



LẬP TRÌNH PYTHON

LÀM QUEN VỚI PYTHON



NỘI DUNG

1. Giới thiệu
2. Biến, biểu thức, lệnh
3. Hàm
4. Cấu trúc điều khiển

GIỚI THIỆU

Giới thiệu ngôn ngữ lập trình Python

- Ngôn ngữ lập trình Python được tác giả Guido van Rossum tạo ra vào những năm cuối thập niên 1980
- Python hướng tới một ngôn ngữ có cú pháp đơn giản nhưng hiệu quả hơn so với các ngôn ngữ thông dụng như Java, C#, C, C++.
- Python đã khẳng định được những ưu điểm của mình như dễ học, dễ đọc, dễ bảo trì, cùng với hệ thống thư viện phong phú, Python dần trở thành một trong những ngôn ngữ phổ biến nhất hiện nay
- Một người mới học lập trình có thể dễ dàng tiếp cận và sử dụng Python để giải quyết các bài toán một cách nhanh chóng so với các ngôn ngữ phức tạp khác.

GIỚI THIỆU

Giới thiệu ngôn ngữ lập trình Python

- Python được sử dụng trong nhiều lĩnh vực như giáo dục, nghiên cứu hay trong công nghiệp.
- Các công ty, tổ chức chuyên về lĩnh vực công nghệ sử dụng Python để phát triển các phần mềm có thể kể đến như: Google, Yahoo, Facebook,
- Tổ chức Nghiên cứu Hạt nhân châu Âu (European Organisation for Nuclear Research - CERN), Industrial Light and Magic, và Cơ quan Hàng không và Vũ trụ Hoa Kỳ (NASA).
- Python hỗ trợ trên đa nền tảng như Microsoft Windows, Mac OS X, Linux. Các thông tin về phiên bản mới nhất của Python có thể xem tại địa chỉ <http://www.python.org>

GIỚI THIỆU

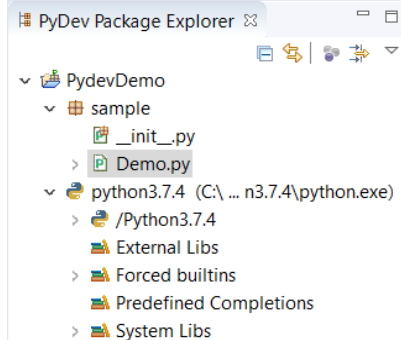
Giới thiệu ngôn ngữ lập trình Python

- Hiện tại Python có 2 phiên bản là Python 2 và Python 3
- Python đang dần chuyển dịch từ Python 2 sang Python 3, ngày càng nhiều các tài nguyên và thư viện cho Python 3 được tạo ra và được sử dụng rộng rãi hơn
- Trong nội dung môn học chúng ta sẽ làm việc với Python 3 ([3.7.4](#))

GIỚI THIỆU

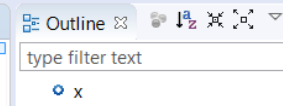
Các công cụ, tiện ích

- Cùng với sự phát triển của ngôn ngữ lập trình Python, ngày càng có nhiều công cụ hỗ trợ việc lập trình và phát triển phần mềm với Python như các IDE (Integrated Development Environment) Pycharm, Wing, Eric, PyDev (Eclipse), Spyder,...
- IDE được lựa chọn sử dụng trong môn học là Eclipse với plugin Pydev.
- Nhìn chung, Eclipse là một IDE khá thông dụng và được sử dụng nhiều, Eclipse có các plugin hỗ trợ nhiều ngôn ngữ như JAVA, C, C++, Python, PHP, JSP,... cùng với đó Eclipse hỗ trợ đa nền tảng, giao diện thân thiện và khá tương đồng trên các hệ điều hành khác nhau.



```
1 print("Hello World")
2 x = 10
3 print(x)
```

Vùng soạn thảo (editor)



Console

PyDev Console [0]

```
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
>>>
C:\Python3.7.4\python.exe 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)]
>>> x = 10
>>> x
10
>>>
```

Vùng hiển thị kết quả
(console - interactive console)

GIỚI THIỆU

Các công cụ, tiện ích

- Việc cài đặt Pydev cũng khá đơn giản theo hướng dẫn trên trang web của Pydev https://www.pydev.org/manual_101_install.html.
- Ngoài ra bạn cũng có thể cài đặt Pydev bằng file zip theo hướng dẫn trên trang web của Pydev https://www.pydev.org/manual_101_install.html.
- Tham khảo thêm hướng dẫn cài đặt và cấu hình Pydev cho Eclipse tại <https://www.rose-hulman.edu/class/csse/resources/Eclipse/eclipse-python-configuration.htm>

GIỚI THIỆU

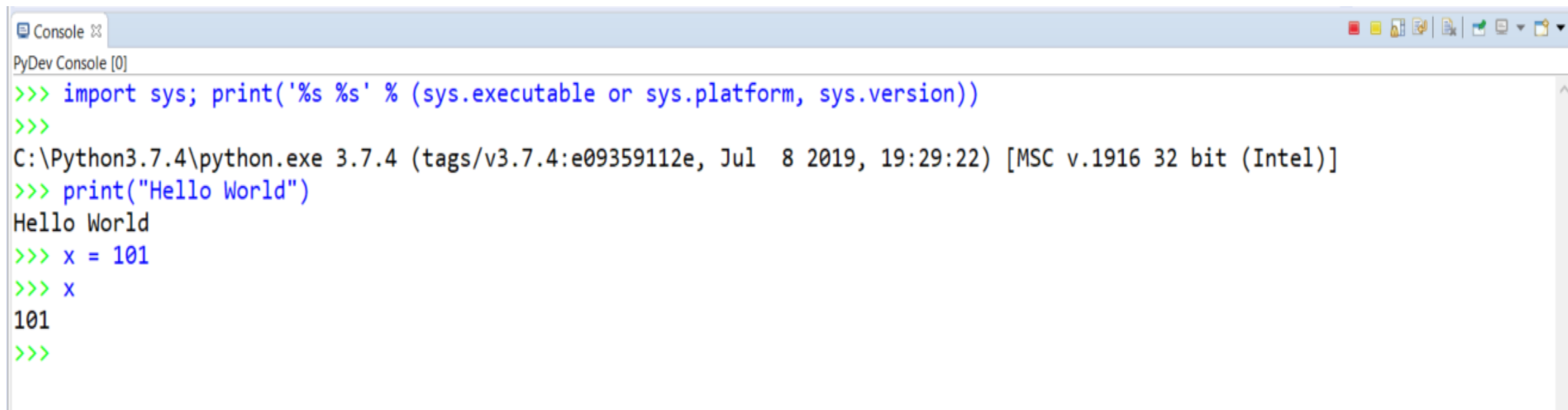
Dịch và chạy chương trình Python

- Để dịch và chạy một chương trình Python, trước hết cần cài đặt Python từ trang chủ <https://www.python.org/downloads/release>.
- Trong môn học này chúng ta sẽ làm việc với phiên bản Python 3.7.4.
- Nếu cài đặt trên hệ điều hành Windows, cần chú ý việc thêm đường dẫn đến thư mục cài đặt Python vào biến môi trường Path.

GIỚI THIỆU

Dịch và chạy chương trình Python

- Có một số cách để chạy một chương trình Python. Cách thứ nhất là sử dụng cửa sổ tương tác trực tiếp (interactive console) của trình thông dịch Python, tại đây các câu lệnh được thông dịch và thực thi trực tiếp.

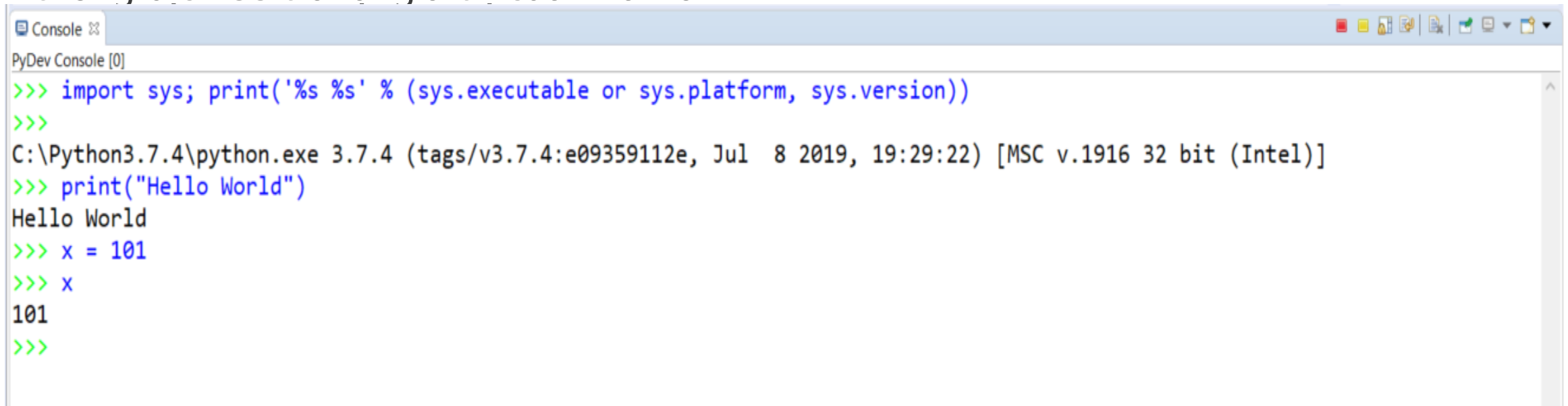
A screenshot of the PyDev Console window in an IDE. The window has a title bar with 'Console' and a close button. Below the title bar, it says 'PyDev Console [0]'. The console contains several lines of Python code and their output. The code starts with an import statement and a print function to show system information. Then, it prints 'Hello World'. Finally, it assigns the value 101 to a variable 'x' and prints 'x', which outputs '101'. The code is color-coded: keywords are blue, strings are red, and numbers are green. The output is in black text.

```
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
>>>
C:\Python3.7.4\python.exe 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)]
>>> print("Hello World")
Hello World
>>> x = 101
>>> x
101
>>>
```

