Python (ngôn ngữ lập trình)

Bởi:

Lê Văn Tâm

Python là một ngôn ngữ lập trình thông dịch do Guido van Rossum tạo ra năm 1990. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động; do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

Theo đánh giá của Eric S. Raymond, 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, như nhận định của chính Guido van Rossum trong một bài phỏng vấn ông.

Ban đầu, Python được phát triển để chạy trên nền Unix. Nhưng rồi theo thời gian, nó đã "bành trướng" sang mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ chốt trong việc quyết định hướng phát triển của Python.

Lịch sử

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 đâu tiền 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. Các đặc điểm mới của Python 3.0 sẽ được trình bày phần cuối bài này.

Đặc điểm

Dễ học, dễ đọc

Python được thiết kế để trở thành một ngôn ngữ dễ học, mã nguồn dễ đọc, bố cục trực quan, dễ hiểu, thể hiện qua các điểm sau:

Từ khóa

- Python tăng cường sử dụng từ khóa tiếng Anh, hạn chế các kí hiệu và cấu trúc cú pháp so với các ngôn ngữ khác.
- Python là một ngôn ngữ phân biệt kiểu chữ HOA, chữ thường.
- Như C/C++, các từ khóa của Python đều ở dạng chữ thường.

Khối lệnh

Trong các ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp kí hiệu hoặc từ khóa. Ví dụ, trong C/C++, cặp ngoặc nhọn { } được dùng để bao bọc một khối lệnh. Python, trái lại, có một cách rất đặc biệt để tạo khối lệnh, đó là thụt các câu lệnh trong

khối vào sâu hơn (về bên phải) so với các câu lệnh của khối lệnh cha chứa nó. Ví dụ, giả sử có đoạn mã sau trong C/C++:

```
#include <math.h> //... delta = b * b - 4 * a * c; if
(delta > 0) { // Khối lệnh mới bắt đầu từ kí tự { đến } x1
= (- b + sqrt(delta)) / (2 * a); x2 = (- b - sqrt(delta))
/ (2 * a); printf("Phuong trinh co hai nghiem phan
biet:\n"); printf("x1 = %f; x2 = %f", x1, x2); }
```

Đoạn mã trên có thể được viết lại bằng Python như sau:

```
import math #... delta = b * b - 4 * a * c if delta > 0: #
Khối lệnh mới, thụt vào đầu dòng x1 = (- b +
math.sqrt(delta)) / (2 * a) x2 = (- b - math.sqrt(delta))
/ (2 * a) print "Phuong trinh co hai nghiem phan biet:"
print "x1 = ", x1, "; ", "x2 = ", x2
```

Ta có thể sử dụng dấu tab hoặc khoảng trống để thụt các câu lệnh vào.

Các bản hiện thực

Python được viết từ những ngôn ngữ khác, tạo ra những bản hiện thực khác nhau. Bản hiện thực Python chính, còn gọi là CPython, được viết bằng C, và được phân phối kèm một thư viện chuẩn lớn được viết hỗn hợp bằng C và Python. CPython có thể chạy trên nhiều nền và khả chuyển trên nhiều nền khác. Dưới đây là các nền trên đó, CPython có thể chạy.

- Các hệ điều hành họ Unix: AIX, Darwin, FreeBSD, Mac OS X, NetBSD, Linux, OpenBSD, Solaris,…
- Các hệ điều hành dành cho máy desktop: Amiga, AROS, BeOS, Mac OS 9, Microsoft Windows, OS/2, RISC OS.
- Các hệ thống nhúng và các hệ đặc biệt: GP2X, Máy ảo Java, Nokia 770 Internet Tablet, Palm OS, PlayStation 2, PlayStation Portable, Psion, QNX, Sharp Zaurus, Symbian OS, Windows CE/Pocket PC, Xbox/XBMC, VxWorks.
- Các hệ máy tính lớn và các hệ khác: AS/400, OS/390, Plan 9 from Bell Labs, VMS, z/OS.

Ngoài CPython, còn có hai hiện thực Python khác: Jython cho môi trường Java và IronPython cho môi trường .NET và Mono.

Khả năng mở rộng

Python có thể được mở rộng: nếu ta biết sử dụng C, ta có thể dễ dàng viết và tích hợp vào Python nhiều hàm tùy theo nhu cầu. Các hàm này sẽ trở thành hàm xây dựng sẵn (built-in) của Python. Ta cũng có thể mở rộng chức năng của trình thông dịch, hoặc liên kết các chương trình Python với các thư viện chỉ ở dạng nhị phân (như các thư viện đồ họa do nhà sản xuất thiết bị cung cấp). Hơn thế nữa, ta cũng có thể liên kết trình thông dịch của Python với các ứng dụng viết từ C và sử dụng nó như là một mở rộng hoặc một ngôn ngữ dòng lệnh phụ trợ cho ứng dụng đó.

