chỉ sau:



```
emails = ["SARAH.BROWN@GMAIL.com",
          "Pablo.Hernandez@live.com",
          "LI.Min@hotmail.com"]
```

- Để đảm bảo tính nhất quán, bạn muốn tất cả địa chỉ email đều viết thường. Vì vậy, bạn hãy thay đổi chúng:

```
> 
    for i in range(len(emails)):
        print("--> Vòng lặp thứ " + str(i))

        #in cac phan tu truoc khi thay doi
        print("Truoc khi thay doi thanh thanh thu " +
              str(i) + " la " + emails[i])

        #thay doi
        emails[i]=emails[i].lower()
        print("Truoc khi thay doi thanh thanh thu " +
              str(i) + " la " + emails[i])

    #in toan bo
    print("Bay gio toan bo ten email la " + str(emails))

[84] ✓ 0.0s
```

Kết quả: .....

- `.lower()` để chuyển đổi tất cả các ký tự của một chuỗi thành chữ thường;
- `.upper()` để chuyển đổi tất cả các ký tự của một chuỗi thành chữ hoa;
- `.title()` để chuyển đổi ký tự đầu tiên của một chuỗi thành chữ hoa và tất cả các ký tự còn lại thành chữ thường;
- `.capitalize()` để chuyển đổi ký tự đầu tiên của mỗi từ trong chuỗi thành chữ hoa và tất cả các ký tự còn lại thành chữ thường.

### Tóm tắt

- Để thay đổi các phần tử trong danh sách, chúng ta luôn cần gán lại phần tử đã thay đổi cho chính nó.
- Các phương thức chuỗi để thay đổi kiểu chữ là: `.lower()`, `.upper()`, `.title()` và `.capitalize()`

### **Bài tập áp dụng:**

1. Đối với mỗi tình huống sau, hãy tạo mã tương tự như mã được trình bày trong chương này:

a. Biên tập một bài viết. Bạn làm việc tại một tờ báo và phải biên tập một bài viết có rất nhiều từ viết tắt:

```
acronyms = ["asap", "faq", "fyi", "diy"]
```

Tất cả các từ viết tắt đều là chữ thường, vì vậy bạn hãy đổi chúng thành chữ hoa.

b. Thẻ tên. Bạn đang tổ chức một sự kiện và có danh sách tên sau:

```
names = ["JOHN", "geetha", "xiao", "LAURA"]
```

Bạn muốn in ra các thẻ tên đẹp, vì vậy bạn viết hoa tất cả các tên.

2. Màu sắc. Cho danh sách sau:

```
colors = ["yellow", "beige", "green", "red", "ultramarine", "coral", "lavender", "silver",  
"cyan", "blue", "black", "magenta", "gold", "pink", "scarlet", "brown"]
```

a. Có bao nhiêu màu? Tính toán xem!

b. Bắt đầu từ phần tử thứ hai (vị trí 1), đổi mỗi từ thứ ba thành chữ in hoa.

c. Bắt đầu từ phần tử thứ ba (vị trí 2), viết hoa mỗi từ thứ ba.

d. Thêm tất cả các màu của nửa đầu danh sách colors vào danh sách sau bằng vòng lặp *for*, đảm bảo chúng được viết thường:

```
some_colors = ["white"]
```

Bây giờ có bao nhiêu màu trong some\_colors?

e. Thêm tất cả các màu của nửa sau danh sách colors vào danh sách sau bằng cách sử dụng lệnh cắt:

```
more_colors = ["purple"]
```

Bây giờ có bao nhiêu màu trong more\_colors? Đổi chúng thành chữ in hoa.

3. Camping. Cho danh sách sau:

```
camping = ["tent", "adventure", "boots", "hiking", "hat", "nature", "path", "lake",
"mountain_sports", "fire", "water bottle", "fishing", "national park", "beach",
"compass", "forest", "trail", "sleeping bag"]
```

- Có bao nhiêu phần tử trong danh sách này?
- Lấy tất cả các từ có ít hơn (bao gồm) 6 chữ cái và thêm chúng vào danh sách sau, viết hoa mỗi từ:

```
short_camping = ["Trip"]
```

- Cắt mỗi từ thứ hai của danh sách camping bắt đầu từ từ đầu tiên (vị trí 0) và gán chúng cho một biến mới gọi là some\_camping\_words.
- Viết hoa mỗi từ của các chuỗi trong some\_camping\_words gồm một số ký tự khác 4.
- Trong some\_camping\_words, xóa từ đầu tiên (vị trí 0) bằng phương pháp danh sách.
- Trong some\_camping\_words, hãy xóa "path" bằng phương thức danh sách.
- Có nhiều từ hơn trong short\_camping hay some\_camping\_words? Sử dụng cấu trúc if/else để in ra danh sách nào có nhiều từ hơn, cũng như số lượng từ mà chúng chứa.

## 12. Thật là một mớ hỗn độn ở hiệu sách!

Vòng lặp **for** để tạo danh sách mới

Cuối cùng, chúng ta hãy cùng học cách sử dụng vòng lặp *for* để tạo danh sách mới.

- Hôm nay có rất nhiều khách hàng đến cửa hàng, và họ đã nhầm lẫn những cuốn sách có họ của tác giả bắt đầu bằng chữ A và S:

```
▶ v
  authors = ["Alcott", "Saint-Exupéry",
  "Arendt", "Sepulveda", "Shakespeare"]
  print(authors)
[85] ✓ 0.0s
```

- Vì vậy, bạn phải đặt những cuốn sách có họ tác giả bắt đầu bằng chữ *A* lên một kệ, và những cuốn sách có họ tác giả bắt đầu bằng chữ *S* lên một kệ khác:

```
> v
  #khai tao danh sach trong
  danhsach_a=[]
  danhsach_s=[]

  for i in range(len(authors)):
    #in tung thanh phan tac gia
    authors[i]

    # Lay phan dau cua tac gia hien tai
    kytudau=authors[i][0]
    #in ky tu dau tien
    print("Ky tu dau tien cua ten tac gia la " + kytudau)

    #bat dau kiem tra ky tu dau tien
    if kytudau == "A":
      #them vao danh sach tac gia bat dau ky tu A
      #dung Lệnh .append()
      danhsach_a.append(authors[i])
      print("Danh sach ta gia ky tu A bay gio chua them "
            + str(danhsach_a) + "\n")
    else:
      #them vao danh sach tac gia bat dau ky tu S
      # dung phep +
      danhsach_s=danhsach_s+[authors[i]]
      print("Danh sach ta gia ky tu S bay gio chua them "
            + str(danhsach_s) + "\n")
86] ✓ 0.0s
```

Kết quả: .....

Theo nguyên tắc chung, khi sử dụng vòng lặp *for* để tạo và điền vào *một danh sách rỗng*, chúng ta phải:

1. Khởi tạo một danh sách rỗng trước vòng lặp *for*
2. Nối hoặc thêm các phần tử mới vào bên trong vòng lặp *for*

### Tóm tắt

- Để tạo và điền vào danh sách trong vòng lặp *for*, chúng ta phải: (1) khởi tạo một danh sách rỗng trước vòng lặp *for* và (2) điền vào danh sách bằng cách sử dụng *.append()* hoặc *nối danh sách* trong vòng lặp *for*.
- Cắt chuỗi hoạt động tương tự như cắt danh sách.
- Trong nhiều lần cắt liên tiếp, chúng ta thực hiện từng lần cắt, bắt đầu từ bên trái.
- Ký tự đặc biệt "*\n*" tạo *một dòng trống* sau khi *in*.

### Bài tập áp dụng

1. Đối với mỗi tình huống sau, hãy tạo mã tương tự như mã được trình bày trong chương này.

a. Bán xe điện. Bạn làm việc tại một công ty xe hơi nổi tiếng và bạn phải vận chuyển những chiếc xe điện mới vừa mới đến. Các đồng nghiệp của bạn đã đánh số những chiếc xe được gửi đến Tây Ban Nha [ES] và Bồ Đào Nha [PT], nhưng họ đã nhầm lẫn chúng:

```
e_cars = ["PT-754J", "ES-096L", "PT-536G", "ES-543H", "PT-653H"]
```

Phân tách hai nhóm xe theo điểm đến của chúng.

b. Dạy động từ tiếng Anh. Bạn là giáo viên tiếng Anh cho sinh viên nước ngoài. Một số sinh viên gặp khó khăn trong việc hiểu khi nào động từ hiện tại được chia ở ngôi thứ ba số ít (he/she/it), hoặc ở các ngôi khác (I/you/we/they). Vậy là bạn đã cung cấp một danh sách các động từ:

```
english_verbs = ["eat", "drink", "eats", "sleep", "drinks", "sleeps"]
```

và bạn giúp học sinh phân biệt các động từ giữa ngôi thứ ba và các ngôi khác.

2. Món tráng miệng. Cho danh sách sau:

```
desserts = ["meringue", "apple pie", "eclair", "rice pudding", "chocolate", "english  
pudding", "cake", "icing"]
```

Lấy tất cả các chữ cái đầu, đổi chúng thành chữ in hoa và nối chúng thành một danh sách mới. Sau đó, đảo ngược danh sách. Bạn sẽ nhận được món tráng miệng nào?

3. Đoán xem các món ăn nào. Cho danh sách sau:

```
jobs = ["photog", "bal", "mu", "inve", "ambas", "si", "ler", "stig", "rapher", "ci",  
"ator", "ina", "an", "sador"]
```

Nhóm các chuỗi gồm 2, 3, 4, 5 và 6 chữ cái vào danh sách mới. Bạn sẽ nhận được những công việc nào? Hãy đảm bảo chữ cái đầu tiên của mỗi công việc đều được viết hoa.

4. Art. Cho danh sách sau:

```
art = ["apor", "refsscu", "atwat", "fetes", "erta", "jtylpt", "aprco", "srap",  
"ruolo", "texture", "gitp", "puors"]
```

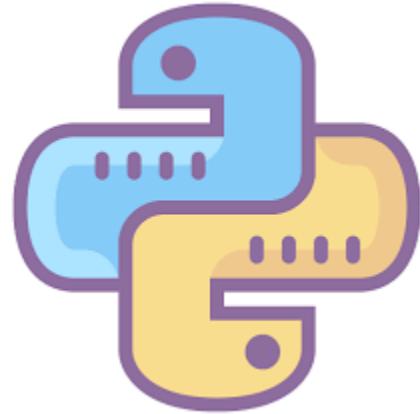
Tạo danh sách mới cho mỗi mục sau:

- Nếu độ dài chuỗi là 4, hãy lấy hai chữ cái bắt đầu từ chữ cái thứ hai (vị trí 1 và 2).
- Nếu độ dài chuỗi là 5, hãy lấy chữ cái thứ ba và thứ tư (vị trí 2 và 3). • Nếu độ dài chuỗi ít nhất là 6, hãy lấy ba chữ cái cuối cùng.

Bạn sẽ lấy được những từ nghệ thuật nào? Hãy đảm bảo tất cả các chuỗi đều viết hoa!

## PHẦN 4

### SỐ VÀ



### THUẬT TOÁN

Trong phần này, bạn sẽ học cách thực hiện các phép tính số học, làm việc với các số ngẫu nhiên và triển khai các thuật toán đầu tiên.

### 13. Triển khai máy tính

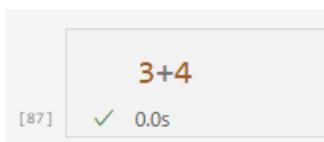
Số nguyên, số thực và các phép toán số học

Trong các chương trước, bạn đã phát triển khá nhiều tư duy tính toán, vì vậy bây giờ bạn đã sẵn sàng cho các con số, một số phép toán đơn giản và các thuật toán! Có một quan niệm sai lầm phổ biến rằng để giỏi lập trình, người ta phải rất giỏi toán. Tuy nhiên, điều đó không nhất thiết đúng, như bạn sẽ thấy trong các chương tiếp theo!

Trong chương này, bạn sẽ bắt đầu làm quen với các con số trong lập trình bằng cách triển khai máy tính. Để làm được điều đó, trước tiên bạn cần học các toán tử số học trong Python và cách yêu cầu người dùng nhập một số.

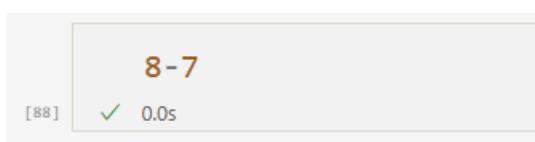
Như trong các chương trước, trước tiên hãy thử tự mình giải bài toán và sau đó so sánh câu trả lời của bạn với mã bên dưới.

#### 1. Phép cộng



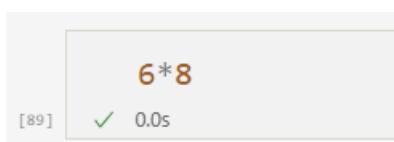
Kết quả: .....

#### 2. Phép trừ



Kết quả: .....

#### 3. Phép nhân



Kết quả: .....

### 3. Phép mũ $a^b$



The screenshot shows a Jupyter Notebook cell with the code `2**8`. The output is `256`, indicated by a green checkmark and the text `0.0s` for execution time.

Kết quả: .....

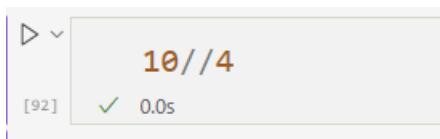
### 4. Phép chia



The screenshot shows a Jupyter Notebook cell with the code `10/4`. The output is `2.5`, indicated by a green checkmark and the text `0.0s` for execution time.

Kết quả: .....

### 5. Phép chia lấy tròn xuống (floor)



The screenshot shows a Jupyter Notebook cell with the code `10//4`. The output is `2`, indicated by a green checkmark and the text `0.0s` for execution time.

Kết quả: .....

Tóm lại, Python cung cấp bảy toán tử số học:

- 1 cho phép cộng (+);
- 1 cho phép trừ (-);
- 2 cho "họ nhân", bao gồm phép nhân (\*) và phép lũy thừa (\*\*);
- 3 cho "họ chia", bao gồm phép chia (/), phép chia lấy phần nguyên (//) và phép chia lấy phần dư (%).

Lưu ý rằng các toán tử chia có thể cung cấp kết quả là số nguyên hoặc số thập phân, độc lập với các đặc điểm của số bị chia và số chia. Khám phá thêm các đặc điểm bằng cách giải bài tập sau.

## 2. Làm thế nào để yêu cầu người dùng nhập một số?

Khi yêu cầu người dùng nhập một số, điều quan trọng là phải cẩn thận về các kiểu biến.

Hãy cùng xem

Điều này có nghĩa là gì!

- Yêu cầu người dùng nhập một số, gán nó cho một biến và in ra biến đó:

```
> v
    number=input("Xin moi nhap mot so") #1
    print(number) #2
[93]   ✓  3.0s
```

Kết quả: .....

Chúng ta sử dụng hàm *input()* tích hợp sẵn để yêu cầu người dùng nhập một số, và lưu kết quả vào biến *number*. Sau đó, in ra giá trị của biến. Bạn mong đợi biến *number* có kiểu dữ liệu là gì? Hãy cùng tìm hiểu nhé!

- Kiểm tra kiểu dữ liệu của biến *number*:

```
▷ v
    type(number)
[94]   ✓  0.0s
```

Kết quả: .....

Để biết kiểu của một biến, chúng ta sử dụng hàm *type()* tích hợp, hàm này lấy một biến làm đầu vào và trả về kiểu của biến đó.

- Chuyển đổi số thành số nguyên, in ra và kiểm tra kiểu của nó:

```
▷ v
    number=int(number)
    print(number)
    type(number)
[95]   ✓  0.0s
```

Kết quả: .....

Hàm `int()` tích hợp sẽ lấy một biến không phải số nguyên làm đầu vào và trả về dưới dạng số nguyên.

- Chuyển đổi số thành số thực, in ra và kiểm tra kiểu của nó:

A screenshot of a Jupyter Notebook cell. The code is:  
number=float(number)  
print(number)  
type(number)  
The output shows a green checkmark and the time 0.0s. The result of the print statement, 0.0s, is displayed below the code.

Kết quả: .....

Hàm `float()` tích hợp sẽ lấy một biến không phải thập phân làm đầu vào và trả về dưới dạng thập phân.

- Chuyển đổi số trở lại thành chuỗi, in ra và kiểm tra kiểu của nó:

A screenshot of a Jupyter Notebook cell. The code is:  
number=str(number)  
print(number)  
type(number)  
The output shows a green checkmark and the time 0.0s. The result of the print statement, 0.0s, is displayed below the code.

Biến số có thể có ba loại:

- `int` (số nguyên), được sử dụng trong tính toán;
- `float` (số thập phân), được sử dụng trong tính toán;
- `String`, khi chúng ta cần số dưới dạng văn bản—ví dụ, khi nối chúng với chuỗi.

### 3. Hãy cùng tạo máy tính!

- Yêu cầu người dùng nhập số đầu tiên, tức là số đầu tiên. Số đầu tiên nên là loại nào?

```
so_a=input("Xin moi nhap so dau tien")
# doi dinh dang sang so thuc
so_a=float(so_a)

type(so_a)
[99]    ✓  2.5s
```

Kết quả: .....

- Yêu cầu người dùng nhập dữ liệu thứ hai, đó là toán tử số học:

```
pheptoan=input("Xin moi nhap phep toan ban muon thuc hien")
type(pheptoan)
[100]   ✓  1.6s
```

Kết quả: .....

- Cuối cùng, hãy yêu cầu người dùng nhập dữ liệu thứ ba và cũng là dữ liệu cuối cùng, tức là số thứ hai. Nên nhập loại dữ liệu nào?

```
so_b=float(input("Xin moi nhap so thu 2"))
type(so_b)
[01]    ✓  1.5s
```

Kết quả: .....

- Hãy cùng viết phần lõi của máy tính! Bạn sẽ làm thế nào? Hãy thử một số ý tưởng trước khi xem xét phần triển khai bên dưới:

```
if pheptoan=="+":
    result=so_a+so_b
elif pheptoan=="-":
    result=so_a-so_b
elif pheptoan=="*":
    result=so_a*so_b
elif pheptoan=="/":
    result=so_a/so_b
elif pheptoan=="//":
    result=so_a//so_b
elif pheptoan=="%":
    result=so_a%so_b
else:
    print("Ban da nhap sai phep tinh")

print(result)
.02] ✓ 0.05
```

Kết quả: .....

- Cuối cùng, hãy in ra kết quả:

```
print(str(so_a) + pheptoan + str(so_b) + " = " + str(result))
.03] ✓ 0.05
```

Kết quả: .....

Sinh viên gom tắt cả đoạn code trên trong 1 ô thực thi.

### **Bài tập áp dụng**

1. Cuộc thi toán. Bạn đang tổ chức một cuộc thi toán, trong đó người tham gia phải chọn giữa ba phong bì và giải các biểu thức số học có trong phong bì đã chọn:

- Nếu người tham gia chọn phong bì 1, người đó sẽ phải giải:  $(3 \times 5^2 \div 15) - (5 - 2^2)$ .

- Nếu người tham gia chọn phong bì 2, người đó sẽ phải giải:  $-1 \times [(3 - 4 \times 7) \div 5] - 2^3 \times 24 \div 6$ .
- Nếu người tham gia chọn phong bì 3, người đó sẽ phải giải:  $[(36 - 3) \times 4] / [(15 - 9) \div 3]$ .

Tính toán các kết quả.

2. Gia sư hình học. Bạn đang giúp con trai hàng xóm làm một số bài tập hình học. Con phải tính diện tích và thể tích của một hình trụ, và bạn muốn kiểm tra tính chính xác của kết quả bằng Python.

Hỏi đứa trẻ về bán kính và chiều cao của hình trụ. Sau đó, tính diện tích và thể tích của hình trụ bằng các công thức sau: diện tích =  $2\pi r^2 + 2\pi rh$  và thể tích =  $\pi r^2 h$ . Gợi ý: Giá trị của  $\pi$  là bao nhiêu? Gán nó cho một biến!

Cậu bé cũng phải tính diện tích bề mặt và diện tích của một hình lập phương có độ dài cạnh  $a = 4$ . Cậu bé không có công thức đúng, vì vậy bạn hãy tìm chúng trên internet. Viết mã để kiểm tra xem các phép tính của cậu bé có chính xác không.

3. Nhiệt độ ngoài kia là bao nhiêu? Bạn đang đi du lịch giữa Châu Âu và Bắc Mỹ, và bạn cần phải mang theo quần áo phù hợp. Hãy viết một công thức chuyển đổi nhiệt độ, biết rằng mối quan hệ giữa độ C và độ F là  $C = 5 \div 9 \times (F - 32)$ . Trả lời hai câu hỏi sau:

a. Nhiệt độ ở Miami là  $75^\circ F$ . Nhiệt độ tính theo độ C là bao nhiêu?

b. Nhiệt độ ở Lisbon là  $17^\circ C$ . Nhiệt độ tính theo độ F là bao nhiêu?

## **14. Làm việc với số**

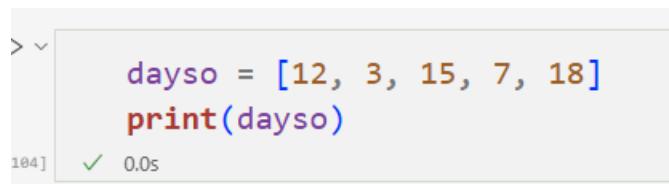
*Các phép toán thường gặp với danh sách số*

Danh sách số là một trong những cấu trúc dữ liệu được sử dụng nhiều nhất trong lập trình. Chúng tuân theo các quy tắc tương tự như danh sách chuỗi—tức là, chúng ta có thể sử dụng phép cắt và các phương thức (ví dụ: .append(), .remove(), v.v.) để thao tác chúng. Trong chương này, chúng ta sẽ khám phá một số tác vụ điển hình được thực hiện với danh sách số.

### *1. Thay đổi số dựa trên điều kiện*

Một trong những tác vụ phổ biến nhất trong lập trình là thay đổi số trong danh sách dựa trên một số điều kiện. Hãy xem ví dụ này!

- Cho danh sách số sau:



```
> 
  dayso = [12, 3, 15, 7, 18]
  print(dayso)
104] ✓ 0.0s
```

Kết quả: .....

- Trừ 1 vào các số lớn hơn hoặc bằng 10, và cộng 2 vào các số nhỏ hơn 10:

```
▷ ▾
# Lặp từng vị trí trong chiều dài dayso
for i in range(len(dayso)):
    #nếu số hiện tại >= 10
    if dayso[i] >=10:
        #trừ nó cho 1
        dayso[i]=dayso[i]-1
    else:
        #cộng nó cho 2
        dayso[i]=dayso[i]+2

#in kết quả cuối cùng
print(dayso)
[105] ✓ 0.0s
```

Kết quả: .....

## 2. Tách số dựa trên điều kiện

Một nhiệm vụ rất phổ biến khác với danh sách số là tách các số thành các danh sách mới dựa trên điều kiện cho trước. Hãy cùng xem một ví dụ sau!

- Cho danh sách số sau:

```
▷ ▾
dayso = [2, 10, 7, 5, 0, 9]
print(dayso)
[106] ✓ 0.0s
```

Kết quả: .....

Chúng ta bắt đầu với một danh sách chứa sáu số nguyên.

- Chia các số thành hai danh sách khác nhau—một danh sách cho số lẻ và một danh sách cho số chẵn:

```
[107] #khởi tạo dãy số chẵn và Lẻ  
  
daysochan=[];  
daysole=[];  
  
# Lặp từng vị trí trong chiều dài dayso  
for i in range(len(dayso)):  
  
    #kiểm tra vị trí hiện tại là chẵn không  
    if dayso[i] % 2 ==0:  
        daysochan.append(dayso[i])  
    else:  
        daysole.append(dayso[i])  
  
#in kết quả 2 day số  
print(daysochan)  
print(daysole)  
[107] ✓ 0.0s
```

Kết quả: .....

### 3. Tìm giá trị lớn nhất của một danh sách số

Một nhiệm vụ rất phổ biến thứ ba khi xử lý danh sách số là tìm giá trị lớn nhất (hoặc nhỏ nhất) trong một danh sách. Hãy tự mình tìm giá trị lớn nhất của danh sách dưới đây, soạn thảo và thử nghiệm với mã, trước khi tìm lời giải.

- Cho danh sách số sau:

```
[108] dayso = [2, -5, 34, 70, 22]  
      print(dayso)  
[108] ✓ 0.0s
```

Kết quả: .....

- Tìm số lớn nhất trong danh sách:

```
#khởi tạo giá trị Lớn nhất Là số đầu tiên vị trí 0
max_dayso=dayso[0]

# Lặp từng vị trí trong chiều dài dayso bắt đầu từ vị trí 1
for i in range(1,len(dayso)):

    #so sánh với giá trị Lớn nhất hiện có
    if dayso[i]>max_dayso:
        #update giá trị Lớn nhất
        max_dayso=dayso[i]

#in giá trị Lớn nhất tìm được
print(max_dayso)
```

[✓] 0.0s

Kết quả: .....

## Bài tập áp dụng

1. Tìm giá trị nhỏ nhất trong một dãy số. Cho dãy số sau:

numbers = [78, -900, 356, -103, 0, -78]

tìm giá trị nhỏ nhất trong dãy số.

2. Nhóm các số theo vị trí. Cho dãy số sau:

numbers = [4, 25, 7, -8, 59, 63, -10, 74]

tách các số ở vị trí lẻ khỏi các số ở vị trí chẵn bằng vòng lặp *for*.

3. Bội số. Cho dãy số sau:

numbers = [20, 24, 69, 15, 100, 16, 40, 80, 33, 57, 2, 200]

tạo một dãy số cho các số là bội số của 10, một dãy số cho các số là bội số của 3, và một dãy số cho các số còn lại. Cuối cùng, xóa các số trong danh sách.

4. Chuỗi dài nhất và ngắn nhất. Cho danh sách các chuỗi sau:

```
dogs = ["labrador", "chihuahua", "basset hound", "bernese shepherd", "poodle", "cocker spaniel"]
```

tìm chuỗi dài nhất và ngắn nhất. In ra hai chuỗi và độ dài của chúng.

5. Tính tổng các số trong một danh sách. Cho danh sách các số sau:

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

tính tổng.

6. Dãy số Fibonacci. Dãy số Fibonacci là một dãy số mà số hiện tại là tổng của hai số trước đó. Viết mã yêu cầu người dùng nhập số n và in ra dãy số Fibonacci của n.

Gợi ý: Bắt đầu chuỗi là [1,1]

Ví dụ:

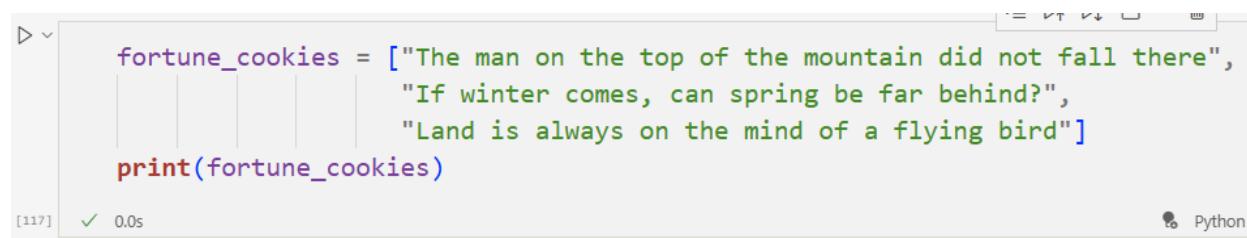
- Đầu vào của người dùng: 10
- Đầu ra: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55].

## 15. Bánh quy may mắn

Mô-đun Python random

Hãy tiếp tục khám phá các con số trong Python bằng cách tìm hiểu cách tạo số ngẫu nhiên. Tính ngẫu nhiên rất hữu ích trong lập trình, ví dụ như để tạo trò chơi hoặc mô phỏng khoa học.

- Bạn đang ở một nhà hàng Trung Quốc, và cuối bữa ăn, bạn nhận được một chiếc bánh quy may mắn. Chỉ còn ba chiếc bánh quy may mắn. Mỗi chiếc đều chứa một thông điệp:



```
fortune_cookies = ["The man on the top of the mountain did not fall there",
                    "If winter comes, can spring be far behind?",
                    "Land is always on the mind of a flying bird"]
print(fortune_cookies)
```

[117] 0.0s Python

Kết quả: .....

- Bạn sẽ nhận được chiếc bánh may mắn nào? Hãy để máy tính quyết định! Để làm được điều này, máy tính cần một *mô-đun* Python có tên là *random*:



```
import random
```

- Đây là thông báo của bạn khi máy tính chọn **một số thứ tự ngẫu nhiên**:

```
#Lấy số thứ tự ngẫu nhiên của lời nhắn

stt_msg=random.randint(0,len(fortune_cookies)-1)

print(stt_msg)

loinhan = fortune_cookies[stt_msg]

print(loinhan)
```

✓ 0.0s

Kết quả: .....

- Và đây là thông báo của bạn khi máy tính **trực tiếp chọn một phần tử ngẫu nhiên**

```
▷ 
    loinhan=random.choice(fortune_cookies)

    print(loinhan)
[138] ✓ 0.0s
```

Kết quả: .....

Tóm tắt

- *Mô-đun* là một đơn vị chứa các hàm cho một tác vụ cụ thể.
- Để nhập một *mô-đun*, chúng ta sử dụng từ khóa **import**. Các lệnh *import* thường được viết ở đầu mã và chỉ một lần.
- Khi gọi một hàm *mô-đun*, chúng ta sử dụng cú pháp sau:  
*module\_name.function\_name()*.
- *random* là một mô-đun để tạo số ngẫu nhiên. Nó chứa một số hàm, bao gồm:
  - *.randint(a,b)*: trả về một số nguyên ngẫu nhiên nằm giữa hai điểm cuối a và b (đã bao gồm);

- `.choice(list_name)`: trả về một phần tử của danh sách.

## **Bài tập áp dụng**

1. Đối với mỗi tình huống sau, hãy tạo đoạn mã tương tự như đoạn mã được trình bày trong chương này:

- Tung đồng xu. Có hai khả năng nào khi tung đồng xu? Viết chúng vào một danh sách. Sau đó, tung đồng xu, một lần bằng `.randint()` và một lần bằng `.choice()`. Bạn sẽ nhận được gì?
- Lăn xúc xác. Có những khả năng nào khi tung xúc xác? Viết chúng vào một danh sách. Sau đó, lăn xúc xác, một lần bằng `.randint()` và một lần bằng `.choice()`. Bạn sẽ nhận được những con số nào? Cuối cùng, chọn một phương pháp và lăn xúc xác ba lần. Bạn sẽ nhận được những con số nào?
- Mười số ngẫu nhiên. Tạo một danh sách gồm 10 số ngẫu nhiên từ 0 đến 100 bằng vòng lặp `for`.
- Các số ngẫu nhiên duy nhất là bội số của một số. Tạo một danh sách gồm 100 số ngẫu nhiên từ 5 đến 60. Chia chúng thành hai danh sách tùy thuộc vào việc chúng có phải là bội số của 4 hay không. Sau đó, tạo một danh sách khác gọi là `unique`, trong đó bạn thêm các bội số duy nhất của 4 từ danh sách trước đó. Điều này có nghĩa là, ví dụ, nếu 42 xuất hiện nhiều hơn một lần, nó sẽ chỉ xuất hiện một lần trong `unique`. Nếu số đó đã có trong `unique`, hãy in ra một câu như sau: Số x đã có trong `unique`. Bạn có thể tạo ngẫu nhiên bao nhiêu bội số duy nhất của 4?
- Thử nghiệm với số nguyên tố. Tạo một danh sách gồm 150 số ngẫu nhiên từ 50 đến 100, và chia chúng thành các danh sách tùy thuộc vào việc chúng có phải là bội số của các số nguyên tố 2, 3, 5 hoặc 7 hay không (một số có thể được thêm vào nhiều danh sách nếu nó là bội số của một số nguyên tố). Sau đó, cộng tất cả các phần tử của mỗi danh sách riêng biệt (không sử dụng các hàm tích hợp mà bạn có thể tìm thấy trực tuyến!). Mỗi

tổng có phải là bội số của số nguyên tố ban đầu không? Nghĩa là, tổng của tất cả các bội số của 3 có phải là bội số của chính 3 không?

## 16. Kéo búa bao

### *Giới thiệu về thuật toán*

Ai cũng biết trò chơi kéo búa bao! Trẻ em ở khắp mọi nơi trên thế giới đều chơi trò chơi này có nguồn gốc từ ít nhất 2.000 năm trước ở Trung Quốc<sup>1</sup>. Trong chương này, chúng ta sẽ học cách triển khai trò chơi này bằng Python. Bạn sẽ làm như thế nào? Hãy viết ý tưởng của bạn vào bài tập tiếp theo và thử viết trình triển khai của riêng bạn. Sau đó, hãy xem giải pháp tính toán bên dưới, cũng được triển khai trong Notebook

### *Hoàn thành các câu*

Hãy nghĩ về ba bước bạn cần để triển khai trò chơi kéo búa bao và viết chúng bên dưới.

