**Python Programming Language**

**Vấn đề 1. What is Python?**

Python là một ngôn ngữ lập trình cấp cao, đa nền tảng và là ngôn ngữ mã nguồn mở được phát hành theo giấy phép tương thích với GPL. Python Software Foundation (PSF), một tổ chức phi lợi nhuận, giữ bản quyền của Python.

Guido Van Rossum đã tạo ra Python vào cuối những năm 1980. Nó được phát hành vào năm 1991 tại Centrum Wiskunde & Informatica (CWI) ở Hà Lan như một sự kế thừa cho ngôn ngữ ABC. Ông đặt tên cho ngôn ngữ này theo tên một chương trình hài kịch nổi tiếng có tên là 'Monty Python's Flying Circus' (và không phải theo tên Python - con rắn).

Trong vài năm gần đây, Python đã phổ biến vô cùng rộng rãi. Theo khảo sát gần đây của stackoverflow.com, Python nằm trong top ba ngôn ngữ lập trình được yêu thích nhất vào năm 2020.

**Python Features – Những đặc tính của Python**

* Python là một ngôn ngữ dựa trên **trình thông dịch** (interpreter), cho phép thực hiện một lệnh tại một thời điểm.
* Các kiểu dữ liệu cơ bản mở rộng được hỗ trợ, ví dụ: số thực (dấu phẩy động), số nguyên(**độ dài không giới hạn**), số phức, chuỗi (cả ASCII và Unicode), danh sách và từ điển.
* Các biến được **khai báo kiểu tự động** qua toán tử gán.
* Hỗ trợ các khái niệm **lập trình hướng đối tượng** như lớp, kế thừa, đối tượng, mô-đun, v.v.
* Hỗ trợ **xử lý ngoại lệ** sạch hơn.
* Hỗ trợ **quản lý bộ nhớ tự động**.
* **Nhiều mô-đun tích hợp sẵn** và của bên thứ ba, có thể được chèn và sử dụng độc lập trong ứng dụng Python.

**Python Advantages – Những ưu điểm của Python**

* Nâng cao khả năng đọc. Vì mục đích đó, thụt lề thống nhất được sử dụng để phân tách các khối câu lệnh thay vì dấu ngoặc nhọn, giống như trong các ngôn ngữ như C, C ++ và Java.
* Python miễn phí và được phân phối dưới dạng phần mềm mã nguồn mở. Một cộng đồng lập trình lớn đang tích cực tham gia vào việc phát triển và hỗ trợ các thư viện Python cho các ứng dụng khác nhau như khuôn khổ web, máy tính toán học và khoa học dữ liệu.
* Python là một ngôn ngữ đa nền tảng. Nó hoạt động như nhau trên các nền tảng hệ điều hành khác nhau như Windows, Linux, Mac OSX, v.v. Do đó, các ứng dụng Python có thể được chuyển dễ dàng qua các nền tảng hệ điều hành.
* Python hỗ trợ nhiều mô hình lập trình bao gồm các kiểu lập trình mệnh lệnh, thủ tục, hướng đối tượng và chức năng.
* Python là một ngôn ngữ có thể mở rộng, có chức năng bổ sung (ngoài những gì được cung cấp bằng ngôn ngữ cốt lõi) có thể được cung cấp thông qua các mô-đun và các gói được viết bằng các ngôn ngữ khác (C, C ++, Java, v.v.)
* Một DB-API tiêu chuẩn cho kết nối cơ sở dữ liệu đã được định nghĩa bằng Python. Nó có thể được kích hoạt bằng cách sử dụng bất kỳ nguồn dữ liệu nào (Oracle, MySQL, SQLite, v.v.) làm phụ trợ cho chương trình Python để lưu trữ, truy xuất và xử lý dữ liệu.
* Bản phân phối chuẩn của Python chứa bộ công cụ Tkinter GUI, là việc triển khai thư viện GUI phổ biến được gọi là Tcl / Tk. Một GUI hấp dẫn có thể được xây dựng bằng Tkinter. Nhiều thư viện GUI khác như Qt, GTK, WxWidgets, v.v. cũng được chuyển sang Python.
* Python có thể được tích hợp với các công nghệ lập trình phổ biến khác như C, C ++, Java, ActiveX và CORBA.

**Vấn đề 2. Where to use Python?**

* **Data Science** (phân tích dữ liệu): với các thư viện **numpy**, **pandas**, **Matplotlib**, **Anaconda**
* **Machine Learning** (học máy): với các thư viện **Scikit-learn, Tensoflow, NLTK**
* **Web Development** (phát triển Web): với các thư viện **django, Pyramid, Flask**
* **Image Processing** (xử lý ảnh): với thư viện **OpenCV**
* **Game Development** (phát triển Game): với thư viện **PyGame**
* **Embedded Systems and IoT (hệ thống nhúng và IoT)**: Một phiên bản nhỏ gọn của Python có tên là **Micropython** đã được phát triển, đặc biệt dành cho vi điều khiển.
* **Android Apps** (phát triển hệ điều hành Android): với thư viện **Kivy library**
* **Automated Jobs** (công việc tự động): viết các tác vụ tự động của hệ điều hành và là ngôn ngữ nhúng trong nhiều phần mềm phổ biến. Điều này tương tự như VBA được nhúng để viết các macro trong Excel, PowerPoint,…
* **Rapid Development Tool** (công cụ phát triển nhanh chóng): Python, Java hay c# có thể sử dụng tham chiếu các thư viện, nhờ đó thời gian phát triển sản phẩm được giảm tối thiểu.

