Chương 7. Hàm (function)

Nội dung

- Định nghĩa Hàm
- Truyền tham số
- Giá trị trả về
- Truyền một danh sách vào hàm
- Truyền một đối số tùy ý
- Lưu trữ hàm trong module

7.1. Định nghĩa Hàm

- Cấu trúc đơn giản nhất của một hàm:
- Dòng lệnh đầu tiên sử dụng từ khóa def để thông báo cho Python rằng ta đang định nghĩa một hàm.
- Bất kỳ dòng thụt lề nào theo sau def greet_user(): tạo nên phần thân của hàm
- Dòng """Display a simple greeting.""" là một chú thích được gọi là mỗi chuỗi tài liệu
- Dòng print("Hello!") Là dòng code duy nhất trong phần thân của hàm này, vì vậy, greet_user() chỉ có một lệnh: print(“Hello!").

```
def greet_user():
        """Display a simple greeting."""
        print("Hello!")

greet_user()
```

```
Hello!
```

Truyền thông tin tới một hàm

- Để truyền thông tin tới một hàm, nhập tham số username trong dấu ngoặc đơn định nghĩa hàm
- Hàm yêu cầu ta cung cấp một giá trị cho username mỗi khi gọi nó

```
def greet_user(username):
        """Display a simple greeting."""
        print(f"Hello, {username.title()}!")

greet_user('jesse')
```

```
Hello, Jesse!
```

Lệnh greet_user('jesse') sẽ gọi hàm greet_user() và cung cấp cho hàm thông tin cần thiết để thực hiện lệnh gọi print(). Hàm chấp nhận tên được chuyển và hiển thị lời chào cho tên đó

Đối số và tham số

❑Biến username của greet_user() là một ví dụ về một *tham số*, một phần thông tin mà hàm cần để thực hiện công việc của nó.

❑Giá trị 'jesse' trong greet_user('jesse') là một ví dụ về *đối số*. Đối số là một phần thông tin được truyền từ một lệnh gọi hàm đến một hàm. Khi chúng ta gọi hàm, chúng ta đặt giá trị mà chúng ta muốn hàm hoạt động trong dấu ngoặc đơn.

❑Trong trường hợp này, đối số 'jesse' đã được chuyển đến hàm greet_user() và giá trị được gán cho tham số username.

7.2. Truyền tham số

- Bởi vì một định nghĩa hàm có thể có nhiều tham số, một lệnh gọi hàm có thể cần nhiều đối số. Chúng ta có thể truyền các đối số cho các hàm của mình theo một số cách.
- Ta có thể sử dụng các đối số có vị trí, các đối số này cần theo cùng thứ tự mà các tham số đã được viết; các đối số từ khóa, trong đó mỗi đối số bao gồm một tên biến và một giá trị; và danh sách và từ điển các giá trị.

Đối số có vị trí

- Khi gọi một hàm, Python phải khớp từng đối số trong lệnh gọi hàm với một tham số trong định nghĩa hàm. Cách đơn giản nhất để làm điều này là dựa trên thứ tự của các đối số được cung cấp

```
def describe_pet(animal_type, pet_name):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.”)
        print(f"My {animal_type}'s name is {pet_name.title()}.”)

describe_pet('hamster', 'harry')
```

```
I have a hamster.
My hamster's name is Harry.
```

Nhiều lời gọi hàm

- Có thể gọi một hàm nhiều lần nếu cần

```
def describe_pet(animal_type, pet_name):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.")
        print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
```

```
I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.
```

Vấn đề thứ tự trong Đối số có vị trí

- Ta có thể nhận được kết quả không mong muốn nếu trộn thứ tự của các đối số trong một lệnh gọi hàm khi sử dụng các đối số vị trí

```
def describe_pet(animal_type, pet_name):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.")
        print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('harry', 'hamster')
```

```
I have a harry.
My harry's name is Hamster.
```

Đối số từ khóa

- Đối số từ khóa là một cặp tên-giá trị được truyền cho một hàm.
- Chúng ta liên kết trực tiếp tên và giá trị bên trong đối số, vì vậy khi ta truyền đối số vào hàm, sẽ không có sự nhầm lẫn nào

```
def describe_pet(animal_type, pet_name):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.")
        print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(animal_type='hamster', pet_name='harry')
```