1. ....

2. ....

3. ....

Giả sử bạn sẽ chơi với máy tính: máy tính sẽ chọn kéo, búa hoặc bao, và bạn cũng sẽ làm như vậy. Ai thắng?

#### *1. Máy tính chọn*

Trong bước đầu tiên, máy tính chọn giữa búa, bao và kéo. Bằng cách nào? Hãy cùng xem mã bên dưới.

- Yêu cầu máy tính chọn trò chơi búa, bao hoặc kéo:

```
import random

#Liệt kê các trạng thái
trangthai= ["kéo", "búa", "bao"]

#máy chọn
maychon=random.choice(trangthai)

print(maychon)
```

1 ✓ 0.0s

Kết quả: .....

### 2. Lựa chọn của người chơi

Ở bước thứ hai, đến lượt người chơi lựa chọn giữa oắn tù tì và kéo. Hãy cùng xem bên dưới.

- Yêu cầu người chơi lựa chọn giữa oắn tù tì và kéo:

```
#hỏi người chơi chọn cái gì

nguoicho Choi=input("kéo, búa, hay bao?")

print/nguoicho Choi
```

140] ✓ 1.65

Kết quả: .....

### 3. Xác định ai thắng

Đã đến lúc xác định ai thắng! Chúng ta làm điều đó như thế nào? Máy tính có ba lựa chọn khả thi, và người chơi cũng vậy. Vậy là có chín kịch bản có thể xảy ra. Làm thế nào để chúng ta mã hóa chúng mà không bỏ sót bất kỳ kịch bản nào? Một phương án là xác

định ba tình huống trong đó lựa chọn của máy tính là cố định và lựa chọn của người chơi thay đổi.

Hãy cùng xem cách thực hiện!

- Nếu máy tính chọn “bao”:



```
if maychon=="bao":  
  
    #so sánh với người chơi  
    if nguoicho == "bao":  
        print("Hoà rồi!")  
    elif nguoicho == "búa":  
        print("Bạn đã thắng")  
    else:  
        print("Bạn đã thua")
```

- Tương tự, Nếu máy tính chọn “kéo”:



```
if maychon=="kéo":  
  
    #so (variable) nguoicho: str  
    if nguoicho == "kéo":  
        print("Hoà rồi!")  
    elif nguoicho == "bao":  
        print("Bạn đã thắng")  
    else:  
        print("Bạn đã thua")
```

- Nếu máy tính chọn “búa”:

```
✓ if maychon=="búa":  
    #so sánh với người chơi  
    if nguoicho=="búa":  
        print("Hòa rồi!")  
    elif nguoicho=="kéo":  
        print("Bạn đã thắng")  
    else:  
        print("Bạn đã thua")
```

Kiểm tra có nghĩa là đánh giá và xác minh rằng mã thực hiện những gì nó được cho là phải làm.

Gỡ lỗi có nghĩa là xác định và loại bỏ lỗi khỏi mã

Tính song song có nghĩa là duy trì một cấu trúc tương ứng cho các dòng hoặc khối mã tiếp theo

## Hợp nhất mã

- Hãy hợp nhất mã:

```
import random

#Liệt kê các trạng thái
trangthai=["kéo","búa","bao"]

#máy chọn
maychon=random.choice(trangthai)

print(maychon)

nguoichoi=input("kéo, búa, hay bao?")

print/nguoichoi
if maychon=="bao":

    #so sánh với người chơi
    if nguoichoi=="bao":
        print("Hoà rồi!")
    elif nguoichoi=="búa":
        print("Bạn đã thắng")
    else:
        print("Bạn đã thua")

if maychon=="kéo":

    #so sánh với người chơi
    if nguoichoi=="kéo":
        print("Hoà rồi!")
    elif nguoichoi=="bao":
        print("Bạn đã thắng")
    else:
        print("Bạn đã thua")

if maychon=="búa":

    #so sánh với người chơi
    if nguoichoi=="búa":
        print("Hoà rồi!")
    elif nguoichoi=="kéo":
        print("Bạn đã thắng")
    else:
        print("Bạn đã thua")
```

✓ 3.9s

Kết quả: .....

Chia để trị nghĩa là chia một dự án thành các dự án con, giải quyết các dự án con và kết hợp các giải pháp của các dự án con để có được giải pháp cho toàn bộ dự án.

Thuật toán là một chuỗi các bước nghiêm ngặt để thực hiện và hoàn thành một nhiệm vụ.

## Tóm tắt

- *Thuật toán* là một chuỗi các bước để thực hiện một tác vụ.
- Khi viết thuật toán (và mã nói chung), chúng ta thường sử dụng phương pháp song song, kiểm thử, gỡ lỗi và chia để trị.

## Bài tập áp dụng

1. **Đêm đố vui!** Đố vui là một trò chơi đố vui, trong đó người chơi phải trả lời các câu hỏi về nhiều chủ đề khác nhau.

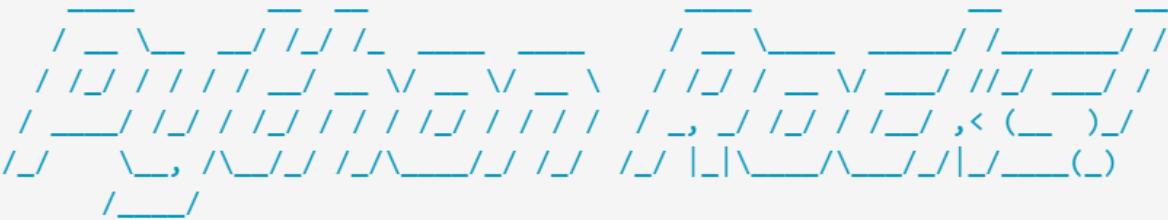
Để thực hiện trò chơi Đố vui này, hãy chuẩn bị 3 câu hỏi và câu trả lời tương ứng cho 3 chủ đề khác nhau. Yêu cầu người chơi chọn một chủ đề, sau đó hỏi một câu hỏi được chọn ngẫu nhiên về chủ đề đó. Cuối cùng, hãy cho người chơi biết câu trả lời có đúng không. Nếu không, hãy in ra câu trả lời đúng.

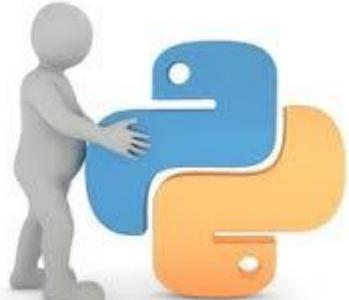
Dưới đây là một số gợi ý:

- Bạn sắp xếp câu hỏi và câu trả lời của mình như thế nào? Bạn sử dụng kiểu dữ liệu Python nào?
- Trình tự các hành động bạn cần thực hiện là gì? Hãy viết chúng ra trước khi viết mã. Bạn luôn có thể cập nhật chúng trong khi thực hiện.
- Làm thế nào để kiểm tra xem mã của bạn có đúng không?
- Hãy nhớ chia nhỏ và chinh phục!

```
pip install pyfiglet
from pyfiglet import Figlet
import random
colors=[
    "\033[91m","\033[92m","\033[93m"
    "\033[94m","\033[95m","\033[96m"
]
fonts=["slant", "block", "banner3-D", "isometric1",
       "standard","digital"]
text=input("Enter your text: ") or "Python Rocks!"
print("\n choice a font:")
for i,f in enumerate(fonts,1):
    print(f'{i}.{f}')
choice = int(input("Enter font number: ") or 1)
f=Figlet(font=fonts[choice-1])
baner=f.renderText(text)
print(random.choice(colors)+baner+"\033[0m")
```

```
choice a font:
1.slant
2.block
3.banner3-D
4.isometric1
5.standard
6.digital
```





## PHẦN 5

# VÒNG LẶP WHILE

## VÀ ĐIỀU KIỆN

Trong phần 5, bạn sẽ học cấu trúc cuối cùng trong lập trình: vòng lặp *while*. Bạn cũng sẽ tìm hiểu các loại điều kiện khác nhau có thể sử dụng trong vòng lặp *while* và các câu lệnh *if/elif/else*.

## 18. Động vật, số duy nhất và tổng

*Các loại điều kiện khác nhau*

Trong chương trước, chúng ta chỉ thấy một loại điều kiện trong vòng lặp ***while***—đó là một biến bằng "có". Nay giờ, hãy cùng xem xét ba ví dụ với các loại điều kiện khác. Trước tiên, hãy thử tự mình giải quyết từng nhiệm vụ: đọc kỹ yêu cầu, liệt kê các bước cần thực hiện, triển khai chúng từng bước một và hợp nhất mã nguồn vào giải pháp (chia để trị!). Lần này, hãy thử tiến thêm một bước nữa: chú ý đến các quá trình mà tâm trí bạn trải qua khi giải quyết các nhiệm vụ. Bạn sẽ thường thấy các mô hình tư duy lặp lại khi lập trình. Việc biết và nhận ra chúng sẽ giúp bạn nhận thức và do đó tăng tốc công việc. Đôi với mỗi ví dụ sau, bạn sẽ thấy một cách khả thi để tiếp cận nhiệm vụ lập trình đang thực hiện. Có thể nó sẽ giống với suy nghĩ của bạn, hoặc có thể nó sẽ khác. Trong mọi trường hợp, nó sẽ cho bạn ý tưởng về các hướng tư duy khá thi.

### 1. Đoán tên con vật!

- Cho danh sách sau:



```
dongvat = ["dog", "cat", "chicken"]
print(dongvat)
```

[153] ✓ 0.0s

Kết quả: .....

- Tạo một trò chơi trong đó máy tính ngẫu nhiên chọn một trong ba con vật và người chơi phải Đoán vật thể của máy tính được chọn. Đảm bảo rằng người chơi tiếp tục chơi cho đến khi được mong đợi con vật của máy tính lựa chọn. Khi kết thúc trò chơi, hãy cho người chơi biết họ đã phải thử bao nhiêu lần để mong đợi được điều đó.

Trò chơi có yêu cầu bốn: (1) máy tính ngẫu nhiên một trong ba con vật; (2) người chơi phải mong đợi một vật thể do máy tính lựa chọn; (3) người chơi tiếp tục chơi cho đến khi được mong đợi con vật của máy tính lựa chọn; và (4) khi kết thúc trò chơi, hãy cho người

chơi biết họ phải thử bao nhiêu lần để mong đợi được điều đó. Hãy cùng xem cách thực hiện theo yêu cầu!

1. Máy tính lựa chọn ngẫu nhiên một lần trong ba con vật. Điều này khá đơn giản:

```
▶ 
import random

#máy tính chọn
maychon=random.choice(dongvat)

print(maychon)
[154] ✓ 0.0s
```

Kết quả: .....

2. Người chơi phải đoán con vật do máy tính chọn. Nhiệm vụ này cũng dễ:

```
▶ 
#nguo Choi đoán
nguoichoi=input("Ban doan la cat? dog? hay chicken?")

print/nguoichoi)
[155] ✓ 2.1s
```

Kết quả: .....

3. Người chơi tiếp tục chơi cho đến khi đoán được con vật do máy tính chọn. Cụm từ "cho đến khi họ đoán được con vật" tương đương với "miễn là họ đoán được con vật", điều này ngay lập tức gợi ý cho chúng ta nên sử dụng vòng lặp *while*. Chúng ta sẽ viết điều kiện gì trong phần đầu? Hãy xem nhé:

```
# Lập đi Lập Lại Là dự đoán
# của người chơi và Lựa chọn của máy tính khác nhau
while maychon!=nguoicho:
    print("Bạn đã đoán sai rồi")

    #người chơi chọn Lại
    nguoicho=input("Bạn đoán là cat? dog? hay chicken?")

    print(negoicho)

    # hiện thị người chơi đoán đúng
    print("Tuyệt vời! Bạn đã đoán đúng")
```

[156] ✓ 10.7s

Kết quả: .....

4. Khi kết thúc trò chơi, hãy cho người chơi biết họ đã đoán được con vật đó bao nhiêu lần. Chúng ta chắc chắn cần một bộ đếm!

```
soluotdoan=1;
# Lập đi Lập Lại Là dự đoán
# của người chơi và Lựa chọn của máy tính khác nhau
while maychon!=nguoicho:
    print("Bạn đã đoán sai rồi")

    # hiện thị số lần đoán
    print("Bạn đã đoán " +str(soluotdoan))

    soluotdoan+=1
    #người chơi chọn Lại
    nguoicho=input("Bạn đoán là cat? dog? hay chicken?")

    print(negoicho)

    # hiện thị người chơi đoán đúng
    print("Tuyệt vời! Bạn đã đoán đúng sau " + str(soluotdoan) + " lần.")
```

[160] ✓ 6.9s

Python

Kết quả: .....

Sau khi giải quyết bốn nhiệm vụ, chúng ta có thể hợp nhất mã lại với nhau!

## 2. Tạo một danh sách gồm 8 số ngẫu nhiên duy nhất!

*Đây là nhiệm vụ tiếp theo của chúng ta:*

- Tạo một danh sách gồm 8 số ngẫu nhiên từ 0 đến 10. Đảm bảo chúng là duy nhất, nghĩa là mỗi số chỉ xuất hiện một lần trong danh sách. Nếu số đó đã có trong danh sách, hãy in ra dòng sau: **Số x đã có trong danh sách.** Bạn đã tạo ra bao nhiêu số trước khi tìm được 8 số duy nhất?

Nhiệm vụ này có bốn yêu cầu: (1) tạo một danh sách gồm 8 số ngẫu nhiên từ 0 đến 10; (2) đảm bảo chúng là duy nhất, nghĩa là mỗi số chỉ xuất hiện một lần trong danh sách; (3) nếu số đó đã có trong danh sách, hãy in ra Số x đã có trong danh sách; và (4) bạn đã tạo ra bao nhiêu số trước khi tìm được 8 số duy nhất? Hãy cùng xem xét từng yêu cầu một!

### 1. Tạo một danh sách gồm 8 số ngẫu nhiên từ 0 đến 10.

Chỉ theo yêu cầu này, chúng ta có thể tạo một danh sách gồm 8 số bằng cách sử dụng vòng lặp *for* và hàm *.randint()* từ mô-đun *.random*:

```
import random

#khởi tạo dãy số

dayso=[]

for i in range(8): # danh sách gồm 8 số ngẫu nhiên

    dayso.append(random.randint(0,10))

#print dãy số ra màn hình
print(dayso)
```

[161] 0.0s Python

Kết quả: .....

Chúng có duy nhất không? .....

2. *Đảm bảo chúng là duy nhất, nghĩa là mỗi số chỉ xuất hiện một lần trong danh sách.*

Hoặc vì lý do này, chúng ta không thể sử dụng vòng lặp *for*—vòng lặp này được sử dụng khi biết chính xác số lần lặp—nhưng chúng ta cần sử dụng vòng lặp *while*, vòng lặp này được sử dụng khi số lần lặp được xác định bởi một điều kiện.

```
▷ v
import random

#khởi tạo dãy số

dayso=[]

while len(dayso) != 8:

    # tạo 1 số ngẫu nhiên
    so=random.randint(0,10)

    #kiểm tra số này có trong dãy số chưa
    if so in dayso:
        a=0 # đặt 1 lệnh vô tri không có ý nghĩa gì cả
    else:
        dayso.append(so)

#print dãy số ra màn hình
print(dayso)

[166] ✓ 0.0s
```

Kết quả: .....

3. *Nếu số đã có trong danh sách, hãy in: Số x đã có trong danh sách*

```
▶ v
import random

#khởi tạo dãy số

dayso=[]

while len(dayso) != 8:

    #tạo 1 số ngẫu nhiên
    so=random.randint(0,10)

    #kiểm tra số này có trong dãy số chưa
    if so in dayso:
        #a=0 #đặt 1 lệnh vô tri không có ý nghĩa gì cả
        print("Số " + str(so) + " đã có trong danh sách.")
    else:
        dayso.append(so)

#print dãy số ra màn hình
print(dayso)

[167] ✓ 0.0s
```

Kết quả: .....

4. Bạn đã tạo ra bao nhiêu số trước khi tìm được 8 số duy nhất?

Để đáp ứng yêu cầu cuối cùng này, chúng ta cần một bộ đếm. Nó sẽ theo dõi số lượng số chúng ta đã tạo ra, trùng khớp với số lần lặp của vòng lặp *while*!

```
import random

# khởi tạo dãy số

dayso=[]

sodem=0

while len(dayso) != 8:

    # tạo 1 số ngẫu nhiên
    so=random.randint(0,10)

    # tăng số đếm lên 1

    sodem+=1

    # kiểm tra số này có trong dãy số chưa
    if so in dayso:
        #a=0 # đặt 1 lệnh vô tri không có ý nghĩa gì cả
        print("Số " + str(so) + " đã có trong danh sách.")
    else:
        dayso.append(so)

#in dãy số ra màn hình
print(dayso)

#in số lần lập tạo dãy số
print("Tổng số lần lập tạo ra dãy số duy nhất là " + str(sodem))
```

Kết quả: .....

### 3. Tính tổng các bội số của 3

- Viết mã tiếp tục yêu cầu người chơi nhập một số nguyên cho đến khi họ nhập một số âm. Cuối cùng, in ra tổng của tất cả các số nguyên đã nhập là bội số của 3.

Nhiệm vụ có hai yêu cầu: (1) tiếp tục yêu cầu người chơi nhập một số nguyên cho đến khi họ nhập một số âm, và (2) cuối cùng, in ra tổng của tất cả các số nguyên đã nhập là bội số của 3. Hãy cùng xem cách thực hiện chúng!

1. Tiếp tục yêu cầu người chơi nhập một số nguyên cho đến khi họ nhập một số âm. Yêu cầu rất đơn giản: chúng ta sử dụng hàm input để yêu cầu người chơi nhập số và một vòng lặp while để tiếp tục yêu cầu. Chúng ta sử dụng điều kiện nào trong phần đầu? Hãy cùng xem:

```
# yêu cầu người chơi nhập

so=input("Xin mời bạn nhập 1 số nguyên, số âm là dừng ?")
so=int(so)
print(so)

while so >=0:

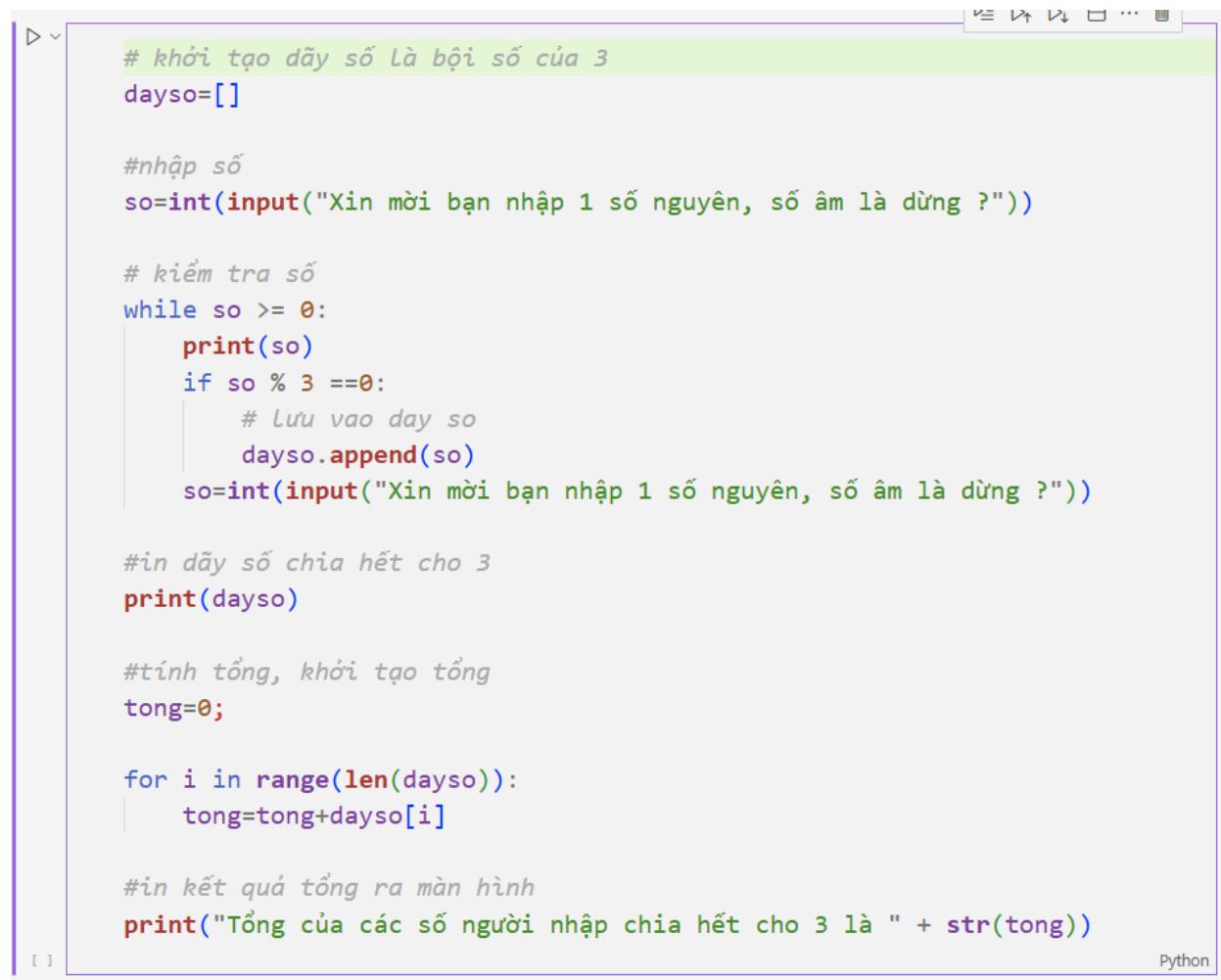
    so=int(input("Xin mời bạn nhập 1 số nguyên, số âm là dừng ?"))
    print(so)
```

[170] ✓ 6.3s Python

Kết quả: .....

2. Cuối cùng, in ra tổng của tất cả các số nguyên đã nhập là bội số của 3.

Chúng ta cần kiểm tra xem các số người dùng nhập vào có phải là bội số của 3 hay không, và nếu có, hãy cộng chúng lại. Bạn có ý tưởng nào không? Hãy bắt đầu soạn thảo mã:



```
# khởi tạo dãy số là bội số của 3
dayso=[]

#nhập số
so=int(input("Xin mời bạn nhập 1 số nguyên, số âm là dừng ?"))

# kiểm tra số
while so >= 0:
    print(so)
    if so % 3 ==0:
        # Lưu vào dãy số
        dayso.append(so)
    so=int(input("Xin mời bạn nhập 1 số nguyên, số âm là dừng ?"))

#in dãy số chia hết cho 3
print(dayso)

#tính tổng, khởi tạo tổng
tong=0;

for i in range(len(dayso)):
    tong=tong+dayso[i]

#print kết quả tổng ra màn hình
print("Tổng của các số người nhập chia hết cho 3 là " + str(tong))
```

Python

Kết quả: .....

### Tóm tắt

- Trong tiêu đề vòng lặp *while*, chúng ta có thể viết nhiều loại điều kiện khác nhau. Điều kiện đúng là điều kiện duy trì vòng lặp (không dừng lại!)
- Khi giải quyết một nhiệm vụ, việc phân tích và phân tích các yêu cầu, giải quyết các nhiệm vụ con, và hợp nhất mã nguồn thành giải pháp (chia để trị!) là điều thường thấy.
- Khi viết mã, chúng ta thường viết bản nháp đầu tiên, sau đó cải thiện bản nháp để mã nhanh hơn và mạnh mẽ hơn (viết mã cũng giống như viết email!)

## **Bài tập áp dụng**

1. Đoán số! Hãy tạo một trò chơi trong đó máy tính chọn một số từ 0 đến 10, và người chơi phải đoán số đó. Nếu người chơi đoán một số quá cao hoặc quá thấp, máy tính sẽ thông báo cho người chơi. Trò chơi dừng lại khi người chơi đoán được số đó. Cuối cùng, hãy cho người chơi biết đã đoán được số đó bao nhiêu lần.

2. 12 số chẵn ngẫu nhiên. Tạo một danh sách gồm 12 số chẵn ngẫu nhiên từ 0 đến 30.

Bạn đã loại trừ bao nhiêu số lẻ?

3. Trò chơi đánh vần cho trẻ em. Tạo một trò chơi giúp trẻ học đánh vần. Trò chơi có các yêu cầu sau: (1) Tạo một danh sách các từ cần đánh vần. Trong số các từ này, hãy chọn ngẫu nhiên một từ, và cho trẻ biết từ đã chọn (ví dụ: "Đánh vần từ 'hello'"). (2) Trẻ phải nhập một chữ cái vào cùng một lúc. Nếu trẻ nhập đúng chữ cái, hãy khen ngợi (ví dụ: "Làm tốt lắm!") và yêu cầu trẻ nhập chữ cái tiếp theo. Nếu trẻ nhập sai chữ cái, hãy nói với trẻ rằng chữ cái đó không đúng và yêu cầu trẻ nhập lại chữ cái.

Thử thách 1: Thay vì chỉ tạo 1 danh sách từ, hãy tạo 3 danh sách, mỗi danh sách một chủ đề, để trẻ có thể chọn chủ đề trước khi đánh vần một từ.

Thử thách 2: Trò chơi tiếp tục cho đến khi trẻ muốn đánh vần một từ mới.

## **19. And, or, not, not in**

### *Kết hợp và đảo ngược điều kiện*

Cho đến nay, chúng ta chỉ xem xét một điều kiện trong các cấu trúc *if/else* và vòng lặp *while*. Nếu chúng ta cần nhiều hơn một điều kiện thì sao? Và nếu chúng ta cần đảo ngược một điều kiện thì sao? Trong chương này, chúng ta sẽ học cách kết hợp hoặc đảo ngược điều kiện bằng cách sử dụng các toán tử *logic and, or, not*, và toán tử thành viên *not in*.

#### **1. and**

- Cho danh sách các số nguyên sau:

```
▷ ▾
dayso=[1, 5, 7, 2, 8, 19]
print(dayso)
[172] ✓ 0.0s
```

Kết quả: .....

- In ra các số từ 5 đến 10:

```
▷ ▾
for i in range(len(dayso)):
    if dayso[i] >= 5 and dayso[i] <=10:
        #in ra
        print("Số " + str(dayso[i]) + " nằm giữa 5 và 10.")
[174] ✓ 0.0s
```

Kết quả: .....

Chúng ta sử dụng toán tử logic **and** khi chúng ta muốn kiểm tra xem tất cả các điều kiện có hợp lệ hay không

## 2. *or*

- Cho chuỗi ký tự sau:

```
▷ ▾
message = "Have a nice day!!!"
print(message)
[175] ✓ 0.0s
```

Kết quả: .....

- Và đưa ra tất cả các dấu câu:



```
daucau = "\\"\\'()[]{}<>.,;:@#$%&*_"
print(daucau)
```

Kết quả: .....

Dấu câu của chuỗi chứa tất cả các dấu câu trên bàn phím chữ cái Latinh. Hãy so sánh các ký hiệu với các ký hiệu trên bàn phím của bạn và xem có thêm dấu câu nào không! Các ký hiệu ở đầu dấu câu chuỗi "\\"\\'()[]{}<>.,;:@#\$%&\*\_\" là ký hiệu hơi khó hiểu, vì vậy hãy cùng phân tích chúng. Dấu ngoặc kép đầu tiên "\\"\\'" là ký hiệu giới thiệu chuỗi. Hai ký hiệu sau "\\"\\'" là các ký tự đặc biệt—you có thể nhớ ký tự đặc biệt "\n", được sử dụng để xuống dòng. Dấu gạch chéo ngược \ cho Python biết dấu ngoặc kép "" sau đây thực sự là một dấu gạch chéo ngược chứ không phải là ký hiệu chúng ta sử dụng để đóng chuỗi.

Dấu gạch chéo ngược cuối cùng "\\"\\'" thực sự là một dấu gạch chéo ngược vì dấu gạch chéo xuôi / sau đó không phải là một ký tự đặc biệt.

- In ra và đếm số ký tự là dấu câu hoặc nguyên âm:

```
▷ ▾
#chuỗi nguyên âm
nguyenam="aeiou"

#khởi tạo số đếm
sodem=0

for i in range(len(message)):
    #kiểm tra đó là nguyên âm hay là dấu câu
    if message[i] in daucau or message[i] in nguyenam:
        #in ra màn hình ký tự là dấu câu hoặc nguyên âm:
        print(message[i] + " là dấu câu hoặc nguyên âm.")

        #tăng số đếm
        sodem+=1

    #in tổng số nguyên âm hay là dấu câu
    print("Tổng số nguyên âm hay là dấu câu: " + str(sodem))
[178] ✓ 0.0s
```

Kết quả: .....

Chúng ta sử dụng toán tử logic **or** khi chúng ta muốn kiểm tra xem ít nhất một điều kiện có hợp lệ hay không

### 3. *not*

- Cho danh sách các số nguyên sau:

```
▷ ▾
dayso=[4, 6, 7, 9]
print(dayso)
[180] ✓ 0.0s
```

Kết quả: .....

- In ra các số không chia hết cho 2:

```
[181] D v
      for i in range(len(dayso)):
          if not dayso[i] % 2 == 0:
              #in day so khong chia het cho 2
              print(dayso[i])
[181] ✓ 0.0s
```

Kết quả: .....

Chúng ta sử dụng toán tử logic **not** khi muốn kiểm tra xem điều ngược lại của một điều kiện có hợp lệ hay không

Ngoài ra, nó tương tự như != như sau:

```
[182] D v
      for i in range(len(dayso)):
          #if not dayso[i] % 2 == 0:
          if dayso[i] % 2 != 0:
              #in day so khong chia het cho 2
              print(dayso[i])
[182] ✓ 0.0s
```

#### 4. not in

- Tạo 5 số ngẫu nhiên từ 0 đến 10. Nếu các số ngẫu nhiên này chưa có trong danh sách sau thì hãy thêm chúng vào:

```
▶ 
dayso=[1,4,7]
print(dayso)
[183] ✓ 0.0s
```

Kết quả: .....

```
▶ 
import random

for _ in range(5):

    so=random.randint(0,10)
    print(so)

    if so not in dayso:
        dayso.append(so)

#in day so
print(dayso)
[184] ✓ 0.0s
```

Kết quả: .....

### Tóm tắt

- Các toán tử logic là *and*, *or*, và *not*.
- Khi kết hợp các điều kiện, thứ tự thực hiện là *not*, *and*, *or* (NAO).
- Các toán tử thành viên là *in* và *not in*.

### Bài tập áp dụng

1. Thiên tông của Python. Hãy giải quyết 4 bước sau, và bạn sẽ khám phá ra Thiên tông của Python!

a. Cho danh sách các chuỗi sau:

```
strings_to_slice = ["reisk", "kpan", "xfsimpleg", "bosolutionb", "pobetterx", "weorb",
"ofworsep", "aathanx", "hoau", "hfcomplexx", "poors", "opcomplicatedx", "rwsolutions",
"re?o"]
```

Tạo một danh sách mới có tên là slices\_strings chứa các chuỗi tương tự nhưng không có hai chữ cái đầu và chữ cái cuối (ví dụ: "gfhio" sẽ thành "hi").

b. Cho danh sách các chuỗi sau:

```
strings_to_invert= ["emos", "elpoep", "kniht", "taht", "xelpmoc", "ro", "detacilpmoc",
"si", "retteb", "naht", "elpmis"]
```

Tạo một danh sách mới có tên là inverted\_strings chứa các chuỗi tương tự nhưng được đảo ngược (ví dụ: "ih" sẽ trở thành "hi").

c. Cho danh sách các chuỗi sau:

```
strings_to_pick = ["this", "sounds", "simple", "but", "is", "it?", "some", "things", "look",
"better", "than", "when", "complex", "but", "complex","again", "is", "worse", "better",
"than", "complicated", "I'm", "confused"]
```

Tìm các từ có trong cả slices\_strings và inverted\_strings, đổi chúng thành chữ in hoa và thêm chúng vào danh sách mới. Bạn nhận được câu nào?

d. Câu nhận được đến từ đâu? Chạy lệnh Python sau: import this.

2. Chơi với số. Cho danh sách các số sau:

```
numbers = [7, 9, 15, 19, 24, 30, 37, 45, 50]
```

a. In ra các số chia hết cho 3 và 5.

b. In ra các số chia hết cho 3 hoặc 5.

c. In ra các số chia hết cho 3 nhưng không chia hết cho 5. Thực hiện nhiệm vụ này theo hai cách khác nhau, một lần dùng not, và một lần không dùng not.

