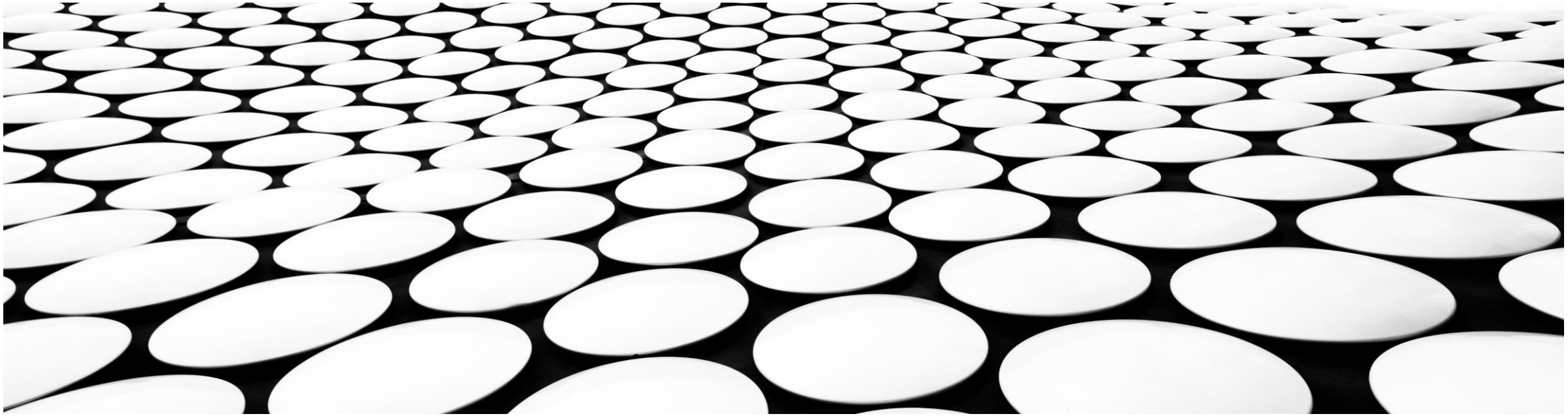
KĨ THUẬT LẬP TRÌNH PYTHON

NGUYỄN MẠNH HÙNG



CHƯƠNG 5: HÀM

HÀM LÀ GÌ?

- Hàm là một nhóm các câu lệnh, được đặt tên, để thực thi một công việc nào đó, và có thể thực thi nhiều lần trong một chương trình.
- Việc định nghĩa và sử dụng hàm là một cách thay thế cho việc ***copy-and-paste*** khi lập trình, nhằm tránh việc lặp lại những đoạn code giống nhau.
- Bằng cách khai báo hàm, khối lượng công việc sẽ giảm đi đáng kể. Đặc biệt nếu muốn thay đổi việc thực thi công việc sau đó, ta chỉ việc thay đổi trong khai báo hàm mà không cần phải xem lại cả chương trình.

- **Ví dụ** về **hàm tích hợp** sẵn trong Python:

  open(‘*tên_file*’): **nhận** ‘*tên_file*’ và **trả về** một *đối tượng file*.

  len(<*biến dữ liệu*>): **nhận** một <*biến dữ liệu*> và **trả về** một *số*.

- Trong bài giảng, ta sẽ tìm hiểu cách **viết một hàm** trong Python được tiến hành như thế nào.

CÁC THÀNH PHẦN CHÍNH CỦA MỘT HÀM TRONG PYTHON

Table 16-1. Function-related statements and expressions

Statement or expression	Examples
Call expressions	`myfunc('spam', 'eggs', meat=ham, *rest)`
def	`def printer(messge):` `  print('Hello ' + message)`
return	`def adder(a, b=1, *c):` `  return a + b + c[0]`

global	`x = 'old'`
	`def changer():`
	`  global x; x = 'new'`

nonlocal (3.X)	`def outer():`
	`  x = 'old'`
	`  def changer():`
	`  nonlocal x; x = 'new'`

yield	`def squares(x):` `  for i in range(x): yield i ** 2`
lambda	`funcs = [lambda x: x**2, lambda x: x**3]`

CÂU LỆNH DEF

- Câu lệnh **def** khởi tạo một đối tượng hàm và gán cho nó một tên.
- Khai báo hàm: **def** *tên_hàm*(arg1, arg2, ..., argN)**:**

  <khối_lệnh>
- Khai báo hàm với câu lệnh **return**: **def** *tên_hàm*(arg1, arg2, ..., argN)**:**

  <khối_lệnh>

  return *giá_trị*

- Dòng tiêu đề của câu lệnh **def** xác định *tên_hàm* được gán cho đối tượng hàm, cùng với một danh sách có hoặc không có các đối số/tham số (arguments/parameters) được viết trong dấu ngoặc đơn. Giá trị tham số sẽ được truyền cho đối tượng hàm tại thời điểm hàm được gọi.