Thứ tự của các đối số từ khóa không quan trọng vì Python biết mỗi giá trị sẽ đi đến đâu. Hai lệnh gọi hàm sau đây là tương đương

```
describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
```

Giá trị mặc định

- Khi viết một hàm, ta có thể xác định một giá trị mặc định cho mỗi tham số. Nếu một đối số cho một tham số được cung cấp trong lệnh gọi hàm, thì Python sẽ sử dụng giá trị đối số. Nếu không, nó sử dụng giá trị mặc định của thông số.
- Khi xác định giá trị mặc định cho một tham số, ta có thể loại trừ đối số tương ứng mà ta thường viết trong lệnh gọi hàm.

```
def describe_pet(pet_name, animal_type='dog'):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.")
        print(f"My {animal_type}'s name is {pet_name.title()}.")


describe_pet(pet_name='willie')
```

```
I have a dog.
My dog's name is Willie.
```

Giá trị mặc định

- Để mô tả một con vật không phải con chó, ta có thể sử dụng một lệnh gọi hàm như sau:

```
describe_pet(pet_name='harry', animal_type='hamster')
```

Bởi vì một đối số rõ ràng cho animal_type được cung cấp, Python sẽ bỏ qua giá trị mặc định của tham số.

Ghi chú: Khi ta sử dụng giá trị mặc định, bất kỳ thông số nào có giá trị mặc định cần được liệt kê sau tất cả các tham số không có giá trị mặc định. Điều này cho phép Python tiếp tục diễn giải các đối số vị trí một cách chính xác.

Lệnh gọi hàm tương đương

- Bởi vì các đối số vị trí, đối số từ khóa và giá trị mặc định đều có thể được sử dụng cùng nhau, nên thường ta sẽ có một số cách tương đương để gọi một hàm.

```
def describe_pet(pet_name, animal_type='dog'):
```

```
# A dog named Willie.
describe_pet('willie')
describe_pet(pet_name='willie')

# A hamster named Harry.
describe_pet('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
```

Tránh lỗi đối số

Các đối số không khớp xảy ra khi ta cung cấp ít hoặc nhiều đối số hơn một hàm cần thực hiện công việc của nó

```
def describe_pet(animal_type, pet_name):
        """Display information about a pet."""
        print(f"\nI have a {animal_type}.")
        print(f"My {animal_type}'s name is {pet_name.title()}.”)

describe_pet()
```

```
Traceback (most recent call last):
        File "pets.py", line 6, in <module>
                describe_pet()
TypeError: describe_pet() missing 2 required positional arguments: 'animal_
type' and 'pet_name'
```

7.3. Giá trị trả về

- Không phải lúc nào một hàm cũng phải hiển thị trực tiếp đầu ra của nó. Thay vào đó, nó có thể xử lý một số dữ liệu và sau đó trả về một giá trị hoặc tập hợp các giá trị.
- Giá trị mà hàm trả về được gọi là giá trị trả về. Câu lệnh trả về nhận một giá trị từ bên trong một hàm và gửi trở lại dòng được gọi là hàm.
- Giá trị trả về cho phép ta chuyển phần lớn công việc khó khăn của chương trình sang các hàm, điều này có thể đơn giản hóa phần nội dung chương trình của mình.

Trả về giá trị đơn

- Hãy xem xét một hàm lấy họ và tên và trả về tên đầy đủ

```
def get_formatted_name(first_name, last_name):
        """Return a full name, neatly formatted. """
        full_name = f"{first_name} {last_name}"
        return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
```

```
Jimi Hendrix
```

Khi gọi một hàm trả về một giá trị, cần cung cấp một biến mà giá trị trả về có thể được gán cho. Trong trường hợp này, giá trị trả về được gán cho biến musician.

