1. Function và Packages
   1. Functions(các hàm)

# Khái niệm

Các hàm giúp chia nhỏ chương trình của chúng ta thành các phần nhỏ hơn và tổ chức lại theo từng mô-đun. Khi chương trình của chúng ta phát triển ngày càng lớn hơn, các hàm giúp cho nó dễ tổ chức và dễ quản lý hơn. Hơn nữa, nó ránh lặp lại các đoạn code và làm cho code có thể được sử dụng lại.

**Cú pháp một hàm**:

A picture containing text

Description automatically generated

Ví dụ, hàm tính giai thừa:

A picture containing graphical user interface

Description automatically generated

Ví dụ, gọi lại hàm: **Giaithua(5)**

Số 5 (được gọi là đối số, argument) được truyền vào hàm thông qua tham số (parameter) n (n sẽ nhận giá trị 5). Ngoài ra, đối số cũng có thể là biến.

Lưu ý: Trong python, định nghĩa hàm phải luôn có trước lệnh gọi hàm. Nếu không, chúng ta sẽ gặp lỗi.

# Lệnh return

Cú pháp:



Ví dụ, hàm trả về max, min

Graphical user interface, application

Description automatically generated

Lệnh return trong hàm MaxMin() trả về hai giá trị, thông qua hai biểu thức

đặt ngay sau nó, cách nhau bởi dấu cách (max và min). Đoạn trình sau sử dụng

hàm MaxMin() để tìm số lớn nhất và nhỏ nhất trong ba số 5, 2, 8

Text

Description automatically generated with medium confidence

Kết quả thu được:

Background pattern

Description automatically generated with low confidence

Nếu không có biểu thức nào sau câu lệnh return, hoặc bản thân câu lệnh return không có bên trong một hàm, thì hàm cũng vẫn trả về nhưng nó sẽ trả về đối

tượng None.

# Phạm vi hoạt động của biến

Phạm vi của một biến là một đoạn trình mà biến đó có thể được nhận ra. Các tham số của hàm và các biến được định nghĩa bên trong một hàm sẽ không được “nhìn thấy” từ bên ngoài hàm. Nói cách khác, phạm vi hoạt động của nó chỉ là ở trong thân hàm. Do đó, chúng có phạm vi cục bộ (local variable).

Thời gian tồn tại của một biến là khoảng thời gian mà biến tồn tại trong bộ nhớ. Thời gian tồn tại của các biến bên trong một hàm là thời gian mà hàm được thực thi. Chúng sẽ bị giải phóng khi chúng ta quay trở lại chỗ đã gọi hàm. Do đó, một hàm không nhớ các giá trị của một biến từ các lần gọi trước của nó.

# Đối số

Khi định nghĩa hàm, nếu hàm có tham số, thì khi gọi hàm, ta cần truyền đầy

đủ giá trị cho các tham số này. Giá trị được truyền vào hàm khi gọi hàm (thông qua

tham số) được gọi là đối số.

## Đối số mặc định

Tham số có thể sử dụng đối số mặc định. Khi gọi hàm, ta có hai lựa chọn:

1). Truyền đầy đủ đối số cho tham số và, 2). Truyền thiếu đối số cho tham số có

định nghĩa đối số mặc định. Như vậy, muốn tham số sử dụng đối số mặc định, ta

cần chỉ rõ trong lúc định nghĩa hàm

A picture containing chart

Description automatically generated

Kết quả thu được:



Bất kỳ tham số nào trong một hàm đều có thể được định nghĩa giá trị đối số

mặc định. Nhưng một khi chúng ta có một tham số có giá trị mặc định, tất cả các

tham số bên phải của nó cũng phải có giá trị mặc định. Nói khác đi, ta không thể

định nghĩa tham số Ten có giá trị mặc định, trong khi tham số Thongdiep lại không

có. Hàm sau sẽ có báo lỗi khi thực thi:

Chart

Description automatically generated

Thông báo lỗi có thể là:



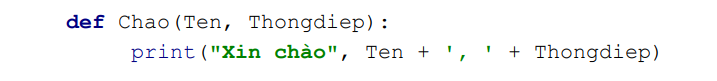
## Truyền tham số theo tên

Python cho phép các hàm được gọi bằng cách sử dụng các đối số không

theo thứ tự nhưng phải chỉ rõ tên của tham số nào sẽ nhận được đối số nào. Khi

chúng ta gọi các hàm theo cách này, thứ tự (vị trí) của các đối số có thể bị thay đổi.