GIỚI THIỆU

Dịch và chạy chương trình Python

- Trình thông dịch sẽ xác định dòng lệnh người dùng nhập vào là biểu thức hay câu lệnh và sẽ thực thi chúng. Nếu người dùng nhập vào $x = 101$, trình thông dịch hiểu nó là một phép gán và nó không trả lại giá trị gì. Nếu người dùng nhập vào x , trình thông dịch sẽ trả lại giá trị của x là 101.

A screenshot of a PyDev Console window. The window has a title bar with 'Console' and a close button. Below the title bar, it says 'PyDev Console [0]'. The console contains several lines of Python code and their outputs. The code starts with an import statement and a print statement for system information. Then, it prints 'Hello World'. Next, it assigns the value 101 to the variable x. Finally, it prints the value of x, which is 101.

```
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
>>>
C:\Python3.7.4\python.exe 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)]
>>> print("Hello World")
Hello World
>>> x = 101
>>> x
101
>>>
```

GIỚI THIỆU

Dịch và chạy chương trình Python

- Cách thứ hai là dịch và chạy cả file mã nguồn Python, việc này có thể thực hiện trực tiếp trên Eclipse hoặc trên cửa sổ dòng lệnh thông qua lệnh **python tên_file.py**

```
D:\eclipse_workspace\PydevDemo\sample>python Demo.py
Hello World
10
D:\eclipse_workspace\PydevDemo\sample>
```

GIỚI THIỆU

Một số đặc điểm của ngôn ngữ Python

- **Học và sử dụng một cách dễ dàng:** Python dễ học và dễ sử dụng bởi vì nó là ngôn ngữ lập trình bậc cao, dễ được tiếp cận bởi người dùng.
- **Ngôn ngữ biểu đạt (Expressive Language):** Ngôn ngữ Python có tính biểu đạt cao có nghĩa là nó dễ đọc và dễ hiểu.
- **Ngôn ngữ thông dịch (Interpreted Language):** Python là một ngôn ngữ thông dịch, trình thông dịch thực thi từng lệnh tại mỗi thời điểm, điều này giúp việc kiểm soát và gỡ lỗi dễ dàng hơn, do đó nó phù hợp với những người mới học lập trình.

GIỚI THIỆU

Một số đặc điểm của ngôn ngữ Python

- **Ngôn ngữ đa nền tảng (Cross-platform Language):** Python hoạt động đồng nhất trên các nền tảng khác nhau như as Windows, Linux, Unix và Macintosh,... Do vậy chúng ta cũng có thể gọi Python là một ngôn ngữ portable (portable language).
- **Miễn phí và mã nguồn mở:** Giấy phép Python Software Foundation, Python được cung cấp miễn phí trên trang chủ <https://www.python.org>.
- **Ngôn ngữ hướng đối tượng (Object-Oriented Language):** Python hỗ trợ ngôn ngữ lập trình hướng đối tượng và đưa ra các khái niệm về các lớp (class) và đối tượng (object).

GIỚI THIỆU

Một số đặc điểm của ngôn ngữ Python

- **Có khả năng mở rộng (Extensible):** Python có thể viết mã chương trình bằng các ngôn ngữ khác như C, C++,... và cũng có thể biên dịch mã chương trình đó trong các ngôn ngữ tương ứng.
- **Thư viện khổng lồ (Large Standard Library):** Python có một hệ thống thư viện khổng lồ và trải rộng trên các lĩnh vực, nó cung cấp một nguồn tài nguyên phong phú giúp việc phát triển các ứng dụng trở nên nhanh chóng.
- **Hỗ trợ lập trình giao diện (GUI Programming Support):** Giao diện người dùng có thể được phát triển bằng Python.

GIỚI THIỆU

Một số đặc điểm của ngôn ngữ Python

- **Khả năng tích hợp (Integrated):** Python có thể dễ dàng tích hợp với các ngôn ngữ khác như C, C++, JAVA,...
- **Ngôn ngữ kiểu dữ liệu động (Dynamically Typed Language):** Kiểu dữ liệu của một biến được quyết định khi thực thi chứ không phải khi khai báo, khởi tạo. Do vậy không cần khai báo kiểu dữ liệu cho các biến trong Python.

BIẾN, BIỂU THỨC, LỆNH

Biến

- Một biến trong Python có thể nhận giá trị số và giá trị không phải là số.
- Ví dụ dưới đây dùng biến x để lưu giá trị số nguyên, và in ra giá trị của biến x
- Chú ý nếu dùng câu lệnh `print('x')` sẽ in ra ký tự x thay vì giá trị của biến x là 10, vì 'x' là biểu diễn của chuỗi ký tự.

```
>>> x = 10
>>> print(x)
10
... |
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Các biến có thể được khai báo và gán giá trị theo bộ (tupe) bằng dấu phẩy ','. Ví dụ
- Chú ý số lượng biến phải bằng số lượng giá trị, nếu không sẽ là một câu lệnh lỗi.

```
>>> x, y, z = 10, 20, 30
>>> x
10
>>> y
20
>>> z
30
>>> |
```

```
>>> x, y, z = 10, 20
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
>>> x, y, z = 10, 20, 30, 40
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: too many values to unpack (expected 3)
>>>
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Hình dưới đây minh họa về việc gán và thay đổi giá trị cho các biến

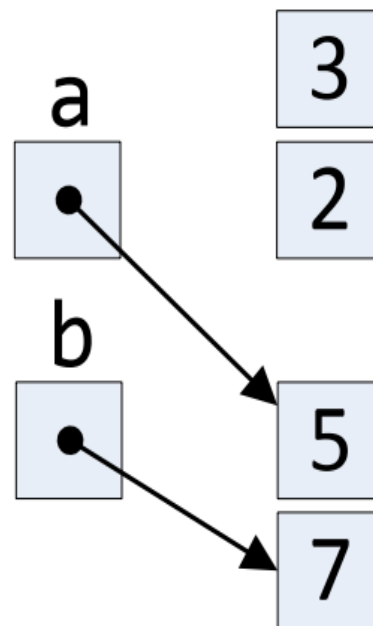
a = 2

b = 5

a = 3

a = b

b = 7



BIẾN, BIỂU THỨC, LỆNH

Biến

- Để lấy ra kiểu giá trị của biến, ta dùng từ khóa `type`
- Python không yêu cầu xác định kiểu dữ liệu cho biến, kiểu dữ liệu của biến chỉ được xác định khi thực thi, do vậy một biến có thể mang các kiểu giá trị khác nhau tại các thời điểm khác nhau.

```
>>> a = 507
>>> print('giá trị của a = ', a, ' kiểu dữ liệu là', type(a))
giá trị của a = 507  kiểu dữ liệu là <class 'int'>
>>> a = 'MIM'
>>> print('giá trị của a = ', a, ' kiểu dữ liệu là', type(a))
giá trị của a = MIM  kiểu dữ liệu là <class 'str'>
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Một biến cần được gán giá trị để thực thi và gọi tới trong các câu lệnh tiếp theo, nếu một biến chưa được gán giá trị và được gọi tới sẽ có thông báo lỗi biến đó chưa được khai báo.