Tạo số đối số tùy chọn

- Đôi khi, việc đặt một đối số tùy chọn để những người sử dụng hàm có thể chọn cung cấp thêm thông tin chỉ khi họ muốn. Chúng ta có thể sử dụng các giá trị mặc định để làm cho một đối số tùy chọn.

```
def get_formatted_name(first_name, middle_name, last_name):
        """Return a full name, neatly formatted. """
        full_name = f"{first_name} {middle_name} {last_name}"
        return full_name.title()

musician = get_formatted_name('john', 'lee', 'hooker ')
print(musician)
```

```
John Lee Hooker
```

Tạo số đối số tùy chọn

- Để đặt tên đệm là tùy chọn, chúng ta có thể cung cấp cho đối số middle_name một giá trị mặc định trống và bỏ qua đối số trừ khi người dùng cung cấp giá trị. Để làm cho get_formatted_name() hoạt động mà không có tên đệm, chúng ta đặt giá trị mặc định của middle_name thành một chuỗi trống và di chuyển nó đến **cuối** danh sách các tham số:

```
def get_formatted_name(first_name, last_name, middle_name=''):
        """Return a full name, neatly formatted."""
        if middle_name:
                full_name = f"{first_name} {middle_name} {last_name}"
        else:
                full_name = f"{first_name} {last_name}"
        return full_name.title()


musician = get_formatted_name('jimi', 'hendrix')
print(musician)


musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

```
Jimi Hendrix
John Lee Hooker
```

Trả về một từ điển

- Một hàm có thể trả về bất kỳ loại giá trị nào ta cần, bao gồm cả các cấu trúc dữ liệu phức tạp hơn như danh sách và từ điển.
- Ví dụ, hàm sau nhận các phần của tên và trả về từ điển đại diện cho một người:

```
def build_person(first_name, last_name):
        """Return a dictionary of information about a person. """
        person = {'first': first_name, 'last': last_name}
        return person

musician = build_person('jimi', 'hendrix')
print(musician)

{'first': 'jimi', 'last': 'hendrix'}
```

Trả về một từ điển

- Hàm này nhận thông tin dạng văn bản đơn giản và đưa nó vào một cấu trúc dữ liệu có ý nghĩa hơn cho phép ta làm việc với thông tin ngoài việc in ra. Các chuỗi 'jimi' và 'hendrix' hiện được gắn nhãn là tên và họ.
- Chúng ta có thể dễ dàng mở rộng chức năng này để chấp nhận các giá trị tùy chọn như tên đệm, tuổi, nghề nghiệp hoặc bất kỳ thông tin nào khác mà ta muốn lưu trữ về một người.

```
def build_person(first_name, last_name, age=None):
        """Return a dictionary of information about a person.a"""
        person = {'first': first_name, 'last': last_name}
        if age:
                person['age'] = age
        return person


musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
```

Sử dụng một hàm với vòng lặp while

- Có thể sử dụng các hàm với tất cả các cấu trúc Python mà ta đã học cho đến thời điểm này

```
def get_formatted_name(first_name, last_name):
        """Return a full name, neatly formatted."""
        full_name = f"{first_name} {last_name}"
        return full_name.title()


# This is an infinite loop!
while True:
        print("\nPlease tell me your name:")
        f_name = input("First name: ")
        l_name = input("Last name: ")

        formatted_name = get_formatted_name(f_name, l_name)
        print(f"\nHello, {formatted_name}!")
```

Sử dụng một hàm với vòng lặp while

```
def get_formatted_name(first_name, last_name):
        """Return a full name, neatly formatted."""
        full_name = f"{first_name} {last_name}"
        return full_name.title()
```

```
while True:
        print("\nPlease tell me your name:")
        print("(enter 'q' at any time to quit)")

        f_name = input("First name: ")
        if f_name == 'q':
                        break
        l_name = input("Last name: ")
        if l_name == 'q':
                break

        formatted_name = get_formatted_name(f_name, l_name)
        print(f"\nHello, {formatted_name}!")
```

```
Please tell me your name:
(enter 'q' at any time to quit)
First name: Eric
Last name: matthes

Hello, Eric Matthes!
Please tell me your name:
(enter 'q' at any time to quit)
First name: q
```

7.4. Truyền một danh sách vào hàm

- Giả sử chúng ta có một danh sách người dùng và muốn in lời chào cho mỗi người dùng. Ví dụ sau đây gửi một danh sách các tên đến một hàm có tên gọi là greet_users(), hàm này chào từng người trong danh sách:

```
def greet_users(names):
        """Print a simple greeting to each user in the list. """
        for name in names:
                msg = f"Hello, {name.title()}!"
                print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