Ví dụ với hàm Chao() ở trên:



Câu lệnh gọi hàm sau là hợp lệ:



Nếu một tham số được truyền giá trị theo tên, tất cả các tham số sau nó cũng

phải được truyền theo tên.

## Truyền tham số với số lượng đối số tùy ý

Đôi khi, chúng ta không biết trước số lượng đối số sẽ được truyền vào một

hàm. Python cho phép chúng ta xử lý tình huống này thông qua các lệnh gọi hàm

với một số lượng đối số tùy ý.

Trong định nghĩa hàm, chúng ta sử dụng dấu hoa thị (\*) trước tên tham số

để biểu thị loại đối số này. Đây là một ví dụ:

Hàm Chao() sau đây muốn in ra lời chào tới nhiều người. Số lượng người

chào là không biết trước nên ta định nghĩa (thêm \* vào trước tham số CacTen):

A picture containing text

Description automatically generated

Khi gọi hàm, ta có thể truyền số lượng đối số tùy thích. Sau đây là lời gọi

hàm truyền ba đối số:



Kết quả thu được:

Background pattern

Description automatically generated with low confidence

# Biến toàn cục

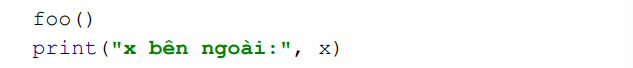
Trong Python, một biến được khai báo bên ngoài hàm hoặc trong phạm vi

toàn cục được gọi là biến toàn cục. Điều này có nghĩa là một biến toàn cục có thể

được truy cập bên trong hoặc bên ngoài hàm.

A picture containing text

Description automatically generated



Kết quả thu được:

Text

Description automatically generated with medium confidence

## Phạm vi hoạt động của biến

Phạm vi của một biến là một đoạn trình mà biến đó có thể được nhận ra.

Các tham số của hàm và các biến được định nghĩa bên trong một hàm sẽ không

được “nhìn thấy” từ bên ngoài hàm. Nói cách khác, phạm vi hoạt động của nó chỉ

là ở trong thân hàm. Do đó, chúng có phạm vi cục bộ (local variable).

Thời gian tồn tại của một biến là khoảng thời gian mà biến tồn tại trong bộ

nhớ. Thời gian tồn tại của các biến bên trong một hàm là thời gian mà hàm được

thực thi. Chúng sẽ bị giải phóng khi chúng ta quay trở lại chỗ đã gọi hàm. Do đó,

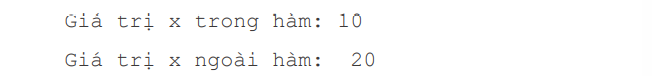
một hàm không nhớ các giá trị của một biến từ các lần gọi trước của nó

Ví dụ:

Graphical user interface, text

Description automatically generated with medium confidence

Kết quả thu được:



Ở đây, chúng ta có thể thấy rằng giá trị của x ban đầu là 20. Mặc dù hàm

my\_func() đã thay đổi giá trị của x thành 10, nó không ảnh hưởng đến giá trị bên

ngoài hàm. Điều này là do biến x bên trong hàm (cục bộ) khác với biến bên ngoài.

Mặc dù chúng có cùng tên, nhưng chúng là hai biến khác nhau với phạm vi khác

nhau

Mặt khác, các biến bên ngoài của hàm có thể nhìn thấy từ bên trong. Chúng

có phạm vi toàn cục.

