Python

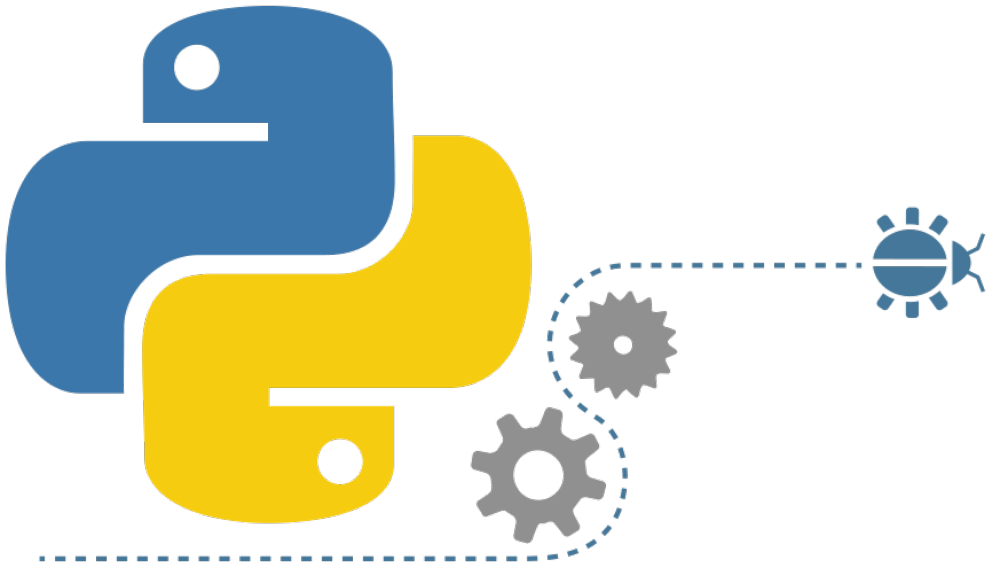
Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình.

Python Basics

Kiến thức ngôn ngữ lập trình Python

Python overview

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu. *(Theo wikipedia.org)*



1. Các phiên bản Python

Sự phát triển Python đến nay có thể chia làm các giai đoạn:

- **Python 1:** bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido làm việc tại CWI (Centrum voor Wiskunde en Informatica - Trung tâm Toán-Tin học tại Amsterdam, Hà Lan). Vì vậy, các phiên bản Python đầu tiên đều do CWI phát hành. Phiên bản cuối cùng phát hành tại CWI là 1.2.
  - Vào năm 1995, Guido chuyển sang CNRI (Corporation for National Research Initiatives) ở Reston, Virginia. Tại đây, ông phát hành một số phiên bản khác. Python 1.6 là phiên bản cuối cùng phát hành tại CNRI.
  - Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn GPL. Sau đó, CNRI và FSF (Free Software Foundation - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Cùng năm đó, Guido được nhận Giải thưởng FSF vì Sự phát triển Phần mềm tự do

(Award for the Advancement of Free Software).
    - Phiên bản 1.6.1 ra đời sau đó là phiên bản đầu tiên tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.
- **Python 2:** vào năm 2000, Guido và nhóm phát triển Python dời đến BeOpen.com và thành lập BeOpen PythonLabs team. Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập Digital Creations.
    - Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của Python Software Foundation (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.
- **Python 3**, còn gọi là Python 3000 hoặc Py3K: Dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python". Trong PEP (Python Enhancement Proposal) có mô tả chi tiết các thay đổi trong Python.

Hiện tại Python có 2 nhánh chính là 2.x và 3.x. Ở nhánh 2.x đã dừng phát triển và đang đứng ở phiên bản 2.7. Nhánh Python 3.x thì vẫn đang được tiếp tục phát triển.

2. Đặc điểm của ngôn ngữ Python

- Ngôn ngữ lập trình phổ biến:

Thành công của nó không đơn thuần là một mối quan tâm ngày càng tăng trong lập trình như một toàn thể trong những năm gần đây. Năm ngoái, Stack Overflow đã phân tích dữ liệu về sự phát triển của các ngôn ngữ lập trình, dựa trên dữ liệu lưu lượng truy cập từ các quốc gia có thu nhập cao. Từ dữ liệu này, họ tuyên bố rằng "*Python có một cơ sở vững chắc để trở thành ngôn ngữ lập trình chính phát triển nhanh nhất*".

- Các công ty lớn sử dụng Python:

Uber, PayPal, Google, Facebook, Instagram, Netflix, Dropbox và Reddit đều sử dụng Python trong quá trình phát triển và thử nghiệm của họ. Hơn nữa, Python cũng được sử dụng rộng rãi trong lập trình robot và các hệ thống nhúng. Ngay cả các hệ thống kế thừa được viết bằng C và C ++ cũng dễ dàng giao tiếp với Python.

Trong các lĩnh vực yêu cầu phân tích dữ liệu - lĩnh vực rất phát triển hiện nay, Python và các thư viện của nó đều phù hợp. Goldman Sachs là một trong nhiều tổ chức tài chính lớn sử dụng Python để thể hiện số lượng lớn dữ liệu của họ.

- Học máy với Python:

Học máy (Machine Learning) rất quan trọng trong thế giới hiện đại. Nó sắp xếp trải nghiệm internet của bạn, làm mọi thứ từ kiểm duyệt các mạng xã hội đến lái xe ô tô.

Điều này liên quan thế nào với Python? Trong khi có các thư viện được thiết kế để làm việc với các ngôn ngữ phổ biến khác, Python là ngôn ngữ thực tế dành cho học máy. TensorFlow của Google

hoạt động chủ yếu với Python. Hầu hết các khóa học về học máy đều sử dụng ngôn ngữ lập trình này. Việc phân tích dữ liệu và phân tích cú pháp cần thiết cho việc học máy đều thực hiện tốt với Python và các thư viện của nó.

- Python được hỗ trợ tốt:

Do ngày càng phổ biến, Python được hỗ trợ trực tuyến tốt ở hầu hết cấp độ. Các trang web như Stack Overflow thường xuyên đưa ra trợ giúp với các nguyên tắc cơ bản ở cấp độ mới làm quen với Python. Các lập trình viên làm việc trên các vấn đề phức tạp và cụ thể cũng có thể tìm thấy sự hỗ trợ hiệu quả hơn cho ngôn ngữ lập trình này, so với các ngôn ngữ khác.