3. Nâng cấp trò chơi kéo búa bao. Trong Chương 16, chúng ta đã triển khai trò chơi kéo búa bao. Trong phiên bản đó, có rất nhiều lần lặp lại. Trong lập trình, chúng ta thường không muốn lặp lại vì chúng có thể gây ra lỗi. Làm thế nào để làm cho đoạn mã ít lặp lại hơn? Bằng cách kết hợp các điều kiện! Bạn có thể kết hợp những điều kiện nào trong trò chơi này? Hãy viết lại trò chơi kéo búa bao một cách ngắn gọn hơn bằng cách sử dụng các toán tử logic. Sau khi bạn đã tối ưu hóa mã, hãy biến nó thành một trò chơi thực sự bằng cách thêm một vòng lặp *while* cho phép người chơi chơi bao lâu tùy thích. Gợi ý: Thay vì suy nghĩ theo hướng máy tính và lựa chọn của người chơi, hãy suy nghĩ theo hướng kết quả, tức là hòa và người chơi (hoặc máy tính) thắng.

## 20. Đằng sau phép so sánh và điều kiện

*Boolean*

Cuối cùng cũng đến lúc hé lộ những điều ẩn sau phép so sánh và điều kiện! Python "nhìn thấy" điều gì khi chúng ta viết một phép so sánh hoặc một điều kiện?

### 1. So sánh hoặc điều kiện đơn lẻ

- Cho bài toán sau:

A screenshot of a Jupyter Notebook cell. The code is as follows:

```
so=6
print(so)
type(so)
```

The cell has a status bar at the bottom left showing [185] and a green checkmark icon with the text 0.0s.

Kết quả: .....

- Kết quả của phép so sánh sau là gì?

A screenshot of a Jupyter Notebook cell. The code is as follows:

```
print(so > 3)
```

The cell has a status bar at the bottom left showing [186] and a green checkmark icon with the text 0.0s.

Kết quả: .....

- Gán phép toán trên cho một biến và in ra. Biến này thuộc kiểu gì?

```
ketqua=so>3
print(ketqua)
type(ketqua)
```

[187] ✓ 0.0s

Kết quả: .....

Tương tự

```
ketqua=so<3
print(ketqua)
type(ketqua)
```

[188] ✓ 0.0s

Kết quả: .....

**Boolean** là một kiểu biến. Chúng chỉ có thể có hai giá trị: **True hoặc False**.

## 2. Kết hợp các phép so sánh hoặc điều kiện

Hãy cùng xem xét kỹ hơn hoạt động này và xem điều gì xảy ra khi chúng ta kết hợp các điều kiện.

- Kết quả của các phép so sánh sau là gì?

```
so=3
print(so>1)
print(so<5)
print(so > 1 and so < 5)
```

[189] ✓ 0.0s

Kết quả: .....

A screenshot of a Jupyter Notebook cell. The code is:

```
so=3
print(so>1)
print(so>5)
print(so > 1 and so > 5)
```

The output shows the results of the print statements and the final boolean expression evaluation:

```
[190]    ✓  0.0s
```

Kết quả: .....

	First condition	Second condition	First condition and Second condition
(1)	True	True	True
(2)	False	True	False
(3)	True	False	False
(4)	False	False	False

	First condition	Second condition	First condition or Second condition
(1)	True	True	True
(2)	False	True	True
(3)	True	False	True
(4)	False	False	False

	Condition	not condition
(1)	True	False
(2)	False	True

### 3. Chúng ta còn sử dụng Boolean ở đâu nữa?

Boolean thường được dùng làm cờ trong vòng lặp *while*. Điều này có nghĩa là gì?

- Hãy xem phiên bản sửa đổi của ví dụ này: Bạn có muốn thêm kẹo không?

```
▷ ▾
#khởi tạo số kẹo
sokeo=0

# sử dụng cờ Luận Lý
flag=True

print("Bạn có " + str(sokeo) + " viên kẹo.")

#sử dụng cờ Luận Lý cho vòng Lặp
while flag:

    traloi=input("Bạn có muốn thêm kẹo không? (yes/no)")

    if traloi=="yes":
        sokeo+=1
        print("Bạn có " + str(sokeo) + " viên kẹo.")
    else:
        print("Bạn có tổng số " + str(sokeo) + " viên kẹo.")
        flag=False

[191] ✓ 9.1s
```

Kết quả: .....

### Tóm tắt

- Khi viết phép so sánh hoặc điều kiện, kết quả là một biến Boolean.
- Boolean là một kiểu dữ liệu của Python, tương tự như danh sách, chuỗi, số nguyên, v.v.
- Chỉ có 2 giá trị Boolean: *True* và *False*.
- Tổ hợp các điều kiện sử dụng and, or, not tuân theo bảng chân trị.
- Boolean có thể được sử dụng làm cờ trong vòng lặp *while* (hoạt động giống như đèn giao thông).

## **Bài tập áp dụng:**

1. Bạn có muốn ít bài tập hơn không? Viết lại vòng lặp *while* từ bài tập Bạn có muốn ít bài tập hơn không?
2. Tung đồng xu! Khi tung đồng xu, chúng ta có hai kết quả: mặt sấp và mặt ngửa. Trong bài tập này, chúng ta sẽ sử dụng True cho mặt sấp và False cho mặt sấp. Tung đồng xu 8 lần và lưu kết quả vào một danh sách có các phần tử kiểu Boolean. Bạn đã nhận được bao nhiêu kết quả mặt sấp và mặt ngửa? Tỷ lệ giữa số lần ngửa và số lần sấp là bao nhiêu? Nay giờ hãy tung đồng xu 1000 lần. Tỷ lệ mới là bao nhiêu? Hai tỷ lệ này khác nhau như thế nào?
3. Bộ so sánh. Bộ so sánh là một thuật toán so sánh hai số. Nó tương tự như máy tính, nhưng thay vì sử dụng các toán tử số học, nó sử dụng các toán tử so sánh. Hãy tạo một bộ so sánh yêu cầu người dùng nhập hai số nguyên và in ra tất cả các phép so sánh có thể có giữa hai số nguyên đó.

Ví dụ: Nếu người dùng nhập 3 và 5, hãy in ra:

$3 > 5$  là Sai

$3 < 5$  là Đúng

v.v.

Hãy đảm bảo: (1) sử dụng tất cả các toán tử so sánh; (2) sử dụng Boolean bất cứ khi nào có thể; và (3) cho phép người dùng sử dụng bộ so sánh bao lâu tùy thích. Bạn đã sử dụng những số nào để kiểm tra xem bộ so sánh có hoạt động chính xác không? Khi nào bạn nhận được kết quả Đúng?



## TẬP TRUNG VÀO DANH SÁCH

### VÀ VÒNG LẶP *FOR*

Trong phần này, bạn sẽ kết hợp kiến thức hiện có về danh sách và vòng lặp *for* với các khái niệm và thuộc tính mới.

## 21. Tổng quan về danh sách

### Các phép toán, phương thức và thủ thuật

Chúng ta đã đi được một nửa chặng đường trong hành trình học tư duy tính toán và lập trình trong Python! Vì vậy, đây là thời điểm thích hợp để nghỉ ngơi và tóm tắt lại mọi thứ chúng ta đã học về danh sách cho đến nay. Trong chương này, chúng ta sẽ áp dụng các quy tắc "ngữ pháp" cho danh sách Python và làm nổi bật một số thuộc tính mới quan trọng cần biết. Chương này chứa rất nhiều ví dụ và chi tiết giúp bạn cải thiện kỹ năng lập trình và hiểu được mã của người khác.

#### 1. Các phép toán số học trên các phần tử danh sách

Như bạn có thể nhớ, trong Python có 7 phép toán số học: **cộng (+), trừ (-), nhân (\*), lũy thừa (\*\*), chia (/), chia hết phần nguyên (//) và lấy dư (%)**. Để thực hiện các phép toán số học theo từng phần tử—tức là trên các phần tử danh sách—chúng ta sử dụng vòng lặp *for*. Các phép toán từng phần tử có thể được thực hiện (1) giữa hai hoặc nhiều danh sách có cùng độ dài hoặc (2) giữa một danh sách và một số. Trong cả hai trường hợp, chúng ta đều sử dụng vòng lặp *for*. Hãy xem hai ví dụ về phép cộng (nhưng chúng có thể áp dụng cho bất kỳ phép toán nào).

- *Tính tổng của hai danh sách theo từng phần tử:*



```
sochan=[2,4,6]
sole=[1,3,5]

tong=[]
for i in range(len(sochan)):
    tong.append(sochan[i]+sole[i])

print(tong)
```

[193] ✓ 0.0s

Kết quả: .....

- *Tính tổng của mỗi phần tử trong danh sách:*

```
> ▷ dayso=[1,7,5]
   so=3

   for i in range(len(dayso)):
       dayso[i]+=so

   print(dayso)
[3] ✓ 0.0s
```

Kết quả: .....

## 2. Các phép toán "số học" giữa các danh sách

Các phép toán giữa các danh sách thực chất không phải là số học, nhưng chúng sử dụng các ký hiệu số học với một ý nghĩa khác. Hai phép toán khả thi là nối, sử dụng ký hiệu + (phát âm là nối với) và nhân bản, sử dụng ký hiệu \* (phát âm là nhân bản bởi [số]). Hãy xem các ví dụ:

- *Nối hai danh sách:*

```
> ▷ sochan=[2,4,6]
   sole=[1,3,5]

   noichuoi=sochan+sole

   print(noichuoi)
[1] ✓ 0.0s
```

Kết quả: .....

- *Sao chép danh sách 3 lần:*

```
▶ ▾
  dayso=[1,2,3]
  so=3

  saocheb=dayso*so
  print(saocheb)
[2] ✓ 0.0s
```

Kết quả: .....

```
▶ ▾
  daynho=[0]

  solan=50

  daydai=daynho*solan
  print(daydai)
[3] ✓ 0.0s
```

Kết quả: .....

### 3. Gán danh sách

Khi gán một danh sách cho một danh sách khác, chúng ta phải rất cẩn thận! Hãy cùng xem lý do.

- Cho một danh sách chứa một vài số nguyên:

```
▶ ▾
  given_list=[1,4,8]
  print(given_list)
[4] ✓ 0.0s
```

Kết quả: .....

- Gán `given_list` cho `new_list`:



```
new_list=given_list
print(new_list)
[5]    ✓  0.0s
```

Kết quả: .....

- *Thay đổi phần tử danh sách đầu tiên của new\_list:*



```
new_list[0]=40
print(new_list)
[6]    ✓  0.0s
```

Kết quả: .....

- *In given\_list:*



```
print(given_list)
[7]    ✓  0.0s
```

Kết quả: .....

**Phần tử đầu tiên của given\_list cũng là 40!** Điều này xảy ra bởi vì khi chúng ta gán một danh sách cho một danh sách khác, chúng ta đặt hai tên cho cùng một danh sách. Nó hơi giống như khi một người có hai tên: ví dụ, tên của anh trai tôi là Flavio Alberto. Dù tôi gọi anh ấy là Flavio hay Alberto, anh ấy luôn là cùng một người!

- *Làm thế nào để tạo một bản sao độc lập của một danh sách?*

```
▶ 
given_list=[1,4,8]
new_list=given_list.copy()

new_list[0]=40

print(given_list)
print(new_list)

[1]   ✓  0.0s
```

Kết quả: .....

#### 4. Thêm một phần tử hoặc một danh sách vào danh sách

Chúng ta có thể thêm một phần tử vào danh sách theo hai cách: thêm vào cuối danh sách bằng phương thức `.append()`, hoặc thêm vào một vị trí cụ thể bằng phương thức `.insert()`. Hãy cùng xem hai ví dụ đơn giản để làm quen với cách thức hoạt động của các phương thức này.

- *Thêm một phần tử vào cuối danh sách:*

```
▶ 
dayso=[1,4,8,7]
dayso.append(9)

print(dayso)

[3]   ✓  0.0s
```

Kết quả: .....

- *Chèn số 2 vào vị trí 1:*

```
▶ 
dayso=[1,2,3,4]
dayso.insert(1,7)

print(dayso)

[4]   ✓  0.0s
```

Kết quả: .....

- *Nối hai danh sách:*

```
daya=[1,4,9]
dayb=[1,3]

dayc=daya+dayb
print(dayc)
[5]    ✓  0.0s
```

Kết quả: .....

- *Thêm một danh sách vào cuối một danh sách khác:*

```
daya=[1,4,9]
dayb=[1,3]

dayb.extend(daya)
print(dayb)
[6]    ✓  0.0s
```

Kết quả: .....

### 5. Xóa phần tử khỏi danh sách

Chúng ta có thể xóa phần tử danh sách dựa trên giá trị của chúng, bằng cách sử dụng `.remove()` hoặc dựa trên vị trí của chúng, bằng cách sử dụng `.pop()`. Chúng ta cũng có thể xóa tất cả phần tử bằng cách sử dụng `.clear()`. Hãy cùng xem một số ví dụ để làm mới các phương thức này và học một số thủ thuật mới.

- *Từ danh sách sau, hãy xóa tất cả các phần tử "ciao":*

```
> 
greetings = ["ciao","ciao","hello"]
greetings.remove("ciao")

print(greetings)
[7]   ✓  0.0s
```

Kết quả: .....

Tại sao chỉ có xoá 1 chữ “ciao” trong khi có 2 chữ.

Trong danh sách chứa nhiều phần tử giống nhau, phương thức `.remove()` chỉ xóa phần tử đầu tiên.

Để xoá hết chúng ta sử dụng kết hợp `while`:

```
> 
greetings = ["ciao","ciao","hello"]
while "ciao" in greetings:
    greetings.remove("ciao")

print(greetings)
[8]   ✓  0.0s
```

Kết quả: .....

- Xóa chuỗi “hello” dựa trên vị trí của nó:

```
> 
greetings = ["ciao","ciao","hello"]
greetings.pop(2)

print(greetings)
[9]   ✓  0.0s
```

Kết quả: .....

- Xóa tất cả các phần tử trong danh sách:

```
> 
greetings = ["ciao","ciao","hello"]
greetings.clear()

print(greetings)
[10] ✓ 0.0s
```

Kết quả: .....

## 6. Sắp xếp danh sách

Sắp xếp danh sách là một tác vụ rất phổ biến trong lập trình. Ví dụ, chúng ta có thể muốn sắp xếp tên theo thứ tự bảng chữ cái (xem bài tập "Một bước nữa!" bên dưới) hoặc danh sách giá cả tăng dần hoặc giảm dần.

- *Sắp xếp danh sách các số nguyên sau:*

```
> 
dayso=[5,7,6]
dayso.sort()

print(dayso)
[11] ✓ 0.0s
```

Kết quả: .....

- *Sắp xếp danh sách các số nguyên sau theo thứ tự giảm dần:*

```
> 
dayso=[5,7,6]
dayso.sort(reverse=True)

print(dayso)
[12] ✓ 0.0s
```

Kết quả: .....

- *Đảo ngược danh sách các số nguyên sau:*

```
▶ 
dayso=[5,7,6]
dayso.reverse()

print(dayso)
[13] ✓ 0.0s
```

Kết quả: .....

Chúng ta sử dụng phương thức `.reverse()` để đảo ngược thứ tự các phần tử trong danh sách. Theo đó, phần tử cuối cùng sẽ trở thành phần tử đầu tiên, phần tử áp chót sẽ trở thành phần tử thứ hai, v.v. Lưu ý rằng `.reverse()` sắp xếp các phần tử dựa trên vị trí của chúng, trong khi `.sort()` (xem ví dụ ở trên) sắp xếp các phần tử dựa trên giá trị của chúng.

## 7. Tìm kiếm phần tử

Hãy kết thúc hành trình dài khám phá các phương pháp danh sách bằng cách tìm hiểu cách tìm kiếm và đếm phần tử.

- *Tạo một danh sách và tìm kiếm một phần tử cụ thể:*

```
▶ 
kytu= ["a", "g", "c", "g"]
vitri=kytu.index("g")

print(vitri)
[14] ✓ 0.0s
```

Kết quả: .....

Chúng ta tạo danh sách các chữ cái chứa các chuỗi và tìm vị trí của phần tử "g" bằng cách sử dụng phương thức `.index()`. Như bạn có thể thấy, `.index()` chỉ cung cấp cho chúng ta vị trí của phần tử đầu tiên, là 1—vì vị trí của phần tử bắt đầu từ 0 trong Python.

- *Làm thế nào để tìm được tất cả các vị trí?*

```
▶ 
kytu= ["a", "g", "c", "g"]

vitri=[]

for i in range(len(kytu)):
    if kytu[i]=="g":
        vitri.append(i)

print(vitri)
[15] ✓ 0.0s
```

Kết quả: .....

- *Đếm số lần một phần tử xuất hiện trong danh sách:*

```
▶ 
kytu= ["a", "g", "c", "g"]
n=kytu.count("g")

print(n)
[16] ✓ 0.0s
```

Kết quả: .....

## Tóm tắt

- Chúng ta có thể thực hiện các phép toán theo từng phần tử trong danh sách bằng các toán tử số học `+`, `-`, `*`, `/`, `**`, `//`, `%`.
- Chúng ta có thể thực hiện các phép toán "số học" trên danh sách bằng cách sử dụng **phép nối +** và **phép nhân bản \***.
- 11 phương thức cho danh sách là: `.append()`, `.clear()`, `.copy()`, `.count()`, `.extend()`, `.index()`, `.insert()`, `.pop()`, `.remove()`, `.reverse()`, `.sort()`.
- Trong số 11 phương thức, 3 phương thức trả về giá trị mới là `.copy()`, `.count()` và `.index()`. 8 phương thức còn lại tự sửa đổi danh sách.

## **Bài tập áp dụng**

2. Đếm ngược năm mới! Cho danh sách sau: numbers = [0,1,2,3,4,5,6,7,8,9], hãy đảo ngược nó bằng cách sử dụng:

a. Phương pháp danh sách.

b. Cắt lát Slicing.

c. Vòng lặp *for*.

Sự khác biệt giữa ba phương pháp là gì?

3. Cửa hàng ứng dụng. Bạn đang thực hiện một nghiên cứu thị trường dựa trên dữ liệu cửa hàng ứng dụng. Đây là giá của các ứng dụng trong cửa hàng:

```
app_prices = [  
    7.99, 7.99, 2.99, 4.99, 7.99, 9.99, 9.99, 1.99, 1.99, 1.99,  
    4.99, 5.99, 3.99, 5.99, 0.99, 3.99, 3.99, 2.99, 1.99, 4.99,  
    8.99, 1.99, 3.99, 1.99, 1.99, 8.99, 6.99, 0.99, 6.99, 8.99,  
    3.99, 1.99, 0.99, 1.99, 0.99, 8.99, 1.99, 7.99, 3.99, 1.99,  
    8.99, 2.99, 4.99, 6.99, 4.99, 7.99, 8.99, 1.99, 2.99, 0.99,  
    7.99, 6.99, 7.99, 6.99, 2.99, 0.99, 0.99, 3.99, 2.99, 5.99,  
    0.99, 0.99, 7.99, 9.99, 5.99, 5.99, 1.99, 4.99, 5.99, 5.99,  
    6.99, 9.99, 5.99, 5.99, 1.99, 8.99, 9.99, 4.99, 9.99, 4.99,  
    0.99, 0.99, 2.99, 9.99, 3.99, 6.99, 8.99, 4.99, 1.99, 9.99,  
    0.99, 7.99, 1.99, 4.99, 4.99, 0.99, 3.99, 3.99, 1.99, 8.99,  
    3.99, 9.99, 5.99, 2.99, 2.99, 5.99, 4.99, 3.99, 8.99,  
    5.99, 8.99, 8.99, 1.99, 9.99, 7.99, 6.99, 7.99, 4.99, 4.99,
```

7.99, 8.99, 7.99, 4.99, 5.99, 5.99, 0.99, 2.99, 8.99, 7.99,  
1.99, 3.99, 3.99, 4.99, 9.99, 0.99, 1.99, 3.99, 9.99, 5.99,  
4.99, 8.99, 6.99, 5.99, 6.99, 7.99, 1.99, 2.99, 9.99, 6.99,  
9.99, 6.99, 8.99, 8.99, 2.99, 1.99, 9.99, 1.99, 7.99, 9.99,  
4.99, 3.99, 9.99, 9.99, 6.99, 6.99, 7.99, 9.99, 2.99, 4.99]

- a. Có bao nhiêu ứng dụng?
- b. Có bao nhiêu ứng dụng có giá 4,99? Tính kết quả theo hai cách, một lần sử dụng phương pháp liệt kê và một lần sử dụng vòng lặp *for*.
- c. Tỷ lệ phần trăm ứng dụng có giá 4,99 là bao nhiêu?
- d. Mức giá duy nhất của các ứng dụng trong cửa hàng là bao nhiêu? Tìm và sắp xếp chúng theo thứ tự tăng dần.
- e. Có bao nhiêu ứng dụng cho mỗi mức giá?
- f. Mức giá phổ biến nhất của một ứng dụng là bao nhiêu?

## 22. Tìm hiểu thêm về vòng lặp for

*Nhiều cách lặp lại lệnh trong danh sách và hơn thế nữa*

Trong các chương trước, chúng ta đã học cách sử dụng vòng lặp *for* để duyệt danh sách, tìm kiếm phần tử trong danh sách, thay đổi phần tử danh sách và tạo danh sách bằng cách thêm một phần tử tại một thời điểm. Ngoài ra, chúng ta đã sử dụng vòng lặp *for* để lặp lại các lệnh độc lập với danh sách.

### 1. Lặp lại lệnh

Như định nghĩa đã nêu,

Vòng lặp *for* là sự lặp lại của một nhóm lệnh  
với số lần xác định.

- In ra 3 số ngẫu nhiên từ 1 đến 10:

```
▶ 
import random

for _ in range(3):
    so=random.randint(1,10)
    print(so)

[17] ✓ 0.0s
```

Kết quả: .....

### 2. Vòng lặp For với danh sách

Có ít nhất 4 cách để sử dụng vòng lặp *for* với danh sách. Bạn đã biết cách đầu tiên: vòng lặp *for* thông qua các chỉ mục. Trong phần này, chúng ta sẽ tìm hiểu về vòng lặp *for* thông qua các phần tử, thông qua các chỉ mục và phần tử, và sự hiểu biết về danh sách. Lưu ý rằng thông qua các chỉ mục, thông qua các phần tử, và thông qua các chỉ mục và phần tử không phải là các thuật ngữ chuyên môn; tuy nhiên, chúng ta sẽ sử dụng chúng

để phân biệt các loại vòng lặp *for* khác nhau. Ngược lại, sự hiểu biết về danh sách là một thuật ngữ chuyên môn mà bạn có thể tìm thấy trong bất kỳ sách hoặc trang web lập trình Python nào. Trong tất cả các ví dụ trong phần này, chúng ta sẽ bắt đầu với danh sách sau, chứa ba chuỗi:



```
tensinhvien=["mai","hoa","lan","diep"]
print(tensinhvien)
```

Kết quả: .....

Nhiệm vụ của chúng ta là chuyển chữ cái đầu tiên của mỗi chuỗi thành chữ in hoa. Để làm được điều đó, chúng ta sẽ áp dụng phương thức *.title()* cho từng phần tử của danh sách và ghi đè lên danh sách hiện có bất cứ khi nào có thể.

## 2.1 Vòng lặp *for* qua chỉ số

Bạn đã biết loại vòng lặp *for* này. Hãy cùng ôn lại kiến thức với ví dụ sau.

- Viết hoa mỗi chuỗi bằng cách sử dụng vòng lặp *for* qua chỉ số:



```
tensinhvien=["mai","hoa","lan","diep"]
print(tensinhvien)

for i in range(len(tensinhvien)):

    print("Số thứ tự " + str(i) + " là "+ tensinhvien[i])

    tensinhvien[i]=tensinhvien[i].title()

print(tensinhvien)
```

Kết quả: .....

## 2.2 Vòng lặp *for* qua các phần tử

Hãy cùng tìm hiểu cách triển khai vòng lặp *for* mới đầu tiên: vòng lặp *for* qua các phần tử. Đọc ví dụ bên dưới và cố gắng hiểu nó hoạt động như thế nào:

- *Viết hoa mỗi chuỗi bằng cách sử dụng vòng lặp *for* qua các phần tử:*

```
tensinhvien = ["mai", "hoa", "lan", "diep"]

inhoaten = []

print(tensinhvien)

for ten in tensinhvien:

    print("Giá trị hiện tại: " + ten)

    inhoaten.append(ten.title())

print(inhoaten)
```

The screenshot shows a Python code editor window with a script named '16.py'. The code defines a list 'tensinhvien' with four elements: 'mai', 'hoa', 'lan', and 'diep'. It then initializes an empty list 'inhoaten'. The script uses a for loop to iterate over each name in 'tensinhvien', printing its value and appending its title case to 'inhoaten'. Finally, it prints the contents of 'inhoaten'. The status bar at the bottom indicates the script is 0.0s and has passed all tests.

Kết quả: .....

## 2.3 Vòng lặp *for* qua chỉ số và phần tử

Đúng như tên gọi, vòng lặp *for* qua chỉ số và phần tử kết hợp vòng lặp *for* qua chỉ số với vòng lặp *for* qua phần tử. Cách triển khai rất đơn giản. Hãy cố gắng hiểu ví dụ bên dưới trước khi đọc phần giải thích tiếp theo.

- *Viết hoa mỗi chuỗi bằng cách sử dụng vòng lặp *for* qua chỉ số và phần tử:*

```
• tensinhvien=["mai","hoa","lan","diep"]

    print(tensinhvien)

    for i,ten in enumerate(tensinhvien):

        print("Giá trị hiện tại vị trí " + str(i) + " : " + ten)

        tensinhvien[i]=tensinhvien[i].title()

    print(tensinhvien)
```

8] ✓ 0.0s Python

Kết quả: .....

```
▷ ✓ print(list(enumerate(tensinhvien)))
[29] ✓ 0.0s
```

Kết quả: .....

Hàm tích hợp `enumerate()` cung cấp danh sách các chỉ số và phần tử được ghép nối.

Vòng lặp *for* thông qua chỉ số và vị trí rất hữu ích khi chúng ta cần trích xuất cả vị trí và phần tử của toàn bộ danh sách.

## 2.4 Hiệu danh sách

Phương pháp thứ tư và cũng là phương pháp cuối cùng để sử dụng vòng lặp *for* kết hợp với danh sách được gọi là hiệu danh sách. Thoạt nhìn có vẻ phức tạp, nhưng chúng ta sẽ tìm hiểu ngay!

- *Viết hoa mỗi chuỗi bằng cách sử dụng hiệu danh sách chứa vòng lặp for thông qua các chỉ mục:*

```
tensinhvien=["mai","hoa","lan","diep"]

tensinhvien=[tensinhvien[i].title() for i in range(len(tensinhvien))]

print(tensinhvien)
```

✓ 0.0s      Python

Kết quả: .....

Để hiểu ta phân tích như sau:



Tương tự ta biến đổi các cách còn lại như sau:

```
tensinhvien=["mai","hoa","lan","diep"]

inhoaтен=[]

print(tensinhvien)

for ten in tensinhvien:
    inhoaтен.append(ten.title())
print(inhoaтен)

#viết gọn đoạn trên thành
tensinhvien=[ten.title() for ten in tensinhvien]
print(tensinhvien)
```

✓ 0.0s

Xét ví dụ sau:

```
tensinhvien=["hoang","hoa","hong","hy"]

teninhoa=[]
print(tensinhvien)
for ten in tensinhvien:
    if len(ten) < 4:
        teninhoa.append(ten.title())

print(teninhoa)
```

0.0s

Kết quả: .....

Bây giờ chúng ta rút gọn như sau

```
tensinhvien=["hoang","hoa","hong","hy"]

teninhoa=[]
print(tensinhvien)
for ten in tensinhvien:
    if len(ten) < 4:
        teninhoa.append(ten.title())

print(teninhoa)

#rút gọn

teninhoa=[ten.title() for ten in tensinhvien if len(ten) < 4]
print(teninhoa)
```

0.0s      Python

Kết quả: .....

### 3. Vòng lặp for lồng nhau

Là chủ đề cuối cùng của chương này, chúng ta hãy tìm hiểu về vòng lặp *for* lồng nhau. Vòng lặp *for* lồng nhau là một vòng lặp *for* nằm trong một vòng lặp *for* khác. Nó hoạt động như thế nào? Hãy đọc ví dụ bên dưới và cố gắng hiểu điều gì xảy ra.

- Cho danh sách các nguyên âm sau:

```
▷ ▾
nguyenam = ["A", "E", "I", "O", "U"]
print(nguyenam)
[37] ✓ 0.0s
```

Kết quả: .....

- Đối với mỗi nguyên âm, hãy in ra tất cả các nguyên âm bên phải:

```
▷ ▾
for i in range(len(nguyenam)):
    print("-----" + nguyenam[i])

    for j in range(i+1, len(nguyenam)):
        print(nguyenam[j])
[38] ✓ 0.0s
```

Kết quả: .....

## Tóm tắt

- Khi sử dụng vòng lặp *for* để lặp lại các lệnh không cần chỉ mục, chúng ta thay thế chỉ mục bằng dấu gạch dưới.
- Có ít nhất 4 loại vòng lặp *for* với danh sách: thông qua chỉ mục (sử dụng *range()*), thông qua các phần tử, thông qua chỉ mục và phần tử (sử dụng *enumerate()*), và dạng hiểu danh sách.
- Các hàm tích hợp *list()* có thể được sử dụng để **chuyển đổi** đầu ra của *range()* và *enumerate()* thành **một danh sách**.
- Hàm tích hợp *enumerate()* đồng thời trích xuất các **chỉ mục và phần tử** được ghép nối từ một danh sách.
- *Tuple* là chuỗi các phần tử được **phân tách bằng dấu phẩy** và nằm giữa hai dấu ngoặc tròn.

- Vòng lặp *for* lồng nhau là vòng lặp *for* bên trong vòng lặp *for*.

### **Bài tập áp dụng:**

1. Ăn thả ga. Những người bạn này đang ở một nhà hàng ăn thả ga:

```
friends = ["Geetha", "Huanxiang", "Megan", "Pedro"]
```

Đây là món ăn nhẹ tại tiệc buffet: food = ["sushi", "nachos", "samosa", "cheese"]

Mỗi người thử từng loại món ăn nhẹ. In ra các câu như:

*Geetha ăn sushi*

*Geetha ăn nachos*

...

*cho tất cả bạn bè:*

a. Sử dụng vòng lặp *for* lồng nhau thông qua các chỉ mục.

b. Sử dụng vòng lặp *for* lồng nhau thông qua các phần tử.

2. Trẻ em chơi. Ở trường mẫu giáo, trẻ em chơi trò chơi bắt cặp với một trẻ khác mỗi khi giáo viên rung chuông. Cuối cùng, mỗi trẻ sẽ bắt cặp với tất cả các trẻ khác. Cho

danh sách trẻ em sau:

```
kids = ["Paul", "Juhee", "Luca", "Maria"]
```

a. In ra tất cả các tổ hợp có thể bắt đầu từ trẻ đầu tiên, tức là:

Paul chơi với Juhee

Paul chơi với Luca

Paul chơi với Maria

Juhee chơi với Luca

Juhee chơi với Maria

Luca chơi với Maria

b. In ra tất cả các tổ hợp có thể bắt đầu từ trẻ cuối cùng (Maria).

3. Các thành phố trên thế giới. Cho danh sách các thành phố sau:

`cities = ["Bogota", "Riga", "Kinshasa", "Damascus", "New Delhi", "Auckland"]`

a. Sử dụng vòng lặp `for` thông qua các chỉ mục, tạo một danh sách mới chứa tên các thành phố có hơn 7 ký tự và đổi chúng thành chữ in hoa.

b. Sử dụng vòng lặp `for` qua các phần tử, tạo một danh sách mới chứa các chữ cái đầu của các thành phố có số ký tự từ 7 đến 10.

c. Sử dụng vòng lặp `for` qua các chỉ số và phần tử, in ra từng phần tử bằng chữ thường và vị trí của chúng.

d. Sử dụng danh sách hiểu biết, tạo một danh sách mới chứa các tên thành phố có ít hơn 7 ký tự và đổi chúng thành chữ thường.

4. Học đếm. In các số liên tiếp từ 10 đến 29 bằng vòng lặp `for` lồng nhau. Vòng lặp `for` bên ngoài sẽ in ra chữ số đầu tiên, trong khi vòng lặp `for` bên trong sẽ in ra chữ số thứ hai, chẳng hạn như:

10

11

12

...

29

5. Tam giác số. Yêu cầu người dùng nhập một số. Sau đó, in ra một tam giác số, trong đó hàng lớn nhất là số được truy vấn. Ví dụ:

Đầu vào: 5

Đầu ra:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Gợi ý: Cân nhắc sử dụng tham số end trong hàm print(). Tìm ví dụ về cách sử dụng end online.

## 23. Danh sách các danh sách

*Cắt lát, vòng lặp for lồng nhau và làm phẳng*

Danh sách các danh sách là gì?

Danh sách các danh sách là danh sách có các phần tử là danh sách.

Danh sách các danh sách tuân theo các quy tắc tương tự như danh sách; chúng chỉ thêm một "lớp" chỉ mục. Trong chương này, bạn sẽ học cách cắt lát danh sách các danh sách, sử dụng vòng lặp for lồng nhau để lặp qua chúng và khám phá các cách làm phẳng chúng.

### 1. Cắt lát

Để cắt lát một danh sách các danh sách, chúng ta sửa đổi các quy tắc cắt lát đã học cho danh sách: bằng cách thêm một lớp chỉ mục. Hãy cùng xem nó hoạt động như thế nào!