```
>>> x = 503
```

```
>>> x
```

```
503
```

```
>>> y
```

```
Traceback (most recent call last):
```

```
  File "<input>", line 1, in <module>
```

```
NameError: name 'y' is not defined
```

```
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Để xóa một biến khỏi phiên làm việc hiện tại sử dụng từ khóa **del**.

```
>>> a,b,c = 1,'abc',4
>>> a,b,c
(1, 'abc', 4)
>>> del a
>>> a
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'a' is not defined
>>> b
'abc'
>>> c
4
>>> del b, c
>>> b
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'b' is not defined
>>> c
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'c' is not defined
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Chú ý đến quy ước của việc đặt tên biến, việc đặt tên biến, hay các định danh (Identifier) trong Python được quy ước như sau:
 - Một định danh cần chứa ít nhất một ký tự
 - Ký tự đầu tiên của một định danh cần là chữ cái viết hoa hoặc thường hoặc dấu gạch dưới '_' (underscore). Python3.7.4 có thể đặt tên biến bằng ký tự unicode
 - Các ký tự sau ký tự đầu tiên có thể là chữ cái, dấu gạch dưới, hoặc số
 - Các ký tự đặc biệt không được sử dụng để đặt tên cho các định danh như (@, #, !, *, vv)

```
>>> x = 10
>>> y = 20
>>> tổng = x+y
>>> print('tổng = ', tổng)
tổng = 30
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Tên của các định danh không được đặt trùng với các từ khóa trong Python, bảng dưới đây là danh sách các từ khóa trong Python

and	del	from	None	try
as	elif	global	nonlocal	True
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

BIẾN, BIỂU THỨC, LỆNH

Biến

- Trong Python, các định danh được phân biệt chữ viết hoa và chữ viết thường, ví dụ 'for' là một từ khóa và không được sử dụng để đặt tên cho một định danh thì 'FOR', hoặc 'For' không phải là một từ khóa và có thể sử dụng để đặt tên cho một định danh.
- Tương tự như vậy, các định danh có cách viết hoa, thường khác nhau là các định danh khác nhau.

```
>>> FOR = 100
```

```
>>> FOR
```

```
100
```

```
>>> for = 10
```

```
File "<input>", line 1
```

```
for = 10
```

```
^
```

```
SyntaxError: invalid syntax
```

BIẾN, BIỂU THỨC, LỆNH

Biến

- Việc đặt tên cho một định danh, hoặc biến cần thể hiện được mục đích và ý nghĩa của định danh đó, nên tránh việc đặt tên định danh gần tương tự nhau
- Việc đặt tên các định danh theo quy tắc giúp cho mã nguồn chương trình dễ đọc, dễ hiểu hơn, đồng thời cũng tránh được các lỗi do nhầm lẫn và dễ dàng tìm và sửa lỗi hơn

BIẾN, BIỂU THỨC, LỆNH

Nhập dữ liệu (input)

- Dữ liệu nhập từ luồng nhập xuất chuẩn được đọc thông qua lệnh 'input()'

```
>>> print('Nhập vào tên của bạn')
Nhập vào tên của bạn
>>> name = input()
Bá Khá
>>> print('Xin chào bạn ',name)
Xin chào bạn  Bá Khá
>>> print(type(name))
<class 'str'>
>>> print('Nhập vào một giá trị số')
Nhập vào một giá trị số
>>> num = input()
123.6
>>> print('Giá trị số = ',num,' Kiểu dữ liệu',type(num))
Giá trị số =  123.6  Kiểu dữ liệu <class 'str'>
>>> num = float(num)
>>> print('Giá trị số = ',num,' Kiểu dữ liệu',type(num))
Giá trị số =  123.6  Kiểu dữ liệu <class 'float'>
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Nhập dữ liệu (input)

- Kiểu dữ liệu mặc định của dữ liệu được trả về từ input() là kiểu chuỗi ký tự
- Có thể chuyển kiểu dữ liệu chuỗi ký tự sang các kiểu dữ liệu số như int hoặc float.
- Lệnh input còn có thể truyền tham số là chuỗi ký tự, và dữ liệu có thể chuyển đổi trực tiếp sang kiểu mong muốn
- Chú ý việc chuyển đổi dữ liệu (ép kiểu) yêu cầu dữ liệu được nhập đúng định dạng, nếu không sẽ có thông báo lỗi.

```
>>> num = int(input('Nhập vào một số nguyên:'))
```

```
Nhập vào một số nguyên:3
```

```
>>> print('Giá trị của số nguyên vừa nhập là num = ',num)
```

```
Giá trị của số nguyên vừa nhập là num = 3
```

```
>>> num = int(input('Nhập vào một số nguyên: '))
```

```
Nhập vào một số nguyên: 3.14
```

```
Traceback (most recent call last):
```

```
File "<input>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: '3.14'
```

```
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Xuất dữ liệu (print)

- Việc nhập dữ liệu từ luồng nhập chuẩn (standard input), ví dụ từ bàn phím được thực hiện thông qua lệnh **input**.
- Việc xuất dữ liệu ra luồng xuất chuẩn, ví dụ cửa sổ dòng lệnh (console) được thực hiện thông qua lệnh **print**
- Một số định dạng in với lện **print** được thể hiện trong ví dụ

```
>>> print('abc')
```

```
abc
```

```
>>> print('abc',end='')
```

```
abc>>> print('abc',end='\n')
```

```
abc
```

```
>>> print('abc',end='abc')
```

```
abcabc>>> print('',end='abc\n')
```

```
abc
```

BIẾN, BIỂU THỨC, LỆNH

Xuất dữ liệu (print)

- Ví dụ về định dạng chuỗi phân cách giữa các giá trị được in ra trong lệnh print qua tham số sep, giá trị mặc định là sep = ''

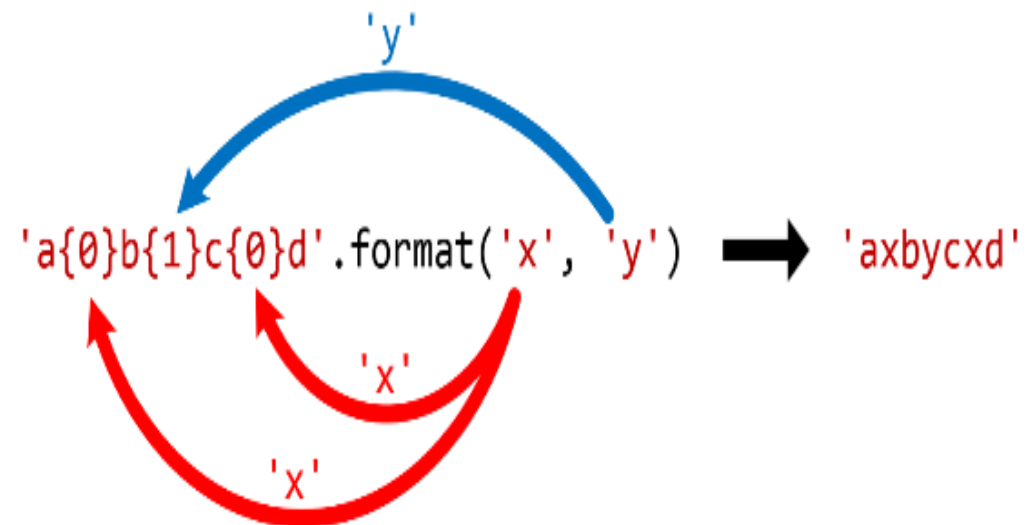
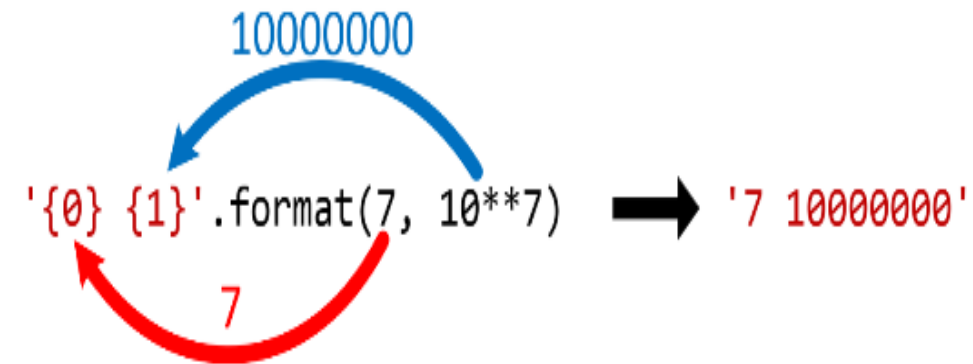
```
>>> w, x, y, z = 10, 15, 20, 25
>>> print(w, x, y, z)
10 15 20 25
>>> print(w, x, y, z, sep=',')
10,15,20,25
>>> print(w, x, y, z, sep='')
10152025
>>> print(w, x, y, z, sep=':')
10:15:20:25
>>> print(w, x, y, z, sep='-----')
10-----15-----20-----25
>>>
```

BIẾN, BIỂU THỨC, LỆNH

Xuất dữ liệu (print)

- Ví dụ về định dạng chuỗi in thông qua hàm format.

```
>>> print(0, 10**0)
0 1
>>> print(0, 10**1)
0 10
>>> print(0, 10**3)
0 1000
>>> print(0, 10**6)
0 1000000
>>> print(0, 10**9)
0 1000000000
>>> print(0, 10**12)
0 1000000000000
>>> print('{0} {1}'.format(0, 10**5))
0 100000
>>> print('{0} {1}'.format(10, 10**10))
10 10000000000
>>> |
```



BIẾN, BIỂU THỨC, LỆNH

Xuất dữ liệu (print)

- Ví dụ về định dạng chuỗi in thông qua hàm format, căn lề bên phải

```
>>> print('{0:>3} {1:>16}'.format(4, 10**4))
4          10000
>>> print('{0:>3} {1:>16}'.format(4, 10**5))
4          100000
>>> print('{0:>3} {1:>16}'.format(4, 10**6))
4          1000000
>>> print('{0:>3} {1:>16}'.format(4, 10**7))
4          10000000
>>> print('{0:>3} {1:>16}'.format(4, 10**8))
4          100000000
>>> print('{0:>3} {1:>16}'.format(4, 10**9))
4          1000000000
>>> print('{0:>3} {1:>16}'.format(4, 10**10))
4          10000000000
```