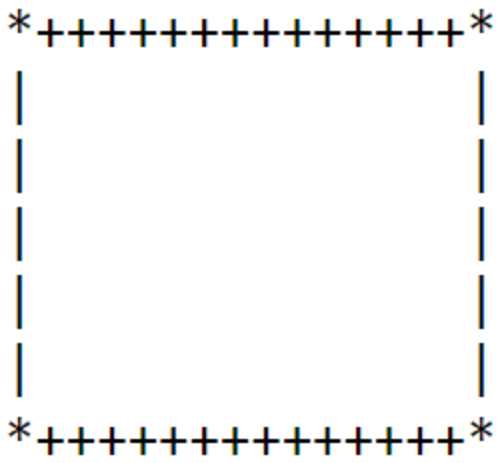
- Câu lệnh **return** có thể xuất hiện ở bất kì đâu trong thân hàm; khi luồng thực thi chạy đến câu lệnh **return**, lệnh gọi hàm kết thúc và trả về *giá_trị*. Nếu không có *giá_trị* trả về, **return** trả về giá trị **None**.

- **def** được xem như một câu lệnh, và <*khối_lệnh*> bên trong phần thân hàm sẽ chưa được thực thi cho đến khi hàm được gọi.

- Để gọi hàm, ta có thể viết tên hàm, sau đó là dấu ngoặc đơn, không chứa hoặc chứa tham số được viết trong phần tiêu đề của hàm.

VÍ DỤ 1: ĐỊNH NGHĨA HÀM

- Viết một hàm vẽ một hình chữ nhật như sau:



```
# Ví dụ 1: viết hàm in ra hình vuông
def rectangle():
    st='*'
    stv='|'
    sth='+'
    print(st.ljust(15,sth)+st)
    for i in range(5):
        print(stv.ljust(15,' ')+stv)
    print(st.ljust(15,sth)+st)
rectangle()
```

VÍ DỤ 2: HÀM TRẢ VỀ MỘT GIÁ TRỊ

- Định nghĩa một hàm nhận hai chuỗi, và trả về tập các kí tự xuất hiện trong cả hai chuỗi đó. Ví dụ:

```
>> st1 = ‘Python’ ; st2 = ‘Pyramid’

>> char_in_common(st1,st2)

>> [ ‘P’, ‘y’ ]
```

```python
# Ví dụ 2: viết hàm xác định
# các kí tự chung giữa hai chuỗi
def char_in_common(st1,st2):
    lst = []
    for x in st1:
        if x not in lst and x in st2:
            lst.append(x)
    return lst
st1='Python'
st2='Pyramid'
char_in_common(st1,st2)
```

VÍ DỤ 3: HÀM TRẢ VỀ NHIỀU GIÁ TRỊ

- Định nghĩa một hàm nhận một mảng số liệu, tính và trả về giá trị trung bình và độ lệch tiêu chuẩn của dữ liệu. Ví dụ:

```
>> data = [1,2,3,4,5,6,7,8,9]

>> summary(data)

>> (5, 2.738612788)
```

```python
# VD3: viết một hàm tính trung bình
# và độ lệch tiêu chuẩn của dữ liệu
def summary(x):
    n = len(x)
    if n==0:
        print("Không có số liệu!")
        return
    elif n==1:
        xtb = x[0]
        sx = 0
    else:
        sumx = sum(x)
        sumxx = 0
        xtb = sumx/n
        for e in x: sumxx += e**2
        sx = math.sqrt(1/(n-1)*(sumxx-n*xtb**2))
    return xtb,sx
```

KHAI BÁO ĐỐI SỐ/THAM SỐ

- Khi gọi hàm nhận một số giá trị để xử lí, ta cần truyền các giá trị này như là đối số của hàm.
- Ví dụ: **char_in_common**(*st1*,*st2*), **summary**(*data*)
- **Giá trị tham số mặc đinh**: tham số được khai báo dưới dạng "**arg** = **value**", thì ta có lựa chọn không truyền giá trị cho tham số này khi gọi hàm. Giá trị đó gọi là giá trị mặc định.