**Các công cụ (Tools) và khuôn khổ (Frameworks) quan trọng để phát triển các loại ứng dụng Python**

* **Web Development**: [Django](https://www.djangoproject.com/), [Pyramid](http://www.pylonsproject.org/), [Bottle](http://bottlepy.org/), [Tornado](http://tornadoweb.org/), [Flask](http://flask.pocoo.org/), [web2py](http://www.web2py.com/)
* **GUI Development**: [tkInter](https://wiki.python.org/moin/TkInter), [PyGObject](https://wiki.gnome.org/Projects/PyGObject), [PyQt](http://www.riverbankcomputing.co.uk/software/pyqt/intro), [PySide](https://wiki.qt.io/PySide), [Kivy](https://kivy.org/), [wxPython](http://www.wxpython.org/)
* **Scientific and Numeric**: [SciPy](https://www.scipy.org/), [Pandas](https://pandas.pydata.org/), [IPython](http://ipython.org/)
* **Software Development**: [Buildbot](https://buildbot.net/), [Trac](http://trac.edgewall.org/), [Roundup](http://roundup.sourceforge.net/)
* **System Administration**: [Ansible](https://www.ansible.com/), [Salt](https://www.saltstack.com/), [OpenStack](https://www.openstack.org/)

**Vấn đề 3. Python Version History**

* Python 0.9.0 phát hành ngày 02-1991
* Python 1.0 phát hành ngày 01-1994
* Python 2.0 phát hành ngày 10-2000
* Python 3 phát hành ngày 12-2008
* Python 3.6 phát hành ngày 12-2016
* Python 3.7 phát hành ngày 5-2018
* Python 3.8 phát hành ngày 10-2019
* Python 3.9 phát hành ngày 10-2020,…

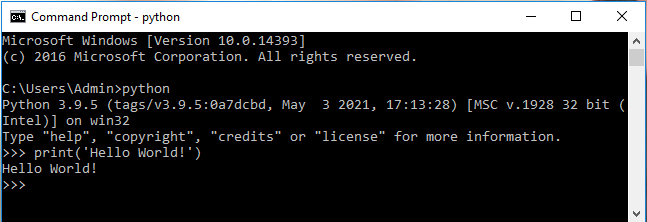
**Vấn đề 4. Install Python**

* Tải các bản cài đặt Python từ <https://www.python.org/downloads>

**Lưu ý**: khi cài đặt, các phiên bản Python yêu cầu sự phù hợp phiên bản của hệ điều hành và hệ thống xử lý (System type) Win32 bit hay Win64 bit.

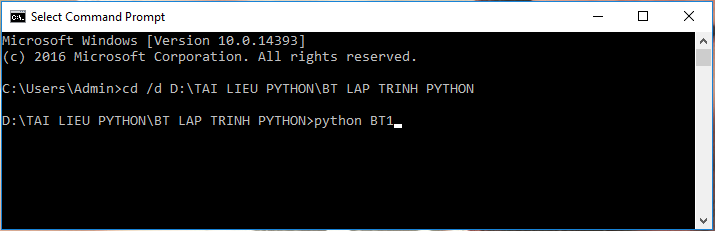
**Vấn đề 5. Python - Shell**

* Chạy trực tiếp lệnh Python: Bạn mở cửa sổ **command prompt** trên Windows, gõ **python** và nhấn **Enter**. Sau đó, sau dấu nhắc lệnh của python, gõ lệnh bạn muốn thực thi.



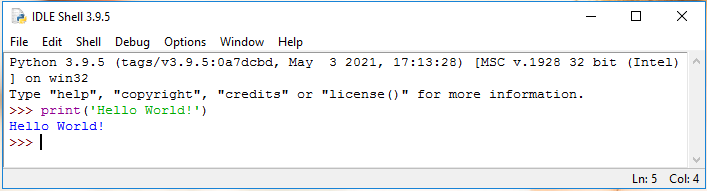
**Lưu ý**: Để kết thúc (chạy) khối lệnh (def, for, while,…) trong Command Prompt – python, bạn nhấn hai lần phím Enter.

* Chạy tập lệnh Python Script: Bạn mở cửa sổ **command prompt** trên Windows, sau đó bạn chuyển điều khiển đến thư mục chứa tệp python, và gõ **python** **<tên tệp>.py** rồi nhấn **Enter**.