BIẾN, BIỂU THỨC, LỆNH

Xuất dữ liệu (print)

- Cặp 3 dấu `'''` `'''`, hoặc `"` `"` `"` `"` `"` `"` dùng để đánh dấu chuỗi trên nhiều dòng, xem ví dụ dưới đây.

```
>>> multiline = '''Đây là chuỗi trên nhiều dòng  
... mỗi dòng sẽ được đánh dấu bởi ký tự xuống dòng '\n' '''
```

```
>>> multiline
```

```
"Đây là chuỗi trên nhiều dòng\nmỗi dòng sẽ được đánh dấu bởi ký tự xuống dòng '\n' "
```

```
>>>
```

BIẾN, BIỂU THỨC, LỆNH

Biểu thức và các phép toán

- Một số tự nhiên 34 hay một biến x được xem là một biểu thức. Chúng ta có thể sử dụng các toán tử để kết hợp các giá trị và biến thành các biểu thức phức tạp hơn.
- Bảng dưới đây liệt kê các phép toán cơ bản trong Python

BIẾN, BIỂU THỨC, LỆNH

Biểu thức	Ý nghĩa	Ví dụ
$x + y$	Phép cộng giá trị x với giá trị y nếu x, y là số. Nối y vào x thành xâu xy nếu x và y là xâu ký tự	$4 + 5 \Rightarrow 9$ 'abc' + 'def' \Rightarrow 'abcdef'
$x - y$	Phép trừ x với y nếu x và y là số	$7 - 8 \Rightarrow -1$
$x * y$	Phép nhân x với y nếu x và y là số Nhân x lên y lần nếu x là xâu ký tự và y là số nguyên Nhân y lên x lần nếu x là số nguyên và y là xâu ký tự	$3 * 5 \Rightarrow 15$ "ab" * 2 \Rightarrow "abab" 2 * "ab" \Rightarrow "abab"
x / y	Phép chia x cho y nếu x và y là số	$6 / 3 \Rightarrow 2$
$x // y$	Phép lấy phần nguyên của phép chia x cho y nếu x và y là số	$7 // 3 \Rightarrow 2$
$x \% y$	Phép lấy phần dư của phép chia x cho y nếu x và y là số	$7 \% 3 \Rightarrow 1$
$x ** y$	Phép tính x mũ y nếu x và y là số	$2 ** 3 \Rightarrow 8$

BIẾN, BIỂU THỨC, LỆNH

Biểu thức và các phép toán

- Các toán tử có thể sử dụng kết hợp với phép gán "=" để có phép gán gọn hơn như
 - $x += y$ tương đương với $x = x + y$
- Các biểu thức đều có một giá trị nào đó. Giá trị của biểu thức được tính thông qua các giá trị và giá trị của các biến, kết hợp với các phép toán trên các giá trị để tính được giá trị của các biểu thức phức tạp.
- Các phép toán trong bảng trên là các phép toán hai ngôi (toán tử hai ngôi), bên cạnh đó còn có các toán tử một ngôi như dấu $-$ cho giá trị âm, và dấu $+$ cho các giá trị dương

BIẾN, BIỂU THỨC, LỆNH

Biểu thức và các phép toán

- Các giá trị bị giới hạn bởi kiểu dữ liệu. Ví dụ dưới đây cho thấy biểu thức `2.0 ** 10000` vượt quá giá trị của kiểu số thực

```
>>> 2.0 ** 1000
```

```
1.0715086071862673e+301
```

```
>>> 2.0 ** 10000
```

```
Traceback (most recent call last):
```

```
  File "<input>", line 1, in <module>
```

```
OverflowError: (34, 'Result too large')
```

```
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biểu thức và các phép toán

- Cần chú ý đến sai số trong các giá trị của biểu thức, ví dụ dưới đây cho thấy có sai số (rất nhỏ) trong phép toán $1 - 1/3 - 1/3 - 1/3 \neq 0$
- Các phép tính trên số thập phân vô hạn không tuần hoàn thường sẽ có sai số

```
>>> one = 1.0
>>> ot = 1.0/3.0
>>> z = one - ot - ot - ot
>>> z
1.1102230246251565e-16
>>> 1.0 - 1/3 - 1/3 - 1/3
1.1102230246251565e-16
>>> |
```

BIẾN, BIỂU THỨC, LỆNH

Biểu thức và các phép toán

- Các phép toán có thứ tự ưu tiên khi thực hiện, bảng dưới đây thể hiện độ ưu tiên của các phép toán được sắp xếp giảm dần
- Các phép toán cùng độ ưu tiên thì sẽ thực hiện theo thứ tự từ trái sang phải hoặc từ phải sang trái.
- Dấu # để đánh dấu một dòng ghi chú trong Python.

Số ngôi	Toán tử	Thứ tự	Ví dụ
Hai	**	Phải sang trái	$2 ** 3 ** 2 = 2 ** (3 ** 2) = 2 ** 9 = 512$
Một	+, -		-3, +4
Hai	*, /, //, %	Trái sang phải	$2*3/2 = (2*3)/2 = 3$
Hai	+, -	Trái sang phải	$2+3-4 = (2+3)-4$
Hai	=	Phải sang trái	$x = y = 4$

HÀM

Gọi hàm

- Thư viện chuẩn của Python cung cấp nhiều hàm để thực hiện các chức năng thông dụng trong việc lập trình như hàm tính mũ, tính giá trị tuyệt đối, tính căn,...
- Ví dụ dưới đây là gọi đến lệnh sqrt trong thư viện math
- Trong ví dụ trên, math là một module (hay thư viện) sqrt là một hàm trong module math.

from *module* import *function list*

```
>>> from math import sqrt
>>> x = 9.0
>>> sqrt(x)
3.0
>>> |
```


HÀM

Gọi hàm

- Danh sách dưới đây là các lệnh (hàm) dựng sẵn (builtins) trong Python, các lệnh này được gọi trực tiếp mà không cần import module nào.

```
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',  
'__format__', '__ge__', '__getattr__', '__getitem__', '__gt__', '__hash__',  
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__',  
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',  
'__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items',  
'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

HÀM

Gọi hàm

- Một số hàm toán học cơ bản trong module math

sqrt

Computes the square root of a number: $\text{sqrt}(x) = \sqrt{x}$

exp

Computes e raised a power: $\text{exp}(x) = e^x$

log

Computes the natural logarithm of a number: $\text{log}(x) = \log_e x = \ln x$

log10

Computes the common logarithm of a number: $\text{log}(x) = \log_{10} x$

cos

And other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyper bolic sine, and hyperbolic tangent

pow

Raises one number to a power of another: $\text{pow}(x;y) = x^y$

degrees

Converts a value in radians to degrees: $\text{degrees}(x) = 180 \pi x$

radians

Converts a value in degrees to radians: $\text{radians}(x) = 180 \pi x$

fabs

Computes the absolute value of a number: $\text{fabs}(x) = |x|$

HÀM

Xây dựng hàm

- Một chương trình sẽ dễ dàng tiếp cận, dễ đọc, dễ hiểu và dễ kiểm soát lỗi nếu được tổ chức thành các hàm chức năng một cách hợp lý
- Việc viết các hàm thực hiện các công việc thành phần trong một chương trình giúp tái sử dụng hàm đó với cùng một công việc được thực hiện nhiều lần, để khoanh vùng xác định lỗi và sửa lỗi.

HÀM

Xây dựng hàm

- Mỗi hàm trong Python có hai khía cạnh
 - Định nghĩa hàm (**Function definition**): định nghĩa của một hàm chứa các dòng lệnh xác định các hành vi của hàm đó.
 - Lời gọi hàm (**Function invocation**): Một hàm được sử dụng trong chương trình thông qua lời gọi hàm
- Mỗi hàm được định nghĩa một lần nhưng có thể gọi đến nhiều lần.
- Khuôn mẫu để định nghĩa một hàm như sau:

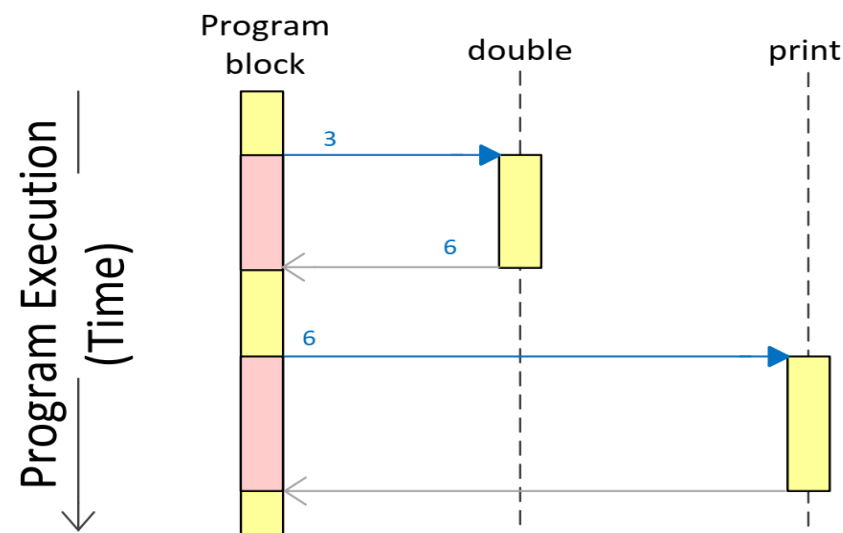
```
def name ( parameter list ) :  
    block
```