**Chú ý**: Chúng ta có thể đọc các giá trị của biến toàn cục từ bên trong hàm

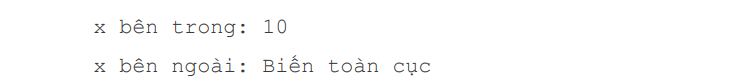
nhưng không thể thay đổi (ghi) giá trị của nó

Ví dụ:

Graphical user interface, text

Description automatically generated

Kết quả thu được:



## Global

Để sửa đổi giá trị của các biến bên ngoài hàm, chúng phải được khai báo là

biến toàn cục bằng cách sử dụng từ khóa global

Graphical user interface, text

Description automatically generated

Kết quả thu được:

A picture containing background pattern

Description automatically generated

# Python modules (Module trong Python)

Một mô-đun là một tệp chứa các câu lệnh Python và các định nghĩa. Một tệp

chứa code Python, ví dụ: example.py, được gọi là mô-đun và tên mô-đun của nó sẽ

là example

Chúng ta sử dụng các mô-đun để chia nhỏ chương trình lớn thành các tệp

mà ta có thể quản lý và tổ chức chúng. Hơn nữa, các mô-đun cung cấp khả năng tái

sử dụng của code.

Chúng ta có thể định nghĩa các hàm mà ta sử dụng nhiều nhất trong một

mô-đun và khi sử dụng trong các chương trình, ta import nó, thay vì sao chép các

hàm vào các chương trình khác nhau

## Tạo và sử dụng một module:

Bước 1. Tạo module có tên MyModule: Thêm một file .py vào project, đặt

tên là MyModule.py. Trong file này, ta định nghĩa hai hàm Cong() và Tru() hai số

a, b.

Bước 2. Sử dụng module MyModule.py: Tong một file .py khác (giả sử file

main.py), để sử dụng hai hàm Cong() và Tru() đang chứa trong MyModule, ta sử

dụng câu lệnh import MyModule

Graphical user interface, text, application, email

Description automatically generated

*Tạo 1 module*

Graphical user interface, text, application, email

Description automatically generated

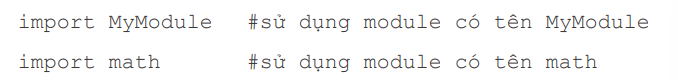
*Sử dụng module với lệnh import*

## Làm việc với module

### Sử dụng toàn bộ module - import

import <Tên\_module>

Ví dụ:



Khi sử dụng toàn bộ module, để truy cập các hàm, biến, hằng,... đã được

định nghĩa trong module, ta viết thêm tên module vào trước của hàm, biến, hằng,...đó. Ví dụ:

Text

Description automatically generated with low confidence

Đặt bí danh cho module:

A picture containing text

Description automatically generated

### Chỉ sử dụng một số thứ trong module

Đôi khi ta không cần import toàn bộ module mà chỉ cần sử dụng một số

thành phần (hàm, biến, hằng, ...) trong module. Khi đó ta sử dụng from/ import. Ví

dụ, ta có thể import riêng rẽ hai hàm cong() và tru() trong MyModule như sau

Text

Description automatically generated

Ta cũng có thể đặt bí danh cho các thành phần mà ta import cho dễ sử dụng,

ví dụ hàm cong() được đặt bí danh là add, hàm tru() được đặt bí danh là minus. Khi

đó, ta gọi hai hàm thông qua bí danh của nó mà không thể gọi bằng tên thật

Text

Description automatically generated

### import tất cả các tên trong module:

Để sử dụng tất cả các tên, bao gồm hàm, biến, hằng,... trong một module mà không cần chỉ rõ tên module khi sử dụng, ta sử dụng lệnh import \* như sau (ví dụ sử dụng module MyModule:



Khi đó, các thành phần trong MyModule được sử dụng qua tên của nó mà không cần đặt tên module phía trước

Text

Description automatically generated with medium confidence

### Đặt đường dẫn tới module

Khi một module được đặt trong cùng thư mục với các file .py sử dụng nó, trình thông dịch sẽ dễ dàng tìm thấy module đó. Nhưng khi chúng ta đặt module ở một nơi nào đó, nếu chương trình không tìm thấy module đó, nó sẽ báo lỗi. Khi đó, ta có thể cần biết cách thiết lập đường dẫn tới module: Trong PyCharm, ta làm như sau:

Bước 1: Chọn tên project, chọn setting (nút setting có biểu tượng  nằm tại góc tên, bên phải của cửa sổ chính Pycharm).