- Python là ngôn ngữ của giáo dục:

Việc sử dụng máy tính trong giáo dục đã thay đổi hoàn toàn trong những năm gần đây. Ngày nay, công nghệ tham gia nhiều vào lĩnh vực giáo dục, trong đó việc dạy lập trình phổ biến hơn trên toàn thế giới.

Python là một ngôn ngữ lập trình dễ đọc, được thiết kế với các mô tả đơn giản và cú pháp thông thường. Trải nghiệm người dùng cũng được ưu tiên. Vì vậy, đây là ngôn ngữ rất phù hợp để dạy cho trẻ em. Đơn giản nhất, trẻ em có thể học Python từ một phiên bản trò chơi Minecraft có tên gọi Minecraft Pi.

Python cũng phù hợp để giáo dục mở rộng. Một số trường đại học dạy ngôn ngữ lập trình này không chỉ trong ngành khoa học máy tính mà còn cho sinh viên toán. Ngoài ra, Matplotlib (một thư viện Python phổ biến) được sử dụng cho các đối tượng ở tất cả các cấp để thể hiện dữ liệu phức tạp.

Python cũng là một trong những ngôn ngữ phát triển nhanh nhất trên Codecademy, vì rất dễ để học từ xa.

- Python là ngôn ngữ miễn phí:

Bản chất mọi ngôn ngữ lập trình đều miễn phí. Sự khác biệt ở đây là Python là mã nguồn mở, nó hoàn toàn tương thích với các phần mềm mã nguồn mở như Open Source và GPL.

Trạng thái không ràng buộc của chuỗi Python làm cho nó trở thành một công cụ hoàn hảo cho tất cả mọi người sử dụng.

Python Installation - IDE

Hướng dẫn cài đặt Python và công cụ lập trình IDE

1. Cài đặt

Python hỗ trợ hầu hết các nền tảng và rất dễ tìm thấy sẵn trên một số hệ điều hành như Mac OS, Linux, Windows.

Để biết là hệ thống của bạn đã cài Python chưa, có thể vào màn hình command line và gõ:

```
$ python --version
```

Nếu đã cài đặt python thì sẽ hiển thị thông tin phiên bản python. Nếu báo lỗi thì đồng nghĩa với bạn chưa cài đặt Python.

Có thể tham khảo cách cài đặt Python tại: https://www.python.org/downloads/

2. Công cụ phát triển - IDE

Chỉ cần dùng một text editor là bạn có thể viết được code python hoặc có thể dùng các công cụ cao cấp hơn (IDE) như Aptana, PyCharm... Các IDE thường hỗ trợ thêm quá trình phân tích cú pháp dòng lệnh trong quá trình lập trình, phát triển cũng như hỗ trợ run, debug chương trình.

Danh sách các IDE được yêu thích nhất:

1. PyCharm
2. Spyder

3. PyDev
4. IDLE
5. Wing
6. Eric Python
7. Sublime Text
8. Rodeo
9. Jupyter Notebook
10. Visual Studio

Python Hello world

Hướng dẫn viết chương trình đầu tiên Hello world bằng Python.

Tạo một file có tên là `helloworld.py` và có nội dung như sau:

```
print ('Hello world')
```

`print` là lệnh cơ bản nhất để xuất một biến ra (thường là màn hình)

Sau đó, vào màn hình command line, di chuyển đến thư mục chứa file này và gõ:

```
$ python helloworld.py
```

Nếu thấy xuất hiện dòng chữ `Hello world` tức là bạn đã hoàn thành việc viết ứng dụng **python** đầu tiên.

Python naming conventions - Quy ước đặt tên trong Python

Các quy ước đặt tên khi lập trình chương trình Python.

1. General

- Tránh sử dụng tên quá chung chung hoặc quá dài dòng. Hãy cân bằng tốt giữa cả hai:
  - Bad: data_structure, my_list, info_map, dictionary_for_the_purpose_of_storing_data_representing_word_definitions
  - Good: user_profile, menu_options, word_definitions
- Không sử dụng các từ hoặc ký tự đặc biệt như như: "O", "l", or "I"
- Khi sử dụng tên CamelCase, viết hoa tất cả các chữ cái viết tắt (ví dụ: HTTPServer)

2. Packages

- Tên package nên được viết thường
- Khi cần nhiều từ, hãy sử dụng ký tự "_" để phân tách

3. Modules

- Tên module nên được viết thường
- Khi cần nhiều từ, hãy sử dụng ký tự "_" để phân tách

4. Classes

- Tên lớp phải tuân theo quy ước UpperCaseCamelCase
- Các lớp ngoại lệ sẽ kết thúc bằng "Error"

5. Global (module-level) Variables

- Các biến toàn cục nên là chữ thường
- Các từ trong tên biến toàn cục phải được phân tách bằng dấu gạch dưới "_"

6. Instance Variables

- Tên biến phải là chữ thường
- Các từ trong tên biến thể hiện phải được phân tách bằng dấu gạch dưới "_"
- Các biến non-public nên bắt đầu bằng một dấu gạch dưới

7. Methods

- Tên phương thức nên viết thường
- Các từ trong một tên phương thức nên được phân tách bằng dấu gạch dưới "_"
- Phương pháp non-public nên bắt đầu bằng một dấu gạch dưới

8. Method Arguments

- Các phương thức instance nên có tham số thứ nhất là `self`
- Các phương thức class nên có tham số thứ nhất là `cls`

9. Functions

- Tên function nên viết thường
- Các từ trong một tên phương thức nên được phân tách bằng dấu gạch dưới "_"

10. Constants

- Tên constant nên viết thường
- Các từ trong một tên phương thức nên được phân tách bằng dấu gạch dưới "_"

Python Basic Syntax - Cú pháp cơ bản trong Python

Các cú pháp cơ bản của Python

- Kiểm tra phiên bản của Python:

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Nhập đoạn code sau đây và nhấn Enter:

```
>>> print "Hello, Python!"
```

Output:

```
Hello, Python!
```

- Lập trình trình Python script mode:

`test.py`

```
#!/usr/bin/python

print "Hello, Python!"
```

```
$ chmod +x test.py     # This is to make file executable
$./test.py
```

Output:

```
Hello, Python!
```

- Từ khóa trong Python:

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python Variable types - Kiểu biến trong Python

Các kiểu biến dữ liệu trong Python

Trong lập trình, biến (variable) là tên của một vùng trong bộ nhớ RAM, được sử dụng để lưu trữ thông tin. Bạn có thể gán thông tin cho một biến, và có thể lấy thông tin đó ra để sử dụng. Khi một biến được khai báo, một vùng trong bộ nhớ sẽ dành cho các biến.