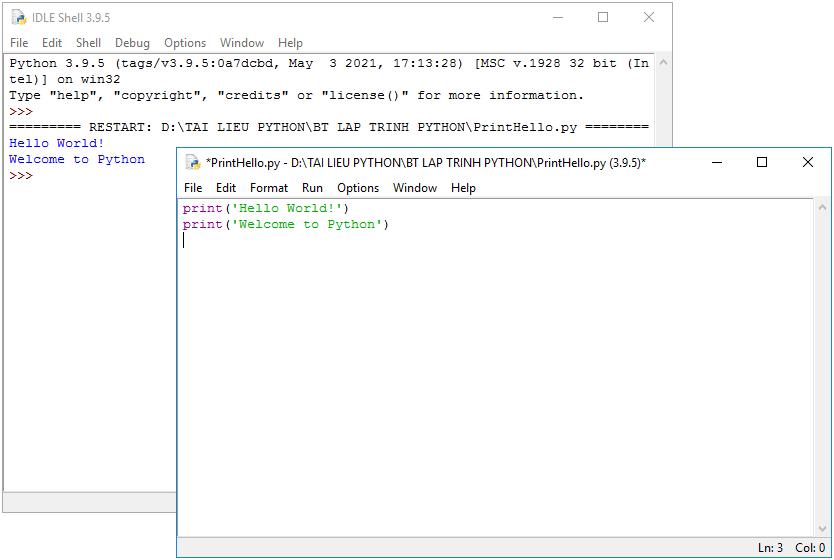


**Vấn đề 6. Python - IDLE**

IDLE (Integrated Development and Learning Environment – môi trường học tập và phát triển tích hợp), nơi mà bạn có thể gõ-chạy trực tiếp lệnh python (tương tự như Python Shell và bạn có thể lưu lại những lệnh đã chạy này) và là nơi thực thi tập lệnh Python Script.



* Để soạn Python Script, trong cửa sổ IDLE, nháy **File**, nháy **New File**, sau đó gõ tập lệnh của bạn và lưu dưới dạng **<tên tệp>.py**. Để chạy tập lệnh này, trên menu, nháy **Run**, nháy **Run Module (F5)**



**Vân đề 7. Python - Syntax**

Câu lệnh Python kết thúc bằng ký tự mã thông báo NEWLINE (ký tự xuống dòng). Nó có nghĩa là mỗi dòng trong tập lệnh Python là một câu lệnh.

Ví dụ:

>>> print('First Name: ', 'Steve')

>>> print('Last Name: ', 'Jobs')

* Sử dụng ký tự gạch chéo \ để nối một câu lệnh kéo dài trên nhiều dòng. Không áp dụng nhiều câu lệnh trên một dòng.

Ví dụ:

>>> if 100 > 99 and \

200 <= 300 and \

True != False:

print('Hello World!')

>>> print('Hello \

World!') # a multi-line statement

>>> print('Hello') \

print(' World!') # two statement in one logical line

SyntaxError: invalid syntax

* Sử dụng dấu chấm phẩy; để tách nhiều câu lệnh trên một dòng.

Ví dụ:

>>> print('First Name: ', 'Steve'); print('Last Name: ', 'Jobs')

* Các biểu thức trong dấu ngoặc đơn (), dấu ngoặc vuông [] hoặc dấu ngoặc nhọn {} có thể được trải dài trên nhiều dòng mà không sử dụng dấu gạch chéo ngược.

Ví dụ:

>>> list = [1, 2, 3, 4,

5, 6, 7, 8,

9, 10, 11, 12]

**Indentation in Python – Thụt lề trong Python**

Khoảng trắng hoặc tab ở đầu dòng được coi là mức thụt lề của dòng, được sử dụng để xác định nhóm câu lệnh. Các câu lệnh có cùng mức thụt lề được coi là một nhóm hoặc khối. Ví dụ: các hàm, lớp hoặc vòng lặp trong Python chứa một khối câu lệnh.

Ví dụ:

>>> def tong(a, b):

s = a + b

return s

**Quy tắt thụt lề**

* Sử dụng dấu hai chấm: để bắt đầu một khối lệnh và nhấn Enter.
* Tất cả các dòng trong một khối lệnh phải sử dụng cùng một thụt đầu dòng, dấu cách hoặc tab.
* Python đề xuất tab (bốn dấu cách) làm thụt đầu dòng để làm cho mã dễ đọc hơn. Không trộn dấu cách và tab trong cùng một khối.
* Một khối lệnh có thể có các khối lệnh bên trong với thụt lề cấp độ tiếp theo.

Ví dụ:

if 10 > 5: # 1st block starts

print("10 is greater than 5") # 1st block

print("Now checking 20 > 10") # 1st block

if 20 > 10: # 1st block

print("20 is greater than 10") # inner block

elif: # 2nd block starts

print("10 is less than 5") # 2nd block

print("This will never print") # 2nd block

**Comments in Python – Chú thích trong Python**

Sử dụng kí hiệu # để viết một dòng chú thích và cặp dấu nháy ba (''' hoặc """) để viết nhiều dòng chú thích trong Python.

Ví dụ:

# this is a comment

print("Hello World")

print("Welcome to Python Tutorial") #comment after a statement.

# multi-line Comments

'''

comment1

comment2

comment3

'''

**Python Naming – Tên trong Python**

Chương trình Python có thể chứa các biến, hàm, lớp, mô-đun, gói, v.v. Tên được đặt cho các đối tượng lập trình này. Tên bắt đầu bằng một chữ cái trong bảng chữ cái (viết thường hoặc viết hoa) hoặc một dấu gạch dưới (\_), sau đó là các chữ cái trong bảng chữ cái (a-z hoặc A-Z), chữ số (0-9) hoặc dấu gạch dưới. Tên không được bắt đầu bằng chữ số.