HÀM

Xây dựng hàm

- Trong ví dụ trên, từ khóa **def** đánh dấu việc bắt đầu định nghĩa hàm, tên hàm là **double**, đối số là **a**, nội dung hàm là trả lại giá trị bằng hai lần giá trị đối số **a**.
- Lời gọi hàm được thực hiện bằng tên hàm (double) cùng với đối số được truyền vào là x, kết quả lời gọi hàm double(x) được in ra màn hình bằng lệnh print. Luồng thực hiện được thể hiện trong hình minh họa dưới đây
- Từ khóa **return** được dùng để trả lại giá trị cho hàm. Một hàm có thể không có đối số, và có thể không cần trả lại giá trị gì

```
1 def double(a):  
2     return 2*a;  
3  
4 x = 3  
5 print(double(x))
```



HÀM

Đối số

- Khi một hàm được gọi tới với tham số được truyền vào, khi đó biến tham số sẽ tham chiếu đến biến hoặc giá trị được truyền làm tham số
- Các kiểu dữ liệu như integer, float-point và string được xem là các đối tượng bất biến (immutable), giá trị của các đối tượng bất biến không bị thay đổi khi được truyền là tham số của các hàm
- Trong ví dụ trên, kết quả của `double(x)` là 6, nhưng `x` vẫn có giá trị bằng 3

```
1 def double(a):  
2     a = a*2  
3     return a;  
4  
5 x = 3  
6 print(double(x))  
7 print(x)
```

HÀM

Đối số

- Tham số của hàm có thể có giá trị mặc định, giá trị này có thể bỏ qua trong lời gọi hàm, các tham số có giá trị mặc định phải đứng sau các tham số không có giá trị mặc định.
- Trong ví dụ dưới đây, tham số a của hàm grow có giá trị mặc định là 2, khi gọi hàm grow ta có thể bỏ qua tham số a, khi đó a sẽ nhận giá trị mặc định là 2, ta cũng có thể thay giá trị mặc định của a bằng giá trị khác.

```
>>> def grow(b, a= 2):  
...     return a*b;  
... x = 3  
>>> print(grow(x))  
6  
>>> print(x)  
3  
>>> print(grow(x,3))  
9  
>>>
```

HÀM

Biến toàn cục

- Biến được định nghĩa bên trong hàm được gọi là biến cục bộ (local variable). Biến cục bộ được lưu trong bộ nhớ khi hàm được gọi đến và thực thi, khi kết thúc hàm hoặc biến không còn được sử dụng tới trong hàm thì bộ nhớ dùng để lưu trữ biến sẽ được giải phóng. Cùng một tên biến có thể sử dụng trong các hàm khác nhau mà không gây xung đột.
- Trái ngược với biến cục bộ là biến toàn cục (global variable), biến toàn cục có phạm vi trên toàn bộ chương trình, trên tất cả các hàm, điều này có nghĩa là tất cả các hàm đều có thể truy cập và thay đổi biến toàn cục. Để khai báo sử dụng biến toàn cục ta sử dụng từ khóa **global**, việc sử dụng từ khóa global chỉ cần thiết khi ta gán, hoặc thay đổi giá trị của biến toàn cục.

```
1 result = 0
2 x, y, z = 5, 6, 3
3
4 def maxInTriple():
5     return max(max(x,y),z);
6
7 def doubleMax():
8     global result
9     result = maxInTriple() * 2;
10
11 print(maxInTriple())
12 doubleMax()
13 print(result)
```


HÀM

Hàm đệ quy

- Hàm đệ quy là các hàm có chứa lời gọi hàm đến chính nó. Hàm đệ quy thường được sử dụng để giải các bài toán có định nghĩa đệ quy hoặc quy nạp
- Nhìn chung, với cùng một bài toán nếu có hàm đệ quy và hàm không đệ quy để giải bài toán đó thì thường hàm đệ quy có lời giải ngắn gọn hơn, tuy nhiên trong một số trường hợp hàm đệ quy sẽ tốn bộ nhớ và chi phí tính toán hơn.
- Ví dụ dưới đây là hàm đệ quy tính giai thừa của một số nguyên.

```
>>> def factorial(n):
...     if(n == 0):
...         return 1
...     else:
...         return n*factorial(n-1)
...
...
...
>>> factorial(6)
720
```

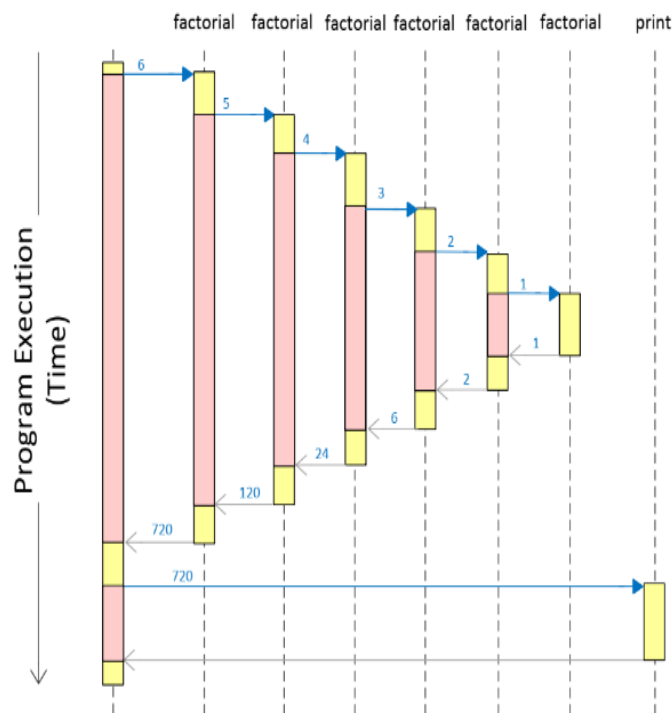
HÀM

Hàm đệ quy

- Việc thực thi lời gọi hàm factorial(6) được mô tả như sau:

$$\text{factorial}(6) = 6 * \text{factorial}(5)$$

[illegible]



```
>>> def factorial(n):
...     if(n == 0):
...         return 1
...     else:
...         return n*factorial(n-1)
...
...
...
>>> factorial(6)
720
```

HÀM

Hàm đệ quy

- Khi xây dựng hàm đệ quy, cần xác định trường hợp cơ sở (base case) để đảm bảo hàm đệ quy có điểm dừng.
- Trong ví dụ trên trường hợp cơ sở là $n == 0$, các trường hợp khác là trường hợp đệ quy.

```
>>> def factorial(n):
...     if(n == 0):
...         return 1
...     else:
...         return n*factorial(n-1)
...
...
...
>>> factorial(6)
720
```

HÀM

Kiểu dữ liệu của hàm

- Trong Python, một hàm là một đối tượng đặc biệt, hàm cũng xác định kiểu giống như số nguyên hay chuỗi ký tự. Ví dụ dưới đây, kiểu của hàm **sqrt** trong module **math** được xác định bằng hàm **type**.
- Trong ví dụ này, x được gán bằng sqrt, khi đó định danh x tham chiếu tới hàm sqrt và thực hiện chức năng tương tự, kết quả của x(9) là 3

```
>>> from math import sqrt
>>> type(sqrt)
<class 'builtin_function_or_method'>
>>> x = sqrt
>>> x(9)
3.0
>>> |
```

HÀM

Kiểu dữ liệu của hàm

- Đối số của một hàm có thể là một hàm

```
1 def add(a,b):  
2     return a+b;  
3  
4 def sub(a,b):  
5     return a-b;  
6 def run(f,a,b):  
7     return f(a,b)  
8  
9 a = 10  
10 b = 21  
11 print(run(add,a,b))  
12 print(run(sub,a,b))
```

HÀM

Biểu thức lambda

- Python hỗ trợ việc định nghĩa hàm thông qua biểu thức lambda.
- Biểu thức lambda giúp việc định nghĩa hàm dễ hơn và nhanh gọn hơn, ngoài ra nó còn cung cấp một cách để định nghĩa các hàm không định danh.

lambda *parameterlist* : *expression*

```
>>> f = lambda x, y: 10 if x == y else 2
>>> f(2,5)
2
>>> f(5,5)
10
>>> type(f)
<class 'function'>
>>>
```