Sửa đổi danh sách trong một hàm

- Khi ta truyền một danh sách vào một hàm, hàm này có thể sửa đổi danh sách.
- Mọi thay đổi được thực hiện đối với danh sách bên trong nội dung của hàm là vĩnh viễn, cho phép ta làm việc hiệu quả ngay cả khi ta đang xử lý một lượng lớn dữ liệu

```
# Start with some designs that need to be printed.
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []

# Simulate printing each design, until none are left.
#  Move each design to completed_models after printing.
while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}”)
        completed_models.append(current_design)
```

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case
```

Sửa đổi danh sách trong một hàm

- Có thể tổ chức lại đoạn code này bằng cách viết hai hàm, mỗi hàm thực hiện một công việc cụ thể

Hàm print_models

```
def print_models(unprinted_designs, completed_models):
        """
        Simulate printing each design, until none are left. Move each
design to completed_models after printing.
        """
        while unprinted_designs:
                current_design = unprinted_designs.pop()
                print(f"Printing model: {current_design}")
                completed_models.append(current_design)
```

Sửa đổi danh sách trong một hàm

Hàm show_completed_models

```
def show_completed_models(completed_models):
        """Show all the models that were printed. """
        print("\nThe following models have been printed:")
        for completed_model in completed_models:
                print(completed_model)
```

Phần chương trình chính:

```
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Ngăn một hàm sửa đổi danh sách

- Ngăn một hàm sửa đổi danh sách bằng cách chuyển cho hàm một bản sao của danh sách, không phải bản gốc.

```
function_name(list_name[:])
```

Kí hiệu lát cắt [:] tạo một bản sao của danh sách để gửi đến hàm. Nếu không muốn làm trống danh sách các thiết kế chưa in, chúng ta có thể gọi print_models() như sau:

```
print_models(unprinted_designs[:], completed_models)
```

7.5. Truyền một đối số tùy ý

- Đôi khi ta sẽ không biết trước có bao nhiêu đối số mà một hàm cần chấp nhận.
- Python cho phép một hàm thu thập một số đối số tùy ý từ câu lệnh gọi

```
def make_pizza(*toppings):
        """Print the list of toppings that have been requested."""
        print(toppings)


make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

*Dấu hoa thị trong tham số * toppings cho Python biết tạo một bộ giá trị trống được gọi là toppings và đóng gói bất kỳ giá trị nào nó nhận được vào bộ này.*

Lời gọi print() trong thân hàm tạo ra kết quả cho thấy Python có thể xử lý một lệnh gọi hàm với một giá trị và một lệnh gọi có ba giá trị.

```
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
```

Truyền một đối số tùy ý

- Thay thế lệnh gọi print() bằng một vòng lặp chạy qua danh sách các topping và mô tả bánh pizza đang được đặt hàng

```
def make_pizza(*toppings):
        """Summarize the pizza we are about to make."""
        print("\nMaking a pizza with the following toppings:")
        for topping in toppings:
                print(f"- {topping}")
make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
Making a pizza with the following toppings:
- pepperoni

Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Cú pháp này hoạt động bất kể hàm nhận được bao nhiêu đối số.

Kết hợp đối số vị trí và tùy ý

- Nếu muốn một hàm chấp nhận một số loại đối số khác nhau, thì tham số chấp nhận một số đối số tùy ý phải được đặt cuối cùng trong định nghĩa hàm.
- Python khớp các đối số vị trí và từ khóa trước rồi thu thập bất kỳ đối số nào còn lại trong tham số cuối cùng.

```
def make_pizza(size, *toppings):
        """Summarize the pizza we are about to make."""
        print(f"\nMaking a {size}-inch pizza with the
following toppings:")
        for topping in toppings:
                print(f"- {topping}")


make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

```
Making a 16-inch pizza with the
following toppings:
- pepperoni

Making a 12-inch pizza with the
following toppings:
- mushrooms
- green peppers
- extra cheese
```

Sử dụng đối số từ khóa tùy ý

- Đôi khi ta muốn chấp nhận một số lượng đối số tùy ý, nhưng ta sẽ không biết trước loại thông tin nào sẽ được chuyển đến hàm. Trong trường hợp này, ta có thể viết các hàm chấp nhận nhiều cặp khóa-giá trị như câu lệnh gọi cung cấp.