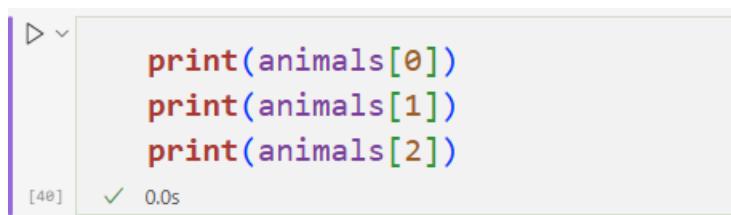
- Cho danh sách các danh sách sau:



```
39]    animals = [[ "dog", "cat"], [ "cow",
  "sheep", "horse", "chicken", "rabbit"],
  [ "panda", "elephant", "giraffe",
  "penguin"]]
  print(animals)
  ✓ 0.0s
```

Kết quả: .....

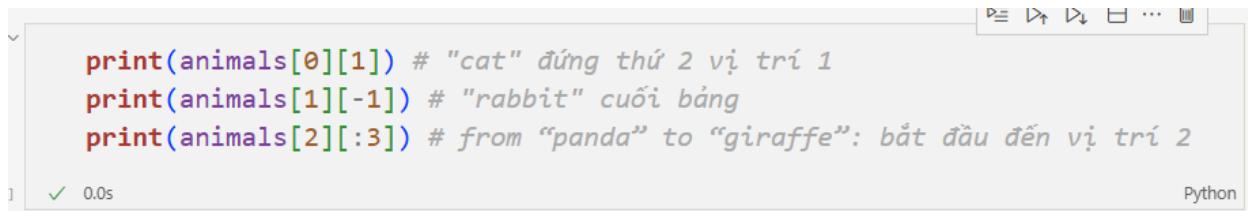
- In các danh sách phụ có chứa vật nuôi, động vật trang trại và động vật hoang dã:



```
[40]    print(animals[0])
  print(animals[1])
  print(animals[2])
  ✓ 0.0s
```

Kết quả: .....

- In ra các phần tử con “cat”, “rabbit”, and from “panda” to “giraffe”:



```
print(animals[0][1]) # "cat" đứng thứ 2 vị trí 1
print(animals[1][-1]) # "rabbit" cuối bảng
print(animals[2][:3]) # from "panda" to "giraffe": bắt đầu đến vị trí 2
1 ✓ 0.0s
```

Kết quả: .....

## 2. Vòng lặp for lồng nhau

Để duyệt các phần tử trong một danh sách các danh sách, chúng ta có thể sử dụng vòng lặp *for* lồng nhau, trong đó vòng lặp bên ngoài duyệt danh sách các danh sách và vòng lặp bên trong duyệt các danh sách con. Hãy cố gắng hiểu ví dụ sau làm gì và sau đó đọc phần giải thích.

- Cho danh sách các danh sách sau:

```
sports = [["skiing", "skating",
"curling"], ["canoeing", "cycling",
"swimming", "surfing"]]
print(sports)
[  ✓  0.0s
```

Kết quả: .....

- In từng phần tử của danh sách con bằng cách sử dụng vòng lặp lồng nhau thông qua các chỉ mục:

```
for i in range(len(sports)): #i chạy từng cell một
    for j in range(len(sports[i])): #j chạy số thứ tự theo từng cell i
        print(sports[i][j])
[44]  ✓  0.0s
```

Python

Kết quả: .....

- In từng phần tử của danh sách con bằng cách sử dụng vòng lặp lồng nhau qua các phần tử:

```
for muagiai in sports:
    for ten in muagiai:
        print(ten)
[45]  ✓  0.0s
```

Kết quả: .....

### 3. Làm phẳng

Làm phẳng nghĩa là biến đổi một **danh sách các danh sách** thành **một danh sách**. Nói cách khác, chúng ta lấy các phần tử con ra khỏi các danh sách con của chúng và đặt chúng vào một danh sách. Có nhiều cách để thực hiện thao tác này. Chúng ta sẽ xem xét bốn cách khác nhau, nhưng có thể còn nhiều cách khác nữa. Với mỗi phương pháp làm phẳng, hãy thử tự mình thực hiện trước, sau đó xem xét ví dụ và giải thích bên dưới.

- Cho danh sách các danh sách sau:

```
# danh sách nhạc cụ
instruments = [["contrabass", "cello",
"clarinet"], ["gong", "guitar"],
["tambourine", "trumpet", "trombone",
"triangle"]]
print(instruments)
```

5] ✓ 0.0s

Kết quả: .....

- Làm phẳng danh sách bằng cách sử dụng vòng lặp lồng nhau thông qua các chỉ mục:

```
# gom thanh 1 danh sach
thongnhat=[]

for group in instruments:
    for nhaccu in group:
        thongnhat.append(nhaccu)

print(thongnhat)
```

47] ✓ 0.0s

Kết quả: .....

- Làm phẳng danh sách bằng cách sử dụng hiểu biết về danh sách:

The diagram illustrates the conversion of nested loops into a list comprehension. It shows two snippets of Python code. The top snippet uses nested loops to build a list:

- `# gom thanh 1 danh sach  
thongnhat=[]`
- `for group in instruments:  
 for nhaccu in group:  
 thongnhat.append(nhaccu)`
- `print(thongnhat)`

The bottom snippet shows the resulting list comprehension:`# viết dạng rút gọn  
thongnhat=[nhaccu for group in instruments for nhaccu in group]  
print(thongnhat)`

A large blue arrow points from the nested loop code to the list comprehension, indicating the transformation process.

Kết quả: .....

### Tóm tắt

- Danh sách của danh sách là danh sách có các phần tử là danh sách.
- Khi cắt, chúng ta sử dụng hai cặp dấu ngoặc vuông. Trong cặp đầu tiên, chúng ta viết vị trí của danh sách con cần cắt; trong cặp thứ hai, chúng ta viết vị trí của (các) phần tử con.
- Chúng ta có thể sử dụng các vòng lặp *for* lồng nhau để duyệt các phần tử con.
- Chúng ta có thể làm phẳng danh sách các danh sách bằng một vòng lặp *for* lồng nhau, một vòng lặp *for* kết hợp với phép nối, hoặc một kỹ thuật đọc hiểu danh sách.

### Lists of lists and images

You surely know that digital images are composed of pixels, that is, small colorful squares organized in a grid. We can think of the grid as a list of lists where each sub-element corresponds to a pixel of a specific color. Let's consider Figure 23.1.

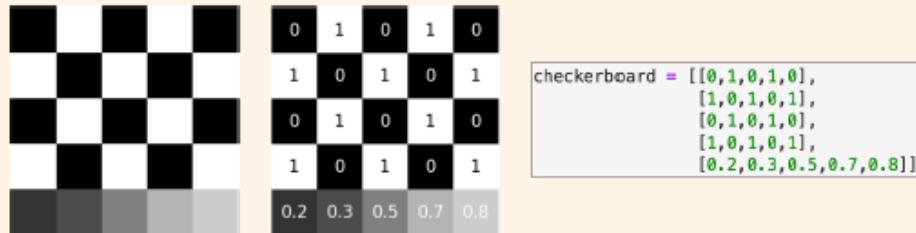


Figure 23.1. Digital representation of a checkerboard. Left: Image rendering. Center: Numerical values corresponding to the checkerboard colors. Right: List of lists encoding the checkerboard colors.

Each black square corresponds to a pixel containing `0`, and each white square corresponds to a pixel containing `1`. Thus, the first (and the third) row of the checkerboard is represented by the sub-list `[0,1,0,1,0]`, and the second (and the fourth) row is represented by the sub-list `[1,0,1,0,1]`. The last row of the checkerboard contains pixels colored with various shades of grey. Each pixel corresponds to a decimal (float) number. Darker greys are closer to `0` (that is, to black), whereas brighter greys are closer to `1` (that is, to white).

What about digital colored images? Each pixel is encoded by an RGB list composed of three numbers, each representing a different color: the first number is for the red (R) component, the second number for the green (G) component, and the third number for the blue (B) component. Let's have a look at Figure 23.2.

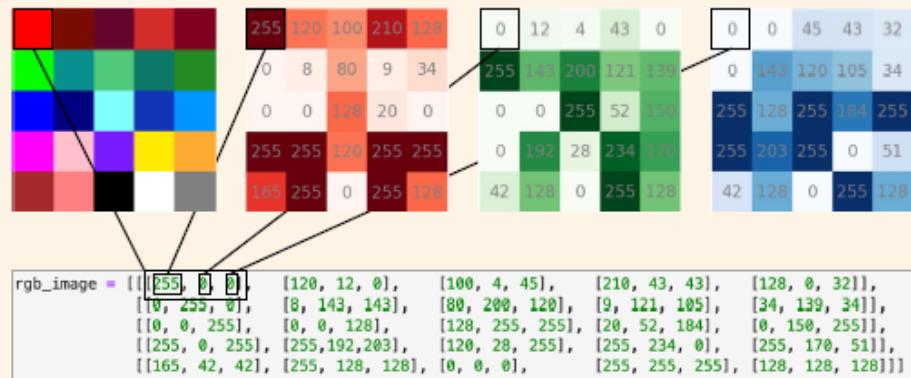


Figure 23.2. Digital representation of a colored image. Top (from left to right): RGB image, red components, green components, and blue components. Bottom: list of lists behind the rendered colored image.

Each pixel is represented by a sub-list composed of three numbers. For example, the top left pixel is red and is represented by the sub-list `[255, 0, 0]`, where `255` represents the amount of red, the first `0` is for the amount of green, and the second zero `0` is for the amount of blue. Each row is a list of lists, enclosed in a list of lists of lists! Finally, note that for both greyscale and colored images, the range of the numbers defining the color can go from `0` to `1` or from `0` to `255`.

## **Bài tập áp dụng**

1. Thủ nghiệm. Cho danh sách các danh sách sau:

numbers = [[3,7,1],[7,6,5,4],[8,9,7,4,5]].

- a. Mỗi danh sách con dài bao nhiêu?
- b. Trong danh sách con đầu tiên, thay thế phần tử thứ ba bằng tổng của hai phần tử trước đó.
- c. Trong danh sách con thứ hai, sắp xếp các phần tử theo thứ tự tăng dần.
- d. Trong danh sách con thứ ba, thay thế số 4 bằng số 3.
- e. Tổng cộng có bao nhiêu số 7? Lưu vị trí của chúng vào danh sách các danh sách (kết quả mong đợi: [[0, 1], [1, 3], [2, 2]]).

2. Tống kết. Cho danh sách các danh sách sau:

numbers = [[1,3,5],[7,2,8],[3,4,9]].

- a. Tạo một danh sách chứa tổng các số trong mỗi danh sách con (kết quả mong đợi: [9, 17, 16]).
- b. Tính tổng tất cả các phần tử của danh sách bằng cách sử dụng (1) vòng lặp *for* qua các chỉ số và (2) vòng lặp *for* qua các giá trị.

3. Đến giờ của ma trận! Cho ma trận sau:

matrix = [[4,1,3,9], [2,1,6,5], [4,0,3,8], [7,2,6,2]]

(Nếu bạn không quen với ma trận, hãy hình dung ma trận như một bảng chứa các số.)

- a. In ma trận dưới dạng bảng 4x4 (kết quả mong đợi:

[4, 1, 3, 9]

[2, 1, 6, 5]

[4, 0, 3, 8]

[7, 2, 6, 2]).

b. Nhân tất cả các phần tử trên đường chéo chính và in kết quả (kết quả mong đợi: 24).

Lưu ý:

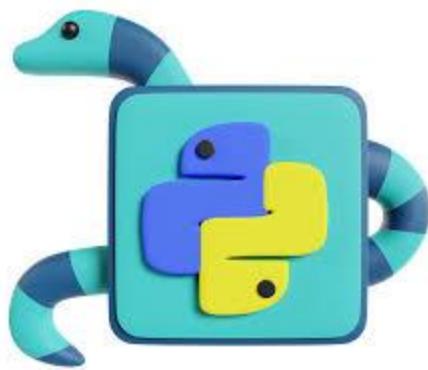
Đường chéo chính đi từ góc trên bên trái xuống góc dưới bên phải. Trong ví dụ này, đường chéo chính chứa: 4, 1, 3, 2.

c. Tính tổng các giá trị ma trận theo chiều dọc (kết quả dự kiến: [17, 4, 18, 24]).

## PHẦN 7

# TỪ ĐIỂN VÀ TỔNG QUAN VỀ CHUỖI

Trong ba chương đầu tiên của phần này, bạn sẽ học một kiểu dữ liệu mới gọi là dictionary. Trong chương cuối, bạn sẽ tích hợp kiến thức về chuỗi với các phương pháp và thủ thuật mới.

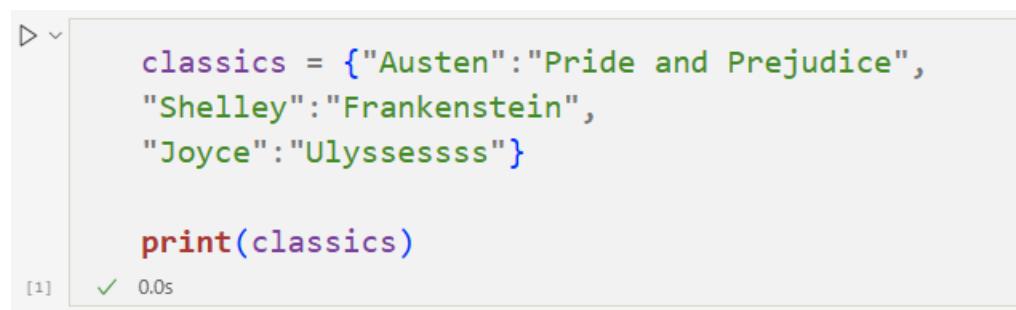


## 24. Kiểm kê tại hiệu sách tiếng Anh

### Từ điển

Bạn đã biết một số kiểu dữ liệu: chuỗi, danh sách, số nguyên, số thực và Boolean. Trong chương này, bạn sẽ tìm hiểu một kiểu dữ liệu mới gọi là từ điển. Từ điển là gì và chúng ta có thể làm gì với chúng?

- Bạn là chủ một hiệu sách tiếng Anh và đây là một số đầu sách kinh điển bạn đang bán:  
“tác giả” : “tác phẩm”



```
classics = {"Austen": "Pride and Prejudice",
            "Shelley": "Frankenstein",
            "Joyce": "Ulysses"}
```

```
print(classics)
```

[1] ✓ 0.0s

Kết quả: .....

- Bạn đang tiến hành kiểm kê và cần in tên tác giả và tiêu đề:



```
# as dict_items
print(classics.items())
# như một danh sách các bộ
print(list(classics.items()))
```

✓ 0.0s

Kết quả: .....

- Sau đó, bạn cần in riêng tên tác giả và tiêu đề:

```
# authors as dict_items
print(classics.keys())
# authors as a list
print(list(classics.keys()))
✓ 0.0s
```

Kết quả: .....

```
# titles as dict_items.
print(classics.values())
# titles as a list
print(list(classics.values()))
✓ 0.0s
```

Kết quả: .....

- Bạn nhận thấy tiêu đề của cuốn sách cuối cùng bị sai nên bạn sửa lại:

Cuốn sách cuối cùng tên tác giả là “Joyce”

```
▷ ▾
    print("Wrong title: " + classics["Joyce"])
[?] ✓ 0.0s
```

Kết quả: .....

```
classics["Joyce"] = "Ulysses"

print("Corrected title: " + classics["Joyce"])

✓ 0.0s
```

Kết quả: .....

- Sau đó, bạn thêm hai cuốn sách mới vừa ra mắt: *Gulliver's Travels* của Swift và *Jane Eyre* của Bronte:

```
> 
# adding the first book (syntax 1)
classics["Swift"] = "Gulliver's travels"

print(classics)
[9] ✓ 0.0s
```

Kết quả: .....

```
> 
# adding the second book (syntax 2)
classics.update({"Bronte": "Jane Eyre"})
print(classics)
[1] ✓ 0.0s
```

Kết quả: .....

- Cuối cùng, bạn loại bỏ những cuốn sách của Austen và Joyce vì bạn vừa bán chúng:

```
> 
# deleting the first book (syntax 1)
del classics["Austen"]
print(classics)
| ✓ 0.0s
```

Kết quả: .....

```
> 
# deleting the second book (syntax 2)
classics.pop("Joyce")
print(classics)
12] ✓ 0.0s
```

Kết quả: .....

**Từ điển** là một chuỗi các cặp khóa:giá trị được phân tách bằng dấu phẩy và nằm giữa hai dấu ngoặc nhọn {}

Sự khác biệt chính giữa danh sách và từ điển nằm ở cách chúng ta xác định vị trí của một phần tử hoặc giá trị. *Trong danh sách, chỉ mục sắp xếp các phần tử từ vị trí 0 đến vị trí  $\text{len}(\text{list})-1$ , theo cách liên tiếp và tăng dần (chúng ta không thể bỏ qua một vị trí!).* Một khác, **trong từ điển, các khóa không theo thứ tự cụ thể nào**. Ngoài ra, lưu ý rằng các số không thể được sử dụng làm khóa!

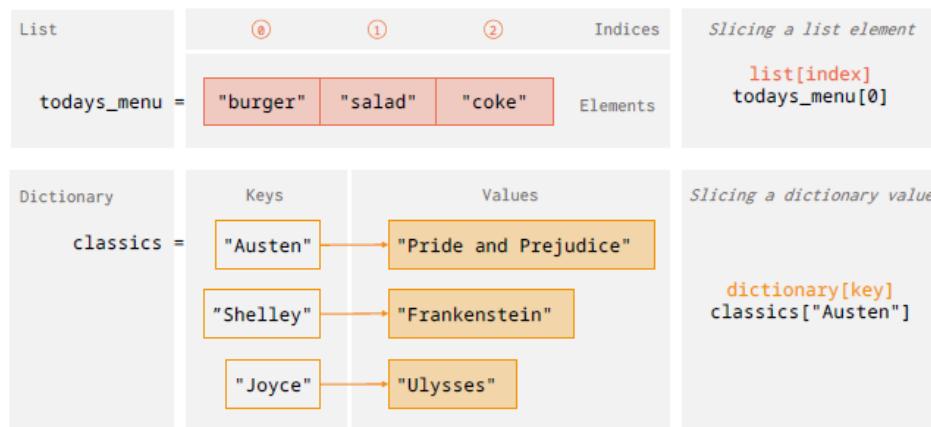


Figure 24.1. Comparing structure and slicing syntax for lists (top) and dictionaries (bottom).

Vì chúng ta không thể thay đổi chỉ số mà chỉ có thể thay đổi phần tử trong danh sách, chúng ta không thể thay đổi khóa mà chỉ có thể thay đổi giá trị trong từ điển.

## Tóm tắt

- Từ điển là một kiểu dữ liệu Python chứa một chuỗi các mục **key:value** được phân tách bằng dấu phẩy và nằm giữa các dấu ngoặc nhọn {}.
- Các phương thức từ điển **.items()**, **.keys()** và **.values()** được sử dụng để truy cập các mục, khóa và giá trị tương ứng.
- Để thay đổi giá trị từ điển, chúng ta ghi đè lên giá trị hiện có bằng cách sử dụng phương thức slicing.
- Để thêm một mục mới, chúng ta sử dụng cú pháp tương tự như phương thức slicing hoặc phương thức **.update()**.
- Để xóa một mục, chúng ta sử dụng từ khóa **del** hoặc phương thức **.pop()**.

## **Bài tập áp dụng**

1. Thông tin sinh viên. Đối với tình huống sau, hãy tạo mã tương tự như mã được trình bày trong chương này. Bạn làm việc tại Phòng Đăng ký của một trường học và đây là dữ liệu của một sinh viên:

```
student = {"First name": "Bruce", "Last name": "Zhiang", "Sex": "Male", "Age": 21,  
"Course": "Literature", "Hobby": "Swimming"}
```

- a. In ra tất cả các khóa và giá trị của chúng.
- b. In ra tất cả các khóa.
- c. In ra tất cả các giá trị.
- d. Bruce gần đây đã chuyển khóa học của mình từ *Literature* to *Foreign Languages*, vì vậy bạn cần cập nhật dữ liệu của anh ấy.
- e. Có hai thông tin bị thiếu: *Address* and *Phone number*, vì vậy bạn cần thêm chúng vào (sử dụng (hai cú pháp khác nhau)).
- f. Cuối cùng, do chính sách bảo mật mới, bạn phải xóa Sex và Hobby.

2. Tại một phòng khám thú y. Bạn là bác sĩ thú y tại một phòng khám thú y, và đây là một số thú cưng bạn hiện đang chăm sóc.

```
pets = [{"name": "Toby", "animal type": "dog", "age": 2},  
{"name": "Kitty", "animal type": "cat", "age": 5},  
{"name": "Tiki", "animal type": "parrot", "age": 1}]
```

- a. Bạn vừa tiếp nhận một bệnh nhân mới, một chú ngựa 4 tuổi tên là Sugar, và bạn thêm nó vào danh sách.

b. Nay giờ, bạn cần in ra tất cả tên các loài động vật. Trước tiên, hãy thực hiện việc này bằng vòng lặp *for* qua các phần tử và sau đó bằng vòng lặp *for* qua các chỉ mục.

c. Cuối cùng, bạn thêm một mục cho biết tất cả các loài động vật hiện đang ở phòng khám (bạn sử dụng kiểu dữ liệu nào?).

3. Nước ép! Bạn sở hữu một quầy nước ép và cần theo dõi lượng nước ép và doanh số bán hàng.

a. Tạo một danh sách các từ điển chứa 3 hương vị nước ép (cam, chanh và lựu), giá cả và màu sắc của chúng.

b. Với mỗi loại nước ép, hãy thêm một mục mới có khóa là 'in shop' và giá trị là Boolean.

c. Bạn vừa nhận được một đơn hàng mới (nước ép nho) và bạn thêm nó vào danh sách.

d. Giá trung bình của một loại nước ép là bao nhiêu?

## 25. Chuyến đi Thụy Sĩ

### Từ điển với danh sách là giá trị

Trong chương trước, bạn đã tìm hiểu về từ điển và danh sách của từ điển. Trong chương này, bạn sẽ học cách lập trình với các từ điển có giá trị là danh sách.

• Bạn của bạn đang lên kế hoạch cho một chuyến đi đến Thụy Sĩ và anh ấy đã hỏi bạn một số mẹo. Bạn bắt đầu với một từ điển trống để điền vào:

```
> 
  tips={} # từ điển Lên dùng {}

[ ]
```

Kết quả: .....

• Anh ấy muốn ghé thăm một số thành phố và thưởng thức ẩm thực đặc trưng. Vì vậy, bạn có thể đưa ra một số gợi ý sau:

```
tips["cities"] = ["Bern", "Lucern"]
tips["food"] = ["chocolate", "raclette"]

print(tips)
[15] ✓ 0.0s
```

Kết quả: .....

- Vì thời gian lưu trú của anh ấy là bốn ngày, bạn hãy thêm hai thành phố nữa và hai món ăn nữa:

```
# dùng .append()
tips["cities"].append("Lugano")
print(tips)
[16] ✓ 0.0s
```

Kết quả: .....

```
# cách 2 để thêm vào dùng +
tips["cities"] += ["Geneva"]
print(tips)
[17] ✓ 0.0s
```

Kết quả: .....

```
# dùng get() và append
tips.get("food").append("onion tarte")
print(tips)
[18] ✓ 0.0s
```

Kết quả: .....

```
# dùng get() và +
tips["food"] = tips.get("food") + ["fondu"]
print(tips)
[  ✓  0.0s
```

Kết quả: .....

- Bạn muốn kiểm tra xem từ điển có chính xác không, vì vậy bạn in từng mục một:

```
> ▾
    for k,v in tips.items(): # in key và value
        print(k,v)
20]  ✓  0.0s
```

Kết quả: .....

- Cuối cùng, bạn cải thiện bản in để dễ đọc hơn:

```
> ▾
    for k,v in tips.items():
        print("{:>6}: {}".format(k,v)) # cần phải trước dấu : 6 ký tự
1  ✓  0.0s
Python
```

Kết quả: .....

Chức năng của phương thức string `.format()` là gì? Nó định dạng các đối số và chèn chúng vào các chỗ giữa chỗ.

Còn nếu chúng ta chỉ muốn in các khóa hoặc các giá trị thì sao?

```
▷ ▾
    for k in tips.keys():
        print(k)
[22]  ✓  0.0s
```

Kết quả: .....

Tương tự cho giá trị.



```
for v in tips.values():
    print(v)
```

Kết quả: .....

### Tóm tắt

- Để khởi tạo một từ điển, chúng ta sử dụng cặp dấu ngoặc nhọn {} rỗng.
- Phương thức .get() của từ điển lấy một khóa làm đối số và trả về giá trị tương ứng.
- Có ít nhất 4 cách khác nhau để truy cập và sửa đổi các giá trị từ điển là danh sách, bằng cách kết hợp:
  - Cắt hoặc .get() để trích xuất một danh sách từ một từ điển.
  - Các thao tác danh sách (chẳng hạn như nối) hoặc các phương thức (ví dụ: .append()) để sửa đổi một danh sách.
- Chúng ta có thể sử dụng vòng lặp for để duyệt qua các giá trị để duyệt các mục, khóa và giá trị của một từ điển.
- Hàm tích hợp print() có thể lấy nhiều biến làm đối số:
  - Phân tách bằng dấu phẩy, hoặc
  - Sử dụng dấu giữ chỗ {} kết hợp với phương thức chuỗi .format().

### Bài tập áp dụng:

1. Cửa hàng nội thất. Bạn là quản lý của một cửa hàng nội thất. Dưới đây là các món đồ nội thất đang được lưu trữ:

```
store = {"furniture": ["chair", "table", "sofa"], "amount": [24, 7, 6], "price": [200, 500, 1200]}
```

- a. Một khách hàng mới đến và mua 4 chiếc ghế. Hãy cập nhật từ điển bằng một phép toán số học.
- b. Sau vài ngày, bạn nhận được những món đồ nội thất mới: 9 tấm thảm, mỗi tấm trị giá 150 đô la và 4 chiếc đèn, mỗi chiếc trị giá 180 đô la. Vì vậy, bạn thêm chúng vào từ điển (sử dụng các cú pháp khác nhau).
- c. Chủ một nhà hàng đến cửa hàng của bạn và mua tất cả các bàn. Hãy cập nhật từ điển (sử dụng ít nhất 2 cú pháp khác nhau).
- d. Để hình dung rõ hơn những gì còn lại, bạn in từ điển bằng cách căn chỉnh các khóa sang phải và các giá trị sang trái.
- e. Tổng giá của đồ nội thất trong kho là bao nhiêu?

2. Dịch chuyển các phần tử danh sách! Cho từ điển sau:

dictionary = {"numbers": [2,3,4,5,6,7,8,9,10]}

- a. Thêm một cặp khóa: giá trị, trong đó khóa là chuỗi chẵn và giá trị là một danh sách chứa True cho các số chẵn và False cho các số lẻ.

(Kết quả mong đợi:

```
{"numbers": [2, 3, 4, 5, 6, 7, 8, 9, 10],  
 "even": [True, False, True, False, True, False, True, False]}).
```

- b. Trừ 1 cho mỗi số.
- c. Làm thế nào để sửa đổi danh sách Boolean sao cho nó tương ứng với danh sách số mới? Gợi ý: Chỉ cần dịch chuyển nó!

4. Các số trong một hình tam giác! Yêu cầu người chơi nhập một số nguyên. Sau đó, in ra một hình tam giác, trong đó mỗi hàng chứa một số nguyên liên tiếp từ 1 đến số do người chơi nhập vào. Ngoài ra, mỗi hàng nên bao gồm một danh sách chứa số từ hàng đó được

lặp lại với số lần bằng chính số đó. Để làm được điều này, hãy sử dụng từ điển và cho phép người chơi chơi bao lâu tùy thích!

(Ví dụ: đầu vào: 5.

Đầu ra dự kiến:

1 [1]

2 [2, 2]

3 [3, 3, 3]

4 [4, 4, 4, 4]

5 [5, 5, 5, 5, 5]).

## 26. Đếm, nén và sắp xếp

Từ điển dùng để làm gì?

Trong chương này, chương cuối cùng dành riêng cho từ điển, bạn sẽ tìm hiểu một số tình huống điển hình mà việc sử dụng từ điển rất tiện lợi. Hãy thử tự giải từng ví dụ trước khi xem xét giải pháp.

### 1. Đếm phân tử

Từ điển cực kỳ tiện lợi khi chúng ta cần lưu các lần xuất hiện, tức là số lần một điều gì đó xảy ra. Hãy cùng tìm hiểu ý nghĩa của điều này qua ví dụ sau.

- Cho chuỗi sau:

```
greetings = "hello! how are you?"  
print(greetings)
```

Kết quả: .....

- Tạo một từ điển trong đó các khóa là các chữ cái trong bảng chữ cái được tìm thấy trong chuỗi, và các giá trị tương ứng là số lần xuất hiện của mỗi chữ cái. Viết mã theo hai cách:  
(1) sử dụng *if/elif*; và (2) sử dụng *.get()*.

### 1. Sử dụng *if/elif*:

```
# tạo thư viện ký tự và giá trị Là số đếm
demkytu={}

for kytu in greetings:
    if kytu not in demkytu.keys(): # nếu ký tự đó chưa có trong
                                    # key của đếm ký từ
        demkytu[kytu]=1;
    else:
        demkytu[kytu]+=1

for k,v in demkytu.items():
    print(k,v)
```

[25] 0.0s Python

Kết quả: .....

## 2. Sử dụng .get():

```
# tạo thư viện ký tự và giá trị là số đếm
demkytu=[]

for kytu in greetings:
    demkytu[kytu]=demkytu.get(kytu,0)+1

for k,v in demkytu.items():
    print(k,v)
```

✓ 0.0s

Phương thức `.get()` chứa hai đối số, *ký tự* (*đang là key*) và *0*, và hoạt động như sau: nếu khóa không tồn tại, `.get()` trả về đối số thứ hai là *0*; nếu khóa đã tồn tại, `.get()` trả về giá trị tương ứng là *ký tự*.

Kết quả: .....

## 2. Nén thông tin

Từ điển cực kỳ tiện lợi cho việc nén thông tin dư thừa: ví dụ, để lưu trữ các tín hiệu mà cảm biến thu được trong một thời gian dài. Hãy nghĩ đến một cảm biến được sử dụng để phát hiện rung động trong trường hợp động đất. Hầu hết thời gian, cảm biến chỉ ghi lại các số *0* vì không có sự kiện địa chấn nào. Tuy nhiên, khi một trận động đất xảy ra, cảm biến sẽ ghi lại một xung (hoặc một nhóm xung) có cường độ khác *0*. Việc lưu trữ ngày và giờ của các số *0* trong một danh sách sẽ đòi hỏi một lượng bộ nhớ máy tính đáng kể, và điều này sẽ hơi vô nghĩa vì thông tin tín hiệu nằm trong các xung. Để giảm dung lượng bộ nhớ lưu trữ trong khi vẫn giữ được thông tin, chúng ta có thể sử dụng từ điển. Bạn sẽ làm điều đó như thế nào? Và sau đó bạn sẽ quay lại từ từ điển về danh sách ban đầu như thế nào?

- Cho danh sách sau:

```
sparse_vector = [0, 0, 0, 1, 0, 7, 0, 0,  
4, 0, 0, 0, 8, 0, 0, 0, 6, 0, 0, 0, 0, 0,  
0, 0, 9, 0, 0]  
  
print(sparse_vector)  
| ✓ 0.0s
```

Kết quả: .....

Chúng ta bắt đầu với một danh sách gọi là sparse\_vector, chứa nhiều số 0 và một vài số nguyên nằm rải rác giữa các số 0. (Lưu ý: trong đại số tuyến tính, vecto thừa thớt là vecto mà phần lớn các thành phần là số 0.)

- *Chuyển đổi nó thành một từ điển:*

Từ điển ở đây ghi lại vị trí khác 0 và giá trị của nó

Cú pháp từ điển: “vị trí” : “giá trị”

```
# tạo từ điển cho nó  
sparse_dict = {}  
for i in range(len(sparse_vector)):  
    if sparse_vector[i] != 0: # nếu giá trị tại đó khác 0  
        sparse_dict[i] = sparse_vector[i] # ghi vào từ điển vị trí i  
        # và giá trị khác 0 đó  
  
print(sparse_dict)  
| ✓ 0.0s
```

Python

Kết quả: .....

```
# thêm Lưu trữ chiều dài của dãy số vào từ điển  
sparse_dict["length"] = len(sparse_vector)  
  
# in ra từng key và value tương ứng  
for k,v in sparse_dict.items():  
    print(k,v)  
| ✓ 0.0s
```

Kết quả: .....