Bước 2: Chọn Project: MyProject (ví dụ tên của project là MyProject), chọn Python Interpreter. Cửa sổ hiện ra như hình

Graphical user interface, text, application

Description automatically generated

Bước 3. Chọn biểu tượng  (góc trên, bên phải), chọn show all. Cửa sổ hiện ra như hình

Graphical user interface, text, application, email

Description automatically generated

Bước 4. Chọn biểu tượng  (màu đỏ trên hình 2.4). Một cửa sổ mới hiện ra, ta chọn biểu tượng dấu +, sau đó chọn thư mục chứa module và chọn OK

# Python Packages (Gói trong module)

Một chương trình rất lớn sẽ được tạo bởi rất nhiều module. Khi đó, Python cho phép ta tổ chức các module trong một cấu trúc cây thư mục đặc biệt, gọi là các gói (packages).

Một thư mục nếu muốn trở thành một gói, ta cần đặt một file có tên ‘\_\_init\_\_.py’ vào trong thư mục đó để Python coi nó như một gói. Tệp này có thể để trống nội dung. Tương tự với các gói con, nó cũng có chứa file ‘\_\_init\_\_.py’. Vì vậy:

o Trong một package/ sub-package có chứa:

- File chỉ dấu: \_\_init\_\_.py (bắt buộc)

- Các file module

- Sub-packages

Hình sau cho thấy một gói có tên là Game, bên trong nó chứa ba gói con (Sound, Image, Level). Trong mỗi gói con là các module. Tất cả các package, subpackage đều chứa file chỉ dấu \_\_init\_\_.py.

Diagram

Description automatically generated

## Sử dụng các module trong gói

Để import các module trong một gói, ta sử dụng toán tử dấu chấm (.): import 〈Tên\_gói〉.[Tên\_gói\_con.]〈Tên\_module〉 Ví dụ: để sử dụng module có tên là open trong hình , ta có thể viết:

A picture containing logo

Description automatically generated

Hoặc:

A picture containing text

Description automatically generated

Hoặc:



Hoặc:



## Đặt đường dẫn tới các gói

Nếu chương trình không tìm thấy gói mà ta sử dụng và nó có sẵn trên máy tính, ta có thể đặt đường dẫn tới gói. Cách đặt tương tự như đặt đường dẫn tới module, đã giới thiệu ở phần trên.

* 1. Bài tập vận dụng

**Bài 1**: Viết hàm tính khoảng cách Euclidean giữa hai điểm A(x1, y1) và B(x2, y2).

Viết hàm kiểm tra xem hai điểm A, B, điểm nào gần tâm O hơn.

**Bài 2**: Hãy tổ chức môt chương trình bằng các module sau:

**Module 1**: Định nghĩa tỷ giá: USD = 23000, EUR = 26000, RUB = 170

**Module 2**: Các hàm quy đổi n USD/ EUR/ RUB ra VND (ba hàm)

Định nghĩa các hàm cộng ba số thực, cộng hai số thực

**Chương trình chính**: - Sử dụng các hàm trong hai module để:

Nhập vào số tiền USD, EUR, RUB hiện có;

In ra tổng số tiền VND sau quy đổi.

**Bài 3**: Tạo một package theo cấu trúc sau Với module1.py và module2.py là hai module đã tạo ra ở **bài 2**

Diagram, timeline

Description automatically generated

Viết chương trình như trong **bài 2** để sử dụng package vừa tạo