```
def build_profile(first, last, **user_info):
        """Build a dictionary containing everything we know about a user."""
        user_info['first_name'] = first
        user_info['last_name'] = last
        return user_info


user_profile = build_profile('albert', 'einstein',
                                        location='princeton',
                                        field='physics')
print(user_profile)
```

Sử dụng đối số từ khóa tùy ý

- Trong phần nội dung của build_profile(), chúng ta thêm họ và tên vào từ điển user_info vì chúng ta sẽ luôn nhận được hai phần thông tin này từ người dùng và chúng chưa được đưa vào từ điển. Sau đó, chúng ta trả lại từ điển user_info cho dòng gọi hàm.
- Gọi build_profile(), truyền cho nó tên là 'albert', họ 'einstein' và hai cặp khóa-giá trị là location = 'Princeton' và field = 'Physics'. Gán hồ sơ đã trả về cho user_profile và in user_profile:

```
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert',
'last_name': 'einstein'}
```

Từ điển trả về chứa họ và tên của người dùng, trong trường hợp này là cả vị trí và lĩnh vực nghiên cứu. Hàm sẽ hoạt động bất kể có bao nhiêu cặp khóa-giá trị bổ sung được cung cấp trong lệnh gọi hàm.

7.6. Lưu trữ hàm trong module

- Một ưu điểm của các hàm là cách chúng tách các khối mã nguồn khỏi chương trình chính. Bằng cách sử dụng tên mô tả cho các hàm, chương trình chính sẽ dễ theo dõi hơn nhiều.
- Ta có thể lưu trữ các hàm của mình trong một tệp riêng được gọi là mô-đun và sau đó nhập mô-đun đó vào chương trình chính. Một câu lệnh nhập (import) yêu cầu Python làm cho code trong một mô-đun có sẵn trong tệp chương trình hiện đang chạy.
- Việc lưu trữ các hàm trong một tệp riêng biệt cho phép ẩn các chi tiết của code chương trình và tập trung vào logic cấp cao hơn của nó. Nó cũng cho phép sử dụng lại các hàm trong nhiều chương trình khác nhau.
- Khi ta lưu trữ các hàm của mình trong các tệp riêng biệt, ta có thể chia sẻ các tệp đó với các lập trình viên khác mà không cần phải chia sẻ toàn bộ chương trình của mình. Biết cách import các hàm cũng cho phép ta sử dụng thư viện các hàm mà các lập trình viên khác đã viết.

Import toàn bộ module

- Để bắt đầu import các hàm, trước tiên chúng ta cần tạo một mô-đun.
- Mô-đun là một tệp kết thúc bằng .py có chứa code muốn import vào chương trình chính.

pizza.py

```
def make_pizza(size, *toppings):
        """Summarize the pizza we are about to make."""
        print(f"\nMaking a {size}-inch pizza with the following toppings:")
        for topping in toppings:
                print(f"- {topping}")
```

make_pizzas.py

```
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Import toàn bộ module

- Để gọi một hàm từ một mô-đun đã import, hãy nhập tên của mô-đun ta đã import, pizza, theo sau là tên của hàm, make_pizza(), được phân tách bằng dấu chấm.

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Nếu ta sử dụng loại câu lệnh import này để import toàn bộ mô-đun có tên module_name.py, mỗi hàm trong mô-đun có sẵn thông qua cú pháp sau:

module_name.function_name()

Ví dụ: pizza.make_pizza(16, 'pepperoni')

Import các hàm cụ thể

- Có thể import một hàm cụ thể từ một mô-đun. Đây là cú pháp chung cho cách tiếp cận này:

```
from module_name import function_name
```

Có thể import bao nhiêu hàm tùy thích từ một mô-đun bằng cách phân tách tên của từng hàm bằng dấu phẩy:

```
from module_name import function_0, function_1, function_2
```

```
from pizza import make_pizza

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Sử dụng as để cấp cho hàm một bí danh (Alias)

- Nếu tên của một hàm ta đang import có thể xung đột với tên tồn tại trong chương trình đang viết hoặc nếu tên hàm dài, có thể sử dụng một bí danh ngắn, duy nhất — một tên thay thế tương tự như biệt hiệu cho hàm.
- Đặt biệt hiệu đặc biệt này cho hàm khi import hàm.

Cú pháp:

```
from module_name import function_name as fn
```

```
from pizza import make_pizza as mp

mp(16, 'pepperoni')
mp(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Sử dụng as để cấp cho một mô-đun một bí danh