VÍ DỤ 4: GIÁ TRỊ MẶC ĐỊNH CỦA THAM SỐ

- Viết một hàm in ra một chuỗi con gồm k kí tự đầu tiên, số k được truyền vào một tham số khi gọi hàm với giá trị mặc định là k_0. Ví dụ:

```
>> message = ‘default value of an argument’

>> display(message)

>> ‘default’

>> display(message,13)

>> ‘default value’
```

```
def display(message,truncate_after=7):
    print(message[:truncate_after])
message='default value of an argument'
display(message)
display(message,13)
```

```
default
default value
```

DANH SÁCH THAM SỐ (SỐ LƯỢNG THAY ĐỔI)

- Python cho phép khai báo một tham số đặc biệt, là một danh sách gồm các tham số với số lượng tùy ý.
- **Cú pháp**: **def** *function*(first, second, ***remaining**):

  statements

- Khi gọi hàm, ta phải truyền giá trị cho hai tham số đầu tiên. Tuy nhiên, vì tham số thứ 3 được đánh **dấu sao** (*) nên các tham số thực tế đứng sau hai tham số đầu tiên sẽ được gom vào một tuple và được đặt là *remaining*.

- **Ví dụ**:

```
def print_tail(first,*tail):
    print(tail)
print_tail(1,5,2,'omega')
```

```
(5, 2, 'omega')
```

BIẾN ĐỊA PHƯƠNG

- Các biến xuất hiện bên trong thân của hàm được mặc định là biến địa phương (local variable).
- Các biến địa phương xuất hiện khi hàm được gọi, và biến mất khi thoát khỏi hàm. Câu lệnh **return** trả về kết quả, nhưng tên biến bị xóa.
- **Ví dụ**:

  lst, **x** là biến địa phương

```
def char_in_common(st1,st2):
    lst = []
    for x in st1:
        if x not in lst and x in st2:
            lst.append(x)
    return lst
```

BIỂU THỨC LAMBDA

- Biểu thức **lambda** khởi tạo một đối tượng hàm, nhưng không gán tên cho nó.
- **Cú pháp**: **lambda** *arg1*, *arg2*,..., *argN* **:** *expression*
- **Ví dụ**: Hàm lấy tổng có thể viết sử dụng **def** hoặc **lambda**

```
def add(a,b):
    return a+b
add(2,3)
```

```
5
```

```
f = lambda a, b: a+b
f(2,3)
```

```
5
```

Chú ý:

- **lambda là biểu thức, không phải câu lệnh**. Do đó, **lambda** có thể xuất hiện ở những chỗ mà câu lệnh **def** không được phép xuất hiện, chẳng hạn bên trong một danh sách, hay là một tham số bên trong hàm gọi.

```
{'nam': 6.0, 'quỳnh': 7.0, 'quyền': 6.5, 'khoa': 8.0}
```

```
lst=sorted(tudien.items(),key=lambda x: x[1])
dict(lst)
```

```
{'nam': 6.0, 'quyền': 6.5, 'quỳnh': 7.0, 'khoa': 8.0}
```

- **Thân của lambda chỉ chứa một biểu thức duy nhất, thay vì một khối lệnh**. Do đó, **lambda** được thiết kế để viết những hàm đơn giản, còn **def** xử lí các tác vụ phức tạp hơn.

CÂU LỆNH YIELD

- Để thiết lập các vòng lặp, ta sử dụng khái niệm **iterator**. Đó là một đối tượng cho phép duyệt qua từng phần tử của một danh sách, mà không cần khai báo toàn bộ danh sách một lúc, do đó cho phép xử lý những danh sách cỡ lớn mà không sử dụng nhiều ô nhớ.
- **Ví dụ**:

```
for i in range(4): print(i)
```

- Có thể tạo ra **iterator**, bằng cách định nghĩa **hàm sinh** (**generator function**) với câu lệnh **yield**.

- **Ví dụ**: Xét hàm thực hiện nối hai danh sách

```
def concat1(a,b):
    return a+b

a = [1,5,'a']
b = ['b',2,-6]
concat1(a,b)
```

```
[1, 5, 'a', 'b', 2, -6]
```

```
def concat2(a, b) :
    for i in a :
        yield i
    for i in b :
        yield i

concat2(a,b)
```

```
<generator object concat2
```

- **Chú ý**: dùng lệnh **yield**

  thay cho **return**

```
print([i for i in concat2(a,b)])
```

```
[1, 5, 'a', 'b', 2, -6]
```

HÀM ĐỆ QUY

- Python hỗ trợ các hàm đệ quy, là **hàm gọi chính nó** để thực hiện phép lặp.
- Đệ quy là một kỹ thuật hữu ích, cho phép thực thi các giải thuật có cấu trúc và độ sâu tùy ý, không thể đoán trước – chẳng hạn lập kế hoạch tuyến đường du lịch, phân tích ngôn ngữ và thu thập thông tin liên kết trên Web.
- Đệ quy là một giải pháp thay thế cho các vòng lặp đơn giản, mặc dù không nhất thiết là đơn giản nhất hoặc hiệu quả nhất.