- *Làm thế nào để chúng ta quay lại vectơ thưa thớt? (khôi phục)*

```
#đầu tiên xác định chiều dài qua thông số Lưu trữ
khoiphuc=[0]*sparse_dict["length"]
# ban đầu tất cả đều là 0
# Bây giờ chúng ta khôi phục các vị trí khác 0

#add thêm các giá trị 0 vào
for k,v in sparse_dict.items():
    if k!="length": # Loại bỏ phần tử cuối cùng
        khoiphuc[k]= v # khôi phục vị trí k giá trị v

# hiển thị
print(khoiphuc)
31] ✓ 0.0s
```

Kết quả: .....

### 3. Sắp xếp từ điển

Trong ví dụ cuối cùng này về từ điển và ứng dụng của chúng, chúng ta sẽ học cách sắp xếp từ điển theo khóa hoặc giá trị của chúng. Hãy xem xét một số đăng ký thành phố được đơn giản hóa, trong đó tên công dân là khóa và tuổi của họ là giá trị. Các sĩ quan có

thể cần sắp xếp số **đăng ký theo tên** để gửi thư, hoặc theo độ tuổi để phân biệt trẻ em với người già. Hãy cùng xem cách thực hiện!

- Cho từ điển sau:

```
> registry = {"Shaili":4, "Chris":90, "Maria":70}

    print(registry)
32] ✓ 0.0s
```

Kết quả: .....

- Sắp xếp các mục từ điển theo khóa của chúng:

```
• # khởi tạo từ điển sau khi được sắp xếp
sapxep=[]

# sắp xếp theo key
# đầu tiên Lấy hết danh sách key

sapxep_keys=list(registry.keys())

sapxep_keys.sort();

print(sapxep_keys)
14] ✓ 0.0s
```

Kết quả: .....

```
#sau đó đi tìm value tương ứng
for k in sapxep_keys:
    sapxep[k]=registry[k]

print(sapxep)
3] ✓ 0.0s
```

Kết quả: .....

- *Sắp xếp các mục từ điển theo giá trị của chúng:*

```
# khởi tạo từ điển sau khi được sắp xếp
sapxep=[]

# sắp xếp theo key theo giá trị
sapxep_key=sorted(registry,key=registry.get)

print(sapxep_key)

✓ 0.0s
```

Kết quả: .....

```
#sau đó đi tìm value tương ứng
for k in sapxep_key:
    sapxep[k]=registry[k]

print(sapxep)

3] ✓ 0.0s
```

Kết quả: .....

Điểm khác biệt nằm ở cách chúng ta sắp xếp (sort) các khóa, tức là theo giá trị từ điển.

Để làm điều đó, chúng ta sử dụng hàm `sorted()` tích hợp sẵn, hàm này nhận hai đối số: (1) từ điển có khóa mà chúng ta muốn sắp xếp và (2) từ điển được kết hợp với phương thức `.get` (lưu ý không có dấu ngoặc tròn). Lưu ý rằng `sorted()` cũng có thể được sử dụng với danh sách và chuỗi—chủ yếu chỉ với một đối số—như một phương án thay thế cho phương thức `.sort()`. Sự khác biệt là `sorted()` trả về **một biến** (ví dụ: `sorted_list = sorted(original_list)`), trong khi `.sort()` tác động trực tiếp lên **danh sách** (ví dụ: `original_list.sort()`).

## Tóm tắt

- Một số ví dụ điển hình về việc sử dụng từ điển bao gồm đếm phần tử, nén thông tin và sắp xếp từ điển theo khóa và giá trị.
- Phương thức **.get(key,initial value)** của từ điển được sử dụng để khởi tạo cặp khóa:giá trị trong từ điển và điền vào cặp này trong vòng lặp *for*.
- Hàm tích hợp **sorted()** được sử dụng để sắp xếp từ điển; lưu ý rằng nó tạo ra một biến mới.

## Bài tập áp dụng

1. Từ từ điển đến danh sách các danh sách và ngược lại! Cho từ điển sau:

```
cars = {"sports car":4, "convertible":5, "limousine":2}
```

a. Chuyển đổi từ điển thành danh sách các danh sách.

(Kết quả mong đợi: [['sports car', 4], ['convertible', 5], ['limousine', 2]]).

b. Chuyển đổi danh sách các danh sách trở lại từ điển ban đầu.

2. Trò chơi bảng cửu chương! Bạn là một lập trình viên tại một công ty trò chơi giáo dục. Nhiệm vụ của bạn là tạo một trò chơi trong đó trẻ em nhập một số và bạn sẽ hiển thị bảng cửu chương tương ứng. Để triển khai trò chơi, hãy tạo một từ điển trong đó các khóa là các số từ 1 đến 10 và các giá trị là kết quả của phép nhân giữa khóa và giá trị do trẻ nhập. Sử dụng vòng lặp *for* và cho phép trẻ chơi bao lâu tùy thích.

(Ví dụ đầu vào: 4

Ví dụ đầu ra:

$$1 \times 4 = 4$$

$$2 \times 4 = 8$$

3 x 4 = 12

4 x 4 = 16

5 x 4 = 20

6 x 4 = 24

7 x 4 = 28

8 x 4 = 32

9 x 4 = 36

10 x 4 = 40).

3. Gia vị và thảo mộc. Bạn làm việc trong một cửa hàng tạp hóa bán gia vị và thảo mộc.

Sau đây là các loại gia vị và thảo mộc trong cửa hàng:

```
spices_herbs = ["basil", "cinnamon", "licorice", "mint", "rosemary", "thyme",
"cardamom", "turmeric", "cilantro", "oregano", "pepper", "chili", "dill", "cayenne
pepper", "ginger", "garlic", "marjoram", "nutmeg", "sage", "saffron", "star anise", "bay
leaves"]
```

a. Bạn phải thay đổi nhãn trên hộp đựng và làm cho chúng trông hiện đại hơn. Chiều dài của nhãn mới tỷ lệ thuận với chiều dài của từ được viết trên đó. Tạo một từ điển trong đó khóa là độ dài của từ và giá trị là danh sách các từ có độ dài đó.

b. Bạn cần biết cần cắt bao nhiêu nhãn cho mỗi độ dài. Tạo một từ điển khác trong đó khóa là độ dài của từ theo thứ tự tăng dần, và giá trị là số nhãn bạn phải cắt cho mỗi độ dài.

c. Nhãn phổ biến nhất là gì? Nó tương ứng với bao nhiêu chữ cái? Hãy tính xem!

Tiếng việt: spices\_herbs = ["húng quế", "quế", "cam thảo", "bạc hà", "hương thảo", "húng
tây", "thảo mộc", "nghệ", "rau mùi", "kinh giới cay", "ót", "ớt", "thì là", "ớt cayenne",

"gừng", "tỏi", "kinh giới cay", "nhục đậu khấu", "xô thơm", "nghệ tây", "đại hồi", "lá nguyệt quế"]

## 27. Tổng quan về chuỗi

Các phép toán, phương thức và in ấn

Trong chương này, chúng ta sẽ tóm tắt các đặc điểm của chuỗi, tương tự như những gì chúng ta đã làm với danh sách trong Chương 21. Bạn sẽ nhận thấy rất nhiều điểm chung giữa hai kiểu dữ liệu này, nhưng cũng có một số điểm khác biệt quan trọng. Hãy theo dõi Sổ tay 27. Như thường lệ, hãy thử giải các bài toán trước khi tìm giải pháp. Hãy bắt đầu thôi!

### 1. Cắt chuỗi

Cắt chuỗi hoạt động giống như cắt danh sách (xem Chương 12). Hãy xem hai ví dụ dưới đây để ôn lại.

- Cho chuỗi sau:

```
two_ways = "rsecwyadrkd"
print(two_ways)
] ✓ 0.0s
```

Kết quả: .....

- Trích xuất từng ký tự thứ hai:

```
# từ đầu đến cuối và bước nhảy là 2

print(two_ways[::2])
] ✓ 0.0s
```

Kết quả: .....

- Trích xuất từ thứ hai và đảo ngược kết quả:

```
# từ cuối lên đầu, theo chiều âm và bước nhảy là 2
print(two_ways[::-2])
[7] ✓ 0.0s
```

Kết quả: .....

## 2. Các phép toán "số học" trên chuỗi

Có hai phép toán "số học" trên chuỗi: nối chuỗi và sao chép chuỗi. Chúng tuân theo cùng các nguyên tắc như danh sách. Hãy cùng xem nhanh cách chúng hoạt động.

- Nối chuỗi hai chuỗi:

```
daya="cười"
dayb="vui"
combo=daya+dayb
print(combo)
[48] ✓ 0.0s
```

Kết quả: .....

- Sao chép một chuỗi 5 lần:

```
motcuoi=":)"
namcuoi=motcuoi*5
print(namcuoi)
[49] ✓ 0.0s
```

Kết quả: .....

## 3. Thay thế hoặc xóa chuỗi con

Chuỗi con là một phần của chuỗi. Trong nhiều ví dụ sau, chúng ta sẽ sử dụng chuỗi con chỉ gồm một ký tự để đơn giản. Tuy nhiên, các quy tắc trong ví dụ cũng áp dụng cho chuỗi con gồm nhiều ký tự. Hãy cùng tìm hiểu cách thay thế hoặc xóa chuỗi con trong

một chuỗi dựa trên vị trí hoặc nội dung của chuỗi con. Hãy bắt đầu bằng cách thay đổi chuỗi con dựa trên vị trí của nó.

- Cho chuỗi sau:

```
favorites = "I like cooking, my family, and my friends"
print(favorites)
```

✓ 0.0s

Kết quả: .....

- Thay thế ký tự ở vị trí 0 bằng U bằng cách sử dụng cắt và gán. Điều gì xảy ra?

```
favorites[0] = "U"
```

i1] ✘ 0.0s

Kết quả: .....

- Làm lại nhiệm vụ tương tự bằng cách sử dụng cắt và nối:

```
from_position_one = favorites [1:] # xoá phần tử đầu tiên

favorites= "U" + from_position_one # nối U với dãy đã xoá

print(favorites)
```

✓ 0.0s

Kết quả: .....

- Làm lại cùng một tác vụ bằng phương thức chuỗi `.split()`:

```
#bằng phương thức chuỗi .split():

favorites = "I like cooking, my family, and my friends"
print(favorites)

parts = favorites.split("I") # cắt bỏ chữ I
print(parts)

favorites= "U" + parts[1] # nối với thành phần part [1]
print(favorites)
```

Kết quả: .....

Nếu chúng ta muốn sửa đổi một chuỗi con dựa trên nội dung thay vì vị trí của nó thì sao?  
Hãy cùng xem nhé!

- Thay thế dấu phẩy bằng dấu chấm phẩy bằng phương thức chuỗi: `.replace()`:

```
favorites = "I like cooking, my family, and my friends"
favorites = favorites.replace(",", ";")
print(favorites)
```

Kết quả: .....

- Bỏ dấu phẩy:

```
favorites = "I like cooking, my family, and my friends"
favorites = favorites.replace(",", "")
print(favorites)
```

Kết quả: .....

#### 4. Tìm kiếm một chuỗi con trong một chuỗi

Làm thế nào để tìm một chuỗi con trong một chuỗi? Hãy xem bên dưới!

- Cho chuỗi sau:



```
us = "we are"
print(us)
[57]    ✓  0.0s
```

Kết quả: .....

- Tìm vị trí của ký tự e bằng phương thức `.find()`:



```
positions = us.find("e")
print(positions)
|    ✓  0.0s
```

Kết quả: .....

Chúng ta sử dụng phương thức `.find()` lấy chuỗi con mà chúng ta muốn tìm làm đối số—trong trường hợp này là "e". Chúng ta chỉ nhận được vị trí 1, trong khi ở us, "e" nằm ở vị trí 1 và 5. Điều này xảy ra vì phương thức `find()` chỉ trả về vị trí của chuỗi con đầu tiên mà nó tìm thấy. Làm thế nào chúng ta có thể tìm vị trí của tất cả các chuỗi con "e"? Hãy thử trả lời câu hỏi này trước khi xem xét giải pháp bên dưới!

- Tìm vị trí của ký tự e bằng một cách khác:

```
#khởi tạo dãy vị trí
positions=[]

# tìm tất cả
for i in range(len(us)):
    if us[i]=="e":
        positions.append(i)

print(positions)
✓ 0.0s
```

Kết quả: .....

- Tìm vị trí của ký tự f bằng phương thức `find()`:

```
> ^
    positions = us.find("f")
    print(positions)
61] ✓ 0.0s
```

Kết quả: .....

Kết quả là -1. Do đó, khi chúng ta tìm kiếm một chuỗi con *không có trong chuỗi*, `find()` trả về -1. Đây là một mẹo thường được sử dụng trong các điều kiện, chẳng hạn như: if `us.find("f") == -1: print("Không tìm thấy ký tự!")`.

## 5. Đếm số chuỗi con trong một chuỗi

- Cho chuỗi sau:

```
^
    hobbies = "I like going to the movies, traveling, and singing"
    print(hobbies)
2] ✓ 0.0s
```

Kết quả: .....

- *Đếm số lượng chuỗi con bằng phương thức .count():*

```
demchuoi= hobbies.count("ing")
print(demchuoi)
```

✓ 0.0s

Kết quả: .....

## 6. Chuỗi vào danh sách và ngược lại

Có thể thuận tiện khi tách các từ trong một chuỗi thành các phần tử danh sách hoặc gộp các chuỗi là phần tử của danh sách thành một chuỗi duy nhất. Hãy cùng xem cách thực hiện cả hai thao tác.

- *Cho chuỗi sau:*

```
string = "How are you"
print(string)
```

Kết quả: .....

- *Chuyển đổi chuỗi thành danh sách các chuỗi trong đó mỗi phần tử là một từ:*

```
> 
list_of_strings = string.split()

print(list_of_strings)
66] ✓ 0.0s
```

Kết quả: .....

Làm thế nào để quay lại danh sách? Hãy cùng tìm hiểu ở ô tiếp theo!

- *Chuyển đổi danh sách các chuỗi thành chuỗi bằng phương thức .join():*

```
[72] | # thêm khoảng trắng giữa các thành phần
      | string_from_list = " ".join(list_of_strings)
      | print(string_from_list)
[72] | ✓ 0.0s
```

Kết quả: .....

## 7. Thay đổi kiểu chữ

Có một số tùy chọn khi thay đổi kiểu chữ. Hãy cùng xem nhanh qua ví dụ đơn giản dưới đây.

- Cho chuỗi ký tự sau:

```
[73] | loichao="Xin chào! Chúng tôi là DHV"
      | print(loichao)
[73] | ✓ 0.0s
```

Kết quả: .....

- Sửa đổi chuỗi thành chữ hoa và chữ thường; chỉ đổi ký tự đầu tiên của chuỗi thành chữ hoa, sau đó là từng từ trong chuỗi; cuối cùng, đảo ngược các trường hợp:

```
#in hoa
print(loichao.upper())

#in thường
print(loichao.lower())

# Viết hoa chữ đầu tiên
print(loichao.capitalize())

# Viết hoa chữ cái đầu tiên mỗi chữ
print(loichao.title())

# Hoán đổi hoa thành thường, thường thành Hoa
print(loichao.swapcase())
```

✓ 0.0s

Kết quả: .....

## 8. In biến

Việc in đặc biệt hữu ích trong lập trình để kiểm tra tính chính xác của các phép toán và thuật toán. Trong các chương trước, chúng ta đã học rằng các đối số của hàm `print()` tích hợp có thể là các biến được nối, các biến được phân cách bằng dấu phẩy, hoặc một chuỗi kết hợp với phương thức `format()`. Bên cạnh việc làm mới các phương thức in này và chỉ ra một số điểm đặc biệt, chúng ta sẽ tìm hiểu về chuỗi f và các cách dễ dàng hơn để in các biến số tốt hơn.

Hãy bắt đầu!

- Cho chuỗi sau:

```
chaosang="morning"
print(chaosang)
```

✓ 0.0s

Kết quả: .....

- In ra Good morning! theo 4 cách khác nhau, sử dụng (1) nối chuỗi, (2) phân tách bằng dấu phẩy, (3) phương thức *format()* và (4) chuỗi *f*:

```
#(1) Nối chữ +
print("Good " + chaosang + "!")

#(2) dùng dấu ","
print("Good" , chaosang , "!")

# (3) the method .format()
print("Good {}".format(chaosang))

# (4) f-strings
print(f"""Good {chaosang}!""")
```

✓ 0.0s

Kết quả: .....

(2) Điều này xảy ra vì trong in ấn phân cách bằng dấu phẩy, các biến luôn được phân cách bằng một khoảng trắng.

(3) Một cách khác là sử dụng phương thức chuỗi `.format()`, phương thức này đặt đối số của nó vào vị trí giữ chỗ `{}` trong chuỗi. Trong trường hợp này, giá trị của `chaosang`—là đối số của `.format()`—được đặt trong dấu ngoặc nhọn tại "Good `{}`!".

(4) Phương thức cuối cùng là sử dụng f-strings, **trong đó f là viết tắt của formatted (được định dạng)**. Trong dấu ngoặc tròn của `print()`, chúng ta viết: (1) `f`, (2) cây mở ngoặc kép `""",` (3) nội dung chúng ta muốn in, và (4) cây đóng ngoặc kép `""".` Trong trường hợp này, nội dung chúng ta muốn in bao gồm một số ký tự (ví dụ: Good và !) và một biến phải được đặt trong cặp dấu ngoặc nhọn—tức là `{chaosang}`.

Nếu chúng ta muốn in một chuỗi kết hợp với một biến số thì sao? Hãy cùng xem nhé!

- Cho một chuỗi và một biến số:

A screenshot of a Jupyter Notebook cell. The code is as follows:

```
chaosang="morning"
thoigian=10
print(chaosang)
print(thoigian)
```

The cell has a progress bar at [83] and a green checkmark indicating it took 0.0s to run.

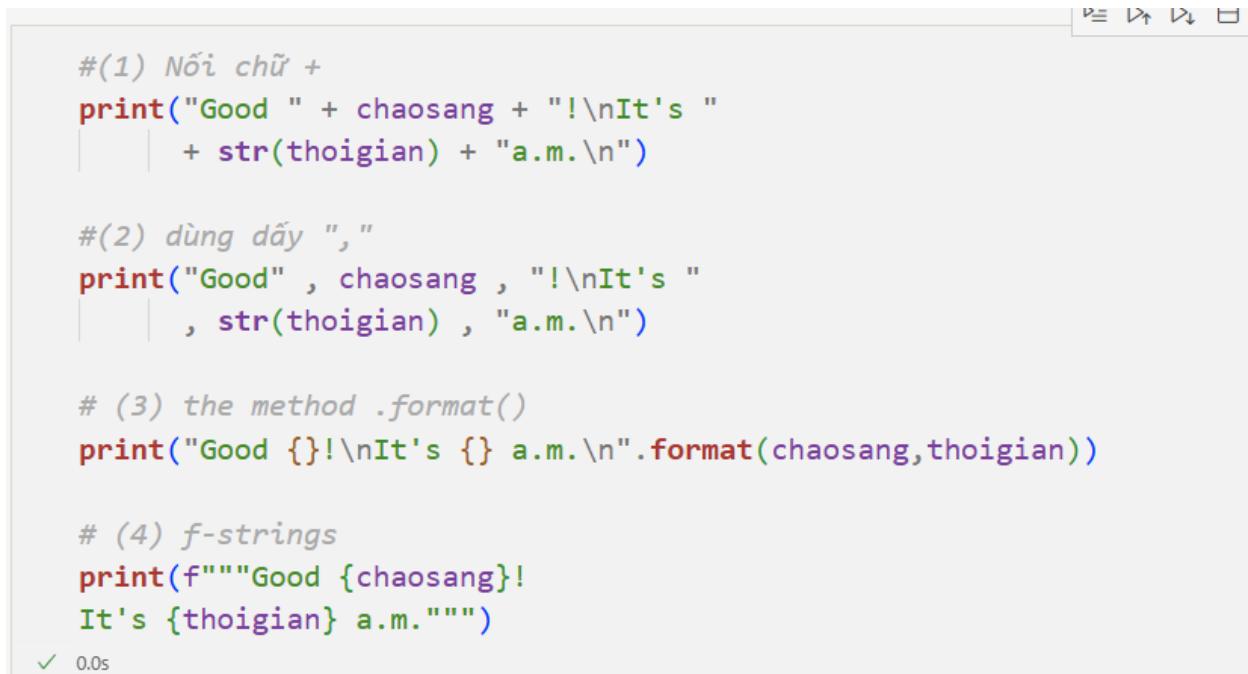
Kết quả: .....

- In

*Good morning!*

*It's 10a.m.*

sử dụng cùng bốn phương pháp trên (lưu ý rằng các câu được viết trên hai dòng riêng biệt):



```
#(1) Nối chữ +
print("Good " + chaosang + "!\nIt's "
      | + str(thoigian) + "a.m.\n")

#(2) dùng dấu ","
print("Good" , chaosang , "!\nIt's "
      | , str(thoigian) , "a.m.\n")

# (3) the method .format()
print("Good {}!\nIt's {} a.m.\n".format(chaosang,thoigian))

# (4) f-strings
print(f"""Good {chaosang}!
It's {thoigian} a.m.""")

✓ 0.0s
```

Kết quả: .....

Vậy còn việc in số với số thập phân ít hơn thì sao? Hãy cùng xem các ví dụ sau.

- Cho biến số sau:

```
giatri=1.23456
print(giatri)
[✓] 0.0s
```

Kết quả: .....

- In Số là 1,23—chỉ lưu ý hai chữ số thập phân đầu tiên—sử dụng bốn phương pháp trên:

```
#(1) Nối chữ +
print("Giá trị là " + str(round(giatri,2)))

#(2) dùng dấu ","
print("Giá trị là " , str(round(giatri,2)))

# (3) the method .format()
print("Giá trị là {:.2f}".format(giatri))

# (4) f-strings
print(f"""Giá trị là {giatri:.2f} """)
```

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

### Tóm tắt

- Trong chuỗi, việc cắt và các phép toán "số học" (nối chuỗi và sao chép) hoạt động theo cùng một cách như trong danh sách.
- Chuỗi không thể thay đổi và do đó không thể gán.

- Chuỗi có 47 phương thức. Trong số đó, 11 phương thức đã học cho đến nay là:  
.capitalize(), .count(), .find(), .format(), .join(), .lower(), .replace(), .split(), .swapcase(),  
.title() và .upper().
- Có ít nhất bốn cách để kết hợp chuỗi và biến số khi in: nối chuỗi, phân tách bằng dấu phẩy, phương thức .format() và f-strings.
- Để làm tròn một số đến số thập phân mong muốn, chúng ta có thể sử dụng hàm round()  
tích hợp sẵn.

## **Bài tập áp dụng**

1. Những câu nói nổi tiếng. Cho chuỗi ký tự sau:

quote = "The future belongs to those who believe in the beauty of their dreams -  
Eleanor Roosevelt"

Sử dụng các phương thức xâu ký tự để:

- Xóa đến những *who*.
- Thay thế *belongs* về *until*.
- Thêm *seems impossible* sau *future*.
- Xóa *The future*.
- Thay thế *believe in the beauty of their dreams* bằng *it's done*.
- Thay thế *Eleanor Roosevelt* bằng *Nelson Mandela*.
- Thêm *It always* vào đầu chuỗi ký tự.

Bạn sẽ nhận được câu trích dẫn nào ở cuối? Hãy đảm bảo các từ được phân cách bằng dấu cách.

2. Điểm chung. Cho các chuỗi sau:

dessert = "lemon meringue pie"

sweet = "honeypot"

- a. Hai chuỗi có những ký tự chung nào? Lưu các ký tự chung vào một danh sách.
- b. Các ký tự chung xuất hiện bao nhiêu lần trong dessert? Lưu kết quả vào một từ điển được tạo theo hai cách khác nhau, tức là sử dụng (1) phương pháp từ điển, và (2) phương pháp chuỗi.
3. Từ đối xứng. Từ đối xứng là những từ đọc xuôi hay ngược đều giống nhau, chẳng hạn như anna hoặc madam. Cho danh sách các chuỗi sau:

words = ["noon", "dog", "dad", "elephant", "jungle", "otto", "night", "bright",  
"kayak", "yeah", "wow"]

Lưu các từ đối xứng vào một danh sách chuỗi mới. Gợi ý: Cân nhắc sử dụng phương pháp cắt chuỗi.

.startswith()	Checks if the string starts with the specified prefix E.g.: print("hello, how are you?".startswith("hello")) returns: True
.strip()	Removes characters on the left and on the right E.g.: print("!!!hello!!!".strip("!")) returns: hello
.swapcase()*	Swaps the case of all characters in the string E.g.: print("Hello, How Are You?".swapcase()) returns: hELLO, hOW aRE yOU?
.title()*	Converts the string to title case E.g.: print("hello, how are you?".title()) returns: Hello, How Are You?
.translate()	Maps the character of a string through a given translation table (see .maketrans()) E.g.: transformation = "bake".maketrans("b", "c"); print("bake".translate(transformation)) returns: cake
.upper()*	Converts the string to uppercase E.g.: print("hello".upper()) returns: HELLO
.zfill()	Adds 0 at the beginning of the string until the string reaches the defined length E.g.: print("5".zfill(4)) returns: 0005

String method	What it does
<code>.capitalize()*</code>	Converts the first character to uppercase and all the others to lowercase E.g.: <code>print("hello".capitalize())</code> returns: Hello
<code>.casefold()</code>	Converts a string into lowercase. Differently from <code>.lower()</code> , it can handle more complex cases. E.g.: <code>print("Straße".casefold())</code> returns: "strasse". <code>print("Straße".lower())</code> returns: "straße" (Straße is street in German)
<code>.center()</code>	Returns a string centered within a given number of characters. E.g.: <code>print("hi".center(6))</code> returns: " hi "
<code>.count()*</code>	Returns the number of times a specified value is present in a string E.g.: <code>print("singing".count("ing"))</code> returns: 2
<code>.encode()</code>	Returns an encoded version of the string using the specified encoding—encodings define how characters are rendered on a screen. E.g.: <code>print("hello".encode(encoding='utf-8'))</code> returns: <code>b'hello'</code>
<code>.endswith()</code>	Returns true if the string ends with the specified value. E.g.: <code>print("hello".endswith('lo'))</code> returns: True
<code>.expandtabs()</code>	Make the tabs in the string of the length defined by the arguments. E.g.: <code>print("h\te\tl\tl\tto".expandtabs(3))</code> returns: h e l l o
<code>.find()*</code>	Return the first position of a substring E.g.: <code>print("singing".find("ing"))</code> returns: 1
<code>.format()*</code>	Formats the string using the specified arguments E.g.: <code>print("Hello, {}".format("how are you?"))</code> returns: Hello, how are you?
<code>.format_map()</code>	Formats specified values—defined in a dictionary—in a string E.g.: <code>print("My dog name is {name} years old".format_map({"name": "Ninja", "age": 7}))</code> returns: My dog Ninja is 7 years old
<code>.index()*</code>	Finds the first substring of a substring E.g.: <code>print("hello".index("l"))</code> returns: 2
<code>.isalnum()*</code>	Checks if all characters in the string are alphanumeric E.g.: <code>print("123hello".isalnum())</code> returns: True
<code>.isalpha()*</code>	Checks if all characters in the string are alphabetic E.g.: <code>print("hello".isalpha())</code> returns: True
<code>.isascii()</code>	Checks if all characters in the string are ASCII. E.g.: <code>print("é".isascii())</code> returns: False
<code>.isdecimal()</code>	Checks if all characters in the string are decimals E.g.: <code>print("123".isdecimal())</code> returns: True
<code>.isdigit()*</code>	Checks if all characters in the string are digits E.g.: <code>print("123".isdigit())</code> returns: True
<code>.isidentifier()</code>	Checks if the string is a valid identifier, that is, if it only contains alphanumeric letters (a-z and 0-9) or underscores (_), and it does not start with a number nor contain spaces E.g.: <code>print("my string".isidentifier())</code> returns: False
<code>.islower()*</code>	Checks if all characters in the string are lowercase E.g.: <code>print("hello".islower())</code> returns: True
<code>.isnumeric()</code>	Checks if all characters in the string are numeric E.g.: <code>print("123".isnumeric())</code> returns: True
<code>.isprintable()</code>	Checks if all characters in the string are printable E.g.: <code>print("\n".isprintable())</code> returns: False

<code>.isspace()</code>	Checks if all characters in the string are space E.g.: <code>print(" ".isspace())</code> returns: True
<code>.istitle()*</code>	Checks if the string is title-cased E.g.: <code>print("Hello, How Are You?".istitle())</code> returns: True
<code>.isupper()*</code>	Checks if all characters in the string are uppercase E.g.: <code>print("HELLO".isupper())</code> returns: True
<code>.join()*</code>	Joins the strings of a list with the specified separator E.g.: <code>print(", ".join(["hello", "hi"]))</code> returns: hello, hi
<code>.ljust()</code>	Left-justifies the string E.g.: <code>print("hello".ljust(10, '-'))</code> returns: hello-----
<code>.lower()*</code>	Converts the string to lowercase E.g.: <code>print("HELLO".lower())</code> returns: hello
<code>.lstrip()</code>	Removes characters at the beginning of the string (l is for left) E.g.: <code>print("hhello".lstrip("h"))</code> returns: ello
<code>.maketrans()</code>	Transforms the transformation of the characters of the first argument into the characters of the second argument. To print the outcome, we need to use the method <code>.translate()</code> E.g.: <code>transformation = "bake".maketrans("b", "c"); print("bake".translate(transformation))</code> returns: cake
<code>.partition()</code>	Partitions the string into tuple elements E.g.: <code>print("hello, how are you?".partition(" "))</code> returns: ('hello,', ' ', 'how are you?')
<code>.removeprefix()</code>	Removes the specified prefix from the string E.g.: <code>print("hello, how are you?".removeprefix("hello, "))</code> returns: how are you?
<code>.removesuffix()</code>	Removes the specified suffix from the string E.g.: <code>print("hello, hi".removesuffix(" ", hi))</code> returns: hello
<code>.replace()*</code>	Replaces substrings in strings E.g.: <code>print("Hello, how is she?".replace("she", "he"))</code> returns: Hello, how is he?
<code>.rfind()</code>	Finds the last substring of a string (r is for right) E.g.: <code>print("hello".rfind('l'))</code> returns: 3
<code>.rindex()</code>	Finds the last substring of a string E.g.: <code>print("hello".rindex('ll'))</code> returns: 2
<code>.rjust()</code>	Right-justifies the string E.g.: <code>print("hello".rjust(10, '-'))</code> returns: -----hello
<code>.rpartition()</code>	Partitions the string into tuple elements starting from the end E.g.: <code>print("hello, how are you?".rpartition(" "))</code> returns: ('hello,', ' ', 'how are you?')
<code>.rsplit()</code>	Splits the string from the end E.g.: <code>print("hello, how are you?".rsplit(", "))</code> returns: ['hello', ' how are you?']
<code>.rstrip()</code>	Removes characters from the end of the string E.g.: <code>print("!!!hello!!!".rstrip("!"))</code> returns: !!!hello
<code>.split()*</code>	Splits the string into a list E.g.: <code>print("hello, how are you?".split(", "))</code> returns: ['hello', ' how are you?']
<code>.splitlines()</code>	Splits the string at line breaks E.g.: <code>print("hello\nhow are you?".splitlines())</code> returns: ['hello', 'how are you?']

# PHẦN 8

# HÀM



Trong phần này, bạn sẽ áp dụng cú pháp lập trình và tư duy tính toán đã học để xây dựng các đơn vị mã có thể tái sử dụng được gọi là hàm.

## 28. In thiệp Cảm ơn

*Đầu vào hàm*

Đến đây, chúng ta đã tìm hiểu về các kiểu dữ liệu Python—danh sách, chuỗi, Boolean, từ điển, số nguyên và số thực. Chúng ta cũng đã học cách kết hợp các kiểu dữ liệu này với các toán tử—gán, thành viên, số học, so sánh và logic—để tạo ra các lệnh—tức là các câu lệnh, điều kiện *if/else*, vòng lặp *for* và vòng lặp *while*. Bước tiếp theo là tìm hiểu cách kết hợp các lệnh thành các đơn vị mã gọi là **hàm**. Chúng ta đã khá quen thuộc với hàm vì đã thường xuyên sử dụng các hàm dựng sẵn của Python, chẳng hạn như *print()*, *len()*, *range()*, v.v. Trong Phần này, chúng ta sẽ tìm hiểu về các hàm và cách viết chúng.

### 1. Thiệp Cảm ơn Cơ bản

- Bạn vừa tổ chức một bữa tiệc và muốn gửi thiệp *Thank you* đến những người đã tham dự. Hãy tạo một hàm lấy tên làm đối số và in ra một thông điệp *Thank you* có chứa tên của người tham dự (ví dụ: *Thank you Maria*):

```
def print_thank_you(ten):  
    """In ra một chuỗi chứa  
    "Thank you" và tên  
  
    Tham số  
    -----  
    tên : string  
        tên là tên của ai đó  
    ...  
  
    print("Thank you ", ten)
```

- In hai tấm thiệp Cảm ơn:



The screenshot shows a Jupyter Notebook cell with the following content:  
[101] • `print_thank_you("Maria")`  
[101] ✓ 0.0s

Kết quả: .....

```
print_thank_you("Xiao")
0.0s
```

Kết quả: .....