- Có thể cung cấp bí danh cho tên mô-đun. Đặt cho mô-đun một bí danh ngắn, như p cho pizza, cho phép gọi các chức năng của mô-đun nhanh hơn.

```
import pizza as p


p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Mô-đun Pizza được đặt bí danh p trong câu lệnh import, nhưng tất cả các chức năng của mô-đun vẫn giữ nguyên tên ban đầu.

Cú pháp:

import module_name as mn

Import tất cả các hàm trong module

- Có thể yêu cầu Python import mọi hàm trong một mô-đun bằng cách sử dụng toán tử dấu hoa thị (*):

```
from pizza import *
make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Dấu hoa thị trong câu lệnh import yêu cầu Python sao chép mọi hàm từ mô-đun pizza vào tệp chương trình này. Vì mọi hàm đều được import nên ta có thể gọi từng hàm theo tên mà không cần sử dụng ký hiệu dấu chấm.

Kết chương

- Trong chương này, chúng ta đã học cách viết hàm và truyền đối số để các hàm có quyền truy cập vào thông tin cần thiết để thực hiện công việc của hàm. Ta đã học cách sử dụng các đối số vị trí và từ khóa cũng như cách để chấp nhận một số lượng đối số tùy ý. Ta đã thấy các hàm hiển thị đầu ra và các hàm trả về giá trị. Chúng ta đã học cách sử dụng các hàm với danh sách, từ điển, câu lệnh if và vòng lặp while. Bạn cũng đã thấy cách lưu trữ các hàm đang viết trong các tệp riêng biệt được gọi là mô-đun, do đó, chương trình sẽ sẽ đơn giản và dễ hiểu hơn.
- Một trong những mục tiêu của một lập trình viên là viết mã đơn giản làm những gì bạn muốn và các chức năng giúp bạn làm điều này. Hàm cho phép bạn viết các khối mã và để chúng một mình và biết chúng hoạt động. Khi nào chúng ta biết một chức năng thực hiện đúng công việc của nó, ta có thể tin tưởng rằng và ta có thể chuyển sang một công việc tiếp theo trong dự án
- Các hàm cho phép ta viết mã một lần và sau đó sử dụng lại mã đó bất kỳ lúc nào ta muốn. Khi ta cần chạy mã trong một hàm, tất cả những gì cần làm là viết một lệnh gọi một dòng và hàm thực hiện công việc của nó. Khi ta cần sửa đổi hành vi của một hàm, ta chỉ phải sửa đổi một khối mã và thay đổi có hiệu lực ở mọi nơi đã thực hiện lời gọi đến hàm đó.

Bài tập chương 7

7-1. Message: Viết một hàm có tên display_message() in ra một câu nói cho mọi người biết ta đang học gì trong chương này. Gọi và đảm bảo rằng thông báo hiển thị chính xác.
7-2. Favorite Book: Viết một hàm được gọi là favourite_book() chấp nhận một tham số, tiêu đề. Hàm sẽ in một tin nhắn, chẳng hạn như Một trong những cuốn sách yêu thích là Alice in Wonderland. Gọi hàm, đảm bảo bao gồm tên sách làm đối số trong lệnh gọi hàm
7-3. T-Shirt: Viết một hàm có tên make_shirt() chấp nhận kích thước và nội dung của thông điệp sẽ được in trên áo. Hàm cần in câu tóm tắt size áo và thông điệp in trên đó. Gọi hàm lần thứ nhất bằng cách sử dụng các đối số vị trí để tạo một cái áo. Gọi hàm lần thứ hai bằng cách sử dụng các đối số từ khóa.
7-4. Large Shirts: Sửa đổi hàm make_shirt() để áo sơ mi có kích thước lớn theo mặc định với thông điệp có nội dung I love Python. Tạo một chiếc áo sơ mi lớn và một chiếc áo sơ mi vừa với thông điệp mặc định và một chiếc áo sơ mi có kích thước bất kỳ với thông điệp khác.
7-5. Cities: Viết một hàm được gọi là description_city() chấp nhận tên của một thành phố và đất nước của thành phố. Hàm sẽ in ra một câu kiểu như: Reykjavik is in Iceland. Cho tham số country giá trị mặc định. Gọi hàm với ba thành phố khác nhau và ít nhất một lời gọi hàm không sử dụng giá trị mặc định.