**Quy ước khi đặt tên**

* Tên trong Python có phân biệt chữ hoa chữ thường, có nghĩa là các biến có tên *name* và *Name* là khác nhau.
* Tên lớp nên sử dụng quy ước chữ tiêu đề (TitleCase). Nó phải bắt đầu bằng một chữ cái viết hoa. Ví dụ: MyClass, Employee, Person,...
* Tên hàm phải ở dạng chữ thường. Nhiều từ phải được phân tách bằng dấu gạch dưới. Ví dụ: tong(num), tinh\_tong (a, b),...
* Tên biến trong hàm phải ở dạng chữ thường. Ví dụ: x, num, salary,...
* Tên mô-đun và gói phải ở dạng chữ thường. Ví dụ: mymodule, tax\_calculation. Sử dụng dấu gạch dưới để cải thiện khả năng đọc.
* Tên biến không đổi phải được viết hoa. Ví dụ: RATE, TAX\_RATE,...
* Sử dụng một hoặc hai ký tự gạch dưới khi đặt tên cho các thuộc tính riêng tư của một lớp.
* Hai dấu gạch dưới đầu và cuối được sử dụng trong chính Python cho một mục đích đặc biệt. Ví dụ: \_\_add\_\_, \_\_init\_\_, v.v.

**Getting User's Input – Nhập dữ liệu từ người dùng**

Hàm input (prompt=None) là một phần của thư viện chính của phân phối Python chuẩn. Nó đọc dữ liệu và trả về một đối tượng chuỗi để gán cho một biến có tên phù hợp. Tham số <prompt> như là một tùy chọn lời nhắc cho người dùng.

Ví dụ:

>>> lastName = input('Input your lastname: ')

**Display Output – Hiển thị dữ liệu ra**

Hàm print () được dùng là một câu lệnh đầu ra trong Python. Nó hiển thị giá trị của bất kỳ biểu thức Python nào trong Python shell.

Ví dụ:

>>> name="Ram"

>>> print(name)

Ram

>>> age = 21

>>> print(age)

21

>>> print('Name:', name, 'Age:', age)

Name: Ram Age: 21

**Vấn đề 8. Python - Keywords**

Cũng giống như các ngôn ngữ tự nhiên, ngôn ngữ lập trình máy tính bao gồm một tập hợp các từ được xác định trước được gọi là từ khóa. Một quy tắc sử dụng quy định cho mỗi từ khóa được gọi là cú pháp.

Python 3.x có 33 từ khóa. Vì chúng có một ý nghĩa xác định trước kèm theo, chúng không thể được sử dụng cho bất kỳ mục đích nào khác. Danh sách các từ khóa Python có thể được lấy bằng cách sử dụng lệnh help() trong Python shell.

>>> help("keywords")

* Sau đây là một tập các từ khóa trong Python 3.9.5

False break for not

None class from or

True continue global pass

\_\_peg\_parser\_\_ def if raise

and del import return

as elif in try

assert else is while

async except lambda with

await finally nonlocal yield

* Ngoại trừ ba từ khóa đầu tiên (False, None và True) có kí tự đầu viết hoa, các từ khóa khác hoàn toàn ở dạng chữ thường.

**Reserved Identifiers – Tên dành riêng**

Các lớp dựng sẵn trong Python chứa một số tên có ý nghĩa đặc biệt. Các tên đặc biệt này được nhận dạng bởi các ký tự gạch dưới ở đầu và cuối:

|  |  |  |
| --- | --- | --- |
| **Mẫu** | **Mô tả** | **Ví dụ** |
| \_\* | - Lưu trữ kết quả của lần tính toán cuối cùng | >>> 5 \* 5  25  >>> print('Hello World')  Hello World  >>> \_  25 |
| \_\_\*\_\_ | Nó đại diện cho các tên do hệ thống xác định. Đây có thể là các hàm hoặc thuộc tính như \_\_new \_\_ (), \_\_init \_\_ (), \_\_name\_\_, \_\_main\_\_, v.v. | >>> \_\_name\_\_  '\_\_main\_\_' |
| \_\_\* | Nó đại diện cho tên riêng của lớp. Những tên này sẽ được sử dụng như tên thành viên riêng của lớp để tránh xung đột tên giữa các thuộc tính riêng của lớp cơ sở và lớp dẫn xuất. |  |

**Vấn đề 9. Python - Variables**

Đối tượng (Object) là sự trừu tượng hóa dữ liệu của Python. Trong Python, dữ liệu được biểu diễn bằng các đối tượng hoặc bằng quan hệ giữa các đối tượng. Sử dụng hàm type () để xem tên lớp của một đối tượng.

Ví dụ:

>>> type(10)

<class 'int'>

**Object's Identity – Nhận dạng đối tượng**

Mỗi đối tượng trong Python có một id. Nó là địa chỉ của đối tượng trong bộ nhớ được biểu thị bằng một giá trị số nguyên. Hàm id () trả về số nhận dạng của đối tượng, chỉ định nơi nó được lưu trữ.

