HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Posts and Telecommunications Institute of Technology

# CHƯƠNG 7: HÀM (FUNCTION)

VŨ HOÀI THƯ
CNTT1. PTIT

# NỘI DUNG

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

# ĐỊNH NGHĨA HÀM

- Hàm là một tập hợp các khối code được viết ra nhằm cho việc tái sử dụng code.
- Ví dụ về một hàm đơn giản:

```
1  def greet_user():
2      """Display a simple greeting!"""
3      print("Hello everyone!")
4
5  greet_user()
```

```
Hello everyone!
```

- Từ khóa *def* được sử dụng để khai báo hàm trong Python.
- Bất kỳ dòng thụt lề nào sau định nghĩa tên hàm đều được gọi là thân hàm.
- Phần chú thích văn bản được gọi là *docstring,* nó mô tả những gì mà hàm cần thực hiện.

# TRUYỀN THÔNG TIN VÀO MỘT HÀM

- Thay đổi hàm bằng cách truyền thông tin tên người dùng vào trong dấu ngoặc đơn trong định nghĩa hàm.

```
1  def greet_user(username):
2      """Display a simple greeting!"""
3      print(f"Hello {username.title()}!")
4
5  greet_user("jesse")
```

```
Hello Jesse!
```

- Hàm chấp nhận bất kỳ giá trị nào của tên người dùng được chỉ định khi gọi hàm.

# ĐỐI SỐ VÀ THAM SỐ

- Tham số (parameter) là tên biến được định nghĩa trong hàm

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

=> Hàm *greet_user* có một tham số là *username*

- Đối số (Argument) là giá trị truyền vào khi gọi hàm

```
5 greet_user("jesse")
```

```
Hello Jesse!
```

=> Giá trị của đối số truyền vào là *jesse*

# TRUYỀN THAM SỐ VÀO HÀM

- Một hàm có thể chứa nhiều tham số
- Khi một tham số được khai báo trong hàm, nó chỉ có phạm vi sử dụng trong hàm, không thể thay đổi giá trị của tham số mà tác động ngoài hàm được
- Một lệnh gọi hàm có thể chứa nhiều đối số
- Khi thực hiện gọi hàm, các đối số truyền vào cần phải đúng với thứ tự các tham số đã được định nghĩa trong hàm

# VỊ TRÍ CỦA ĐỐI SỐ

- Khi gọi một hàm trong Python, các đối số trong lệnh gọi hàm phải khớp với các tham số đã định nghĩa trong hàm.

```
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.
```

- Đối số *hamster* được gán cho tham số *animal_type*
- Đối số *harry* được gán cho tham số *pet_name*

# GỌI MỘT HÀM NHIỀU LẦN

- Một hàm có thể được sử dụng để gọi nhiều lần.
- Gọi một hàm nhiều lần là cách làm việc hiệu quả.

```
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', 'Will')
```

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

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

- Python hoạt động thông qua các đối số cung cấp khi gọi hàm và khớp từng đối số với tham số tương ứng trong định nghĩa của hàm.

# ĐỐI SỐ CHỨA TỪ KHÓA

- Đối số chứa từ khóa là một cặp tên – giá trị để truyền vào một hàm.
- Các đối số từ khóa sẽ giúp làm rõ vai trò của từng giá trị trong lệnh gọi hàm

```
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')
```

→

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

- Thứ tự của đối số từ không quan trọng, 2 lệnh gọi hàm sau tương đương:

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

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

→

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

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

# GIÁ TRỊ MẶC ĐỊNH

- Khi viết một hàm, có thể xác định một giá trị mặc định cho mỗi tham số.
- Nếu đối số được truyền trong lệnh gọi hàm, Python sẽ sử dụng giá trị của đối số, nếu không, Python sẽ sử dụng giá trị mặc định của tham số trong định nghĩa hàm.

```
def describe_pet(pet_name, animal_type = 'hamster'):
    """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 = 'harry')
```

=>

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

# GIÁ TRỊ MẶC ĐỊNH

- Khi sử dụng giá trị mặc định, tất cả các tham số có giá trị mặc định phải được khai báo phía sau các tham số không có giá trị mặc định.

```
def describe_pet(animal_type = 'hamster', 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()}.")
```

Lỗi cú pháp

```
File "C:\Users\HoaiT\AppData\Local\Temp/ipykernel_12256/474175083.py", line 1
  def describe_pet(animal_type = 'hamster', pet_name):
                                                    ^
SyntaxError: non-default argument follows default argument
```

# LỆNH GỌI HÀM TƯƠNG ĐƯƠNG

- Vị trí của đối số, đối số chứa từ khóa và giá trị mặc định đều có thể được sử dụng cùng nhau, nên một số cách gọi hàm sau đây là tương đương:

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

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

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



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

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

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

# TRÁNH LỖI KHI GỌI HÀM

- Khi thực hiện gọi hàm, lỗi thường gặp khi đối số không khớp với tham số trong phần định nghĩa hàm. Các đối số không khớp khi cung cấp thiếu hoặc thừa so với phần đã định nghĩa trước đó.

```
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()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12256/773814891.py in <module>
      4     print(f"My {animal_type}'s name is {pet_name.title()}.")
      5
----> 6 describe_pet()

TypeError: describe_pet() missing 2 required positional arguments: 'animal_type' and 'pet_name'
```

# TRÁNH LỖI KHI GỌI HÀM

- Python hữu ích trong việc đọc mã và chỉ ra các đối số cần phải cung cấp trong lệnh gọi hàm.
- Nếu cung cấp quá nhiều đối số so với định nghĩa hàm, Python cũng sẽ đưa ra một cảnh báo lỗi tương tự

```
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', 'peter')
```

```
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12256/412407265.py in <module>
      4     print(f"My {animal_type}'s name is {pet_name.title()}.")
      5
----> 6 describe_pet('hamster', 'harry', 'peter')

TypeError: describe_pet() takes 2 positional arguments but 3 were given
```

# BÀI TẬP

- Bài tập 1: Viết một hàm có tên là ***description_city()*** nhận hai tham số là tên quốc gia (country) và thành phố (city) của nó.
  - ✓ Nội dung thân hàm cần in ra một câu đơn giản.

    Ví dụ: “Hà Nội là thành phố của Việt Nam”.
  - ✓ Cung cấp một giá trị mặc định cho tham số quốc gia (country). Thực hiện gọi hàm với 3 thành phố khác nhau, và một trong số đó không thuộc quốc gia mặc định.

# GIÁ TRỊ TRẢ VỀ

- Không phải lúc nào, hàm cũng hiển thị trực tiếp đầu ra của nó
- Thay vào đó, hàm 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 return nhận một giá trị bên trong một hàm và trả về kết quả nơi mà hàm được gọi.
- Giá trị trả về cho phép chuyển phần lớn các khối chương trình phức tạp thành các hàm, từ đó giúp đơn giản hóa nội dung của chương trình.

# TRẢ VỀ MỘT GIÁ TRỊ ĐƠN GIẢN

- Khi gọi một hàm có giá trị trả về, cần gán một biến cho kết quả trả về đó. Trong ví dụ dưới đây, kết quả trả về được gán cho biến *musician*

```
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
```

# LÀM VIỆC VỚI ĐỐI SỐ TÙY CHỌN

- Đôi khi, có thể đặt một đối số là tùy chọn để người sử dụng có thể truyền thông tin khi cần.
- Có thể sử dụng giá trị mặc định để làm cho một đối số là tùy chọn.

```python
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 DICTIONARY

- Một hàm có thể trả về bất kì kiểu dữ liệu nào, bao gồm cả các cấu trúc dữ liệu phức tạp hơn như danh sách hay từ điển.

```python
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 DICTIONARY

- Có thể dễ dàng mở rộng hàm bằng cách thêm các tham số tùy chọn khác nhau vào hàm.

```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    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 HÀM VỚI VÒNG LẶP WHILE

- Hàm có thể được sử dụng với tất cả các cấu trúc đã học trong Python: cấu trúc điều kiện, vòng lặp,…

```
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}!")
```

```
Please tell me your name:
First name: Nguyen
Last name: Van Son

Hello, Nguyen Van Son!
```

# SỬ DỤNG HÀM VỚI VÒNG LẶP WHILE

- Cần xác định điều kiện dừng cho vòng lặp. Và từ khóa *break* cung cấp một cách đơn giản để thoát khỏi vòng lặp khi thỏa mãn điều kiện dừng.

```
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: Nguyen
Last name: Van Son

Hello, Nguyen Van Son!

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

# BÀI TẬP

- Bài tập 2: Viết một hàm nhận đầu vào là giá trị của ba số bất kỳ. Kết quả trả về là giá trị lớn nhất trong ba số đó.

- Bài tập 3: Viết một hàm nhận đầu vào là một số nguyên không âm và kết quả trả về là giai thừa của số nguyên đó.

# BÀI TẬP

- Bài tập 4: Viết một hàm nhận đầu vào là một chuỗi và đếm số lượng các chữ hoa và chữ thường của chuỗi đó.

- Bài tập 5: Định nghĩa một hàm có tên là ***distance_from_zero***, với một đối số bất kỳ. Nếu kiểu của đối số là int hoặc float, thì hàm sẽ trả về giá trị tuyệt đối của giá trị đầu vào. Nếu không, hàm sẽ trả về "Nope".

# BÀI TẬP

- Bài tập 6: Định nghĩa một hàm có tên là ***check_perfect_number(),*** nhận đầu vào là một số nguyên dương. Viết hàm để kiểm tra số đó có phải số hoàn hảo hay không?

  ***Lưu ý***: Số hoàn hảo (hay số hoàn thiện) là một số nguyên dương có tổng các ước (ngoại trừ số đó) bằng chính nó.

  ***Ví dụ***: 6 là một số hoàn hảo, vì có các ước (ngoại trừ 6) là 1, 2, 3 và có tổng 1+2+3=6