Dựa trên kiểu dữ liệu của một biến, trình thông dịch phân bổ bộ nhớ và quyết định những gì có thể được lưu trữ trong bộ nhớ dành riêng. Do đó, bằng cách gán các loại dữ liệu khác nhau cho biến, bạn có thể lưu trữ số nguyên, số thập phân hoặc ký tự trong các biến này.

- Khai báo biến bằng một câu lệnh gán:

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Output:

```
100
1000.0
John
```

- Khai báo nhiều loại giá trị (số, chuỗi) cho một biến:

```
a = 1
a = 'Hello World'
a = [1, 2, 3]
a = [1.2, 'Hello', 'W', 2]
```

- Khai báo 1 giá trị cho nhiều biến:

```
a = b = c = 1
```

- Khai báo kết hợp:

```
a, b, c = 1, 2, "john"
```

Khi đó, tương đương với:

```
a = 1
b = 2
c = "john"
```

Python Data type - Kiểu dữ liệu trong Python

Các kiểu dữ liệu trong Python

Python bao gồm các kiểu dữ liệu cơ bản sau:

- Numbers
- String
- List
- Tuple
- Dictionary

1. Python Number

Number được sử dụng để lưu trữ các giá trị số. Một đối tượng Number được tạo khi khai báo và gán giá trị cho chúng.

```
var1 = 1
var2 = 10
```

Bạn có thể delete tham chiếu tới 1 đối tượng Number bằng cách sử dụng cú pháp `del`:

```
del var1[,var2[,var3[....,varN]]]
```

Các kiểu Number trong Python bao gồm:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j

-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAE I	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

2. Python String

String trong Python được định nghĩa như một tập ký tự được thể hiện trong cặp dấu nháy (nháy đơn hoặc đôi):

- Để truy cập từng ký tự trong String, sử dụng mảng [] (hoặc [:]) với vị trí bắt đầu từ 0.
- Toán tử (+) String là lệnh thực hiện ghép chuỗi (concat) và toán tử (*) là lệnh lặp chuỗi.

```
str = 'Hello World!'

print str           # Prints complete string
print str[0]        # Prints first character of the string
print str[2:5]      # Prints characters starting from 3rd to 5th
print str[2:]       # Prints string starting from 3rd character
print str * 2       # Prints string two times
print str + "TEST"  # Prints concatenated string
```

Output:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

3. Python List

List là một kiểu dữ liệu dãy (sequence) các phần tử (element), nó cho phép loại bỏ, hoặc thêm các phần tử vào danh sách, đồng thời cho phép cắt lát (slice) các phần tử.

- Để truy cập từng phần tử trong Python List, sử dụng mảng [] (hoặc [:]) với vị trí bắt đầu từ 0.
- Toán tử (+) Python List là lệnh thực hiện ghép 2 List thành một và toán tử (*) là lệnh lặp List để thành một List có độ dài gấp đôi.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

Output:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

4. Python Tuple

Tuple là một kiểu dữ liệu chuỗi khác tương tự như List. Tuple bao gồm một số giá trị được phân tách bằng dấu phẩy. Tuy nhiên, không giống như List, các phần tử trong Tuple được đặt trong dấu ngoặc đơn.

Tuple có thể được coi là danh sách chỉ đọc (read-only), các phần tử trong Tuple là cố định và không thể thay đổi.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple           # Prints complete list
print tuple[0]        # Prints first element of the list
print tuple[1:3]      # Prints elements starting from 2nd till 3rd
print tuple[2:]       # Prints elements starting from 3rd element
print tinytuple * 2   # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

Output:

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

- Đoạn lệnh sau là không hợp lệ vì thực hiện update giá trị phần tử của Tuple:

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

5. Python Dictionary

Dictionary của Python là loại bảng băm (hash table). Chúng hoạt động như các mảng (array) hoặc băm (hash) được tìm thấy trong Perl và bao gồm các cặp key-value. Dictionary key có thể là hầu hết mọi loại Python, nhưng thường là Number hoặc String. Mặt khác, các giá trị có thể là bất kỳ đối tượng Python tùy ý.

Từ điển được bao quanh bởi dấu ngoặc nhọn {} và các giá trị có thể được chỉ định và truy cập bằng dấu ngoặc vuông ([]).

```
ict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}



print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

Output:

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

6. Chuyển đổi kiểu dữ liệu

Có một số hàm dựng sẵn để thực hiện chuyển đổi từ loại dữ liệu này sang loại dữ liệu khác. Các hàm này trả về một đối tượng mới biểu thị giá trị được chuyển đổi.

#	Function	Mô tả
1	int(x [,base])	Chuyển đổi x sang kiểu Integer
2	long(x [,base])	Chuyển đổi x sang kiểu Long
3	float(x)	Chuyển đổi x sang kiểu Float
4	complex(real [,imag])	Tạo một số phức
5	str(x)	Chuyển đổi x sang kiểu String
6	repr(x)	Chuyển đổi x sang kiểu expression string
7	eval(str)	Đánh giá một chuỗi và trả về một đối tượng
8	tuple(s)	Chuyển đổi sang kiểu Tuple
9	list(s)	Chuyển đổi sang kiểu List
10	set(s)	Chuyển đổi sang kiểu Set
11	dict(d)	Tạo một Dictionary, d phải là một sequence (key, value) tuple
12	frozenset(s)	Chuyển đổi sang kiểu Frozen Set
13	chr(x)	Chuyển đổi một Integer thành Character
14	unichr(x)	Chuyển đổi Integer thành Unicode Character

15	ord(x)	Chuyển đổi Character thành kiểu Integer
16	hex(x)	Chuyển đổi Integer thành Hex String
17	oct(x)	Chuyển đổi Integer thành Octal String

Python Arithmetic Operator - Toán tử số học trong Python

Python hỗ trợ một số toán tử toán học thông dụng như:

- `+` Phép cộng
- `-` Phép trừ
- `*` Phép nhân
- `/` Phép chia
- `%` Phép chia lấy dư (modulo)
- `**` Hàm mũ
- `//` Phép chia làm tròn xuống

Các ví dụ sau được thực hiện với: a = 10 và b = 20

Operator	Description	Example
+ Addition	Phép cộng	a + b = 30
- Subtraction	Phép trừ	a – b = -10
* Multiplication	Phép nhân	a * b = 200
/ Division	Phép chia	b / a = 2
% Modulus	Phép chia lấy dư	b % a = 0
** Exponent	Hàm mũ	a**b =10^{20}

//	Phép chia làm tròn xuống (Floor Division)	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

- Ví dụ:

```
a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c

c = a % b
print "Line 5 - Value of c is ", c

a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c

a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
```

```
Line 5 - Value of c is 1
Line 6 - Value of c is 8
Line 7 - Value of c is 2
```

Python Logical Operator - Toán tử logic trong Python

Các toán tử Logical trong Python.

- Giá trị đúng và sai tương ứng là: True và False

- `not` Để đảo giá trị.
- `and` Phép tính logic và (AND)
- `or` Phép tính logic hoặc (OR)

- Ví dụ:

```
a = True
b = False
print(('a and b is',a and b))
print(('a or b is',a or b))
print(('not a is',not a))
```

Output:

```
('a and b is', False)
('a or b is', True)
('not a is', False)
```

Python Membership Operator - Toán tử thành viên trong Python

Toán tử thực hiện kiểm tra phần tử trong một tập hợp (sequence).

- `in` Kiểm tra có tồn tại
- `not in` Kiểm không tồn tại

```
'good' in 'this is a greate example' # False
'good' not in 'this is a greate example' # True
```

Tập hợp (sequence) có thể là: String, List, Tuple

- Ví dụ:

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"

if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"

a = 2
if ( a in list ):
   print "Line 3 - a is available in the given list"
```

```
else:
   print "Line 3 - a is not available in the given list"
```

Output:

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Python Comparison Operator - Toán tử so sánh trong Python

Một số phép so sánh thông thường trong Python để so sánh 2 giá trị như:

- `<` Bé hơn
- `<=` Bé hơn hoặc bằng
- `>` Lớn hơn
- `>=` Lớn hơn hoặc bằng
- `==` Bằng
- `!=` Khác

Hỗ trợ dạng so sánh kép như:

```
x = 2
1 < x < 3 # True
10 < x < 20 # False
3 > x <= 2 # True
2 == x < 4 # True
```

Ví dụ:

```
a = 21
b = 10
c = 0

if ( a == b ):
   print "Line 1 - a is equal to b"
else:
   print "Line 1 - a is not equal to b"

if ( a != b ):
   print "Line 2 - a is not equal to b"
```

```
else:
   print "Line 2 - a is equal to b"

if ( a <> b ):
   print "Line 3 - a is not equal to b"
else:
   print "Line 3 - a is equal to b"

if ( a < b ):
   print "Line 4 - a is less than b"
else:
   print "Line 4 - a is not less than b"

if ( a > b ):
   print "Line 5 - a is greater than b"
else:
   print "Line 5 - a is not greater than b"

a = 5;
b = 20;
if ( a <= b ):
   print "Line 6 - a is either less than or equal to  b"
else:
   print "Line 6 - a is neither less than nor equal to  b"

if ( b >= a ):
   print "Line 7 - b is either greater than  or equal to b"
else:
   print "Line 7 - b is neither greater than  nor equal to b"
```

Output:

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
Line 4 - a is not less than b
Line 5 - a is greater than b
Line 6 - a is either less than or equal to b
Line 7 - b is either greater than or equal to b
```

Python Assignment Operator - Toán tử gán trong Python

Các toán tử khai báo và gán giá trị biến trong Python

Operator	Description	Example
=	Gán giá trị	`c = a + b` gán giá trị a + b cho c
+= Add AND	Cộng giá trị	`c += a` tương đương c = c + a
-= Subtract AND	Trừ giá trị	`c -= a` tương đương c = c - a
*= Multiply AND	Nhận giá trị	`c *= a` tương đương c = c * a
/= Divide AND	Chia giá trị	`c /= a` tương đương c = c / ac /= a tương đương c = c / a
%= Modulus AND	Phép chia lấy dư	`c %= a` tương đương c = c % a
**= Exponent AND	Hàm mũ	`c **= a` tương đương c = c ** a
//= Floor Division	Phép chia lấy phần nguyên	`c //= a` tương đương c = c // a

- Ví dụ: Thực hiện với a = 10 và b = 20

```
#! /usr/bin/python

a = 21
b = 10
c = 0

c = a + b
```

```
print "Line 1 - Value of c is ", c

c += a
print "Line 2 - Value of c is ", c

c *= a
print "Line 3 - Value of c is ", c

c /= a
print "Line 4 - Value of c is ", c

c  = 2
c %= a
print "Line 5 - Value of c is ", c

c **= a
print "Line 6 - Value of c is ", c

c //= a
print "Line 7 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864
```

Python Bitwise Operator - Toán tử với Bit trong Python

Các toán tử thực hiện trên `bit` trong Python

Operator	Description	Example
& Binary AND	AND	(a & b) (means 0000 1100)
| Binary OR	OR	(a | b) = 61 (means 0011 1101)
^ Binary XOR	XOR	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	Dịch bit sang trái	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	Dịch bit sang phải	a >> 2 = 15 (means 0000 1111)

- Ví dụ:

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0

c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c = a | b;        # 61 = 0011 1101
print "Line 2 - Value of c is ", c

c = a ^ b;        # 49 = 0011 0001
print "Line 3 - Value of c is ", c
```

```
c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c

c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;       # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Python Identity Operator - Toán tử định danh trong Python

Toán tử định danh (Identity) được thực hiện bằng cách so sánh vị trí bộ nhớ của hai đối tượng. Có hai toán tử Identity bao gồm:

Operator	Description	Example
is	Đánh giá là **true** nếu các biến ở hai bên của toán tử trỏ đến cùng một đối tượng và ngược lại là **false**.	
is not	Đánh giá là false nếu các biến ở hai bên của toán tử trỏ đến cùng một đối tượng và ngược lại là **true**.	

- Ví dụ:

```
a = 20
b = 20

if ( a is b ):
   print "Line 1 - a and b have same identity"
else:
   print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):
   print "Line 2 - a and b have same identity"
else:
   print "Line 2 - a and b do not have same identity"

b = 30
if ( a is b ):
```

```
   print "Line 3 - a and b have same identity"
else:
   print "Line 3 - a and b do not have same identity"

if ( a is not b ):
   print "Line 4 - a and b do not have same identity"
else:
   print "Line 4 - a and b have same identity"
```

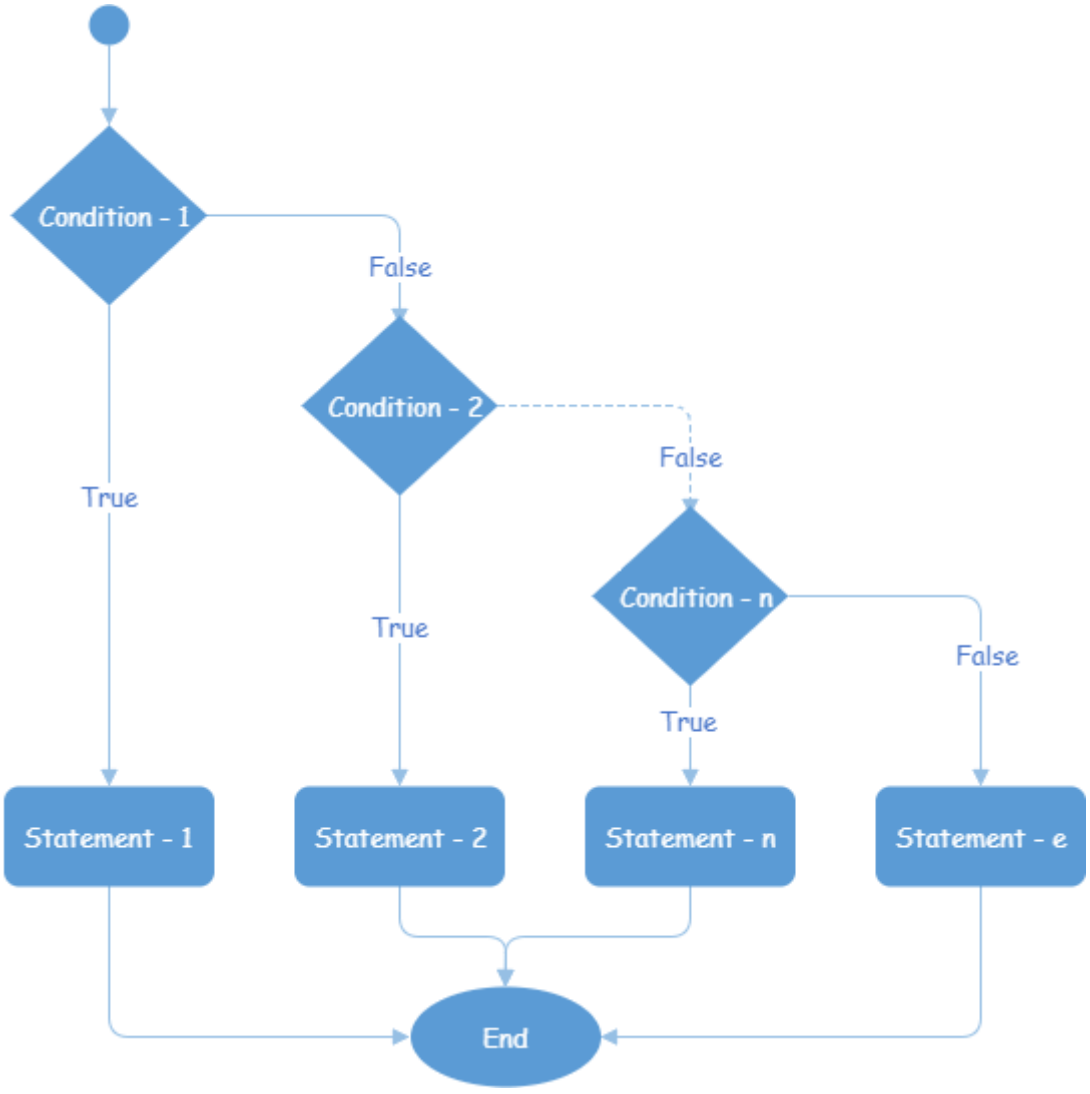
Output:

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

Python Decision Making - IF, else, switch, for, while trong Python

Python hỗ trợ một số cấu trúc điều khiển thông dụng (if, for, while). Hầu hết các cấu trúc điều khiển đều dựa vào thụt đầu dòng (indention) để tạo thành một block xử lý, thay vì sử dụng {...} như các ngôn ngữ khác (Java, PHP, Javascript)

1. If elif else



- Cú pháp:

```
if condition1:
    indentedStatementBlockForTrueCondition1
elif condition2:
    indentedStatementBlockForFirstTrueCondition2
elif condition3:
    indentedStatementBlockForFirstTrueCondition3
elif condition4:
    indentedStatementBlockForFirstTrueCondition4
else:
    indentedStatementBlockForEachConditionFalse
```

- Ví dụ:

```
var = 100
if var == 200:
   print "1 - Got a true expression value"
   print var
elif var == 150:
   print "2 - Got a true expression value"
   print var
elif var == 100:
   print "3 - Got a true expression value"
   print var
else:
   print "4 - Got a false expression value"
   print var

print "Good bye!"
```

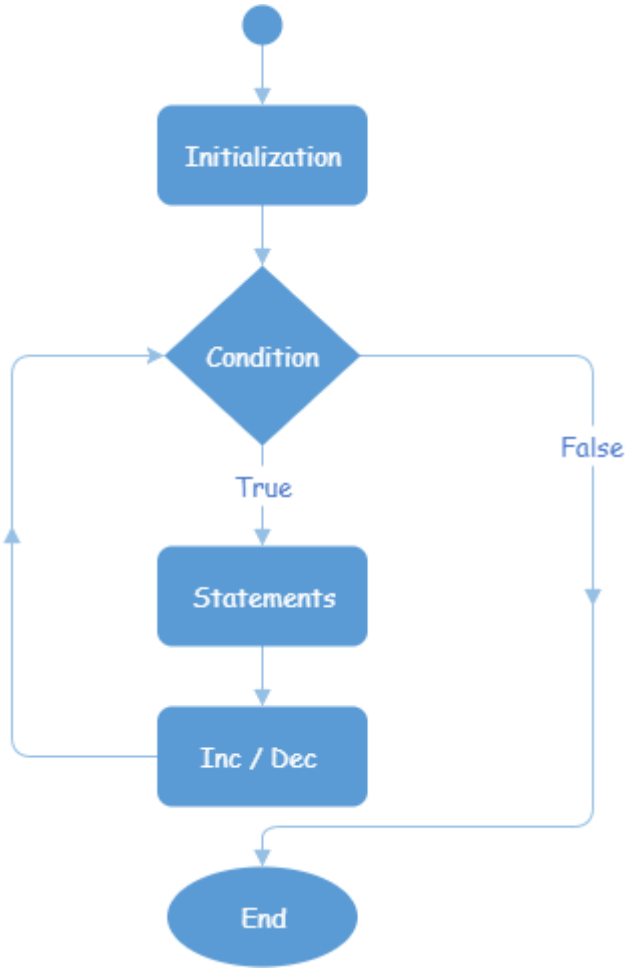
Output:

```
3 - Got a true expression value
100
Good bye!
```

2. Switch case

Python không có cấu trúc switch case

3. For in



- Cú pháp:

```
for iterating_var in sequence:
    statements(s)
```

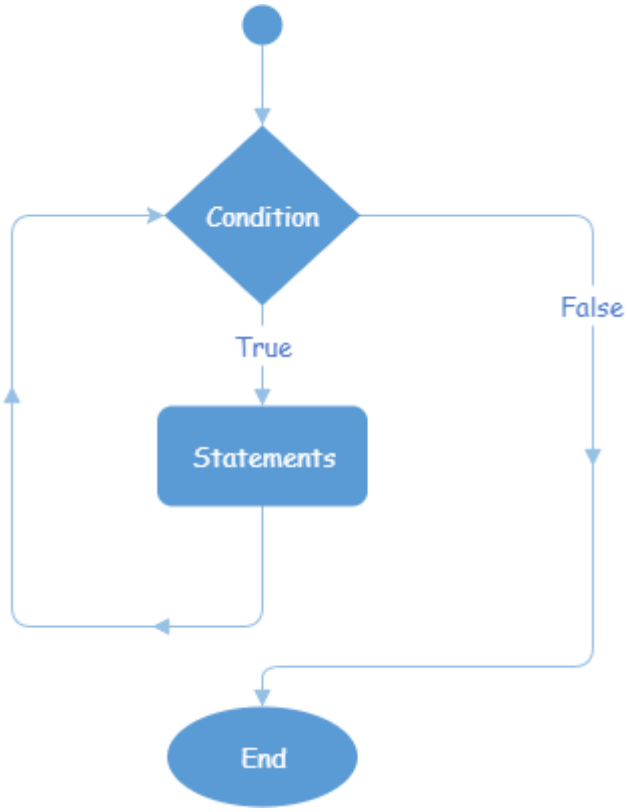
- Ví dụ:

```
for letter in 'Python':  # First Example
    print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:  # Second Example
    print 'Current fruit :', fruit

print "Good bye!"
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

4. While



- Cú pháp:

```
while expression:
    statement(s)
```

- Ví dụ:

```
count = 0
while (count < 9):
    print 'The count is:', count
```

```
    count = count + 1

print "Good bye!"
```

Output:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Python Function - Hàm trong Python

Python function (*hàm*) là một khối lệnh đặc biệt giúp mã chương trình dễ đọc hơn, được đặt tên và có thể gọi để sử dụng ở các nơi khác nhau trong chương trình.

1. Cú pháp

```
def functionname(param, param2,..):
    statements(s)
```

Hàm nếu không trả dữ liệu thì mặc định sẽ trả về giá trị `None`

Ví dụ: Khai báo hàm tính và trả về giá trị tổng của 2 tham số đầu vào:

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

Cách gọi hàm:

```
sum(1, 2)
(trả về giá trị là 3)
```

2. Hàm có đối số mặc định

Hàm có hỗ trợ giá trị mặc định cho tham số khi không truyền vào.

Ví dụ hàm sau:

```
def plus(c, d = 10):
return (c+d)
```

Nếu gọi hàm trên như sau:

```
plus(2)
(kết quả trả về là 12)
```

Một khác biệt trong cách gọi hàm của Python so với PHP, Java... là chúng ta có thể thay đổi thứ tự tham số truyền vào bằng cách đặt tên tham số khi gọi hàm.

Ví dụ: Ta có thể gọi hàm sum(a,b) ở ví dụ trên bằng cách truyền tham số b trước a như sau:

```
sum(b = 1, a = 10)
```

Python String - Kiểu chuỗi trong Python

1. Python String là gì

String (chuỗi) là một trong số những kiểu dữ liệu được sử dụng nhiều nhất trong Python. Một chuỗi có thể khai báo bằng dấu nháy đôi " hoặc đơn '.

Ví dụ:

```
str1 = "Hello"
str2 = ' world'
```

2. Truy cập ký tự trong String

Có thể truy xuất từng ký tự trong một String theo hình thức index,

Ví dụ:

```
var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

3. Khai báo String trên nhiều dòng sử dụng 3 dấu nháy

Có thể sử dụng 3 dấu nháy (đôi hoặc đơn) để khai báo chuỗi trên nhiều dòng.

Ví dụ:

```
paragraph = """This is line 1
This is line 2
This is line 3"""
```

4. Các toán tử trên String

4.1. Nối chuỗi

Có thể tạo một chuỗi dài từ việc nối các chuỗi lại theo cú pháp:

```
str = str1 + " " + str2
```

4.2. Trích xuất chuỗi con

Có thể tạo các chuỗi con thông qua toán tử lấy khoản [start:end] (range). Mặc định start là từ vị trí đầu chuỗi
(0) và end là đến vị trí cuối chuỗi.

Ví dụ:

```
str = 'Hello world'
print str[0:4]
(Hiển thị "Hell")
print str[:4]
(Hiển thị "Hell")
print str[-3:]
(Hiển thị "rld")
print str[6:-3]
(Hiển thị "wo")
```

4.3. Lấy độ dài của chuỗi

Sử dụng hàm len(...) để trả về độ dài của chuỗi.

Ví dụ:

```
count = len("Hello world")
(count có giá trị 11)
```

4.4. Tìm & thay thế nội dung

Có thể tìm và thay thế trong chuỗi bằng cách gọi phươngthức `replace(search, replace[, max])` của một chuỗi.

Ví dụ:

```
str = 'Hello world'
newstr = str.replace('Hello', 'Bye')
print newstr
```

```
(Sẽ hiển thị chuỗi "Bye world" trên màn hình)
```

4.5. Tìm vị trí chuỗi con

Có thể tìm vị trí của một chuỗi con trong chuỗi lớn bằng cách gọi phương thức find(str, beg=0, end=len(string)). Bắt đầu là vị trí 0, nếu không tìm ra thì trả về -1.

Ví dụ:

```
str = 'Hello world'
print str.find('world')
(hiển thị 6)
print str.find('Bye');
(hiển thị -1)
```

Hàm `find()` sẽ tìm theo thứ tự từ trái qua phải của chuỗi,tức là từ lần xuất hiện đầu tiên. Có thể dùng hàm `rfind()` để tìm theo vị trí từ cuối chuỗi về phía trước.

4.6. Tách chuỗi

Có thể tách chuỗi dựa theo một chuỗi delimeter bằng cách gọi phương thức split(str="", num=string.count(str)) .

Ví dụ:

```
str = 'Hello world'
print str.split(' ')
(Trả về một mảng có 2 phần tử là 2 chuỗi "Hello" và "world")
```

Có thể sử dụng hàm `splitlines()` để tách chuỗi theo từng hàng và loại bỏ ký tự `NEWLINE`.

4.7. Trim ký tự khoảng trắng

Có thể loại bỏ các ký tự (mặc định là ký tự khoảng trắng) trước và sau một chuỗi, bằng cách gọi các phương thức sau:

- `strip([chars]):` loại bỏ trước và sau chuỗi
- `lstrip([chars]):` loại bỏ phía trước chuỗi
- `rstrip([chars]):` loại bỏ phía sau chuỗi

4.8. Một số hàm xử lý chuỗi

- `isnumeric():` Kiểm tra một chuỗi có phải là chuỗi số
- `lower():` Chuyển chuỗi hết thành chữ thường
- `upper():` Chuyển chuỗi hết thành chữ HOA

Python List - Kiểu List trong Python

List trong Python là cấu trúc mảng và các phần tử có index có thứ tự. Không như PHP, key của một mảng có thể vừa
là số, vừa là chuỗi (associated array).

Trong Python, muốn tạo một mảng có key là chuỗi thì sẽ sử dụng cấu trúc Dictionary (phần tiếp). Trong phần
này, chúng ta sẽ nói đến List. Một List được khai báo như mảng trong .JSON. Sử dụng [...] để khai báo một mảng.

Ví dụ:

```
numbers = [1, , , , ]


names = ['Marry', 'Peter']
```

Có thể truy xuất từng phần tử của mảng bằng index, phần tử đầu tiên có thứ tự là `0`

Ví dụ:

```
Print numbers[0]
(Hiển thị 1)

Print numbers[-3]
(Hiển thị 3)

Print names[1]
(Hiển thị 'Peter')
```

Để biết được số lượng phần tử của 1 List, có thể sử dụng hàm `len(array)` để lấy số lượng phần tử của mảng tham
số truyền vào.

1. Kiểm tra sự tồn tại của một phần tử

a. Kiểm tra theo index

Trong nhiều trường hợp bạn muốn truy xuất một phần tử bất kỳ (dựa vào index) của mảng thì nếu truy xuất đến một
phần tử không tồn tại thì ứng dụng sẽ báo lỗi. Do đó, trước khi truy xuất một phần tử, bạn cần kiểm tra xem phần tử này đã tồn tại hay chưa. Hiện tại python không hỗ trợ hàm nào để kiểm tra sự tồn tại của một phần tử trong mảng.

Có 2 cách thường thấy để kiểm tra đó là "Look before you leap" (LBYL) và "Easier to ask forgiveness than
permission" (EAFP).

Ví dụ về "Look before you leap (LBYL)":

```
1f index < len(array):
□array[index]

else:
□# handle this
```

Ví dụ về "Easier to ask forgiveness than permission" (EAFP):

```
try:
□array[Lindex]

except IndexError:
□# handle this
```

b. Kiểm tra theo giá trị

Để kiểm tra một giá trị có tồn tại / không tồn tại trong mảng hay không thì có thể sử dụng toán tử `in` / `not in`.

Ví dụ:

```
mylist = ['a', 'b', 'c']

Print 'a' in mylist
(Hi□n thị True)

Print 'P' not 1n mylist
(Hi□n thị False)
```

2. Trích xuất mảng con

Tương tự như chuỗi, tó thể tạo các mảng con thông qua toán tử lấy khoản `[start: end]` (range). Mặc định start là
từ vị trí đầu chuỗi `0` và `end` là đến vị trí cuối chuỗi.

Ví dụ:

```
numbers = ['a', 'b', 'c', 'd']

Print numbers[:2]
(Hiển thị ['a', 'b'])

Print numbers[-2:]
(Hiển thị ['c', 'd'])
```

3. Xóa phần tử của mảng

Có thể xóa một phần tử thông qua toán tử `del`. Thứ tự của các phần tử sẽ dịch chuyển tùy vào vị trí của phần tử bị
xóa.

Ví dụ:

```
numbers = [1, 2, 3, 4, 5]
del numbers[0]

Print numbers

(Hiển thị [2, 3, 4, 5)
```

Bạn có thể xóa một khoản dựa vào toán tử lấy khoảng [start:end]

Ví dụ:

```
numbers = |1, 2, 3, 4, 5, 6, 7]
del numbers[2:4]

Print numbers

(Hiển thị [1, 2, 5, 6, 7])
```

4. Nối 2 mảng

Bạn có thể sử dụng toán tử + để nối giá trị của 2 mảng và tạo ra một mảng lớn có số lượng phần tử là tổng số lượng
phần tử của 2 mảng con.

Ví dụ:

```
a = [1,2]
b = [1,3]

Print a + b
(Hiển thị [1, 2, 1, 3])
```

5. Thêm phần tử vào mảng

Nếu bạn muốn thêm phần tử vào một mảng đã tồn tại, hãy dùng phương thức `list.append (newvalue)` để thêm phần
tử có giá trị `newvalue` vào cuối mảng `list`.

Ví dụ:

```
numbers = [1,2,3]
numbers.append(4)

Print numbers

(Hiển thị [1, 2, 3, 4])
```

6. Lấy phần tử cuối mảng

Nếu muốn lấy phần tử cuối cùng của mảng ra khỏi mảng, có thể sử dụng phương thức `list.pop()`, sẽ trả về giá trị
của phần tử cuối cùng và mảng bây giờ sẽ không còn phần tử này.

Ví dụ:

```
numbers = [1,2,3]

mynumber = numbers.pop()

Print mynumber

(Hiển thị 3)
```

```
Print numbers
(Hiển thị [1, 2])
```

7. Tìm một giá trị trong mảng

Nếu bạn muốn tìm vị trí (index) của một giá trị trong một mảng, có thể dùng phương thức `list.index(obj)`. Nếu
tìm thấy sẽ trả về index của phần tử đầu tiên tìm thấy. Nếu không tìm thấy sẽ có Exception.

Ví dụ:

```
aList = [123, 'xyz', 'zara', 'abc'];
Print "Index for xyz : ", aLlist.index('xyz')
Print "Index for zara : ", alist.index('zara')