Ví dụ:

>>> x = 10

>>> id(x)

10383632

**Single Variable Assignment – Gán giá trị cho một biến đơn**

Các biến trong Python là tên được đặt cho các đối tượng, dùng để tham chiếu đến một giá trị. Giá trị được gán cho một biến bằng cách sử dụng toán tử =, trong đó phía bên trái của nó là tên của một biến và phía bên phải là một giá trị hay một biểu thức.

Ví dụ:

>>> num = 10

>>> print(num)

10

>>> result = 20 + 10

>>> print(result)

30

**Multiple Variables Assignment – Gán giá trị cho nhiều biến**

Bạn có thể khai báo nhiều biến và gán giá trị cho từng biến trong một câu lệnh duy nhất.

Ví dụ:

>>> x, y, x = 10, 20, 30 # Valid

>>> x = 10, y = 20, z = 30 #Error

SyntaxError: cannot assign to literal

**Variable Naming Conventions – Quy ước đặt tên biến**

* Tên biến phải tuân thủ theo quy tắt đặt tên trong Python.
* Tên biến không được là một từ khóa dành riêng trong Python.
* Tên biến được đặt là chữ thường. Ví dụ: name, age,…
* Tên biến dài thì các từ được nối với nhau bằng dấu gạch dưới. Ví dụ: long\_name
* Đặt tên biến sao cho gợi nghĩa. Ví dụ: Nên đặt tên các biến của phương trình bậc hai là *a*, *b*, *c* hay tên của một người là *name*,…

**Vấn đề 10. Python - Data Types**

* **Các kiểu vô hướng: int, float, complex, bool, None**

**Kiểu int**

Trong Python, số nguyên là số 0, số nguyên dương hoặc âm không có phần thập phân và có độ lớn không giới hạn. Ví dụ: 0, 100, -10

* Để lấy kiểu của một số, bạn sử dụng hàm type()

Ví dụ:

>>>type(100)

<class 'int'> # type of x is int

* Sử dụng dấu gạch dưới \_ để phân chia các phần

Ví dụ:

>>> x=1\_234\_567\_890

>>> x

1234567890

* Dùng hàm **int()** để chuyển chuỗi hoặc số thực sang số nguyên

Ví dụ:

>>> int('100')

100

>>> int('-10')

-10

>>> int('5.5')

5

>>> int('100', 2) #chuyển chuỗi trong hệ 2

4

* Đặt các tiền tố 0b (hệ 2), 0o hoặc 0O (hệ 8) và 0x hoặc 0X trước các số để thể hiện hệ của số đó.

Ví dụ:

>>> x=0b11011000

>>> x

216

>>> x=0o12

>>> x

10

>>> x=0x12

>>> x

18

**Kiểu Float**

* Trong Python, kiểu số thực Float bao gồm số dương, số âm được nhận diện bằng dấu chấm thập phân hoặc số kiểu khoa học. Ví dụ: 123.456, -1.55, 0.23 hoặc 1e3, 1E5

Ví dụ:

>>> f=1.2

>>> f

1.2

>>> type(f)

<class 'float'>

* Sử dụng dấu gạch dưới \_ để phân chia các phần

Ví dụ:

>>> f=123\_42.222\_013

>>> f

12342.222013

* Dùng hàm **float()** để chuyển chuỗi hoặc số nguyên sang số thực

Ví dụ:

>>> float('5.5')

5.5

>>> float('5')

5.0

>>> float(' -5')

-5.0

>>> float('1e3')

1000.0

**Các hàm sẳn có (Built-in functions) làm việc với kiểu số**

|  |  |  |
| --- | --- | --- |
| **Cú pháp** | **Mô tả** | **Ví dụ** |
| int (x, base = 10) | Trả về đối tượng số nguyên từ một số thực hoặc một chuỗi chứa các chữ số. Hệ mặc định là hệ cơ số 10. | >>> x = int()  >>> x  0  >>> int(10.5)  10  >>> int(0.12)  0  >>> int('123')  123  >>> int('1011', 2)  11 |
| float (x = 0) | Trả về một đối tượng số dấu phẩy động từ một số hoặc chuỗi chứa các chữ số có dấu thập phân hoặc ký hiệu khoa học. | >>> f = float()  >>> f  0.0  >>> float(10)  10.0  >>>float('10.45')  10.45  >>>float('1e-003')  0.001  >>>float('+1E6')  1000000.0 |
| complex (real = 0, imag = 0) | Trả về một số phức với các thành phần thực và ảo. | >>> complex(4,5)  (4+5j)  >>>complex(1.5, -2.5)  (1.5-2.5j)  >>>complex(1.5j, 2.5j)  (-2.5+1.5j) |
| hex (number) | Chuyển đổi một số nguyên thập phân thành một số thập lục phân có tiền tố 0x. | >>> hex(18)  '0x12'  >>>hex(255)  '0xff'  >>>hex(-64)  '-0x40' |
| oct (number) | Chuyển đổi một số nguyên thập phân sáng một biểu diễn bát phân với tiền tố 0o. | >>> oct(12)  '0o14'  >>>oct(10)  '0o12'  >>>oct(8)  '0o10'  >>>oct(-23)  '-0o27' |
| pow (base, exp, mod=None) | Trả về lũy thừa của các số được chỉ định. | >>> pow(10,2)  100  >>>pow(100,0.5)  10.0  >>>pow(25,-2)  0.0016  >>>pow(10,-2)  0.01  >>>pow(100,2+0j)  (10000+0j) |
| abs (x) | Trả về giá trị tuyệt đối của một số mà không cần xét đến dấu của nó. | >>> abs(-10)  10  >>>abs('10')  TypeError: bad operand type for abs(): 'str'  >>>abs(-1.5)  1.5  >>>abs(1+2j)  2.23606797749979 #this is 5 |
| round (number, ndigits=None) | Trả về số được làm tròn. | >>> round(1234.456,2)  1234.46  >>>round(1234.456,1)  1234.5  >>>round(1234.456,0)  1234.0  >>>round(1234.456, -1)  1230.0 |