## Chúng ta hãy cùng tìm hiểu qua sau



Một hàm là một khối mã thực hiện một nhiệm vụ cụ thể

## 2. Thiệp cảm ơn trang trọng

- Sau khi suy nghĩ kỹ, bạn quyết định in thiệp *Thank you* trang trọng sẽ phù hợp hơn. Sửa đổi hàm trước đó để lấy ba đối số—tiền tố, tên và họ—and in một thông điệp cảm ơn chúc chúng (ví dụ: *Thank you Mrs Maria Lopez*):

```
def print_thank_you(tiento, ho, ten):
    """In ra một chuỗi chứa
    "Thank you" và đầu vào

    Tham số
    -----
    tiento: string
        là ông hay bà
    họ: string
    tên : string
        tên là tên của ai đó
    """

    print("Thank you", tiento, ho, ten)
✓ 0.0s
```

- In hai tấm thiệp *Thank you* trang trọng:

```
> ▾
  print_thank_you ("Mrs", "Maria", "Lopez")
[104] ✓ 0.0s
```

Kết quả: .....

```
> ▾
  print_thank_you ("Mr", "Xiao", "Li")
[105] ✓ 0.0s
```

Kết quả: .....

Điều gì xảy ra nếu thiếu một trong các đối số như trong ví dụ dưới đây?

```
> ▾
  print_thank_you ("Mr", "Xiao")
[106] ✘ 0.0s
Python
-----
TypeError                                         Traceback (most recent call last)
Cell In[106], line 1
      1 print_thank_you ("Mr", "Xiao")
----> 1 print_thank_you ("Mr", "Xiao")

TypeError: print_thank_you() missing 1 required positional argument: 'ten'
```

### 3. Thiếu tên!

- Bạn rất hài lòng với thiệp *Thank you*, nhưng đột nhiên nhận ra một số người tham gia đã không cung cấp tên! Hãy điều chỉnh hàm sao cho tên có một chuỗi rỗng làm giá trị mặc định:

```
def print_thank_you(tiento, ho, ten=""):

    """In ra một chuỗi chứa
    "Thank you" và đầu vào

    Tham số
    -----
    tiento: string
        là ông hay bà
    họ: string
    tên : string
        tên là tên của ai đó, mặc định
        đang thiết lập là trống
    """

    print("Thank you", tiento, ho, ten)
```

- In hai tấm thiệp *Thank you* một tấm có họ và một tấm không có họ:

```
[104] print_thank_you ("Mrs", "Maria", "Lopez")
```

Kết quả: .....

```
[105] print_thank_you ("Mr", "Xiao")
```

Kết quả: .....

#### 4. Thiếu tiền tố và/hoặc họ!

- Cuối cùng, bạn nhận ra rằng tiền tố và/hoặc họ cũng bị thiếu đối với một số khách. Hãy sửa đổi hàm cho phù hợp:

```
def print_thank_you(tiento="", ho="", ten=""):

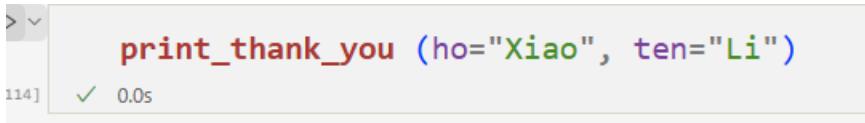
    """In ra một chuỗi chứa
    "Thank you" và đầu vào

    Tham số
    -----
    tiento: string
        là ông hay bà, mặc định
        đang thiết lập là trống
    họ: string
        mặc định đang thiết lập là trống
    tên : string
        tên là tên của ai đó, mặc định
        đang thiết lập là trống
    """

    print("Ông/bà: ", tiento)
    print("Họ:", ho)
    print("Tên: ", ten)
    print("Thank you", tiento, ho, ten)
```

```
▷ ▾ print_thank_you ("Mrs", "Maria","Lopez")
[104] ✓ 0.0s
```

Kết quả: .....



A screenshot of a Python code editor window. The code in the editor is:

```
print_thank_you (ho="Xiao", ten="Li")
```

The status bar at the bottom shows "114]" and a green checkmark icon followed by "0.0s".

## Tóm tắt

- Hàm là các khối mã thực hiện một nhiệm vụ cụ thể. Chúng rất quan trọng cho việc tái sử dụng và mô-đun hóa mã.
- Một hàm bao gồm ít nhất ba thành phần:
  - **Phần đầu**, bắt đầu bằng từ khóa **def**, sau đó là **tên hàm**, và **cặp ngoặc tròn chứa các tham số** được phân tách bằng **dấu phẩy**. Các tham số có thể có giá trị mặc định.
  - **Chuỗi tài liệu**, mô tả chức năng của hàm và các tham số của nó.
  - **Mã giải quyết** một nhiệm vụ.
- Để gọi một hàm, chúng ta viết **tên hàm** theo sau là **cặp ngoặc tròn** chứa các **đối số** được phân tách bằng dấu phẩy.
- **Tham số và đối số là đầu vào của hàm**. Về mặt kỹ thuật, chúng ta gọi tham số là các biến được liệt kê trong phần đầu hàm, và đối số là các biến trong lệnh gọi hàm.
- **Chuỗi tài liệu** là nền tảng khi viết và sử dụng hàm và có thể được truy cập bằng hàm **help()** tích hợp sẵn—xem phần Tìm hiểu sâu hơn bên dưới.

## Bài tập áp dụng

1. Các trường hợp chuỗi. Viết một hàm in một chuỗi cho trước dưới dạng chữ thường, chữ hoa, chữ cái đầu, chữ hoa, và đổi chỗ các chữ cái đầu. Sau đó, gọi hàm hai lần. Thực hiện một lần với chuỗi gồm một từ, và một lần với chuỗi gồm ít nhất hai từ. Cuối cùng, gọi hàm cho mỗi phần tử của danh sách các chuỗi sau bằng vòng lặp *for*:  
summer\_vacation = ["Hiking trails", "weekEndcampIng", "enjoying nature", "fishing"].

2. Độ dài chuỗi. Viết một hàm nhận một danh sách các chuỗi và một số nguyên, và chỉ in ra các chuỗi có độ dài khớp với số nguyên cho trước. Gọi hàm bằng hai độ dài từ khác nhau.

3. Nhiều số. Viết một hàm nhận một danh sách các số và một số nguyên, và chỉ in ra các số chia hết cho số nguyên đó. Nếu người dùng không cung cấp số, thì hàm sẽ chia cho 2 theo mặc định. Gọi hàm bằng hai ước số khác nhau. Cuối cùng, gọi hàm mà không có ước số.

4. Nhân đôi số. Viết một hàm yêu cầu người dùng nhập một số và in ra một từ điển trong đó các khóa là các số có giá trị nhỏ hơn số đầu vào, và các giá trị là số kép của mỗi khóa. Lưu ý rằng hàm không nhận bất kỳ đối số nào. Hàm input() để yêu cầu nhập số nằm bên trong hàm.

(Ví dụ: người dùng nhập: 5

Kết quả mong đợi: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10})

## 29. Cơ sở dữ liệu đăng nhập cho cửa hàng trực tuyến

Đầu ra hàm và thiết kế mô-đun Trong chương trước, chúng ta đã tìm hiểu về hàm và đầu vào của chúng. Trong chương này, chúng ta sẽ đi sâu vào đầu ra hàm. Ngoài ra, chúng ta sẽ xem xét việc thiết kế và tổ chức nhiều hàm trong một dự án lớn hơn. Hãy cùng giải quyết tất cả những điều này bằng cách giải quyết nhiệm vụ sau.

- Bạn là chủ sở hữu của một cửa hàng trực tuyến và cần lưu trữ an toàn tên người dùng và mật khẩu của khách hàng. Hãy tạo một cơ sở dữ liệu trong đó tên người dùng bao gồm chữ cái đầu của tên khách hàng, tiếp theo là họ (ví dụ: "jsmith"), và mật khẩu bao gồm một mã bốn chữ số.

Trước tiên, chúng ta phải tạo một cơ sở dữ liệu. Cơ sở dữ liệu là một tập hợp dữ liệu được tổ chức, có thể dễ dàng truy cập và quản lý. Ví dụ về cơ sở dữ liệu bao gồm kho hàng tại một cửa hàng tạp hóa, danh mục thư viện, hoặc danh sách liên hệ qua điện thoại.

Trong trường hợp của chúng ta, cơ sở dữ liệu sẽ là tập hợp tên người dùng và mật khẩu của khách hàng. Nhìn chung, các cơ sở dữ liệu đơn giản có thể được triển khai dưới dạng từ điển.

Bạn sẽ tạo cơ sở dữ liệu này như thế nào và bạn sẽ chèn tên người dùng và mật khẩu như thế nào? Bạn sẽ sử dụng những biến nào và thuộc kiểu nào? Bạn sẽ viết bao nhiêu hàm, và mỗi hàm sẽ làm gì? Hãy dành chút thời gian để suy nghĩ về giải pháp của bạn trước khi chuyển sang đoạn tiếp theo!

Để giải quyết nhiệm vụ của chúng ta, điều đầu tiên cần làm là "chia để trị" bằng cách xác định những biến và hàm nào chúng ta cần tạo. Hãy bắt đầu với các biến và kiểu dữ liệu của chúng. Với mỗi khách hàng, chúng ta cần hai chuỗi—một cho tên người dùng và một cho mật khẩu. Chúng ta sẽ lưu chúng trong một từ điển—tức là một cơ sở dữ liệu—trong đó tên người dùng sẽ là khóa và mật khẩu sẽ là giá trị. Nay giờ, hãy nghĩ về cách mã hóa mã—tức là cách tổ chức nó thành các hàm. Chúng ta có thể viết ba hàm: một để tạo tên người dùng, một để tạo mật khẩu, và một để gọi hai hàm trước đó và thêm tên người dùng và mật khẩu đã tạo vào cơ sở dữ liệu. Hãy cùng xem xét kỹ hơn cách triển khai giải pháp này!

## 1. Tạo tên người dùng

Hãy đọc đoạn văn bản và mã sau và cố gắng suy ra chức năng của mã.

- Viết một hàm tạo tên người dùng bao gồm **chữ cái đầu của tên và họ**:

```
def create_username (first_name, last_name):
    #tên người dùng bao gồm chữ cái đầu của tên và họ:
    username=first_name[0]+last_name

    #chuyển về chữ in thường
    username=username.lower()
    return username
```

- Kiểm tra chức năng cho hai khách hàng:

```
username_1 = create_username("Julia", "Smith")
```

```
print(username_1)
```

✓ 0.0s

Kết quả: .....

```
username_2 = create_username("Mohammed", "Seid")
```

```
print(username_2)
```

✓ 0.0s

Kết quả: .....

Bạn có thể thấy đường dẫn của các biến đầu ra

```
[1]: 1 def create_username(first_name, last_name):
2     """Create a lowercase username made
3         of initial of first name and last name
4
5     Parameters
6     -----
7     first_name : string
8         First name of a person
9     last_name : string
10        Last name of a person
11
12     Returns
13     -----
14     username : string
15         Created username
16
17     * concatenate initial of first name
18     and last name
19     username = first_name[0] + last_name
20     # make sure username is lowercase
21     username = username.lower()
22
23     # return username
24     return username
```

```
[2]: 1 username_1 = create_username("Julia", "Smith")
2 print(username_1)
jsmith
```

```
[1]: 1 def create_username(first_name, last_name):
2     """Create a lowercase username made
3         of initial of first name and last name
4
5     Parameters
6     -----
7     first_name : string
8         First name of a person
9     last_name : string
10        Last name of a person
11
12     Returns
13     -----
14     username : string
15         Created username
16
17     * concatenate initial of first name
18     and last name
19     username = first_name[0] + last_name
20     # make sure username is lowercase
21     username = username.lower()
22
23     # return username
24     return username
```

```
[3]: 1 username_2 = create_username("Mohammed", "Seid")
2 print(username_2)
mseid
```

## 2. Tạo mật khẩu

Chúng ta cần triển khai một hàm tạo mật khẩu gồm bốn số nguyên. Bạn sẽ làm như thế nào? Hãy thử tự triển khai trước khi xem giải pháp bên dưới.

- Viết một hàm tạo mật khẩu gồm **bốn số nguyên ngẫu nhiên**:

```
import random

def create_password():
    #tạo mật khẩu gồm bốn số nguyên ngẫu nhiên:
    password=str(random.randint(1000,9999))

    return password
```

[4] ✓ 0.0s

- *Kiểm tra chức năng cho hai khách hàng:*

```
password_1 = create_password()

print(password_1)
```

[5] ✓ 0.0s

Kết quả: .....

```
password_2 = create_password()

print(password_2)
```

[6] ✓ 0.0s

Kết quả: .....

## 3. Tạo cơ sở dữ liệu

- Viết một hàm, khi được cung cấp một danh sách các danh sách khách hàng, sẽ tạo và trả về một cơ sở dữ liệu—tức là một từ điển—gồm tên người dùng và mật khẩu. Hàm này cũng trả về số lượng khách hàng trong cơ sở dữ liệu:

```
v def create_database (customers):
    """Tạo một cơ sở dữ liệu dưới dạng từ điển với
       tên người dùng làm khóa và mật khẩu làm giá trị

    Tham số
    -----
    customers: danh sách các danh sách
        Mỗi danh sách con chứa tên và
        họ của khách hàng

    Giá trị trả về
    -----
    db : từ điển
        Đã tạo cơ sở dữ liệu (viết tắt là db)
    n_customers : int
        Số lượng khách hàng trong cơ sở dữ liệu
    """

    db={} # từ điển nên dùng {}

    # cho từng khách hàng customers
    for khach in customers:
        #tao username, nhớ là trong customer mỗi cell có 2 phần tử
        username=create_username(khach[0],khach[1])

        # tạo pass
        password=create_password()

        # Lưu 2 biến trên vào db
        db[username]=password #key = username, value = password

    n_customers=len(db)

    return db,n_customers
```

Việc kiểm tra tính đúng đắn của một hàm bằng cách gọi hàm đó luôn rất quan trọng. Vậy hãy gọi hàm đó và kiểm tra hành vi của nó!

- Cho danh sách khách hàng sau:

```
customers = [["Maria", "Lopez"], ["Julia", "Smith"], ["Mohammed", "Seid"]]

print(customers)
28] ✓ 0.0s Python
```

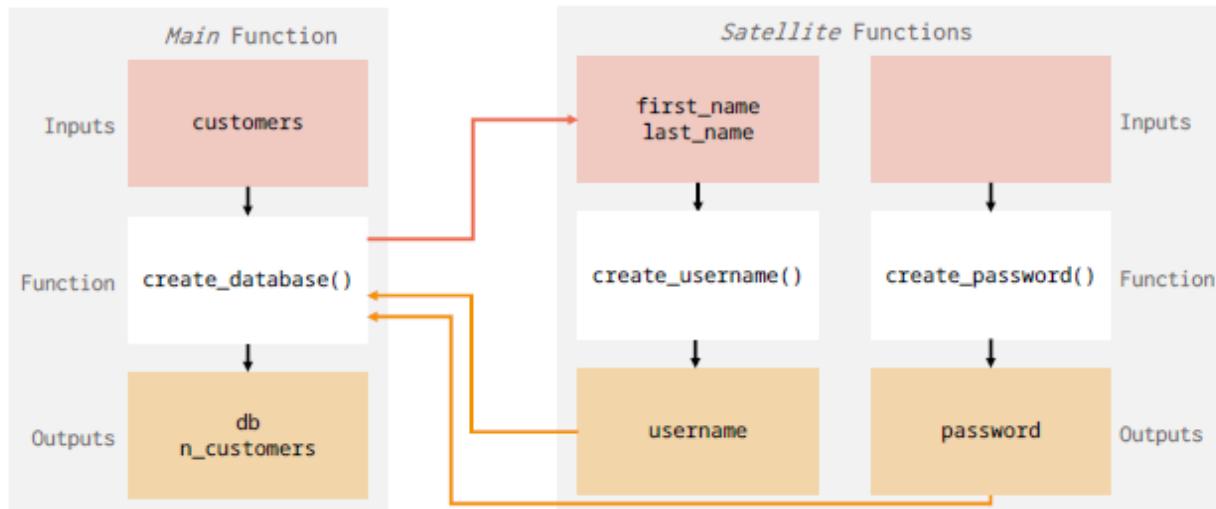
Kết quả: .....

```
# tạo database
database,number_customers=create_database(customers)

# in các giá trị
print(database)
print("Số lượng khách hàng: ", number_customers)
[] ✓ 0.0s
```

Kết quả: .....

Kết quả: .....



## Tóm tắt

- Từ khóa **return** có hai vai trò:
  - Nó chuyển các biến đầu ra từ thân hàm đến lệnh gọi hàm. Khi nhiều biến được trả về, chúng được phân tách bằng dấu phẩy cả trong thân hàm và trong lệnh gọi hàm. Trong lệnh gọi hàm, chúng cũng có thể được tập hợp thành một bộ.
  - Nó đánh dấu *kết thúc một hàm*. Các lệnh được viết sau lệnh return sẽ không được thực thi.
- **Bộ** là một kiểu dữ liệu trong đó các phần tử là bất biến, nghĩa là chúng không thể bị thay đổi. Việc cắt bộ tuân theo các quy tắc tương tự như cắt danh sách (hoặc chuỗi).
- Trong docstring, **cú pháp** của các biến được trả về giống với cú pháp của các tham số đầu vào.
- Điều quan trọng là phải kiểm tra tính đúng đắn của hàm bằng cách gọi chúng với các đối số phù hợp.
- Một dự án thường bao gồm một hàm chính và một số hàm vệ tinh. Hàm chính thực thi giải pháp cho toàn bộ nhiệm vụ, trong khi mỗi hàm vệ tinh thực thi một nhiệm vụ con cụ thể.

## 30. Vé miễn phí tại bảo tàng

*Xác thực đầu vào và các biến thể đầu ra*

Điều gì xảy ra nếu chúng ta cung cấp đầu vào sai cho một hàm? Đôi khi hàm bị lỗi—tức là chúng ta nhận được lỗi—and đôi khi chúng ta nhận được kết quả vô nghĩa. Trong cả hai trường hợp, có thể khó hiểu được điều gì đã xảy ra. Trong chương này, chúng ta sẽ tìm hiểu cách đảm bảo rằng các đầu vào của hàm có đúng kiểu và giá trị. Ngoài ra, chúng ta cũng sẽ tìm hiểu cách trả về đầu ra trong các trường hợp cụ thể, tức là dựa trên điều kiện

hoặc trực tiếp dưới dạng giá trị. Hãy cùng giải quyết tất cả những điều này thông qua ví dụ dưới đây.

- Bạn làm việc tại một bảo tàng và phải cập nhật hệ thống trực tuyến để mua vé. Bản cập nhật là **những người từ 65 tuổi trở lên hiện đủ điều kiện nhận vé miễn phí**. Hãy viết một hàm **yêu cầu khách truy cập nhập tiền tố, họ và tuổi của họ**; kiểm tra kiểu và giá trị của các đầu vào này; và trả về một thông báo cho khách truy cập biết liệu họ có đủ điều kiện nhận vé miễn phí hay không.

```
def free_museum_ticket (prefix, last_name, age):
    # (1) yêu cầu khách truy cập nhập tiền tố, họ và tuổi của họ
    # --- kiểm tra các Loại tham số ---

    # kiểu tiền tố phải là chuỗi
    if not isinstance(prefix,str):
        raise TypeError("Prefix phải là dạng chuỗi str")

    # kiểu Last_name phải là chuỗi
    if not isinstance(last_name,str):
        raise TypeError("Last_name phải là dạng chuỗi str")

    # kiểu tuổi phải là số nguyên
    if not isinstance(age,int):
        raise TypeError("Age phải là dạng số nguyên int")

    #note chú ý dòng Lệnh trên nhé!
    # --- kiểm tra giá trị tham số ---
    # prefix phải là Ms, Mrs, or Mr
    if prefix not in ["Ms", "Mrs", "Mr"]:
        raise ValueError("Prefix phải là Ms, Mrs, or Mr")

    # last_name chỉ được chứa các ký tự
    if not last_name.isalpha():
        raise ValueError("Last_name chỉ được chứa các ký tự")

    # age phải nằm trong khoảng từ 0 đến 125
    if age<0 or age>125:
        raise ValueError("Age phải nằm trong khoảng từ 0 đến 125")

    #--- trả về đầu ra ---
    if age>=65:
        return prefix+ ". " + last_name + ", bạn đủ điều "
        "kiện nhận vé vào bảo tàng miễn phí vì bạn "
        + str(age) + " tuổi"
    else:
        return prefix+ ". " + last_name + ", bạn KHÔNG "
        "đủ điều kiện nhận vé vào bảo tàng miễn phí vì bạn "
        + str(age) + " tuổi"
```

✓ 0.0s

Python

Bây giờ chúng ta tiến hành check

```
# checking prefix type
message = free_museum_ticket (1, "Holmes", 66)

print(message)
[!] ✘ 0.0s
```

```
# checking last_name type
message = free_museum_ticket ("Mrs", 1.2, 66)

print(message)
[!] ✘ 0.0s
```

```
# checking age type
message = free_museum_ticket ("Mrs", "Holmes", "Hi")

print(message)
[!] ✘ 0.0s
```

```
# checking prefix value
message = free_museum_ticket ("Dr", "Holmes", 66)
print(message)
```

```
[ ] # checking last_name value
message = free_museum_ticket ("Mrs", "82", 66)
print(message)
```

```
[ ] ▷ # checking age value
message = free_museum_ticket ("Mrs", "Holmes", 130)
print(message)
```

Bây giờ check trường hợp nhập đúng

```
[ ] # người đủ điều kiện
message = free_museum_ticket ("Mrs", "Holmes", 66)
print(message)
```

Kết quả: .....

```
[ ] > # người không đủ điều kiện
message = free_museum_ticket ("Mrs", "Holmes", 38)
print(message)
```

Kết quả: .....

## Tóm tắt

- Chúng ta triển khai kiểm tra tham số trong các hàm main hoặc khi có đầu vào bên ngoài. Việc kiểm tra được thực hiện bằng cách sử dụng cấu trúc *if/else*. Trong điều kiện *if*:

- *Khi kiểm tra một kiểu*, chúng ta sử dụng toán tử logic không theo sau bởi hàm dựng sẵn *isinstance()*, với các tham số là biến cần kiểm tra và kiểu mong muốn. Các kiểu có thể là str, int, list, dict, v.v.
- *Khi kiểm tra một giá trị*, chúng ta phải định nghĩa các ràng buộc. Chúng ta có thể sử dụng thuộc tính cho một danh sách, các phương thức biến —chẳng hạn như *.isalpha()* cho chuỗi—hoặc các khoảng cho số.

Trong câu lệnh, chúng ta sử dụng *raise* theo sau là ngoại lệ ***TypeError() hoặc ValueError()*** với một thông báo cho biết cách tránh lỗi.

- Khi muốn trả về các đầu ra khác nhau dựa trên các điều kiện, chúng ta có thể sử dụng cấu trúc *if/else* trong đó các câu lệnh chứa từ khóa *return* theo sau là đầu ra mong muốn.
- Có thể trả về giá trị thay vì biến. Trong trường hợp này, trong docstrings, chúng ta chỉ chỉ ra loại.
- Trong docstrings, có thể viết một ví dụ sau định nghĩa hàm để tăng tính rõ ràng.

## Bài tập áp dụng

1. Chọn một từ ngẫu nhiên. Tạo một hàm vệ tinh cho một danh sách các từ, trả về một từ được chọn ngẫu nhiên ở dạng chữ thường.

(Ví dụ: đầu vào: ["muỗng", "Nĩa", "DAO"]. Ví dụ: đầu ra: "nĩa").

2. Hiển thị các chữ cái đã đoán. Tạo một hàm vệ tinh cho một từ và một chữ cái đã đoán, hiển thị từ với chữ cái đã đoán được hiển thị đúng vị trí, trong khi các chữ cái còn lại, chưa đoán được, được hiển thị dưới dạng dấu gạch dưới.

(Ví dụ: đầu vào: ("l", "hello"). Ví dụ: đầu ra:    l l   ).

### 31. Giai thừa

#### Hàm đệ quy

Trong chương này, bạn sẽ tìm hiểu một loại hàm cụ thể được gọi là hàm đệ quy. Chúng có thể khó hiểu và khó triển khai, nhưng rất hữu ích trong một số trường hợp nhất định, như bạn sẽ thấy. Để hiểu rõ hơn cách thức hoạt động của hàm đệ quy, hãy cùng tính giai thừa. Bạn đã từng nghe đến chúng chưa? Giai thừa là tích của tất cả các số nguyên dương nhỏ hơn hoặc bằng một số nguyên dương cho trước. Ví dụ, giai thừa của 4 là 24, được tính bằng  $1 \times 2 \times 3 \times 4$ —hoặc  $4 \times 3 \times 2 \times 1$ . Bạn sẽ viết một hàm tính giai thừa của một số nguyên như thế nào? Hãy tự viết hàm của riêng bạn trước khi xem giải pháp được đề xuất bên dưới.

- Viết một hàm tính giai thừa của một số nguyên cho trước bằng cách sử dụng vòng lặp *for*:

```
def giaithuacho (n):  
  
    #khởi tạo gaii thừa  
    giaithua = 1  
  
    for i in range(2,n+1):  
        giaithua*=i  
  
    return giaithua  
  
# tính 1 gaii thừa nào đó, ví dụ 4!  
tinh=giaithuacho(4)  
print(tinh)
```

✓ 0.0s

Kết quả: .....

- So sánh hàm lặp trước đó với hàm đệ quy sau:

```
> ▾  
def giaithuahq(n):  
    if n>1:  
        return giaithuahq(n-1)*n  
    else: # điểm neo  
        return 1
```

```
# tính 1 gaii thừa nào đó, ví dụ 4!  
tinh=giaithuahq(4)  
print(tinh)
```

141] ✓ 0.0s

Kết quả: .....

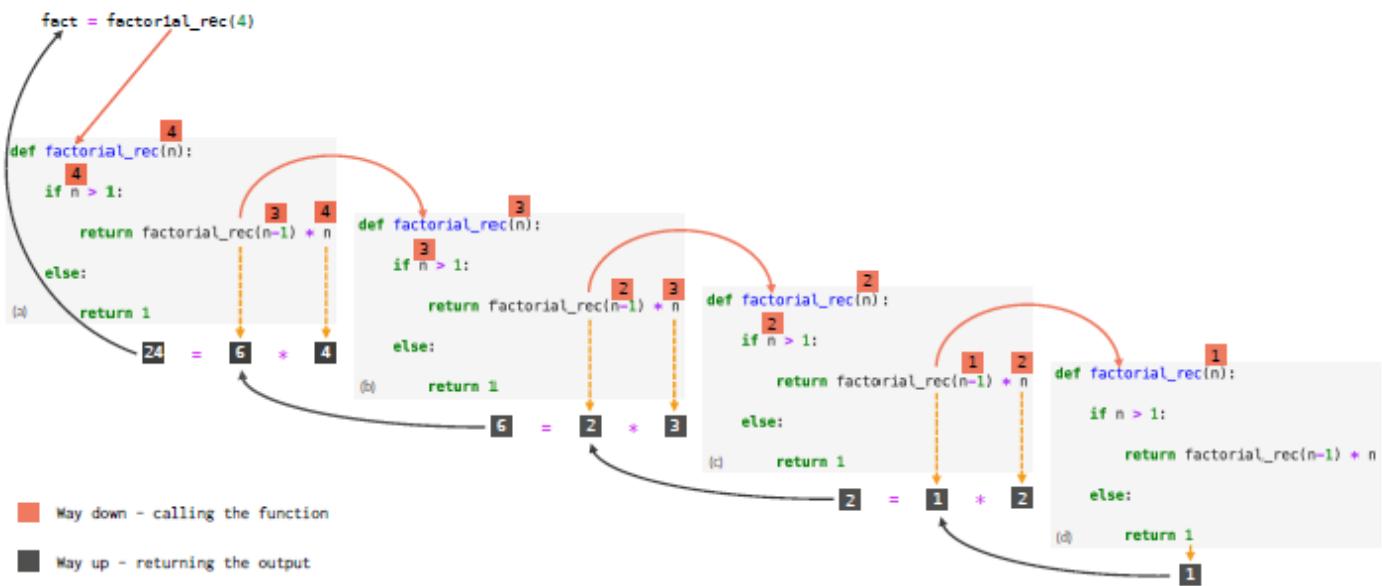


Figure 31.1. The mechanism of a recursive function.

## Tóm tắt

- Các hàm lặp chứa một vòng lặp để lặp lại một số mă.
- Các hàm đệ quy tự gọi chính nó để lặp lại một số mă.
- Các hàm đệ quy thường chứa một cấu trúc if/else, trong đó một câu lệnh là trường hợp cơ sở, và câu lệnh còn lại là trường hợp đệ quy.
- Các hàm đệ quy chứa mã nhỏ gọn và phù hợp cho các bài toán đệ quy nội tại.

Tuy nhiên, chúng sử dụng một lượng lớn bộ nhớ tính toán và có thể khó gỡ lỗi hơn.

### 32. Làm thế nào tôi có thể tái sử dụng các hàm?

Một mô-đun là một tệp chứa các hàm cho  
một nhiệm vụ cụ thể và có phần mở rộng là .py

Trước tiên chúng ta tạo 1 file python tên là *setup\_database.py*

Trong file này chúng ta có 3 hàm *create\_username*, *create\_password*, *create\_database*

The screenshot shows a Jupyter Notebook interface with the following code:

```
1 def create_username(first_name, last_name):
2     #tên người dùng bao gồm chữ cái đầu của tên và họ:
3     username=first_name[0]+last_name
4
5     #chuyển về chữ in thường
6     username=username.lower()
7     return username
8
9 import random
10
11 def create_password():
12     #tạo mật khẩu gồm bốn số nguyên ngẫu nhiên:
13     password=str(random.randint(1000,9999))
14
15     return password
16
17 def create_database(customers):
18     """Tạo một cơ sở dữ liệu dưới dạng từ điển với
19     tên người dùng làm khóa và mật khẩu làm giá trị
20
21     Tham số
22     -----
23     customers: danh sách các danh sách
24         Mỗi danh sách con chứa tên và
25             họ của khách hàng
26
27     Giá trị trả về
28     -----
29     db : từ điển
30     |   Đã tạo cơ sở dữ liệu (viết tắt là db)
31     n_customers : int
32     |   Số lượng khách hàng trong cơ sở dữ liệu
33     """
34
35     db={} # từ điển nên dùng {}
36
37     # cho từng khách hàng customers
38     for khach in customers:
39         #tao username, nhớ là trong customer mỗi cell có 2 phần tử
40         username=create_username(khach[0],khach[1])
41
42         # tạo pass
43         password=create_password()
44
45         # Lưu 2 biến trên vào db
46         db[username]=password #key = username, value = password
47
48     n_customers=len(db)
49
50     return db,n_customers
```

The code defines three functions: `create_username`, `create_password`, and `create_database`. The `create_database` function takes a list of customers as input and returns a dictionary where each customer's name is a key and their generated password is the value. A tooltip for the `db` parameter in the `create_database` docstring is visible, stating "Đã tạo cơ sở dữ liệu (viết tắt là db)".

## 2. Nhập mô-đun và chạy hàm

Tạo 1 file mới .ipynb trong cùng folder với *setup\_database.py*

Hãy quay lại số ghi chép của chúng ta và gọi *create\_username()* từ đó! Chúng ta cần (1) nhập mô-đun và (2) gọi hàm. Có bốn cách để thực hiện. Hãy cùng xem qua chúng!

- *Nhập mô-đun theo nguyên trạng, sau đó gọi hàm:*



```
import setup_database
username = setup_database.create_username ("John", "Gelb")
print(username)
```

[1] ✓ 0.0s

Kết quả: .....

- *Nhập một mô-đun và tạo một bí danh, sau đó gọi hàm:*



```
import setup_database as sdb
# Lúc này sdb như là setup_database "as"
username = sdb.create_username ("John", "Gelb")
print(username)
```

[4] ✓ 0.0s

Kết quả: .....

- *Nhập một hàm duy nhất từ một mô-đun, sau đó gọi hàm đó:*



```
from setup_database import create_username
#gọi trực tiếp hàm đó ra
username = create_username ("John", "Gelb")
print(username)
```

[5] ✓ 0.0s

Kết quả: .....

- Nhập hàm từ một mô-đun và tạo một bí danh, sau đó gọi hàm:

```
> v
    from setup_database import create_username as cu

    #gọi trực tiếp hàm đó ra và cho nó tên "cu"
    username = cu ("John", "Gelb")
    print(username)
[6]   ✓  0.0s
```

Kết quả: .....

- Gọi `create_password()`:

```
▷ v
    import setup_database

    password = setup_database.create_password()
    print(password)
[7]   ✓  0.0s
```

Kết quả: .....