VÍ DỤ 5: HÀM TỔNG

- Viết một hàm đệ quy tính tổng các phần tử của một danh sách.

```
>> numbers = [1, 2, 4, 7]

>> mysum(numbers)

>> 14
```

```
def mysum(lst):
    if len(lst)==0:
        return 0
    else:
        return lst[0] + mysum(lst[1:])

numbers = [1,2,4,7]
mysum(numbers)
```

```
14
```

```
def mysum(lst):
    if len(lst)==0:
        return 0
    else:
        print(lst)
        return lst[0] + mysum(lst[1:])

numbers = [1,2,4,7]
mysum(numbers)
```

In ra danh sách tại mỗi vòng đệ quy

```
[1, 2, 4, 7]
[2, 4, 7]
[4, 7]
[7]

14
```

VÒNG LẶP VÀ ĐỆ QUY

- Đệ quy không được thường xuyên sử dụng trong Python vì tốn bộ nhớ.
- Trong nhiều trường hợp, có thể sử dụng các vòng lặp **while** hay **for** để làm cho chương trình đơn giản và rõ ràng hơn.

```
s = 0
for num in numbers:
    s += num
s
```

```
14
```

```
s = 0
while numbers:
    s +=numbers[0]
    numbers = numbers[1:]
s
```

```
14
```

DỮ LIỆU CÓ CẤU TRÚC BẤT KÌ

- Đệ quy cho phép duyệt qua các dữ liệu có cấu trúc bất kì.
- **Ví dụ**: tính tổng tất cả các số trong danh sách lồng nhau như sau:

[1, [2, [3, 4], 5], 6, [7, 8]]

- Các vòng lặp đơn, hay lồng nhau đều không giải quyết được vấn đề trên. Bởi vì các danh sách lồng nhau với độ sâu thay đổi. Do đó ta không thể biết cần bao nhiêu vòng lặp để code trong trường hợp tổng quát.

```
def sumtree(L):
    tot = 0
    for x in L: # For each item at this level
        if not isinstance(x, list):
            tot += x # Add numbers directly
        else:
            tot += sumtree(x) # Recur for sublists
    return tot

L = [1, [2, [3, 4], 5], 6, [7, 8]] # Arbitrary nesting
print(sumtree(L))
```

```
36
```

- Câu lệnh **isinstance**(*object*, *type*): trả về **True** nếu '*object*' có kiểu '*type*'

BÀI TẬP 1

- Viết một hàm nhận một số nguyên dương n và tính giai thừa (n)!

$$n! = 1.2 \ldots n$$

BÀI TẬP 2

- Viết một hàm giải phương trình bậc 2, các hệ số của phương trình bậc 2 được truyền cho các tham số của hàm.

  Ví dụ: $x^2+2x+3=0$

  ```
  >> solver(1,2,3)

  >> 'Phương trình vô nghiệm'
  ```

BÀI TẬP 3

- Viết hàm mô phỏng quá trình gieo 2 con xúc xắc, hàm nhận một tham số = số lần gieo và trả về một từ điển, có key = tổng số chấm của 2 con xúc xắc, và value = tần số xuất hiện.

Ví dụ: >> Dices(100)

>> {2:10, 3:15, 4:25, ... }

BÀI TẬP 4

- Sử dụng đệ quy để viết một hàm, nhận 2 số nguyên $\boldsymbol{n} > \boldsymbol{0}$, $\boldsymbol{p} \geq \boldsymbol{0}$ và tính lũy thừa $\boldsymbol{n^p}$ theo thuật toán sau đây:

  (a) Nếu $p = 0$: trả về 1.0

  (b) Nếu p lẻ: trả về $n \times \text{fastexp}(n, p-1)$

  (c) Nếu p chẵn:

  tính $t \leftarrow \text{fastexp}(n, \frac{p}{2})$;

  trả về $t \times t$

BÀI TẬP 5

- Sử dụng đệ quy để viết một hàm, nhận một số nguyên $\boldsymbol{n}$ và tính giá trị của số Fibonaci $\boldsymbol{F_n}$ theo thuật toán sau:

  (a) Trường hợp $\boldsymbol{F_0}$: nếu $n = 0$, trả về 0

  (b) Trường hợp $\boldsymbol{F_1}$: nếu $n = 1$, trả về 1

  (c) Trường hợp $\boldsymbol{F_n}$: nếu $n > 1$, trả về $\boldsymbol{F_{n-1} + F_{n-2}}$