**Kiểu Complex**

Không chỉ số thực, Python còn có thể xử lý các số phức và các hàm liên quan của nó bằng cách sử dụng tệp “cmath”. Số phức có công dụng trong nhiều ứng dụng liên quan đến toán học và python cung cấp các công cụ hữu ích để xử lý và thao tác chúng.

* Chuyển số thực thành số phức: Một số phức được biểu diễn bằng "x + yi". Python chuyển đổi các số thực x và y thành phức bằng cách sử dụng hàm complex (x, y). Phần thực có thể được truy cập bằng cách sử dụng hàm real () và phần ảo có thể được biểu diễn bằng virtual ().

Ví dụ:

>>> import cmath

>>> x = 5

>>> y = 3

>>> z = complex(x, y)

>>> print(z)

(5+3j)

**Kiểu Bool**

Kiểu Bool là một trong những kiểu được cung cấp bởi Python, đại diện cho một trong hai giá trị là True hoặc False. Nó được sử dụng để biểu diễn giá trị chân lí của biểu thức. Ví dụ 5 > 3 có giá trị là đúng (True), 3 > 5 có giá trị là sai (False).

* Bạn sử dụng hàm type () để lấy kiểu dữ liệu biến bool.

Ví dụ:

>>> a = True

>>> type(a)

<class 'bool'>

* Bạn sử dụng hàm bool () để chuyển một hằng, một biến hoặc một biểu thức sang kiểu bool.

Ví dụ:

>>> bool(0)

False

>>> bool(1)

True

>>> a = 0

>>> bool(a)

False

>>> b = 1

>>> bool(b)

True

>>> s = ''

>>> bool(s)

False

>>> s = 'Hello'

>>> bool(s)

True

>>> l1 = []

>>> bool(l1)

False

>>> l2 = [1, 2, 3]

>>> bool(l2)

True

>>> bool(5 + 3)

True

* Biểu thức lôgic trả về giá trị có kiểu bool.

Ví dụ:

>>> x = 5; y = 10

>>> y >= x

True

**Boolean Operators – Các toán tử Boolean**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| not | Không | >>> not True  False  >>> not False  True |
| and | Và | >>> True and True  True  >>> True and Fals­e  False  >>> False and False  False |
| or | Hoặc | >>> True or True  True  >>> True or False  True  >>> False or False  False |

**Operators return Boolean values – Các toán tử trả về giá trị Boolean**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| ==, !=, <, >, <=, >= | So sánh | >>> 5 == 3  False  >>> 5 != 3  True  >>> 5 > 3  True  >>> 5 >= 3  True  >>> 3 < 2  False  >>> 3 <= 2  False |
| is, is not | Là, không là | >>> x = 2; y = 3  >>> x is y  False |
| in, not in | Có trong, không có trong | >>> string = ('Welcome to Python!')  >>> 'Welcome' in string  True  >>> 'python!' not in string  True |

**Kiểu None**

Từ khóa None được sử dụng để xác định giá trị null hoặc không có giá trị nào cả. None không phải là 0, False hoặc một chuỗi rỗng. None có kiểu dữ liệu của riêng nó là NoneType.

>>> type(None)

<class 'NoneType'>

* **Kiểu trình tự: String, List, Tuple**

**Kiểu String**

Trong Python, chuỗi là một kiểu dữ liệu dãy bất biến. Nó là chuỗi các ký tự Unicode được bao bọc bên trong các dấu ngoặc kép đơn, đôi hoặc ba.

Ví dụ:

'This is a string in Python'

"This is a string in Python"

'''This is a string in Python'''

"""This is a string in Python"""

* Đưa 'dấu nháy đơn' hoặc "dấu nháy kép" trong chuỗi ra màn hình

Ví dụ:

>>> str1='Đưa "dấu nháy kép" ra màn hình'

>>> print(str1)

Đưa "dấu nháy kép" ra màn hình

>>> str2="Đưa 'dấu nháy đơn' ra màn hình"

>>> print(str2)

Đưa 'dấu nháy đơn' ra màn hình

* Sử dụng hàm len () để lấy độ dài của một chuỗi.

Ví dụ:

>>> greet='Hello'

>>> len(greet)

5

* Một chuỗi là một tập hợp các ký tự có thứ tự. Chuỗi sử dụng chỉ mục, bắt đầu bằng số 0.