- Gọi `create_database()`:

```
> v
    import setup_database
    customers = [[ "Maria", "Lopez"], [ "Julia", "Smith"], [ "Mohammed", "Seid"]]

    db, n = setup_database.create_database(customers)

    print(db)
    print("Tong so khach hang: ", n)
[8]   ✓  0.0s
```

Kết quả: .....



## PHẦN 9

# NHỮNG ĐIỀU CƠ BẢN

## CUỐI CÙNG

## PYTHON

Trong phần này, bạn sẽ học cách đọc và ghi tệp, cũng như một số kiểu dữ liệu cuối cùng, từ khóa, hàm dựng sẵn và mô-đun.

### 33. Quà sinh nhật

#### Đọc và ghi tệp .txt

Một khía cạnh quan trọng trong lập trình là biết cách đọc và ghi tệp. Hãy cùng tìm hiểu những điều cơ bản trong chương này!

- Ba người bạn của bạn đã tổ chức sinh nhật trong tháng này, và bạn đã mua quà cho họ trực tuyến. Giờ là lúc thực hiện phân tích mua hàng và lưu vào hồ sơ. Số tiền mua hàng nằm trong tệp purchases.txt.

Chúng ta sẽ giải quyết nhiệm vụ này như thế nào? Hãy phác thảo các bước bạn sẽ thực hiện trước khi bắt đầu giải quyết bên dưới.

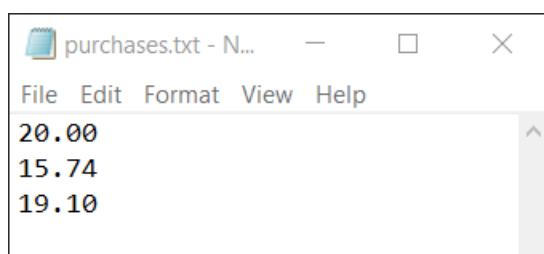
Để giải quyết nhiệm vụ này, chúng ta sẽ chia (và chinh phục!) vấn đề thành ba bước. Đầu tiên, chúng ta sẽ đọc tệp .txt và lưu trữ nội dung của nó vào một danh sách. Thứ hai, chúng ta sẽ phân tích các giao dịch mua bằng cách tính toán số tiền tối thiểu, tối đa và tổng số tiền trong danh sách. Và thứ ba, chúng ta sẽ lưu phân tích vào một tệp .txt mới.

Bắt đầu thôi!

#### 1. Đọc tệp .txt

Trước khi bắt đầu viết mã, điều quan trọng là phải xem tệp đầu vào trông như thế nào.

Tạo và lưu tệp purchases.txt có nội dung sau



Tệp chứa ba số, mỗi số nằm trên một hàng riêng biệt, biểu thị số tiền mua hàng. Hãy đọc tệp và lưu ba số này vào một danh sách.

- *Viết một hàm đọc tệp .txt chứa một số trên mỗi hàng và lưu các số này vào danh sách:*

```
def read_txt (file_name_in):
    """Đọc một tệp .txt với một số
       mỗi hàng và trả về chúng dưới dạng danh sách

    Tham số
    -----
    file_name_in : string
        tên file cần đọc

    Giá trị trả về
    -----
    number: list
        Nội dung tệp trong danh sách số
    """

#khởi tạo
numbers= [] # khai báo list

# open file để đọc
with open (file_name_in,"r") as file:
    # file_name_in bây giờ được mở dưới tên biến là "file"

    for line in file:
        print("Dòng đang đọc: ", line)

        #di chuyển "\n" ra khỏi line
        #vì sao thì xem cấu trúc dưới
        line=line.rstrip("\n")

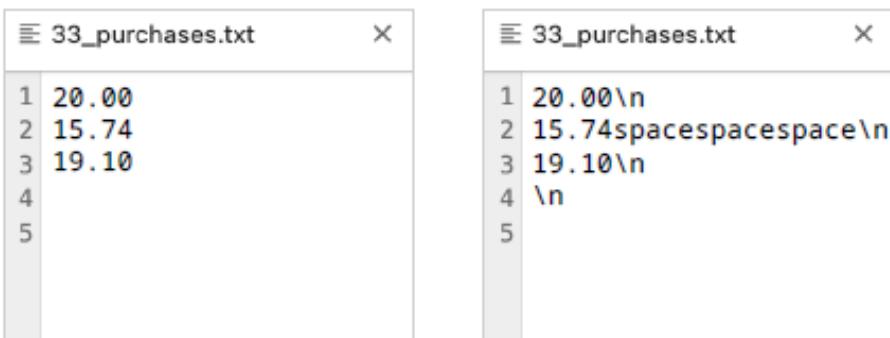
        print("Đdòng in sau khi tách dòng: ",line)
        print("-----") #vì tách \n nên dòng này in sau dòng
                      #trên

    #check dòng đó Là kết thúc "empty"
    if line != "":
        #chuyển số đó sang định dạng float
        so=float(line)

        #thêm số đó vào List của numbers
        numbers.append(so)

return numbers
```

Vì sao chúng ta phải tách bỏ “\n” xem dưới đây:



```
33_purchases.txt
1 20.00
2 15.74
3 19.10
4
5
```

```
33_purchases.txt
1 20.00\n
2 15.74 spacespace\n
3 19.10\n
4 \n
5
```

Bây giờ chúng ta cho chạy file trên



```
> 
purchases= read_txt("purchases.txt")
print(purchases)
[13] ✓ 0.0s
```

Kết quả: .....

## 2. Phân tích các số

- Viết một hàm lấy một danh sách các số làm đầu vào và trả về giá trị nhỏ nhất, lớn nhất và tổng dưới dạng các biến riêng biệt.

```
def calculate_stats (numbers):
    # tìm giá trị Lớn nhất, nhỏ nhất và tổng

    minimum=min(numbers)
    maximum=max(numbers)
    total=sum(numbers)

    return minimum,maximum,total

mi,ma,to=calculate_stats(purchases)
print("Giá trị nhỏ nhất: ",mi)
print("Giá trị lon nhất: ",ma)
print("Giá trị tổng: ",to)
```

[4] | ✓ 0.0s

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

## 3. Lưu phân tích

- Tạo một hàm cho phép ghi các giá trị nhỏ nhất, lớn nhất và tổng vào một tệp trên ba dòng liên tiếp và chỉ rõ chúng biểu diễn điều gì:

```
def write_txt (file_name_out, minimum,maximum, total):  
    # mở file cần ghi thông tin lên chế độ write  
  
    with open (file_name_out,"w") as file:  
  
        file.write("Gia tri nho nhat: "+ str(mi) + "\n")  
  
        file.write("Gia tri lon nhat: " + str(ma) + "\n")  
  
        file.write("Gia tri tong: " + str(to) + "\n")  
  
    # không có giá trị trả về  
✓ 0.0s
```

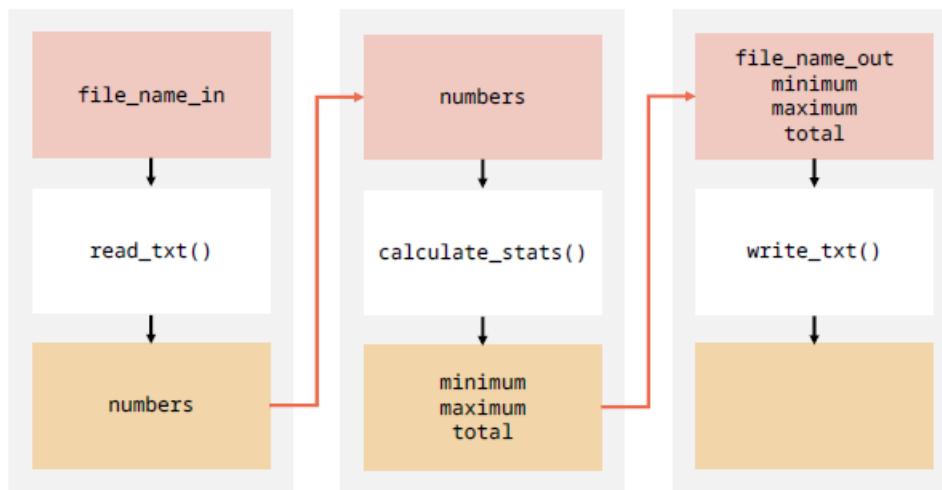
Python

```
write_txt("purchases_out.txt",mi,ma,to)  
✓ 0.0s
```

Generate + Code + Markdown

Sau đó mở file *purchases\_out.txt*

Quá trình trên được tóm tắt như sau:



## Tóm tắt

- Để mở và đọc hoặc tạo và lưu một tệp, chúng ta sử dụng lệnh `with open() as file`, trong đó:
  - ***with*** là một từ khóa hỗ trợ mở và đóng tệp;
  - ***open()*** là một hàm dựng sẵn có các đối số là tên tệp cần đọc hoặc ghi và chuỗi "**r**" khi đọc hoặc "**w**" khi ghi tệp;
  - ***as*** là một từ khóa để đổi tên biến đại diện cho tệp;
  - ***file*** (hoặc `file_object` hoặc `fo`) là tên tệp thông thường cho biến đại diện cho tệp.
- Để đọc nội dung tệp, chúng ta sử dụng vòng lặp `for` duyệt qua từng dòng của nội dung tệp, hoặc phương thức `.read()` trả về nội dung tệp dưới dạng một chuỗi dài. Để xóa nội dung tệp, chúng ta sử dụng các phương thức chuỗi như `.rstrip()` hoặc `.split()`.
- Để ghi nội dung vào một tệp, chúng ta sử dụng phương thức `.write()`, phương thức này nhận một chuỗi (hoặc một chuỗi chuỗi) làm đối số.
- Để tính giá trị nhỏ nhất, lớn nhất và tổng của một danh sách, chúng ta sử dụng các hàm tích hợp `min()`, `max()` và `sum()`.
- Để giải các bài toán có lời giải tuyến tính, chúng ta tổ chức các hàm theo dạng pipeline, trong đó đầu ra của hàm trước là đầu vào của hàm sau.

## 34. Python còn có gì nữa?

Các kiểu dữ liệu, từ khóa, hàm dựng sẵn, mô-đun bổ sung

Chúng ta đang đi đến cuối hành trình phát triển tư duy tính toán khi học Python. Trước khi đến phần cuối của cuốn sách này, nơi chúng ta sẽ học lập trình hướng đối tượng, hãy dành chút thời gian để phân tích thêm một vài kiểu dữ liệu, từ khóa, hàm dựng sẵn và

mô-đun rất hữu ích khi viết mã bằng Python. Đối với những phần còn lại, bạn sẽ được giới thiệu đến các trang web đáng tin cậy.

## 1. Kiểu dữ liệu

Chúng ta đã tìm hiểu sâu về chuỗi, danh sách, số nguyên và số thực, Boolean và từ điển. Nay, hãy xem xét các đặc điểm chính của bộ và tập hợp.

### 1.1 Tuple

Trước đây, chúng ta đã tìm hiểu về tuple như một kiểu dữ liệu cho đầu ra của hàm tích hợp `enumerate()`, của phương thức từ điển `.item()`, và của bất kỳ hàm nào trả về nhiều hơn một biến. Như bạn có thể nhớ, tuple là một chuỗi các phần tử được đặt trong cặp ngoặc tròn và được phân cách bằng dấu phẩy. Chúng không thể thay đổi—tức là, các phần tử của chúng không thể bị thay thế, thêm vào hoặc xóa đi—do đó, chúng là kiểu dữ liệu lý tưởng cho các biến không thay đổi trong suốt mã. Ngoài ra, tuple chỉ có hai phương thức, đó là `.count()`, trả về số lần một phần tử xuất hiện trong một tuple, và `.index()`, trả về vị trí của một phần tử trong một tuple. Hãy cùng xem một ví dụ đơn giản về tuple và hai phương thức của nó.

- Cho tuple sau:

```
▶ ▾
    image_size = (256, 256, 3)

    print(image_size)
[27] ✓ 0.0s
```

Kết quả: .....

- Tính xem có bao nhiêu lần số 256 xuất hiện:

```
▼
    print(image_size.count(256))
I ✓ 0.0s
```

Kết quả: .....

- *Tính vị trí của 3:*

```
▶ ▾
  print(image_size.index(3))
[29] ✓ 0.0s
```

Kết quả: .....

Phân biệt đang tìm vị trí của giá trị 3 không phải là tại vị trí số 3!

## 1.2 Tập hợp

Tập hợp là một kiểu dữ liệu có các phần tử được đặt trong cặp dấu ngoặc nhọn và phân cách bằng dấu phẩy. Tập hợp có ba đặc điểm chính: (1) chúng không thể thay đổi—giống như bộ; (2) chúng không được sắp xếp, tức là các phần tử của chúng không có vị trí cố định—do đó, tập hợp không thể được cắt lát; và (3) **chúng chứa các phần tử duy nhất, do đó không cho phép các phần tử trùng lặp.** Hãy cùng xem xét một tập hợp.

- *Cho tập hợp sau, hãy in ra:*

```
cities = ["Buenos Aires", "Prague", "Delhi", "Delhi"]

print(cities)
print("Số thành phần trong cities là ", len(cities))
✓ 0.0s
```

Kết quả: .....

Kết quả: .....

Chúng ta tạo một tập hợp gọi là cities (thành phố) chứa 4 phần tử là chuỗi (dòng 1) và in nó ra (dòng 2). Từ kết quả in, ta có thể thấy tập hợp kết quả khác với tập hợp đã định nghĩa ở dòng 1. Bởi vì các tập hợp không có thứ tự—đặc điểm (2) ở trên—nên các phần tử được sắp xếp bên trong khác với định nghĩa của chúng, và chúng có thể được in theo thứ tự khác nhau mỗi lần. Thứ hai, vì các tập hợp chỉ chứa các phần tử duy nhất—đặc

điểm (3) ở trên—nên "Delhi" chỉ xuất hiện một lần. Do đó, cities thực tế chỉ chứa 3 phần tử, như ta có thể thấy khi in độ dài của nó (dòng 3).

Với các đặc điểm của chúng, các tập hợp là các trung gian rất thuận tiện cho các thao tác danh sách. Chúng ta có thể sử dụng các thuộc tính set (tập hợp) để loại bỏ các phần tử trùng lặp khỏi danh sách và hai phương thức set (tập hợp)—union() và .intersection()—để hợp nhất hai danh sách và tìm các phần tử chung của chúng—các tập hợp có thêm 15 phương thức mà bạn có thể dễ dàng khám phá trên internet. Hãy cùng xem một số ví dụ.

- Cho danh sách sau:



```
cities = ["San Francisco", "Melbourne", "San Francisco", "Milan"]
```

Python

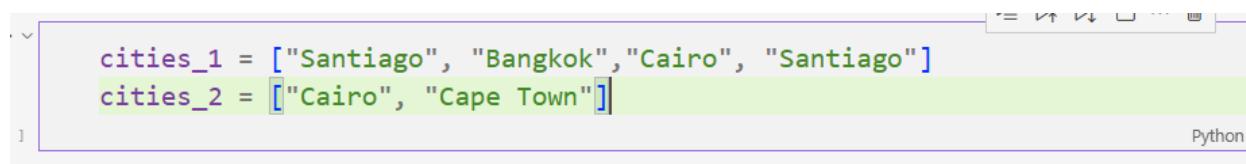
- Xóa các bản sao:



```
> 
    cities = list(set(cities))
    print(cities)
32]   ✓  0.0s
```

Kết quả: .....

- Cho các danh sách sau:



```
1 
    cities_1 = ["Santiago", "Bangkok", "Cairo", "Santiago"]
    cities_2 = ["Cairo", "Cape Town"]
```

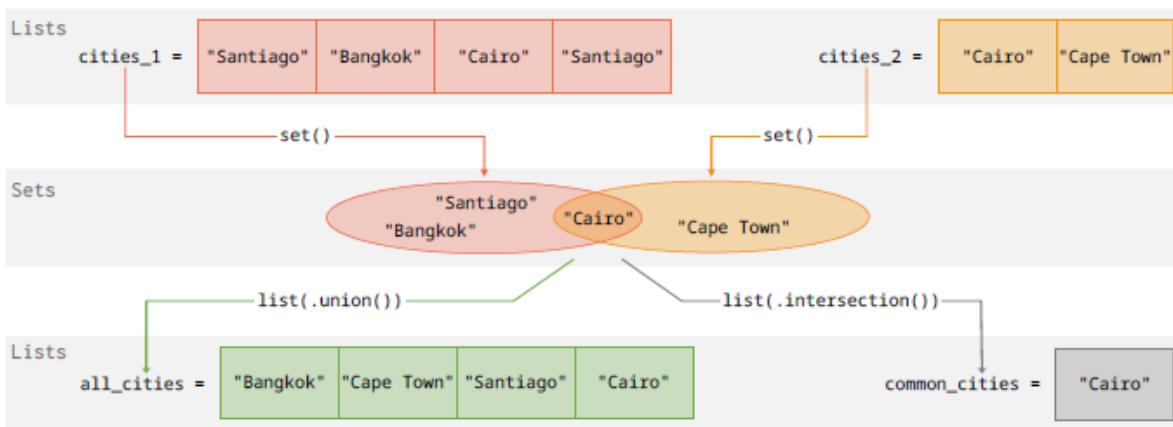
Python

- Tạo một danh sách mới chứa các phần tử duy nhất từ cả hai danh sách:



```
1 
    all_cities = list(set(cities_1).union(set(cities_2)))
    print(all_cities)
1]   ✓  0.0s
```

Kết quả: .....



- Tạo một danh sách mới chứa các phần tử chung cho cả hai danh sách:

```
> common_cities = list(set(cities_1).intersection(set(cities_2)))
   print(common_cities)
35]    ✓ 0.0s
```

Kết quả: .....

## 2. Từ khóa

Bạn đã biết nhiều từ khóa Python—bao gồm `for`, `del`, `def`, `if`, v.v. Bạn có thể tìm thấy danh sách đầy đủ các từ khóa trên một số trang web<sup>2</sup>. Trong phần này, chúng ta sẽ tập trung vào ***lambda***.

### 2.1 lambda

Từ khóa `lambda` được sử dụng để tạo các hàm một dòng chứa một phép toán duy nhất. Những hàm rút gọn này được gọi là hàm ẩn danh—vì chúng không có tên!—hoặc hàm `lambda`—theo sau từ khóa. Để hiểu cách chúng hoạt động, hãy so sánh một hàm chính quy đơn giản tính toán gấp đôi của một số với hàm `lambda` tương ứng.

- Dưới đây là hàm chính quy:

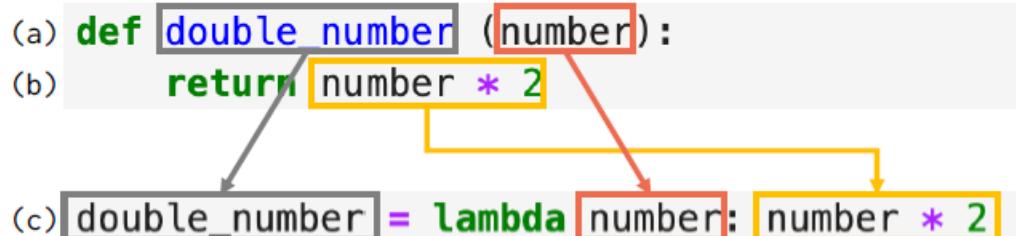
```
[36] def double_number (number):  
    return number * 2  
  
    print(double_number(2))  
[36] ✓ 0.0s
```

Kết quả: .....

- Sau đây là hàm lambda tương ứng:

```
[37] double_number = lambda number: number * 2  
  
print(double_number(5))  
[37] ✓ 0.0s
```

Kết quả: .....



### 3. Các hàm dựng sẵn

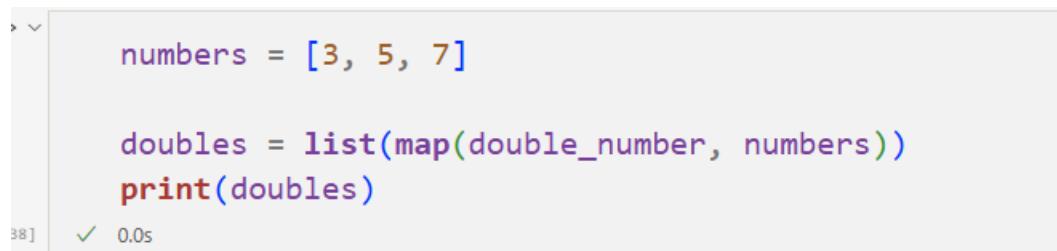
Trong các chương trước, bạn đã tìm hiểu một số hàm dựng sẵn của Python, bao gồm `print()`, `input()`, `len()`, `sum()`, v.v. Python có tổng cộng 71 hàm dựng sẵn, bạn có thể tìm hiểu trong tài liệu chính thức của Python 3. Trong chương này, chúng ta sẽ tìm hiểu sơ lược về hàm `map()`.

#### 3.1 map()

Hàm dựng sẵn `map()` thường được sử dụng để áp dụng hàm **lambda** cho mỗi phần tử của **danh sách**. Bằng cách nào đó, nó hoạt động giống như một vòng lặp `for`, trích xuất

một phần tử danh sách tại một thời điểm và áp dụng một hàm mong muốn cho phần tử đó. Hãy cùng xem hàm map() hoạt động như thế nào với ví dụ sau.

- *Nhân đôi mỗi phần tử danh sách bằng hàm lambda:*



```
numbers = [3, 5, 7]

doubles = list(map(double_number, numbers))
print(doubles)
```

Kết quả: .....

## 4. Mô-đun

Python cung cấp nhiều mô-đun tích hợp sẵn có thể được tìm thấy trên trang web chính thức. Trong phần này, chúng ta sẽ tìm hiểu thêm một hàm của mô-đun *random* và giới thiệu về mô-đun *time*.

### 4.1 random

Chúng ta đã biết hai hàm của mô-đun random, đó là *.randint()* để tạo các số ngẫu nhiên trong một phạm vi và *.choice()* để chọn ngẫu nhiên một phần tử từ danh sách. Khi sử dụng các hàm này, có thể khó gỡ lỗi mã hoặc xác minh tính chính xác của kết quả vì đầu ra ngẫu nhiên được tạo ra khác nhau ở mỗi lần thực thi. Ví dụ: khi chúng ta tạo một số ngẫu nhiên hai lần, chúng ta sẽ nhận được hai kết quả khác nhau, như chúng ta có thể thấy trong ví dụ dưới đây.

- *Tạo một số nguyên ngẫu nhiên từ 1 đến 10 hai lần:*



```
import random

n = random.randint(1, 10)
print("n:", n)
n = random.randint(1, 10)
print("n:", n)
```

[39] ✓ 0.05

Kết quả: .....

Kết quả: .....

- Tạo một số nguyên ngẫu nhiên từ 1 đến 10 hai lần, sử dụng số hạt giống:



```
random.seed(18)
n = random.randint(1, 10)
print("n:", n)

random.seed(18)
n = random.randint(1, 10)
print("n:", n)
```

40] ✓ 0.0s

Kết quả: .....

Kết quả: .....

Do đó, khi sử dụng số hạt giống, việc tạo ra một số ngẫu nhiên có thể tái tạo được và chúng ta có thể gỡ lỗi mã và xác minh kết quả dễ dàng hơn nhiều.

## 4.2 thời gian

Việc biết thời gian tính toán—tức là thời gian cần thiết để thực hiện một phép tính—rất quan trọng, đặc biệt khi chạy các dự án lớn. Trong Python, chúng ta có thể sử dụng module time. Hãy cùng xem nó hoạt động như thế nào với ví dụ sau.

- So sánh thời gian cần thiết để tạo một danh sách có mười, một trăm và một nghìn số không, sử dụng vòng lặp *for* so với sao chép danh sách. Bạn nghĩ sự chênh lệch thời gian sẽ là bao nhiêu?

```
> <ipython>
    import time

    #lists Length
    n_of_elements=[10,100,1000]

    # for each Length
    for n in n_of_elements:

        print("N. of zeros:", len(numbers))

        # create the List using the for Loop
        start = time.time()
        numbers = []
        for _ in range(n):
            numbers.append(0)
        end = time.time()
        print("For loop: {:.6f} sec".format(end-start))

        # create the list using List replication

        start = time.time()
        numbers = [0]*n

        end = time.time()
        print("List repl: {:.6f} sec \n".format(end-start))

[45] ✓ 0.0s
```

Python

Kết quả: .....

## 5. Hoán đổi biến

Hãy kết thúc bài viết bằng một mèo hay trong Python—không có trong nhiều ngôn ngữ lập trình khác!—đó là khả năng hoán đổi biến trên một dòng. Hãy xem cách thức hoạt động của nó trong ví dụ dưới đây:

The screenshot shows a Jupyter Notebook cell with the following code:

```
v1="a"  
v2="b"  
  
print("v1: ",v1)  
print("v2: ",v2,"-----")  
  
v1,v2=v2,v1  
  
print("v1: ",v1)  
print("v2: ",v2)
```

The cell has a status bar at the bottom left indicating [46] and a green checkmark icon at the bottom right indicating 0.0s execution time.

Kết quả: .....

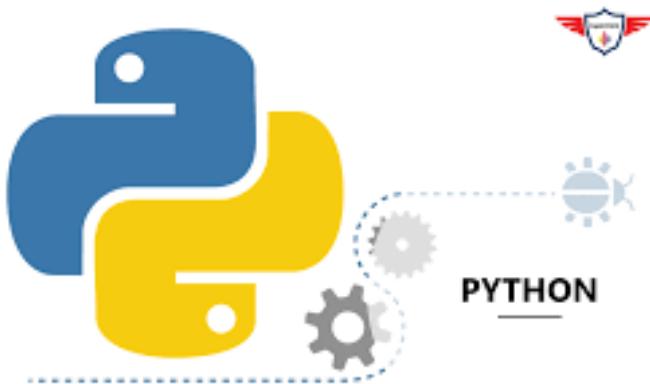
## Tóm tắt

- Tuple là một kiểu dữ liệu chứa các phần tử bất biến và có hai phương thức: `.count()` và `.index()`.
- Set là một kiểu dữ liệu có các phần tử bất biến, không có thứ tự và duy nhất. Chúng có 17 phương thức, bao gồm `.union()` và `.intersection()`. Set có thể là các trung gian hữu ích cho các thao tác danh sách như xóa trùng lặp, hợp nhất hai danh sách hoặc tìm các phần tử chung trong hai danh sách.
- Từ khóa *lambda* cho phép tạo các hàm ẩn danh, là các hàm nhỏ gọn bao gồm (1) *lambda*, (2) *input*, (3) *colon* và (4) *operation* tạo ra *output*.
- *Hàm map()* tích hợp hữu ích khi áp dụng hàm *lambda* cho mỗi phần tử của danh sách.
- Hàm `.seed()` từ mô-đun random cho phép chúng ta tạo các số ngẫu nhiên có thể tái tạo—hay nói một cách kỹ thuật hơn là giả ngẫu nhiên.
- Để tính thời gian tính toán, chúng ta sử dụng mô-đun time. Hàm `.time()` của mô-đun này hoạt động giống như điểm bắt đầu hoặc dừng của đồng hồ bấm giờ.

## PHẦN 10

# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Trong phần cuối cùng này, bạn sẽ học cách lập trình bằng cách sử dụng các lớp và đối tượng, cùng với các thuộc tính và phương thức của chúng.



## 1. Hãy cùng xây dựng một cửa hàng trực tuyến!

*Lớp và đối tượng, thuộc tính và phương thức*

Cho đến thời điểm này, chúng ta đã học ngữ pháp của Python, bao gồm các kiểu dữ liệu, toán tử, vòng lặp và hàm. Chúng ta cũng đã học cách lập trình theo các nguyên tắc của lập trình thủ tục, tức là tạo ra các hướng dẫn từng bước được thực thi tuần tự để đi đến giải pháp cho một nhiệm vụ. Trong phần cuối cùng của cuốn sách, chúng ta sẽ chuyển hướng và tìm hiểu những kiến thức cơ bản về lập trình hướng đối tượng—thường được viết tắt là OOP. Đây là một cách tư duy khác về mã, dựa trên việc biểu diễn thế giới trong các lớp và đối tượng, với các thuộc tính và phương thức của chúng. Hãy cùng làm quen với những khái niệm mới này trong khi giải quyết nhiệm vụ sau.

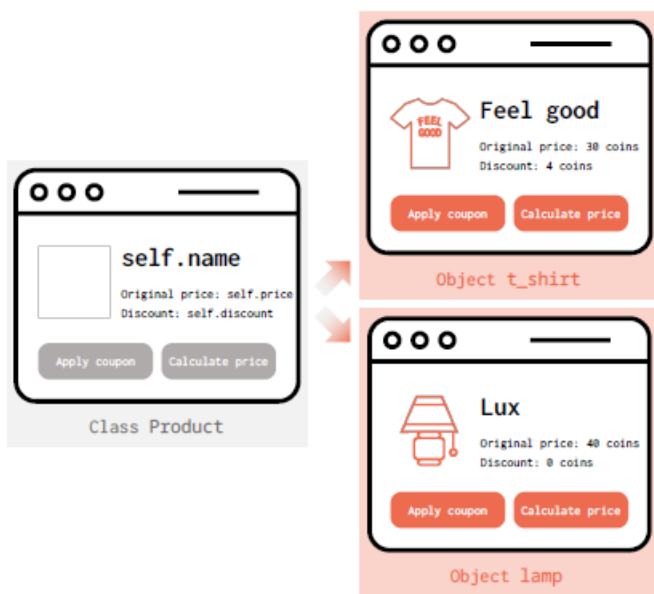
- Bạn đang xây dựng một cửa hàng trực tuyến mới, nơi bạn sẽ bán nhiều loại sản phẩm khác nhau. Mỗi sản phẩm sẽ có một trang web tương tự chứa các đặc điểm như tên, giá gốc và chiết khấu. Ngoài ra, còn có hai hành động khả thi: áp dụng phiếu giảm giá (để cộng giá trị của phiếu giảm giá vào khoản giảm giá) và tính giá (để tính giá sản phẩm).

1. *Tạo một mẫu cho trang web và sau đó tùy chỉnh mẫu đó cho áo phông và đèn với các đặc điểm sau:*

- Áo phông: tên: Feel good, giá gốc: 30 xu, giảm giá khi ra mắt: 4 xu;
- Đèn: tên: Lux, giá gốc: 40 xu, không giảm giá khi ra mắt.

2. *Sau đó, thực hiện các hành động này cho từng sản phẩm:*

- Tính giá sau khi giảm giá;
- Tính giá sau khi thêm phiếu giảm giá SAVE4 trị giá 4 xu vào áo phông và phiếu giảm giá SUMMER10 trị giá 10 xu vào đèn.



## 1. Lớp, đối tượng và thuộc tính

Hãy bắt đầu từ điểm đầu tiên của bài tập. Ở đây, chúng ta có một lớp đại diện cho mẫu trang web và một đối tượng đại diện cho chiếc áo phông—chúng ta sẽ tạo đối tượng đại diện cho chiếc đèn ở cuối phần đầu tiên này. Hãy đọc đoạn mã bên dưới và cố gắng hiểu cú pháp và chức năng của nó.

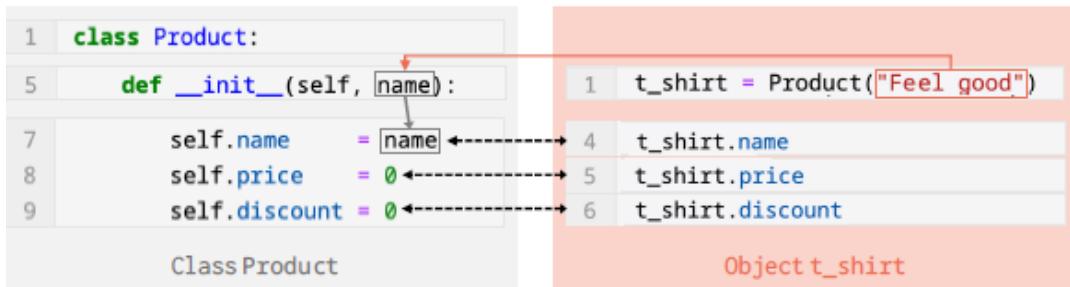
- Đây là lớp đại diện cho một sản phẩm và các thuộc tính của nó:

```

class Product:
    """Lớp đại diện cho một sản phẩm"""
    #

    #--- hàm khởi tạo ---
    def __init__(self, name):
        #khởi tạo
        self.name = name
        self.price = 0
        self.discount = 0

```



- Dưới đây là đối tượng `t_shirt` với các thuộc tính của nó khi khởi tạo và sau khi tùy chỉnh:

```
t_shirt = Product("Feel good")

print("-> Thuộc tính của đối tượng t_shirt khi khởi tạo:")
print("Tên:",t_shirt.name)
print("Giá:", t_shirt.price)
print("Giảm giá:",t_shirt.discount)

print("-> Thuộc tính của đối tượng t_shirt sau khi tùy chỉnh:")
t_shirt.price=30
print("Giá:", t_shirt.price,"coins")
t_shirt.discount=4
print("Giảm giá:",t_shirt.discount,"coins")
```

The screenshot shows a Python code editor with the code above. The status bar indicates a execution time of 0.0s and the word "Python".