Trình thông dịch

Python là một ngôn ngữ lập trình dạng thông dịch, do đó có ưu điểm tiết kiệm thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác. Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix, tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ; hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm. Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

Lệnh và cấu trúc điều khiển

Mỗi câu lệnh trong Python nằm trên một dòng mã nguồn. Ta không cần phải kết thúc câu lệnh bằng bất kì kí tự gì. Cũng như các ngôn ngữ khác, Python cũng có các cấu trúc điều khiển. Chúng bao gồm:

Cấu trúc rẽ nhánh: cấu trúc if (có thể sử dụng thêm elif hoặc else), dùng để thực thi có điều kiện một khối mã cụ thể.

Cấu trúc lặp, bao gồm:

- Lệnh while: chạy một khối mã cụ thể cho đến khi điều kiện lặp có giá trị false.
- Vòng lặp for: lặp qua từng phần tử của một dãy, mỗi phần tử sẽ được đưa vào biến cục bộ để sử dụng với khối mã trong vòng lặp.

Python cũng có từ khóa class dùng để khai báo lớp (sử dụng trong lập trình hướng đối tượng) và lệnh def dùng để định nghĩa hàm.

Hệ thống kiểu dữ liệu

Python sử dụng hệ thống kiểu duck typing, còn gọi là latent typing (tự động xác định kiểu). Có nghĩa là, Python không kiểm tra các ràng buộc về kiểu dữ liệu tại thời điểm dịch, mà là tại thời điểm thực thi. Khi thực thi, nếu một thao tác trên một đối tượng bị thất bại, thì có nghĩa là đối tượng đó không sử dụng một kiểu thích hợp.

Python cũng là một ngôn ngữ định kiểu mạnh. Nó cấm mọi thao tác không hợp lệ, ví dụ cộng một con số vào chuỗi kí tự.

Sử dụng Python, ta không cần phải khai báo biến. Biến được xem là đã khai báo nếu nó được gán một giá trị lần đầu tiên. Căn cứ vào mỗi lần gán, Python sẽ tự động xác định kiểu dữ liệu của biến. Python có một số kiểu dữ liệu thông dụng sau:

- int, long: số nguyên (trong phiên bản 3.x long được nhập vào trong kiểu int). Độ dài của kiểu số nguyên là tùy ý, chỉ bị giới hạn bởi bộ nhớ máy tính.
- float: số thực
- complex: số phức, chẳng hạn 5+4j
- list: dãy trong đó các phần tử của nó có thể được thay đổi, chẳng hạn [8, 2, 'b', -1.5] . Kiểu dãy khác với kiểu mảng (array) thường gặp trong các ngôn ngữ lập trình ở chỗ các phần tử của dãy không nhất thiết có kiểu giống nhau. Ngoài ra phần tử của dãy còn có thể là một dãy khác.
- tuple: dãy trong đó các phần tử của nó không thể thay đổi.
- str: chuỗi kí tự. Từng kí tự trong chuỗi không thể thay đổi. Chuỗi kí tự được đặt trong dấu nháy đơn, hoặc nháy kép.
- dict: từ điển, còn gọi là "hashtable": là một cặp các dữ liệu được gắn theo kiểu {từ khóa: giá trị}, trong đó các từ khóa trong một từ điển nhất thiết phải khác nhau. Chẳng hạn {1: "Python", 2: "Pascal"}
- set: một tập không xếp theo thứ tự, ở đó, mỗi phần tử chỉ xuất hiện một lần.

Ngoài ra, Python còn có nhiều kiểu dữ liệu khác. Xem thêm trong phần "Các kiểu dữ liệu" bên dưới.

Module

Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Nó cũng cung cấp sẵn một tập hợp các modules chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng (socket),…

Đa năng

Python là một ngôn ngữ lập trình đơn giản nhưng rất hiệu quả.

- So với Unix shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.
- So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn. Nó cũng có sẵn nhiều kiểu dữ liệu cấp cao, ví dụ như các mảng (array) linh hoạt và từ điển (dictionary) mà ta sẽ phải mất nhiều thời gian nếu viết bằng C.