Ví dụ:

>>> greet='hello'

>>> greet[0]

'h'

>>> greet[1]

'e'

>>> greet[2]

'l'

>>> greet[3]

'l'

>>> greet[4]

'o'

* Python cũng hỗ trợ lập chỉ mục âm, bắt đầu bằng - (độ dài của chuỗi) cho đến -1.

Ví dụ:

>>> greet='hello'

>>> greet[-5]

'h'

>>> greet[-4]

'e'

>>> greet[-3]

'l'

>>> greet[-2]

'l'

>>> greet[-1]

'o'

**Lưu ý**: Do chuỗi là một loại đối tượng **bất biến** nên bạn không thể dùng toán tử gán để thay đổi giá trị của chuỗi tại một chỉ mục (sẽ phát sinh lỗi).

Ví dụ:

>>> greet='hello'

>>> greet[0]='A'

TypeError: 'str' object does not support item assignment

* Sử dụng hàm str () để chuyển đổi một số thành một chuỗi.

Ví dụ:

>>> str(100)

'100'

>>> str(True)

'True'

**Escape Sequences** – Chuỗi thoát

Trong Python, dấu gạch chéo ngược \ được sử dụng như một ký tự thoát. Sử dụng ký tự gạch chéo ngược theo sau là ký tự bạn muốn chèn vào chuỗi.

Ví dụ:

>>> str1='Đây là \'kí tự thoát\' trong Python'

>>> print(str1)

Đây là 'kí tự thoát' trong Python

>>> str2="Đây là \"kí tự thoát\" trong Python"

>>> print(str2)

Đây là "kí tự thoát" trong Python

* Sử dụng kí tự r hoặc R để bỏ qua chuỗi thoát (kí tự thoát được xem như kí tự thông thường).

Ví dụ:

>>> str2=r"Đây là \"kí tự thoát\" trong Python"

>>> print(str2)

Đây là \"kí tự thoát\" trong Python

* Một số chuỗi thoát thường dùng:

|  |  |  |
| --- | --- | --- |
| \\ | Backslash | >>> print("Hello\\Hi")  Hello\Hi |
| \b | Backspace | >>> print("ab\bcd")  abcd |
| \n | Newline | >>> print("hello\nworld")  hello  world |
| \t | Tab | >>> print('Hello\tPython')  Hello Python |

**String Operators** – Các toán tử kiểu chuỗi

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Nối chuỗi thứ hai vào chuỗi đầu tiên | >>> a = "Hello "  >>> b = "Python"  >>> a + b  'Hello Python' |
| \* | Nhân lên nhiều bản sao của cùng một chuỗi | >>> a = "Hello"  >>> a\*3  'HelloHelloHello' |
| [] | Trả về ký tự tại chỉ mục đã cho | >>> a = 'Python'  >>> a[0]  'P' |
| [ : ] | Tìm các ký tự trong phạm vi được chỉ định bởi hai chỉ mục toán hạng được phân tách bằng ký hiệu : | >>> = 'Python'  >>> a[0:2]  'Py' |
| in | Trả về True nếu tồn tại chuỗi con trong chuỗi đã cho | >>> 'Py' in 'Python'  True  >>> 'Pi' in 'Python'  False |
| not in | Trả về True nếu không tồn tại chuỗi con trong chuỗi đã cho | >>> 'Pi' not in 'Python'  True |

**String Comparison** – So sánh chuỗi

Các toán tử quan hệ so sánh các giá trị Unicode của các ký tự trong chuỗi từ chỉ số 0 cho đến cuối chuỗi. Sau đó, nó trả về một giá trị Boolean theo toán tử được sử dụng. Bạn có thể sử dụng hàm ord (c) để kiểm tra mã Unicode của kí tự c. Ví dụ: ord("A") 🡪 65, ord("a") 🡪 97

|  |  |  |
| --- | --- | --- |
| **Phép toán** | **Mô tả** | **Ví dụ** |
| == | Bằng | >>> "Hello" == "Hello"  True |
| != | Khác | >>> "hello" != "Hello"  True |
| < | Bé hơn | >>> "Hello" < "hello"  True |
| > | Lớn hơn | >>> "hello" > "Hello"  True |
| <= | Bé hơn hoặc bằng | >>> "Hello" <= "hello"  True |
| >= | Lớn hơn hoặc bằng | >>> "Hello" >= "Hello"  True |