Kết quả: .....

- Đây là vật thể `lamp` với các thuộc tính của nó:

```
lamp=Product("Lux")
lamp.price=40
print("Tên:", lamp.name)
print("Giá:", lamp.price, "coins")
print("Giảm giá:", lamp.discount, "coins")
```

[3] 0.0s Python

Kết quả: .....

Kết quả: .....

Kết quả: .....

## 2. Phương thức

Hãy cùng giải quyết điểm thứ hai của nhiệm vụ! Chúng ta cần triển khai hai hành động, đó là áp dụng phiếu giảm giá và tính giá một mặt hàng. Để làm như vậy, chúng ta sao chép lớp *Product* và thêm hai phương thức là *apply\_coupon()* và *calculate\_price()*-trong đoạn mã bên dưới, bạn cũng sẽ tìm thấy phương thức tích hợp *\_\_str\_\_()* mà chúng ta sẽ thảo luận ở cuối chương. Vì lớp đã thay đổi, chúng ta cần khởi tạo lại đối tượng *t\_shirt* để thêm các chức năng mới.

Hãy đọc đoạn mã sau và cố gắng hiểu chức năng của nó!

- Đây là lớp hoàn chỉnh với các thuộc tính và phương thức của nó:

```

class Product:
    """Lớp đại diện cho một sản phẩm"""
    #

    #--- hàm khởi tạo ---
    def __init__(self, name):
        #khởi tạo
        self.name=name
        self.price=0
        self.discount=0

    #----- phương pháp -----
    def apply_coupon(self, coupon):
        """Cap nhat discount dựa trên coupon"""
        if coupon == "SAVE4":
            self.discount+=4
            print("Coupon SAVE4 ĐƯỢC ÁP DỤNG")
        elif coupon=="SUMMER10":
            self.discount+=10
            print("Coupon SUMMER10 ĐƯỢC ÁP DỤNG")
        else:
            print("Coupon của bạn KHÔNG ĐƯỢC ÁP DỤNG")

    def calculate_price(self):
        """Tính số tiền phải trả"""
        updated_price=self.price-self.discount
        return updated_price

    ## --- BUILT-IN METHOD -----
    def __str__(self):
        """In các thuộc tính ra"""
        return "Tên: " + self.name

```

✓ 0.0s

Python

- Khởi tạo đối tượng *t\_shirt* với các thuộc tính của nó. Sau đó, tính giá sau khi giảm giá, và giá sau khi giảm giá và phiếu giảm giá:

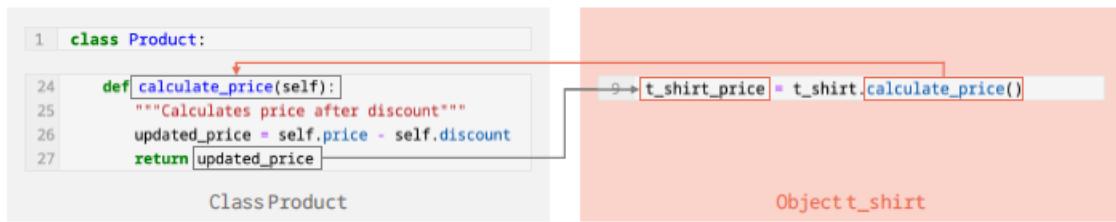
```
# Thiết lập sản phẩm với thuộc tính của nó
t_shirt=Product("Feel good")
t_shirt.price=30
t_shirt.discount=4
print("Tên:",t_shirt.name,"! Giá ban đầu:", t_shirt.price,
      "coins! Khuyến mãi:",t_shirt.discount, "coins.")

# Tính toán giá sau khi áp dụng mã giảm giá
print("->Giá sau khi áp dụng giảm giá")
t_shirt_price=t_shirt.calculate_price()
print("Giá:",t_shirt_price, "coins.")

# áp dụng coupon
print("Áp dụng coupon")
t_shirt.apply_coupon("SAVE4")
print("Khuyến mãi:",t_shirt.discount, "coins.")

# tính toán giá sau cùng sau khi giảm giá và discount
print("->Giá sau khi áp dụng giảm giá và coupon")
t_shirt_price=t_shirt.calculate_price()
print("Giá:",t_shirt_price, "coins.")
```

Kết quả: .....



- Khởi tạo đèn vật thể với các thuộc tính của nó và tính giá của nó trước và sau khi áp dụng phiếu giảm giá SUMMER10:

```
lamp=Product("Lux")  
lamp.price=40  
  
print("Giá ban đầu")  
lamp.price=lamp.calculate_price()  
print("Giá:", lamp.price, "coins.")  
  
print("->Giá sau khi áp dụng Coupon")  
lamp.apply_coupon("SUMMER10")  
lamp.price=lamp.calculate_price()  
print("Giá:", lamp.price, "coins.")
```

✓ 0.0s

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

- In ra hai đối tượng:

```
print(t_shirt)  
print(lamp)
```

✓ 0.0s

Kết quả: .....

Lớp là một khuôn mẫu chứa các thuộc tính và hành động

Đối tượng là một thể hiện của lớp

Thuộc tính là một biến biểu diễn một thuộc tính của đối tượng

Phương thức là một hàm biểu diễn một hành động mà đối tượng có thể thực hiện

- Trong lập trình hướng đối tượng, chúng ta biểu diễn thế giới trong các lớp và đối tượng bằng các thuộc tính và phương thức.
- Lớp là một khuôn mẫu và được giới thiệu bởi từ khóa class.
- Đối tượng là một thể hiện của lớp.
- Thuộc tính là các biến biểu diễn các thuộc tính của đối tượng.
- Phương thức là các hàm trong lớp biểu diễn các hành động mà đối tượng có thể thực hiện. *Tham số đầu tiên của chúng thường là self*.
- Lớp có thể có các phương thức tích hợp, chẳng hạn như hàm tạo `__init__()` và `__str__()`, dùng để định nghĩa `print` của đối tượng.

## **Bài tập áp dụng**

1. Thêm sản phẩm trong cửa hàng trực tuyến! Tạo thêm hai đối tượng từ lớp *Product* với các đặc điểm sau:

- a. Một quả bóng bãi biển tên là *Giant ball*, giá gốc: 10 xu, giảm giá khi ra mắt: 0,50 xu. Tính giá của nó trước và sau khi áp dụng phiếu giảm giá *SAVE4*.
- b. Một cuốn nhật ký có tên *My adventures*, giá 15 xu, giảm giá khi ra mắt: 3 xu. Tính giá của nó trước và sau khi áp dụng phiếu giảm giá *SPRINGSALES30*.

Bạn sẽ nhận được gì khi in từng đối tượng?

## 36. Bảo mật cửa hàng trực tuyến

### Đóng gói

Trong chương này, chúng ta sẽ dựa trên ví dụ về cửa hàng trực tuyến để tìm hiểu về đóng gói, tức là cách định nghĩa quyền truy cập vào các thuộc tính và phương thức bằng cách đặt chúng ở chế độ công khai hoặc riêng tư. Để tìm hiểu ý nghĩa của nó, hãy xem xét nhiệm vụ và cách triển khai giải pháp bên dưới. Bạn đã nắm được hầu hết mã. Hãy tìm kiếm sự khác biệt giữa cách triển khai này và cách triển khai trong chương trước, và cố gắng hiểu ý nghĩa của chúng.

- Khi điền thông tin vào cửa hàng trực tuyến, bạn nhận ra rằng mình muốn giảm thiểu các lỗi có thể làm giảm doanh thu. Vì vậy, bạn đặt các thuộc tính biểu diễn giá và chiết khấu ở chế độ riêng tư, **đồng thời tạo các phương thức get và set để truy cập chúng**. Ngoài ra, bạn nhận ra rằng cần phải bao gồm số **tiền thuế** trong việc tính toán giá cuối cùng. Do đó, bạn triển khai một phương thức riêng tư để tính toán số tiền thuế và sửa đổi **calculate\_price()** cho phù hợp:

```
class Product:  
    """Lớp đại diện cho một sản phẩm"""  
    #  
  
    #--- hàm khởi tạo ----  
    def __init__(self, name):  
        #khởi tạo  
        self.name=name  
        self.__price=0  
        self.__discount=0  
        self.__tax_rate=0.02  
  
    #--- hàm get/set ----  
    def get_price(self):  
        # trả giá trị giá về  
        return self.__price  
  
    def set_price(self, price):  
        # đặt giá trị mới  
        # để chắc chắn ta nên kiểm tra giá trị đầu vào  
            #(1) phải là int hay float  
            #(2) phải là số dương  
        if isinstance(price, (int, float)) and price>0:  
            self.__price=price  
        else:  
            raise ValueError("Bạn đã nhập sai, Giá phải là một " \  
                "số lớn hơn 0.")
```

```
def get_discount(self):
    # trả giá trị giá về
    return self.__discount

def set_discount(self, discount):
    # đặt giá trị mới
    # để chắc chắn chúng ta nên kiểm tra giá trị đầu vào
    #(1) phải là int hay float
    #(2) phải là số dương
    if isinstance(discount, (int, float)) and 0 < discount < self.__price:
        self.__discount = discount
    else:
        raise ValueError("Bạn đã nhập sai, Giảm giá phải là " \
                         "một số lớn hơn 0 và nhỏ hơn giá.")

#----- phương pháp -----
def apply_coupon(self, coupon):
    """Cap nhat discount dựa trên coupon"""
    if coupon == "SAVE4":
        self.__discount += 4
        print("Coupon SAVE4 ĐƯỢC ÁP DỤNG")
    elif coupon == "SUMMER10":
        self.__discount += 10
        print("Coupon SUMMER10 ĐƯỢC ÁP DỤNG")
    else:
        print("Coupon của bạn KHÔNG ĐƯỢC ÁP DỤNG")

def __calculate_tax(self, price):
    # tính giá trị tax
    tax = round(price * self.__tax_rate, 2) # nhân làm tròn 2 số
    print("Tiền thuế trong số", price, "coins:", tax, "coins")
    return tax
```

```
def calculate_price(self):
    """Tính số tiền phải trả"""
    # tính số tiền được giảm giá
    discounted_price=self.__price - self.__discount

    # tính tiền thuế trong giá trị được giảm giá trên
    tax=self.__calculate_tax(discounted_price)

    # cộng tiền thuế vào số tiền sau giảm giá
    taxed_price=discounted_price+tax

    return taxed_price

## --- BUILT-IN METHOD -----
def __str__(self):
    """In các thuộc tính ra"""
    return "Tên: " + self.name
```

4] ✓ 0.0s Python

- Để kiểm tra mã mới, bạn hãy điền lại thông tin trên áo thun, bao gồm: Tên: *Feel good*; Giá gốc: 30 xu; và Giảm giá khi ra mắt: 4 xu. Sau đó, bạn tính giá áo thun trước và sau khi áp dụng mã giảm giá **SAVE4**:

```
# thiết lập tên sản phẩm
t_shirt = Product("Feel good")
print("Tên áo thun:", t_shirt.name)

# thiết lập giá dùng set và đọc lại giá trị dùng get
print("-> Giá ban đầu")
t_shirt.set_price(30)
print("Giá:", t_shirt.get_price(), "coins")
```

21] ✓ 0.0s Python

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

```
# thiết lập giá dùng set và đọc lại giá trị dùng get
print("-> Giảm giá")
t_shirt.set_discount(4)
print("Giảm giá:", t_shirt.get_discount(), "coins")
```

✓ 0.0s

Python

Kết quả: .....

Kết quả: .....

```
print("-> Giá sau khi tính giảm giá và thuế")
t_shirt_price = t_shirt.calculate_price()
print("Giá:", t_shirt_price, "coins")
```

✓ 0.0s

Kết quả: .....

Kết quả: .....

Kết quả: .....

```
print("-> Giá sau khi tính coupon, giảm giá, và thuế")
t_shirt.apply_coupon("SAVE4")
t_shirt_price = t_shirt.calculate_price()
print("Giá:", t_shirt_price, "coins")
```

37] ✓ 0.0s

Python

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

Sự khác biệt chính giữa thuộc tính **public** và **private** là gì? Nó liên quan đến cách chúng ta truy cập chúng, tức là cách chúng ta gán giá trị cho một thuộc tính hoặc lấy giá trị từ một thuộc tính. Trước tiên, hãy cùng làm quen với cách truy cập các thuộc tính **public**.

Thuộc tính *public* có thể được truy cập trực tiếp cả trong một lớp và bởi một đối tượng.

Như bạn đã biết, trong một lớp, chúng ta truy cập một thuộc tính *public* bằng cách sử dụng cú pháp *self.attribute\_name*. Ví dụ: trong lớp *Product*, chúng ta truy cập trực tiếp thuộc tính *public* *self.name* trong phương thức tích hợp *\_\_str\_\_()*. Tương tự, từ một đối tượng, chúng ta truy cập một thuộc tính *public* bằng cách sử dụng *object\_name.attribute\_name*. Ví dụ: từ *t\_shirt*—mà chúng ta khởi tạo trong ô 2, dòng 2—chúng ta truy cập trực tiếp thuộc tính *public* *t\_shirt.name* (dòng 3) để in nó (*print(a)*).

*Còn thuộc tính *private* thì sao?*

Thuộc tính *private* chỉ có thể được truy cập trực tiếp trong một lớp;

Chúng chỉ có thể được truy cập bởi một đối tượng thông qua các phương thức *get/set*.

Trong một lớp, chúng ta truy cập một thuộc tính riêng tư bằng cách sử dụng cú pháp *self.\_\_attribute\_name*, tương tự như những gì chúng ta làm với một thuộc tính công khai.

- Phương thức *get*—còn gọi là *getter*—trả về một thuộc tính riêng tư
- Một phương thức *set*—còn gọi là *setter*—gán giá trị của một tham số cho một thuộc tính riêng tư.

Bây giờ chúng ta hãy xem xét việc đóng gói các phương thức. Nó hoạt động như sau:

Các phương thức *public* có thể được truy cập trực tiếp cả bên trong một lớp và bởi một đối tượng.

Các phương thức *private* chỉ có thể được truy cập bên trong một lớp—không phải bởi một đối tượng.

## Tóm tắt

- Trong đóng gói, chúng ta định nghĩa quyền truy cập công khai hoặc riêng tư vào các thuộc tính và phương thức.
- Tên của một thuộc tính hoặc phương thức riêng tư bắt đầu bằng một dấu gạch dưới kép.

- Một thuộc tính hoặc phương thức công khai có thể được truy cập cả bên trong lớp và bởi đối tượng, trong khi một thuộc tính hoặc phương thức riêng tư chỉ có thể được truy cập bên trong lớp.
- Một đối tượng có thể gọi phương thức **get**—hay getter—để truy xuất giá trị của một thuộc tính riêng tư và phương thức **set**—hay setter—để gán giá trị cho **một thuộc tính riêng tư**.

### **Bài tập áp dụng**

1. Hãy thêm nhiều mặt hàng hơn vào cửa hàng trực tuyến! Từ lớp *Product* đã cập nhật, hãy khởi tạo các đối tượng sau từ chương trước:

- Một chiếc đèn tên là Lux, giá gốc: 40 xu, không giảm giá khi ra mắt. Tính giá của nó trước và sau khi áp dụng phiếu giảm giá SUMMER10.
- Một quả bóng bãi biển tên là Giant ball, giá gốc: 10 xu, giảm giá khi ra mắt: 0,50 xu. Tính giá của nó trước và sau khi áp dụng phiếu giảm giá SAVE4.
- Một cuốn nhật ký tên là My adventures, giá 15 xu, giảm giá khi ra mắt: 3 xu. Tính giá của nó trước và sau khi áp dụng phiếu giảm giá SPRINGSALES30.

### **37. Làm thế nào để thêm mẫu sách?**

#### **Kế thừa**

Cho đến nay, chúng ta đã học những kiến thức cơ bản về lập trình hướng đối tượng. Tức là, chúng ta đã học các khái niệm về lớp và đối tượng, với các thuộc tính *public* và *private* của chúng. Trong chương này, chúng ta sẽ mở rộng khả năng của lớp và đối tượng bằng cách sử dụng kế thừa. Tại đây, bạn sẽ làm quen với các khái niệm về lớp cha và lớp con cùng các thuộc tính của chúng. Bạn đã sẵn sàng chưa?.

- Đã đến lúc thêm sách vào cửa hàng trực tuyến. Đối với trang web của họ, bạn cần thêm nút Đọc mẫu để khách hàng có thể xem trước sách trước khi mua. Tuy nhiên, bạn phải đảm bảo rằng nút này không xuất hiện trên các trang của các sản phẩm khác, chẳng hạn như quần áo hoặc đồ nội thất. Bạn có thể làm điều đó như thế nào?



Tiếp tục với đoạn mã!

- Bạn giữ nguyên lớp *Product* như hiện tại. Đó sẽ là lớp cha.
- Bạn tạo một lớp con đại diện cho sách, kế thừa tất cả các thuộc tính và phương thức từ lớp *Product*. Sau đó, bạn thêm một thuộc tính riêng tư đại diện cho mẫu sách—cùng với các phương thức **get** và **set** của nó—and tạo một phương thức công khai để in mẫu:

```
class Book(Product):
    """lớp con của Product tên là Book"""

    # khởi tạo
    def __init__(self, name):
        super().__init__(name)
        self.__book_sample=""

    # ----- khai báo hàm set và get
    def get_book_sample(self):
        return self.__book_sample

    def set_book_sample(self, name):
        # kiểm tra tên nhập vào phải chuỗi str ko
        if isinstance(name, str):
            self.__book_sample=name
        else:
            raise TypeError ("Tên book phải là dạng chuỗi (str)")

    #----- methods
    def read_sample(self):
        if self.__book_sample!="":
            print(self.__book_sample+ "[...] - Đã có chưa? Mua đi!")
        else:
            print("Book không tồn tại.")

✓ 0.0s
```

Python

- Để kiểm tra mã mới, bạn khởi tạo một đối tượng đại diện cho một cuốn sách lập trình có tên là *Let's code*, giá gốc là 20 xu và giảm giá 2 xu. Sau đó, bạn in ra các đặc điểm của cuốn sách, giá của nó sau khi áp dụng mã giảm giá *SUMMER10* và mẫu của nó:



```
#tạo tên book
coding_book=Book("Let's code!")

#vì Là Lớp con nên được sử dụng hàm Lớp cha
# set giá và giảm giá
coding_book.set_price(20)
coding_book.set_discount(2)

#print các giá trị trên, dùng get để lấy
print("Book tên:", coding_book.name) # name này ở Lớp cha (Product)
print("Giá ban đầu",coding_book.get_price(),"! Giảm giá: ",
      coding_book.get_discount(),"coins")
```

Kết quả: .....

Kết quả: .....



```
#in giá sau cùng
print("Giá sau khi giảm giá, coupon, và tiền thuế")
coding_book.apply_coupon("SUMMER10")
print("Giá",coding_book.calculate_price(),"coins")
```

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....



```
• # cuối cùng đọc tên
print("=> Đang đọc sách")
coding_book.set_book_sample("Coding is a lot about telling" \
    "a computer what to do")
coding_book.read_sample()
```

Kết quả: .....

Kết quả: .....

Một lớp con—hay lớp con—kế thừa các thuộc tính và phương thức của nó từ lớp cha—hay lớp cha—và có thể thêm các thuộc tính và phương thức mới độc đáo cho chính nó.

Nói cách khác:

Kế thừa là một cơ chế cho phép một lớp con

kế thừa và mở rộng các thuộc tính và phương thức của lớp cha.

## Tóm tắt

- Kế thừa bao gồm việc tạo ra một hoặc nhiều lớp con (hoặc lớp con) từ một lớp cha (hoặc lớp cha).
- Lớp con kế thừa các thuộc tính và phương thức từ lớp cha và có thể thêm các thuộc tính và phương thức mới.
- Để tạo một lớp con, chúng ta sử dụng từ khóa class theo sau là tên lớp con và tên lớp cha trong cặp dấu ngoặc tròn.
- Một lớp con chỉ cần một hàm tạo khi nó thêm các thuộc tính mới. Trong trường hợp này, dòng đầu tiên của hàm tạo bao gồm hàm dựng sẵn super() theo sau là dấu chấm và hàm tạo của lớp cha.

## Bài tập áp dụng

1. Kiểm thử lớp cha. Khởi tạo một đối tượng t\_shirt từ lớp cha *Product* có các đặc điểm giống như trong chương trước, tức là tên: Feel good, giá khởi điểm: 30 xu, giảm giá khi ra mắt: 4 xu, phiếu giảm giá SAVE4. Hãy thử thiết lập một mẫu sách và gọi phương thức *read\_sample()*. Điều gì xảy ra và tại sao?

2. Mua một chiếc ô tô điện. Bạn muốn mua một chiếc ô tô điện mới, và đây là những chiếc xe bạn thích:

- Model E-Nature, mức tiêu thụ năng lượng: 15 kWh trên 100 km, dung lượng pin: 75 kWh;
- Model E-Green, mức tiêu thụ năng lượng: 18 kWh trên 100 km, dung lượng pin: 40 kWh.

Bạn muốn tính toán xem dung lượng pin của những chiếc xe này có đủ cho một chuyến đi dài 300 km hay không.

Để làm như vậy, bạn tạo:

- Một lớp cha có tên là *Vehicle*. Hàm khởi tạo của nó chứa các thuộc tính biểu diễn model và mức tiêu thụ năng lượng. Phương pháp này tính toán năng lượng cần thiết bằng cách lấy mức tiêu thụ năng lượng chia cho 100 và nhân với quãng đường lái xe.
- Một lớp con tên là *ElectricCar*. Lớp này thêm một thuộc tính biểu thị dung lượng pin và một phương thức in ra liệu chuyến đi có thể hoàn thành với năng lượng đã tính toán hay không.

Bạn sẽ mua xe nào?

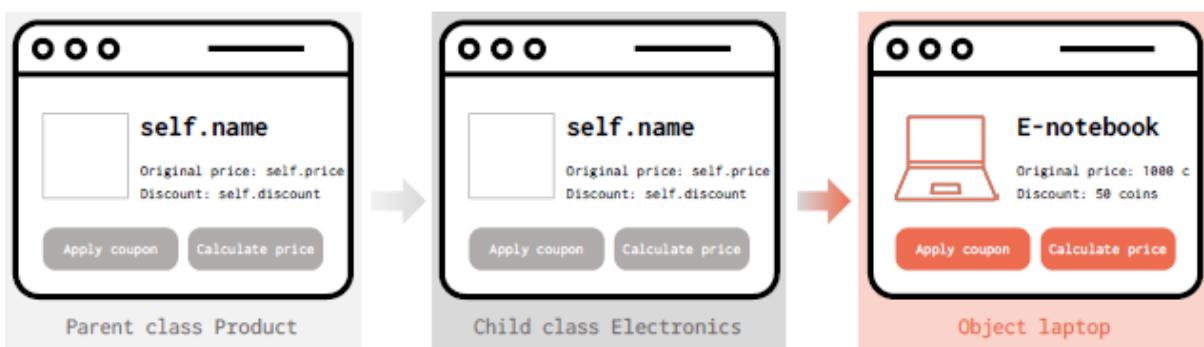
### 38. Tùy chỉnh phiếu giảm giá cho thiết bị điện tử

#### Đa hình

Trong chương cuối cùng này, bạn sẽ tìm hiểu về đa hình, theo nghĩa đen là nhiều hình thức trong tiếng Hy Lạp cổ đại. Đây là thuộc tính của lớp con cho phép chúng ta ghi đè một hoặc nhiều phương thức của lớp cha. Kết quả là, các đối tượng từ lớp cha và các lớp con của nó có thể sử dụng một phương thức có cùng tên nhưng thực hiện các hành động khác nhau. Hãy cùng xem điều này có ý nghĩa gì trong thực tế bằng cách thực hiện phép cộng cuối cùng vào cửa hàng trực tuyến. Hãy theo dõi Sô tay 38!

- Để hoàn thiện cửa hàng trực tuyến, bạn thêm các sản phẩm điện tử. Khi mua hàng, khách hàng chỉ có thể sử dụng phiếu giảm giá TECH100, trị giá 100 xu, thay vì phiếu giảm giá SAVE4 và SUMMER10, có thể được sử dụng cho các sản phẩm khác. Bạn sẽ thay đổi chức năng áp dụng phiếu giảm giá như thế nào để đáp ứng yêu cầu mới này?

Giải pháp cho nhiệm vụ của chúng ta được mô tả trong dưới. Chúng ta sẽ tạo một lớp con tên là Electronics, trông tương tự như lớp cha *Products*. Tuy nhiên, phương thức *apply\_coupon()* sẽ chứa mã khác để cho phép sử dụng phiếu giảm giá TECH100. Chúng ta sẽ kiểm tra mã của mình với một đối tượng mới tên là laptop.



Hãy cùng xem mã!

- Bạn giữ nguyên lớp cha *Product*
- Bạn tạo lớp con mới *Electronics* kế thừa từ *Product* và ghi đè phương thức của nó *apply\_coupon()*:

```
class Electronics(Product):
    """Lớp con đại diện cho một sản phẩm điện tử"""

    #-----methods-----
    # bây giờ chúng ta chỉ viết đè lên phương thức apply_coupon()
    def apply_coupon(self, coupon):

        if coupon=="TECH100":
            self.set_discount(self.get_discount()+100)
            # vì biến discount là private nên bạn phải set và get!
            print("Mã coupon TECH100 được áp dụng!")
        else:
            print("Mã coupon của bạn không có giá trị.")

15] ✓ 0.0s Python
```

```
# thiết lập các thông số
laptop = Electronics("E-notebook")
laptop.set_price(1000)
laptop.set_discount(50)

# Tính toán giá sau khi giảm giá
print("-> Giá sau khi giảm giá")
laptop_price = laptop.calculate_price()
print("Giá:", laptop_price, "coins")

15] ✓ 0.0s Python
```

Kết quả: .....

Kết quả: .....

Kết quả: .....

```
# Giá sau khi áp dụng coupon
print("-> Giá sau khi tính coupon")
laptop.apply_coupon("TECH100")
laptop_price = laptop.calculate_price()
print("Giá:", laptop_price, "coins")

15] ✓ 0.0s Python
```

Kết quả: .....

Kết quả: .....

Kết quả: .....

Kết quả: .....

Đa hình là một cơ chế cho phép lớp con ghi đè lên phương thức của lớp cha trong khi vẫn giữ nguyên tên phương thức.

Như bạn thấy, *laptop* đã gọi *apply\_coupon()* từ lớp *Electronics*—không phải từ lớp *Product*!—và *calculate\_price()* từ lớp *Product* vì trong lớp *Electronics*, chúng ta không ghi đè *calculate\_price()*.

Điều gì sẽ xảy ra nếu chúng ta áp dụng một phiếu giảm giá hợp lệ cho lớp cha? Hãy thử với *SAVE4*:

```
> <ipython> laptop.apply_coupon("SAVE4")
[✓] 0.0s
```

Kết quả: .....

Nếu chúng ta sử dụng phiếu giảm giá *SAVE4* và *TECH100* cho các đối tượng *t\_shirt* của lớp *Product* thì sao?

```
> <ipython> t_shirt = Product("Feel good")
      t_shirt.apply_coupon("SAVE4")
      t_shirt.apply_coupon("TECH100")
[52] ✓ 0.0s
```

Kết quả: .....

Kết quả: .....

Kết quả: .....

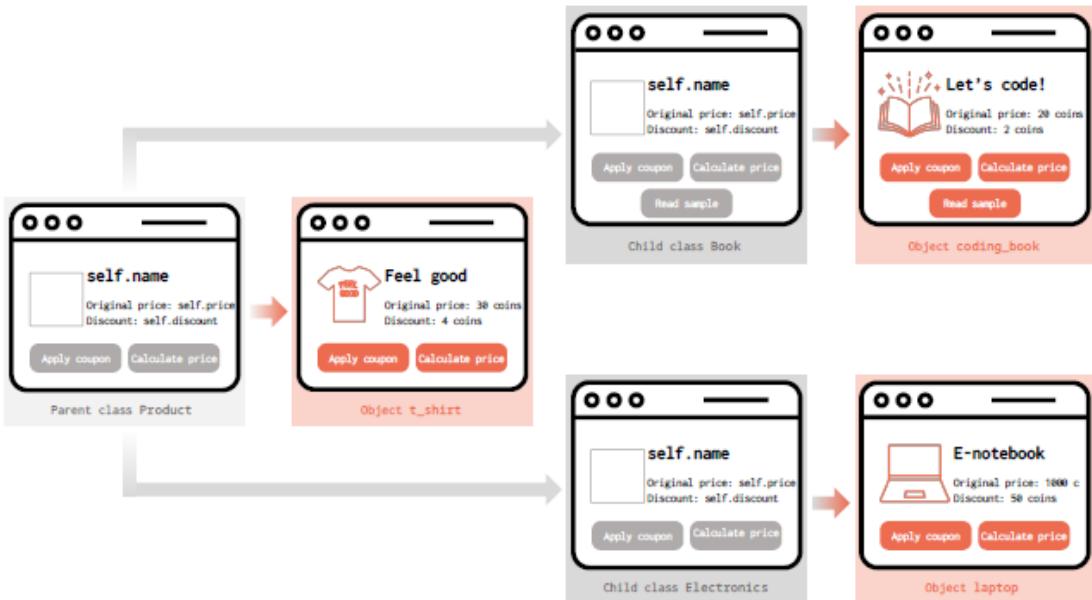
Tương tự như vậy, điều gì xảy ra khi chúng ta sử dụng phiếu giảm giá SAVE4 và TECH100 cho coding\_book của lớp Book?

```
coding_book = Book("Let's code!")
coding_book.apply_coupon("SAVE4")
coding_book.apply_coupon("TECH100")
```

✓ 0.0s

Kết quả: .....

Kết quả: .....



## Tóm tắt

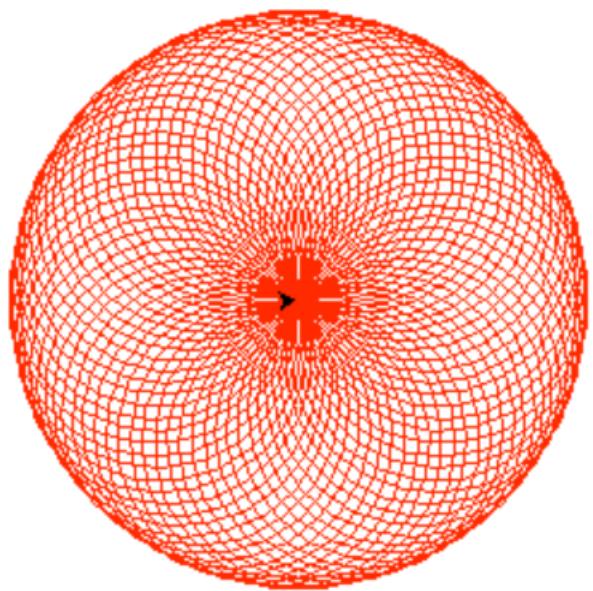
- Đa hình cho phép các lớp con ghi đè lên các phương thức của lớp cha trong khi vẫn giữ nguyên tên phương thức.
- Khi được gọi, phương thức chính xác sẽ tự động được chọn dựa trên lớp của đối tượng.

## **Bài tập áp dụng**

1. Đặt đồ ăn trực tuyến. Bạn phải tạo một chương trình phần mềm mới cho việc đặt đồ ăn trực tuyến. Khách hàng yêu cầu một hệ thống đặt hàng cơ bản với hai tùy chọn tùy chỉnh, một cho pizza và một cho burger. Đơn hàng cơ bản có giá ban đầu là 10 xu. Đối với pizza, giá tăng 20% nếu kích thước là cỡ vừa, và tăng 50% nếu kích thước là cỡ lớn. Đối với burger, giá tăng 1,5 xu nếu khách hàng thêm phô mai và tăng 1 xu nếu họ thêm hành tây. Để kiểm tra mã của bạn, hãy tạo bốn đơn hàng:một pizza nhỏ, một pizza cỡ vừa, và một pizza cỡ lớn, cũng như một burger có thêm phô mai và hành tây. Tổng chi phí là bao nhiêu?

```
import turtle,colors
turtle.bgcolor("black")
t=turtle.Turtle()
t.speed(0)
turtle.colormode(255)
n=36
h=0
for i in range(72):
    col=colors.hsv_to_rgb(h,1,1)
    t.pencolor(int(col[0]*255),int(col[1]*255),int(col[2]*255))
    for _ in range(6):
        t.circle(60)
        t.left(60)
    t.left(5)
    h=1/n
    h%=1
turtle.done()
```

⌚ 54.3s      Python



pip install qrcode

pip install Pillow

```
import qrcode
from PIL import Image

data=input("Xin moi nhap cai ban muon tao ma QR: ")
qr=qrcode.QRCode(version=3,box_size=10,border=4)
qr.add_data(data)
qr.make(fit=True)
image=qr.make_image(fill="Black",back_color="Peru")

image.save("qr_code.png")
Image.open("qr_code.png")
```

✓ 3.1s      Python