Python là một ngôn ngữ lập trình cấp cao có thể đáp ứng phần lớn yêu cầu của lập trình viên:

- Python thích hợp với các chương trình lớn hơn cả AWK và Perl.
- Python được sử dụng để lập trình Web. Nó có thể được sử dụng như một ngôn ngữ kịch bản.
- Python được thiết kế để có thể nhúng và phục vụ như một ngôn ngữ kịch bản để tuỳ biến và mở rộng các ứng dụng lớn hơn.
- Python được tích hợp sẵn nhiều công cụ và có một thư viện chuẩn phong phú, Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, sử dụng các thành phần COM hay CORBA, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ đánh dấu khác. Python cũng được cung cấp các thư viện xử lý các giao thức Internet thông dụng như HTTP, FTP,…
- Python có khả năng giao tiếp đến hầu hết các loại cơ sở dữ liệu, có khả năng xử lí văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.
- Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, MayaVi 3D Visualization Toolkits, Numeric Python, ScientificPython,…
- Python có thể được sử dụng để phát triển các ứng dụng desktop. Lập trình viên có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa (GUI) chất lượng cao. Python còn hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, Delphi, X11, Motif, Tk, Fox, FLTK, …
- Python cũng có sẵn một unit testing framework để tạo ra các các bộ test (test suites).

Multiple paradigms (đa biến hóa)

Python là một ngôn ngữ đa biến hóa (multiple paradigms). Có nghĩa là, thay vì ép buộc mọi người phải sử dụng duy nhất một phương pháp lập trình, Python lại cho phép sử dụng nhiều phương pháp lập trình khác nhau: hướng đối tượng, có cấu trúc, chức năng, hoặc chỉ hướng đến một khía cạnh. Python kiểu kiểu động và sử dụng bộ thu gom rác để quản lí bộ nhớ. Một đặc điểm quan trọng nữa của Python là giải pháp tên động, kết nối tên biến và tên phương thức lại với nhau trong suốt thực thi của chương trình.

Sự tương đương giữa true và một giá trị khác 0

Cũng như C/C++, bất kì một giá trị khác 0 nào cũng tương đương với true và ngược lại, một giá trị 0 tương đương với false. Như vậy:

```
if a != 0:
```

tương đương với

```
if a:
```

Cú pháp

Sau đây là cú pháp cơ bản nhất của ngôn ngữ Python:

Toán tử

```
+ - * / // (chia làm tròn) % (phần dư) ** (lũy thừa) ~
(not) & (and) | (or) ^ (xor) << (left shift) >> (right
shift) == (bằng) <= >= != (khác)
```

Python sử dụng kí pháp trung tố thường gặp trong các ngôn ngữ lập trình khác.

Các kiểu dữ liệu

Kiểu số

```
1234585396326 (số nguyên dài vô hạn) -86.12 7.84E-04 2j 3
+ 8j (số phức)
```

Kiểu chuỗi (string)

```
"Hello" "It's me" '"OK"-he replied'
```

Kiểu bộ (tuple)

```
(1, 2.0, 3) (1,) ("Hello",1,())
```

Kiểu danh sách (list)

```
[4.8, -6] ['a','b']
```

Kiểu từ điển (dictionary)

```
{"Vietnam":"Hanoi",              "Netherlands":"Amsterdam",
"France":"Paris"}
```

Chú thích

```
# dòng chú thích
```

Lệnh gán

```
tên biến = biểu thức x = 23.8 y = -x ** 2 z1 = z2 = x + y
loiChao = "Hello!" i += 1 # tăng biến i thêm 1 đơn vị
```

In giá trị

```
print biểu thức print (7 + 8) / 2.0 print (2 + 3j) * (4 -
6j)
```

Nội suy chuỗi (string interpolation)

```
print "Hello %s" %("world!") print "i = %d" %i print "a =
%.2f and b = %.3f" %(a,b)
```

Cấu trúc rẽ nhánh

Dạng 1:

```
if biểu_thức_điều_kiện: # lệnh ...
```

Dạng 2:

```
if biểu_thức_điều_kiện: # lệnh ... else: # lệnh ...
```

Dạng 3:

```
if biểu_thức_điều_kiện_1: # lệnh ... (được thực hiện nếu
biểu_thức_điều_kiện_1 là đúng/true) elif
biểu_thức_điều_kiện_2: # lệnh ... (được thực hiện nếu
biểu_thức_điều_kiện_1 là sai/false, nhưng
biểu_thức_điều_kiện_2 là đúng/true) else: # lệnh ... (được
thực hiện nếu tất cả các biểu thức điều kiện đi kèm if và
elif đều sai)
```