Khi chạy sẽ hiển thị kết quả:

Index for xyz : 1

Index for zara : 2
```

8. Đảo ngược giá trị của mảng

Để đảo ngược thứ tự các giá trị của một mảng, sử dụng phương thức `list.reverse()`. Phương thức này không trả
về kết quả mà thay đổi trực tiếp mảng `list`.

Ví dụ:

```
numbers = |1, 2, 3, 4]
numbers.reverse()

Print numbers

(Hiển thị [4, 3, 2, 1])
```

9. Sắp xếp giá trị các phần tử

Để sắp xếp thứ tự của giá trị trong mảng, sử dụng phương thức `list.sort([func])` để sắp xếp. Nếu tham số đầu
vào là hàm func không truyên vào thì mặc định là sắp xếp theo giá trị tăng dần. Phương thức này không trả về kết
quả mà thay đổi trực tiếp mảng `list`

Ví dụ:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.sort()

Print "List : ", aList

(Hiển thị List : 123 ; 'abc', 'xyz', 'xyz', 'zara'])
```

Cách triển khai hàm compare func() cũng giống như hàm usort trong PHP. Hàm trả về các giá trị o, -1 và 1.

Python Tuple - Tuple trong Python

Python Tuple cũng là một cấu trúc mảng, tương tự như cấu trúc List. Một số điểm khác nhau cơ bản là khai báo Tuple sử
dụng cặp dấu ngoặc `(...)` và một tuple đã được khai báo rồi thì không thay đổi được giá trị (immutable) và không hỗ
trợ các phương thức như append(), pop() ....

Ví dụ:

```
mytuple = ('x', 'y', 'z')
print mytuple
(Hiển thị ('x', 'y', z'))
```

Tuple vẫn hỗ trợ các cách để truy xuất phần tử giống List như là truy xuất theo index, range, tìm kiếm...

1. So sánh List và Tuple

List và Tuple đều là một dẫy (sequence) các phần tử. Chúng có các khác biệt sau:

- Khi viết một List bạn sử dụng cặp dấu ngặc vuông [ ], trong khi viết một Tuple bạn sử dụng dấu ngặc tròn ()

```
# Đây là một Tuple
aTuple = ("apple", "apricot", "banana")
# Đây là một List (Danh sách)
aList = ["apple", "apricot", "banana"
```

- List là kiểu dữ liệu có thể biến đổi (mutable), bạn có thể sử dụng phương thức như append() để thêm phần tử vào List, hoặc sử dụng phương thức remove() để xóa các phần tử ra khỏi List mà không làm tạo ra thêm một thực thể 'List' khác trên bộ nhớ.

```
list1 = [1990, 1991, 1992]


print ("list1: ", list1)
```

```
# Địa chỉ của list1 trên bộ nhớ.
list1Address = hex ( id(list1) )

print ("Address of list1: ", list1Address )

print ("\n")
print ("Append element 2001 to list1")

# Nối (append) một phần tử vào list1.
list1.append(2001)

print ("list1 (After append): ", list1)

# Địa chỉ của list1 trên bộ nhớ.
list1Address = hex ( id(list1) )

print ("Address of list1 (After append): ", list1Address )
```

- Tuple là một đối tượng bất biến (immutable), nó không có các phương thức append(), remove(),.. như list. Một số phương thức, hoặc toán tử mà bạn nghĩ rằng nó dùng để cập nhập Tuple, nhưng không phải vậy, nó dựa trên Tuple ban đầu để tạo ra một Tuple mới.

```
tuple1 = (1990, 1991, 1992)

# Địa chỉ của tuple1 trên bộ nhớ.
tuple1Address = hex ( id(tuple1) )

print ("Address of tuple1: ", tuple1Address )

# Nối một tuple vào tuple1.
tuple1 = tuple1 + (2001, 2002)

# Địa chỉ của tuple1 trên bộ nhớ.
tuple1Address = hex ( id(tuple1) )

print ("Address of tuple1 (After concat): ", tuple1Address )
```

2. Truy cập các phần tử của Tuples

- Sử dụng vòng lặp for để truy cập vào các phần tử của Tuple:

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

for fruit in fruits :
    print ("Fruit: ", fruit)
```

- Truy cập thông qua chỉ số (index):

Bạn cũng có thể truy cập vào các phần tử của Tuple thông qua chỉ số. Các phần tử của Tuple được đánh chỉ chỉ từ trái sang phải, bắt đầu từ 0.

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

print (  fruits )

# Số phần tử
print ("Element count: ", len( fruits) )



for i in range ( 0, len( fruits) ) :
    print ("Element at ", i, "= ", fruits[ i] )

# Một Tuple con chứa các phần tử từ index 1 đến 4 (1, 2, 3)
subTuple = fruits[ 1: 4]

# ('apricot', 'banana', 'coconut')
print ("Sub Tuple [ 1: 4] ", subTuple )
```

Bạn cũng có thể truy cập vào các phần tử của Tuple theo chỉ số âm (Negative index), các phần tử được đánh chỉ số từ phải sang trái với các giá trị -1, -2, ...

```
fruits = ("apple", "apricot", "banana", "coconut", "lemen", "plum", "pear")

print (  fruits )

print ("Element count: ", len( fruits) )

print ("fruits[-1]: ", fruits[-1])
print ("fruits[-2]: ", fruits[-2])

subTuple1 = fruits[-4: ]
```

```
print ("\n")
print ("Sub Tuple fruits[-4: ] ")
print (subTuple1)

subTuple2 = fruits[-4:-2]

print ("\n")
print ("Sub Tuple fruits[-4:-2] ")
print (subTuple2)
```

3. Cập nhập Tuples

Chú ý rằng Tuple là bất biến (immutable), vì vậy nó chỉ có các phương thức hoặc toán tử để truy cập, hoặc tạo ra một Tuple mới từ các Tuple ban đầu.

```
tuple1 = (1990, 1991, 1992)

print ("tuple1: ", tuple1)

print ("Concat (2001, 2002) to tuple1")

tuple2 = tuple1 + (2001, 2002)

print ("tuple2: ", tuple2)

# Một Tuple con, chứa các phần tử từ chỉ số 1 đến 4 (1,2,3)
tuple3 = tuple2[1:4]

print ("tuple2[1:4]: ", tuple3)

# Một Tuple con, chứa các phần tử từ chỉ số 1 đến cuối.
tuple4 = tuple2[1: ]

print ("tuple2[1: ]: ", tuple4)
```