Bài tập chương 7 (t)

7-6. City Names: Viết một hàm có tên city_country() lấy tên của một thành phố và quốc gia của nó. Hàm sẽ trả về một chuỗi có định dạng như sau:
"Santiago, Chile"
Gọi hàm của ta với ít nhất ba cặp thành phố-quốc gia và in các giá trị được trả về.
7-7. Album: Viết một hàm có tên make_album() để xây dựng một từ điển mô tả một album nhạc. Hàm sẽ nhận tên nghệ sĩ và tiêu đề album và nó sẽ trả về một từ điển chứa hai phần thông tin này. Sử dụng chức năng này để tạo ba từ điển đại diện cho các album khác nhau. In từng giá trị trả về để cho thấy rằng từ điển đang lưu trữ thông tin album một cách chính xác. Sử dụng None có để thêm một tham số tùy chọn vào make_album() cho phép lưu trữ số lượng bài hát trong một album. Nếu gọi hàm bao gồm một giá trị cho số lượng bài hát, hãy thêm giá trị đó vào từ điển của album. Thực hiện ít nhất một lệnh gọi chức năng mới bao gồm số lượng bài hát trong album
7-8. User Albums: Bắt đầu với chương trình tại bài 7-7. Xây dựng vòng lặp while cho phép nhập tác giả và tiêu đề của album nhạc. Sau khi ta có thông tin đó, hãy gọi make_album() với thông tin nhập của người dùng và in từ điển đã tạo. Đảm bảo bao gồm một giá trị thoát trong vòng lặp while.
7-9. Messages: Tạo danh sách chứa một loạt tin nhắn văn bản ngắn. Chuyển danh sách đến một hàm có tên show_messages(), hàm này sẽ in từng tin nhắn văn bản.
7-10. Sending Messages: Bắt đầu với chương trình trong bài 7-9. Viết một hàm tên là send_message() in ra thông điệp chuỗi và chuyển các thông điệp đã in ra vào danh sách sent_messages. Sau khi gọi hàm, hãy in cả hai danh sách của ta để đảm bảo rằng các tin nhắn đã di chuyển một cách chính xác.
7-11. Archived Messages: Bắt đầu với chương trình trong bài 7-10. Gọi hàm send_messages() với một bản sao của danh sách messages. Sau khi gọi hàm, hãy in cả hai danh sách của ta để cho thấy rằng danh sách ban đầu vẫn giữ nguyên tin nhắn.

Bài tập chương 7 (t)

7-12. Sandwiches: Viết một hàm chấp nhận danh sách các mục mà một người muốn có trên bánh mì sandwich. Hàm phải có một tham số thu thập nhiều mục như lời gọi hàm cung cấp và nó sẽ in ra một bản tóm tắt về bánh sandwich đang được đặt hàng. Gọi hàm ba lần, mỗi lần sử dụng một số đối số khác nhau.
7-13. User Profile: Bắt đầu với bản sao của user_profle.py trong bài học. Xây dựng hồ sơ của chính ta bằng cách gọi build_profile(), sử dụng họ và tên của ta và ba cặp khóa-giá trị khác mô tả ta.
7-14. Cars: Viết một hàm lưu thông tin về ô tô trong từ điển. Hàm phải luôn nhận được nhà sản xuất và tên kiểu máy. Sau đó, nó sẽ chấp nhận một số lượng đối số từ khóa tùy ý. Gọi hàm với thông tin bắt buộc và hai cặp tên-giá trị khác, chẳng hạn như màu hoặc tính năng tùy chọn. Hàm của ta sẽ hoạt động cho một lời gọi như sau:
car = make_car('subaru', 'outback', color='blue', tow_package=True)
In từ điển được trả lại để đảm bảo tất cả thông tin đều được lưu trữ một cách chính xác.
7-15. Printing Models: Đặt các hàm cho ví dụ print_models.py trong một tệp riêng có tên là print_functions.py. Viết câu lệnh nhập ở đầu print_models.py và sửa đổi tệp để sử dụng các hàm đã nhập.
7-16. Imports: Sử dụng một chương trình ta đã viết có một chức năng trong đó, hãy lưu trữ chức năng đó trong một tệp riêng biệt. Nhập hàm vào tệp chương trình chính của ta và gọi hàm bằng cách sử dụng từng phương pháp sau:
import module_name
from module_name import function_name
from module_name import function_name as fn
import module_name as mn
from module_name import *