Cấu trúc lặp

```
while biểu_thức_đúng: # lệnh ... for phần_tử in dãy: #
lệnh ... L = ["Ha Noi", "Hai Phong", "TP Ho Chi Minh"] for
thanhPho in L: print thanhPho for i in range(10): print i
```

Hàm

```
def tên_hàm (tham_biến_1, tham_biến_2, tham_biến_n): # lệnh ... return giá_trị_hàm def binhPhuong(x): return x*x
```

Hàm với tham số mặc định:

```
def luyThua(x, n=2): """Lũy thừa với số mũ mặc định là 2""" return x**n print luyThua(3) # 9 print luyThua(2,3) # 8
```

Lớp

```
class Tên_Lớp_1: # ... class Tên_Lớp_2(Tên_Lớp_1): """Lớp 2 kế thừa lớp 1""" x = 3 # biến thành viên của lớp # def phương_thức(self, tham_biến): # ... # khởi tạo a = Tên_Lớp_2() print a.x print a.phương_thức(m) # m là giá trị gán cho tham biến List Comprehension
```

List Comprehension là dạng cú pháp đặc biệt (syntactic sugar) (mới có từ Python 2.x) cho phép thao tác trên toàn bộ dãy (list) mà không cần viết rõ vòng lặp. Chẳng hạn y là một dãy mà mỗi phần tử của nó bằng bình phương của từng phần tử trong dãy x:

```
y = [xi**2 for xi in x]
```

Xử lý lỗi

```
try: câu_lệnh except Loại_Lỗi: thông báo lỗi
```

Tốc độ thực hiện

Là một ngôn ngữ thông dịch, Python có tốc độ thực hiện chậm hơn nhiều lần so với các ngôn ngữ biên dịch như Fortran, C, v.v... Trong số các ngôn ngữ thông dịch, Python được đánh giá nhanh hơn Ruby và Tcl, nhưng chậm hơn Lua.

Các đặc điểm mới trong Python 3.x

Nội dung phần này được trích từ tài liệu của Guido van Rossum. Phần này không liệt kê đầy đủ tất cả các đặc điểm; chi tiết xin xem tài liệu nói trên.

Một số thay đổi cần lưu ý nhất

Lệnh print trở thành hàm print. Theo đó sau print() ta cần nhớ gõ vào cặp ngoặc ():

```
print("Hello") print(2+3)
```

Trả lại kết quả không còn là list trong một số trường hợp.

- dict.keys(), dict.items(), dict.values() kết quả cho ra các "view" thay vì list.
- map và filter trả lại các iterator.
- range bây giờ có tác dụng như xrange, và không trả lại list.

So sánh

Không còn hàm cmp, và cmp(a, b) có thể được thay bằng (a > b) - (a < b)

Số nguyên

- Kiểu long được đổi tên thành int.
- 1/2 cho ta kết quả là số thực chứ không phải số nguyên.
- Không còn hằng số sys.maxint
- Kiểu bát phân được kí hiệu bằng 0o thay vì 0, chẳng hạn 0o26.

Phân biệt văn bản - dữ liệu nhị phân thay vì Unicode - chuỗi 8-bit

- Tất cả chuỗi văn bản đều dưới dạng Unicode, nhưng chuỗi Unicode mã hóa lại là dạng dữ liệu nhị phân. Dạng mặc định là UTF-8.
- Không thể viết u"a string" để biểu diễn chuỗi như trong các phiên bản 2.x

Các thay đổi về cú pháp

Cú pháp mới

- Các tham biến chỉ chấp nhận keyword: Các tham biến phía sau *args phải được gọi theo dạng keyword.
- Từ khóa mới nonlocal. Muốn khai báo một biến x với có phạm vi ảnh hưởng rộng hơn, nhưng chưa đến mức toàn cục, ta dùng nonlocal x.
- Gán giá trị vào các phần tử tuple một cách thông minh, chẳng hạn có thể viết (a, *rest, b) = range(5) để có được a = 0; b = [1,2,3]; c = 4.
- Dictionary comprehension, chẳng hạn {k: v for k, v in stuff} thay vì dict(stuff).
- Kiểu nhị phân, chẳng hạn b110001.

Cú pháp được thay đổi

- raise [biểu_thức [from biểu_thức]]
- except lệnh as biến
- Sử dụng metaclass trong đối tượng:

```
class C(metaclass=M): pass
```

Cách dùng biến __metaclass__ không còn được hỗ trợ.

Cú pháp bị loại bỏ

- Không còn dấu ``, thay vì đó, dùng repr.
- Không còn so sánh <> (dùng !=).
- Không còn các lớp kiểu classic.