HÀM

Bộ sinh

- Một chương trình sinh, hay bộ sinh tạo ra một chuỗi các giá trị. Từ khóa **return** dùng để trả lại giá trị cho một hàm, đồng thời nó cũng kết thúc hàm ngay sau khi trả lại giá trị.
- Thay vì sử dụng từ khóa **return**, chúng ta sử dụng từ khóa **yield** để tạo ra bộ sinh. Từ khóa **yield** thực hiện trả lại một giá trị nhưng không kết thúc hàm.

```
>>> def gen():
...     yield 1
...     yield 'aaa'
...     yield 3.5
...
...
>>> type(gen)
<class 'function'>
>>> f = gen()
>>> type(f)
<class 'generator'>
>>> next(f)
1
>>> next(f)
'aaa'
>>> next(f)
3.5
>>> next(f)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
>>> for i in gen():
...     print(i)
...
1
aaa
3.5
>>>
```

CẤU TRÚC ĐIỀU KHIỂN

Biểu thức điều kiện

- Biểu thức điều kiện là biểu thức nhận giá trị logic hoặc là đúng hoặc là sai (True hoặc False). Python cung cấp một kiểu dữ liệu để lưu trữ giá trị logic là kiểu **bool**.
- Chú ý, vì Python là ngôn ngữ phân biệt hoa thường, từ khóa cho giá trị logic đúng là True (không phải true hay TRUE,...), tương tự từ khóa cho giá trị logic sai là False (không phải false hay FALSE). Trong ví dụ trên nếu gán `a = true`, trình thông dịch sẽ báo lỗi, kiểu của giá biến lưu giá trị logic là **bool**.

```
>>> a = true
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'true' is not defined

>>> a = True
>>> print(a)
True

>>> type(a)
<class 'bool'>

>>> |
```


CẤU TRÚC ĐIỀU KHIỂN

Biểu thức điều kiện

- Các biểu thức logic hay biểu thức điều kiện được tạo thành từ các biến và các phép toán logic. Bảng dưới đây mô tả các phép toán quan hệ logic trong Python.

Biểu thức	Ý nghĩa	Ví dụ
$x == y$	Biểu thức nhận giá trị đúng (True) nếu như $x = y$ (phép so sánh bằng, không phải phép gán); ngược lại biểu thức nhận giá trị sai (False)	$5 == 5 \Rightarrow \text{True}$ $5 == 4 \Rightarrow \text{False}$
$x < y$	Biểu thức nhận giá trị đúng (True) nếu $x < y$; ngược lại nhận giá trị sai (False)	$3 < 5 \Rightarrow \text{True}$ $4 < 4 \Rightarrow \text{False}$
$x \leq y$	Biểu thức nhận giá trị đúng (True) nếu $x \leq y$; ngược lại nhận giá trị sai (False)	$5 \leq 5 \Rightarrow \text{True}$ $5 \leq 4 \Rightarrow \text{False}$
$x > y$	Biểu thức nhận giá trị đúng (True) nếu $x > y$; ngược lại nhận giá trị sai (False)	$6 > 5 \Rightarrow \text{True}$ $5 > 5 \Rightarrow \text{False}$
$x \geq y$	Biểu thức nhận giá trị đúng (True) nếu $x \geq y$; ngược lại nhận giá trị sai (False)	$5 \geq 5 \Rightarrow \text{True}$ $5 \geq 6 \Rightarrow \text{False}$
$x \neq y$	Biểu thức nhận giá trị đúng (True) nếu $x \neq y$; ngược lại nhận giá trị sai (False)	$5 \neq 6 \Rightarrow \text{True}$ $5 \neq 5 \Rightarrow \text{False}$

CẤU TRÚC ĐIỀU KHIỂN

Biểu thức điều kiện

- Các phép toán quan hệ, kết hợp với các phép toán logic sẽ tạo ra các biểu thức logic phức tạp hơn, nhằm đạt được các biểu thức điều kiện phù hợp với các bài toán thực tế. Bảng dưới đây mô tả các phép toán logic trong Python
- Cũng giống như các phép toán số học, các phép toán logic cũng có thứ tự ưu tiên. Bảng dưới đây mô tả thứ tự ưu tiên của các phép toán logic.

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False
Số ngôi		Toán tử		Thứ tự
Hai ngôi		>, <, >=, <=, ==, !=		Trái sang phải
Một ngôi		not		
Hai ngôi		and		Trái sang phải
Hai ngôi		or		Trái sang phải

CẤU TRÚC ĐIỀU KHIỂN

Biểu thức điều kiện

- Các phép toán logic thì có độ ưu tiên thấp hơn so với các phép toán số học.

```
>>> x = 3
```

```
>>> x * 3 + 2 < 6 or x*2 + 3 > 8
```

```
True
```

```
>>> |
```

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False
Số ngôi		Toán tử		Thứ tự
Hai ngôi		>, <, >=, <=, ==, !=		Trái sang phải
Một ngôi		not		
Hai ngôi		and		Trái sang phải
Hai ngôi		or		Trái sang phải

CẤU TRÚC ĐIỀU KHIỂN

Cấu trúc điều kiện if

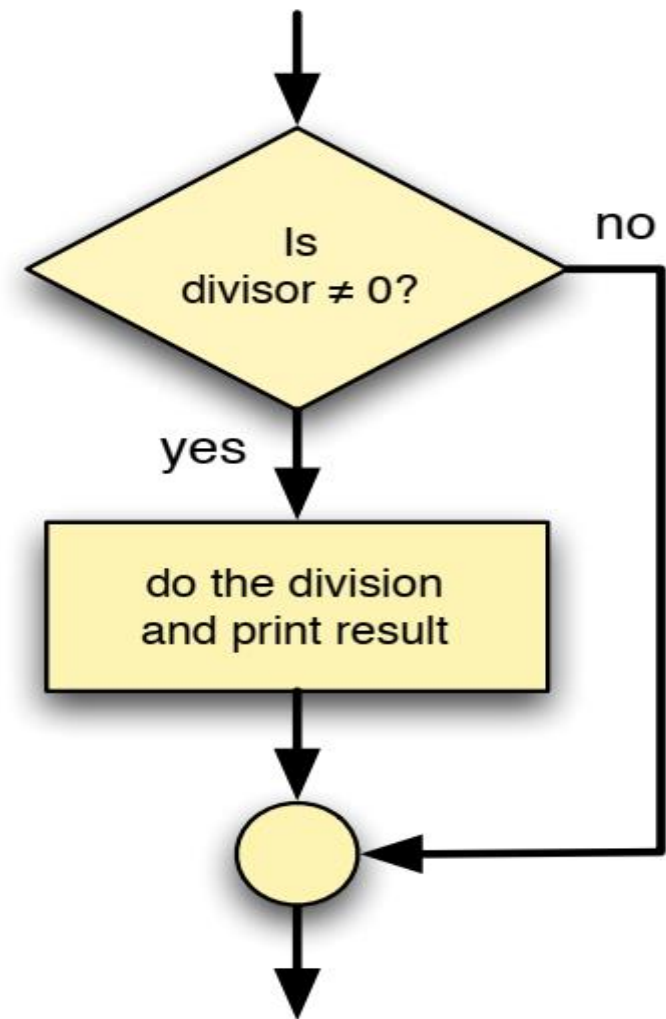
- Lệnh if thực hiện tính giá trị của biểu thức điều kiện, nếu biểu thức nhận giá trị đúng True, các lệnh trong khối block sẽ được gọi tới, nếu giá trị của biểu thức điều kiện nhận giá trị sai thì các lệnh trong khối block sẽ không được gọi tới.
- Chú ý đến dấu ':' sau biểu thức điều kiện, và khối lệnh block cần được thụt lề so với lệnh if (điều này là bắt buộc) vì Python không giống hàng lệnh bằng dấu {}.

if

condition

:

block



CẤU TRÚC ĐIỀU KHIỂN

Cấu trúc điều kiện if

- Trong Python, giá trị của biểu thức điều kiện có thể là giá trị số, một giá trị số có giá trị khác 0, khác rỗng sẽ nhận giá trị logic đúng True, và giá trị số bằng 0 hoặc 0.0 sẽ nhận giá trị logic sai False.
- Lệnh if có thể phát biểu như một mệnh đề điều kiện, nếu ... thì ...

```
>>> if 2:
...     print('Giá trị khác 0 tương đương với giá trị logic đúng True')
...
Giá trị khác 0 tương đương với giá trị logic đúng True
>>> if 0:
...     print('Giá trị bằng 0 tương đương với giá trị logic sai False')
...

```

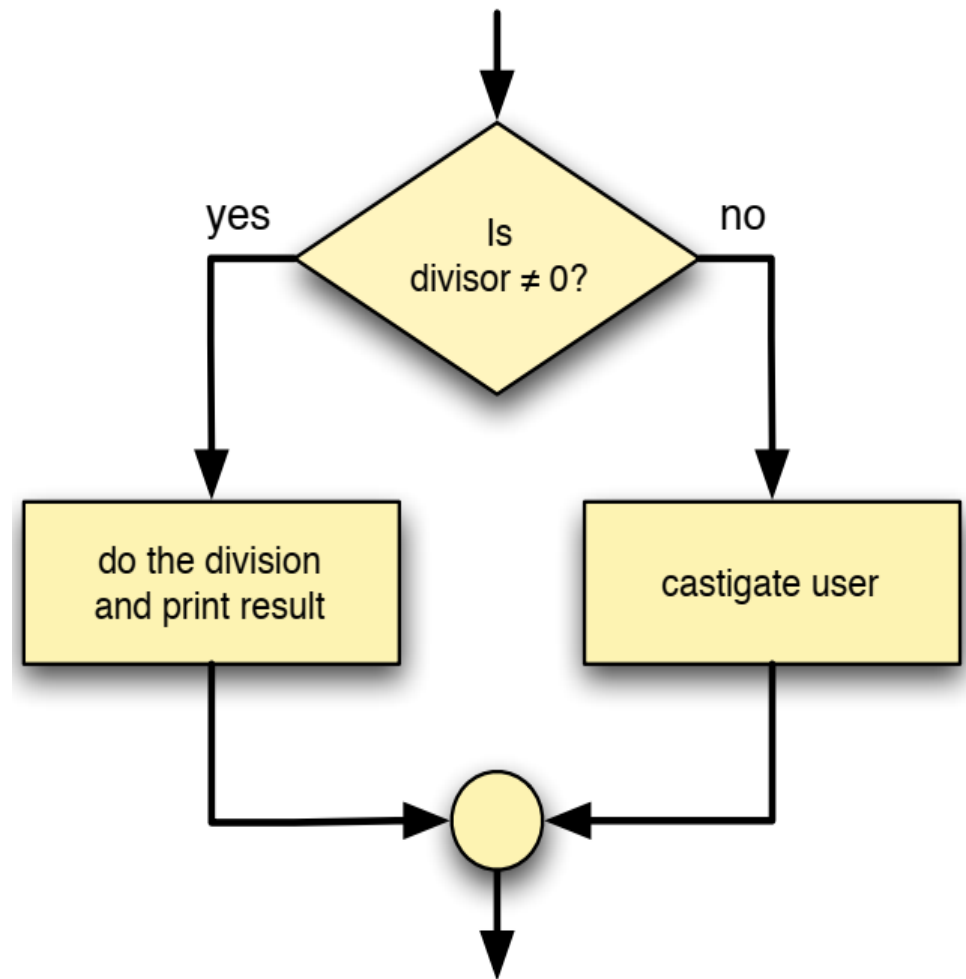
```
>>> x = 10
>>> y = 2
>>> if(y!=0):
...     print(x,'/',y,'=',x/y)
...
10 / 2 = 5.0
>>> y = 0
>>> if( y == 0)
File "<input>", line 1
    if( y == 0)
                ^
SyntaxError: invalid syntax
>>> if(y == 0):
...     print('Không thực hiện được phép chia cho 0')
...
Không thực hiện được phép chia cho 0
>>> |

```

CẤU TRÚC ĐIỀU KHIỂN

Cấu trúc điều kiện if/else

- Lệnh if sẽ thực hiện các lệnh trong khối block nếu biểu thức điều kiện nhận giá trị đúng True, và nếu chúng ta muốn thực hiện các lệnh khi biểu thức nhận giá trị sai ta cần viết thêm một lệnh if nữa, sẽ tốt hơn nếu ta có lệnh để tránh việc này.
- Python cung cấp khối lệnh if / else để việc viết các khối lệnh với biểu thức điều kiện linh hoạt hơn.



CẤU TRÚC ĐIỀU KHIỂN

Cấu trúc điều kiện if/else

- Chú ý đến dấu ':' sau từ khóa else và việc thụt lề của khối lệnh trong else-block là bắt buộc

if

condition

:

if-block

else:

else-block

```
>>> x = 10
>>> y = int(input('Nhập giá trị y = '))
Nhập giá trị y = 5
>>> if(y == 0):
...     print('Không thực hiện được phép chia cho 0')
... else:
...     print(x, '/', y, ' = ', x/y)
...
10 / 5 = 2.0
```

CẤU TRÚC ĐIỀU KHIỂN

Lệnh pass

- Python cung cấp lệnh pass để thực hiện lệnh đặc biệt là “không thực hiện gì”.
- Khối lệnh trong ví dụ trên là không hợp lệ trong Python, bên trong khối lệnh if cần có ít nhất một câu lệnh, để thực hiện yêu cầu như ví dụ trên ta có thể sử dụng lệnh pass.

```
1 x = 5
2 if(x < 0):
3     #Không thực hiện gì
4 else:
5     print('x = ',x)
6
```

```
1 x = 5
2 if(x < 0):
3     pass #Không thực hiện gì
4 else:
5     print('x = ',x)
6
```


CẤU TRÚC ĐIỀU KHIỂN

Sai số với phép toán trên số thực

- Chú ý đến sai số trong các phép so sánh với giá trị số thực.

```
>>> x = 1.11 - 1.1
```

```
>>> y = 2.11 - 2.1
```

```
>>> print('x = ',x,'; y = ',y)
```

```
x = 0.0100000000000000009 ; y = 0.0099999999999999787
```

```
>>> x == y
```

```
False
```

```
>>> |
```

CẤU TRÚC ĐIỀU KHIỂN

Sai số với phép toán trên số thực

- Các khối lệnh if/else có thể lồng nhau để mô tả các biểu thức hoặc chương trình phức tạp, một lệnh if có thể đứng độc lập, tuy vậy một lệnh else thì cần đi cùng với 1 lệnh if, lệnh else sẽ bắt cặp với lệnh if gần nhất không chứa else.
- Python không có cấu trúc switch/case như các ngôn ngữ Java, C, C++,... Tuy nhiên Python có cặp lệnh if/elif/else thực hiện chức năng tương tự và linh hoạt hơn.

```
if condition-1 :  
    block-1  
elif condition-2 :  
    block-2  
elif condition-3 :  
    block-3  
elif condition-4 :  
    block-4  
    :  
else:  
    default-block
```

```
1 month = int(input('Nhập vào tháng 1 - 12'))  
2 if month == 1:  
3     print('tháng 1')  
4 elif month == 2:  
5     print('tháng 2')  
6 elif month == 3:  
7     print('tháng 3')  
8 elif month == 4:  
9     print('tháng 4')  
10 elif month == 5:  
11     print('tháng 5')  
12 elif month == 6:  
13     print('tháng 6')  
14 elif month == 7:  
15     print('tháng 7')  
16 elif month == 8:  
17     print('tháng 8')  
18 elif month == 9:  
19     print('tháng 9')  
20 elif month == 10:  
21     print('tháng 10')  
22 elif month == 11:  
23     print('tháng 11')  
24 elif month == 12:  
25     print('tháng 12')  
26 else:  
27     print('Giá trị nhập không đúng (1 - 12)')
```

CẤU TRÚC ĐIỀU KHIỂN

Sai số với phép toán trên số thực

- Ngoài ra, biểu thức điều kiện có thể viết dưới dạng như sau:

expression-1 **if** *condition* **else** *expression-2*

```
1 số_bị_chia = int(input('nhập vào số chia = '))
2 số_chia = int(input('nhập vào số chia = '))
3
4 thương = số_bị_chia / số_chia if số_chia != 0 else 'Lỗi, không chia được cho 0'
5 print(thương)
```

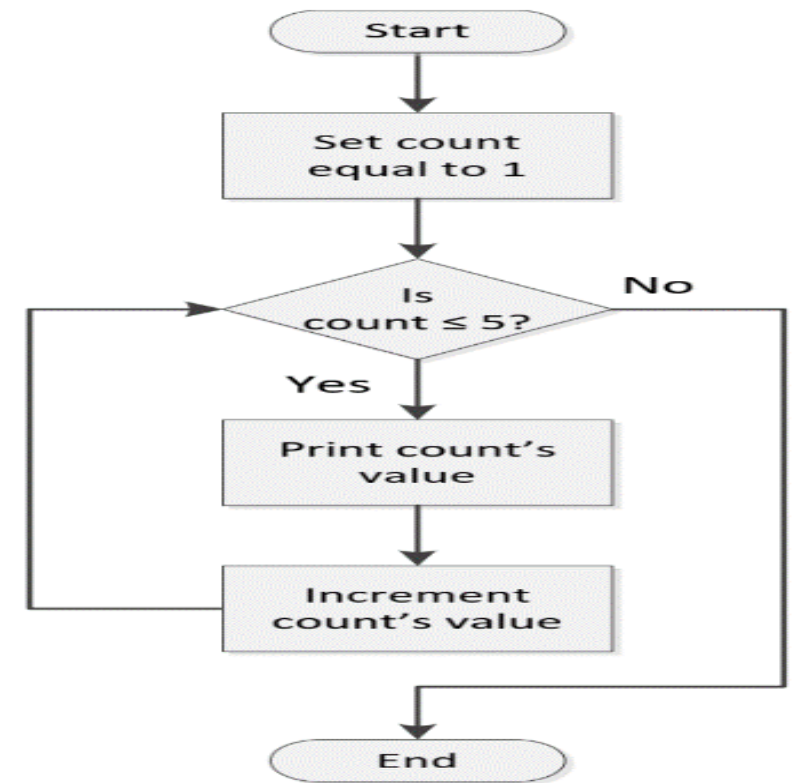
CẤU TRÚC ĐIỀU KHIỂN

Vòng lặp while

- Khối lệnh block được thực hiện trong khi điều kiện condition còn nhận giá trị đúng True.

while *condition* :
block

```
count = 1  
while count <= 5:  
    print(count)  
    count += 1
```



CẤU TRÚC ĐIỀU KHIỂN

Vòng lặp for

- Vòng lặp for thực hiện duyệt qua lần lượt các phần tử trong danh sách và gán cho biến variable, ứng với mỗi phần tử trong list khối lệnh block sẽ được thực hiện một lần.
- Các vòng lặp có thể được sử dụng lồng nhau để giải quyết các bài toán như in ma trận, duyệt các phần tử trên ma trận, hay khi làm việc với dữ liệu kiểu mảng nhiều chiều (danh sách của các danh sách).

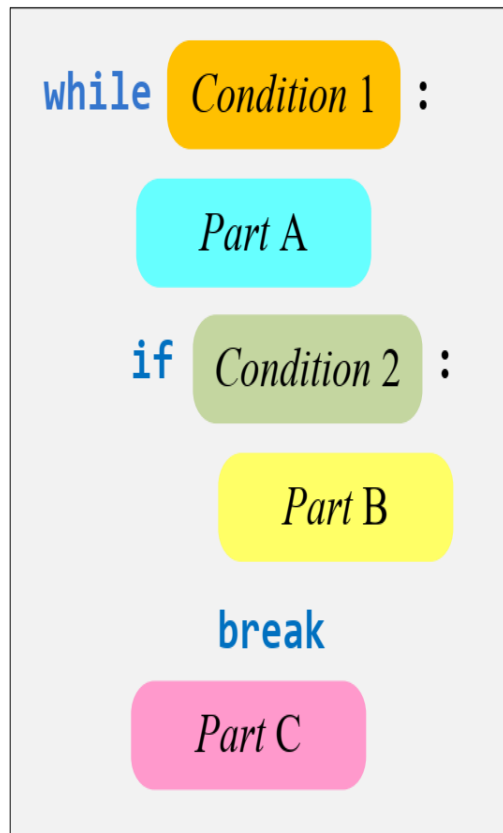
```
for variable in list:  
    block
```

```
for i in range(1, 10):  
    print(i)
```

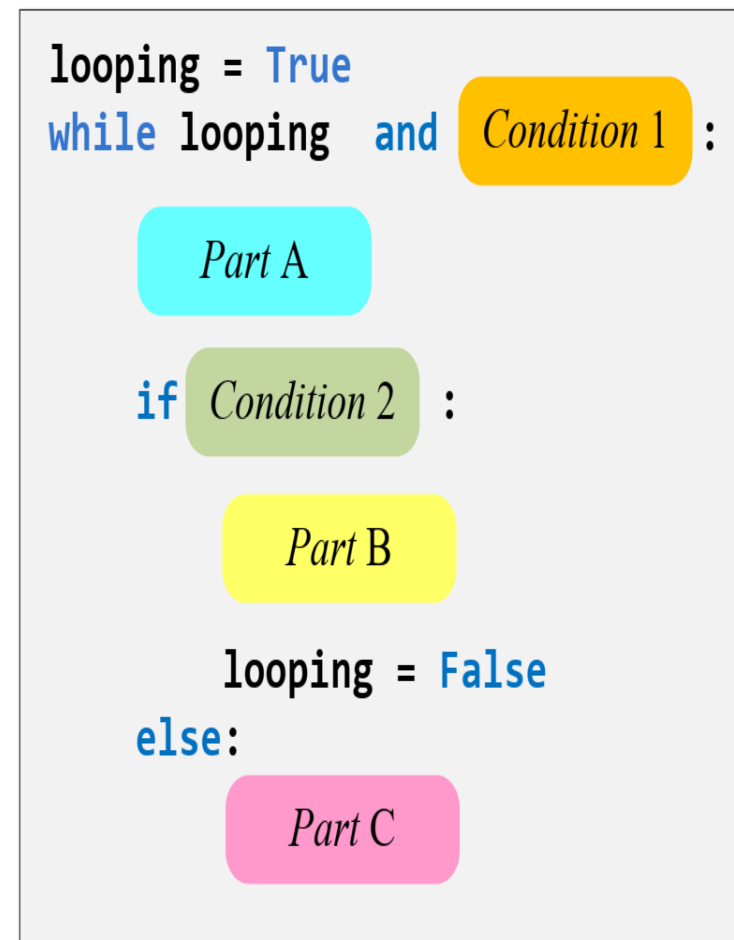
CẤU TRÚC ĐIỀU KHIỂN

Lệnh break

- Các vòng lặp có thể kết thúc bằng lệnh **break**
- Lệnh **break** thường được dùng để kết thúc vòng lặp khi đạt được điều kiện không, hoặc khó xác định bởi số lần lặp, đôi khi là vòng lặp vô hạn cho đến khi tìm thỏa mãn điều kiện thì dừng



→
Eliminate
the
break
statement



CẤU TRÚC ĐIỀU KHIỂN

Lệnh break

- Các vòng lặp có thể kết thúc bằng lệnh **break**
- Lệnh **break** thường được dùng để kết thúc vòng lặp khi đạt được điều kiện không, hoặc khó xác định bởi số lần lặp, đôi khi là vòng lặp vô hạn cho đến khi tìm thỏa mãn điều kiện thì dừng

```
import math
a , b = 1, 10 # hai điểm đầu mút của đoạn [1, 10]
e = 0.000000001 # sai số
f = lambda x : 2*x-4
y = (a+b)/2

while math.fabs(f(y)) > e:
    y = (a+b)/2
    if(math.fabs(f(y)) < e):
        break
    elif f(y)*f(a) < 0:
        b = y
    else:
        a = y

print('nghiệm của phương trình là y = ',y)
```

CẤU TRÚC ĐIỀU KHIỂN

Lệnh continue

- Khi chương trình gặp lệnh **break** bên trong một vòng lặp, các câu lệnh phía sau lệnh break trong vòng lặp sẽ bị bỏ qua (không thực hiện) và chương trình thoát khỏi vòng lặp.
- Trong vòng lặp, đôi khi với một điều kiện nào đó chúng ta muốn bỏ qua lượt lặp hiện tại, nhưng vẫn muốn thực hiện các vòng lặp tiếp theo ta có thể dùng lệnh **continue**.

```
count = 1
sum = 0

while count < 1000: #
    Tính tổng các số chẵn
    nhỏ hơn 1000
    count += 1
    if(count % 2 ==
1):
        continue
    sum += count

print('Tổng = ', sum)
```

while Condition 1 :

Part A

if Condition 2 :

Part B

continue

Part C

→
Eliminate
the
continue
statement

while Condition 1 :

Part A

if Condition 2 :

Part B

else:

Part C

CẤU TRÚC ĐIỀU KHIỂN

Lệnh while / else và for / else

- Python hỗ trợ cấu trúc while / else và for / else cho các vòng lặp while và for.
- Với cấu trúc này, khi vòng lặp kết thúc, các lệnh trong khối else sẽ được gọi tới và thực thi
- Chú ý, nếu vòng lặp bị kết thúc bởi lệnh break thì khối lệnh trong else sẽ không được gọi tới.

```
count = 1
sum = 0

while count < 1000: # Tính
    tổng các số chẵn nhỏ hơn
    1000
    count += 1
    if(count % 2 == 1):
        continue
    sum += count
else:
    print('Tổng = ', sum)
```

```
count = 1
sum = 0

while count < 1000: # Tính
    tổng các số chẵn nhỏ hơn 1000
    count += 1
    if(count % 2 == 1):
        continue
    sum += count
    if(sum > 2500):
        break
else:
    print('Tổng = ', sum)
```

CẤU TRÚC ĐIỀU KHIỂN

Ví dụ với vòng lặp

- Ví dụ dưới đây minh họa chương trình đếm ngược từ 10 về 0.
- Chương trình sinh số ngẫu nhiên với module random

```
from time import sleep
for count in range(10, -1, -1): # Range 10, 9, 8, ..., 0
    print(count) # Hiển thị giá trị count hiện tại
    sleep(1) # Tạm dừng (ngủ) chương trình 1 giây
```

```
from random import randrange, seed

seed(23) # Set random number seed
for i in range(0, 100): # Print 100 random numbers
    print(randrange(1, 101), end=' ') # Số ngẫu nhiên từ 1-100
print() # Print newline
```

```
from random import randrange, seed
from datetime import datetime
seed(datetime.now())
for i in range(0, 100): # Print 100 random numbers
    print(randrange(1, 101), end=' ') # Số ngẫu nhiên từ 1 đến 100
print() # Print newline
```

The image features a rectangular chalkboard with a light-colored wooden frame, tilted at an angle. It is set against a background of vertical wooden planks. The text 'Hết bài' is written in white on the black surface of the chalkboard. At the top of the image, there are two horizontal bars: a dark purple one on the left and a light grey one on the right.

Hết bài