**String Methods** – Các phương thức kiểu chuỗi

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Mô tả** | **Ví dụ** |
| string.capitalize() | Trả về một **bản sao của chuỗi** gốc string với ký tự đầu tiên được viết hoa và phần còn lại của chuỗi viết thường. | >>> a = 'hello world!'  >>> a.capitalize();  'Hello world!' |
| string.casefold() | Trả về **một chuỗi** trong đó tất cả các ký tự đều ở dạng chữ thường. Nó tương tự như phương thức low(), nhưng phương thức casefold() chuyển đổi nhiều ký tự hơn thành chữ thường. Ví dụ, chữ cái thường trong tiếng Đức 'ß' tương đương với chữ cái hoa 'ss'. | >>> mystr = 'Welcome to Python!'  >>> mystr.casefold()  'welcome to python!' |
| string.center(width, fillchar = ' ') | Trả về **một chuỗi** căn giữa mới có độ dài *width*, được chèn thêm bằng ký tự *fillchar*, mặc định là khoảng trắng. | >>> greet='Hi'  >>> print(greet.center(4, '-'))  '-Hi-' |
| string.count(substring[, start, [end]]) | Trả về **số lần xuất hiện** không chồng chéo của chuỗi con *substring* trong chuỗi string gốc. Các đối số tùy chọn *start* và *end* được diễn giải như trong ký hiệu lát cắt. | >>> mystr = 'Welcome to Python!'  >>> mystr.count('to')  1 |
| string.endswith(suffix[, start[, end]]) | Trả về **True** nếu chuỗi gốc string kết thúc bằng *suffix*, nếu không thì là False, với tùy chọn kiểm tra bắt đầu tại vị trí *start* và kết thúc tại vị trí *end*. | >>> mystr = 'Welcome to Python!'  >>> mystr.endswith('Python!')  True |
| string.expandtabs(tabsize = 8) | Trả lại **một bản sao của chuỗi** gốc string, trong đó tất cả các ký tự tab được mở rộng bằng cách sử dụng dấu cách. Kích thước tab mặc định gồm 8 ký tự khoảng cách. | >>> '1234\t'.expandtabs()  '1234 ' |
| string.find(sub[, start[, end]]) | Trả về **chỉ số nhỏ nhất** trong chuỗi gốc string nơi tìm thấy chuỗi con *sub* trong khoảng từ chỉ số *start* đến chỉ số *end*. Chỉ số *start* và *end* là không bắt buộc và được hiểu như trong ký hiệu lát cắt. Trả về -1 khi không tìm thấy. | >>> mystr = 'Welcome to Python!'  >>> mystr.find('Python!')  11  >>> mystr.find('python!')  -1 |
| string.index(sub[, start[, end]]) | Trả về **chỉ số thấp nhất** trong chuỗi gốc string nơi tìm thấy chuỗi con *sub* từ chỉ mục *start* đến chỉ mục *end*. Các chỉ mục tùy chọn *start* và *end* được hiểu như trong ký hiệu lát cắt.  Khi không tìm thấy chuỗi con sẽ phát sinh lỗi **ValueError**. | >>> s = 'Hello World'  >>> print(s.index('World'))  6  >>> print(s.index('world'))  ValueError: substring not found |
| string.isalnum() | Trả về **True** nếu chuỗi gốc string là một chuỗi chữ-số, trả về **False** nếu không phải. Một chuỗi là chữ-số nếu tất cả các ký tự trong chuỗi là chữ và số và có ít nhất một ký tự trong chuỗi. | >>> s = 'HelloWorld2021'  >>> print(s.isalnum())  True  >>> s = 'Hello World 2021'  >>> print(s.isalnum())  False |
| string.isalpha() | Trả về **True** nếu chuỗi gốc string là một chuỗi chữ cái, trả về **False** nếu không phải. Một chuỗi là bảng chữ cái nếu tất cả các ký tự trong chuỗi có trong bảng chữ cái và ở đó có ít nhất một ký tự trong chuỗi. | >>> s = 'HelloWorld'  >>> print(s.isalpha())  True  >>> s = 'Hello World'  >>> print(s.isalpha())  False |
| string.isascii() | Trả về **True** nếu tất cả các ký tự trong chuỗi có ASCII, nếu không thì trả về **False**. Chuỗi trống cũng là ASCII. | >>> mystr = 'ABC123456'  >>> print(mystr.isascii())  True |
| string.isdecimal() | Trả về **True** nếu chuỗi gốc string là chuỗi thập phân, nếu không thì trả về **False**. Một chuỗi là một chuỗi thập phân nếu tất cả các ký tự trong chuỗi là chữ số thập phân (hệ 10) và có ít nhất một ký tự trong chuỗi. | >>> numstr = '12345'  >>> print(numstr.isdecimal())  True  >>> numstr = '123.45'  >>> print(numstr.isdecimal())  False |
| string.isdigit() | Trả về **True** nếu chuỗi gốc string là một chuỗi chữ số, trả về **False** nếu không phải. Một chuỗi là một chuỗi chữ số nếu tất cả các ký tự trong chuỗi là chữ số và ở đó có ít nhất một ký tự trong chuỗi. | >>> mystr = '12345'  >>> print(mystr.isdigit())  True  >>> mystr = '123.45'  >>> print(mystr.isdigit())  False |
| string.isidentifier() | Trả về **True** nếu chuỗi gốc string là một định danh (tên) Python hợp lệ, nếu không thì là trả về **False**. | >>> name ='Python'  >>> print(name.isidentifier())  True  >>> name ='Hello World'  >>> print(name.isidentifier())  False |
| string.islower() | Trả về **True** nếu chuỗi là chuỗi gốc string viết thường, nếu không thì trả về **False**. Một chuỗi là chữ thường nếu tất cả các ký tự được viết trong chuỗi là chữ thường và có ít nhất một ký tự trong chuỗi. | >>> mystr ='hello world'  >>> print(mystr.islower())  True  >>> mystr ='Hello World'  >>> print(mystr.islower())  False |
| string.isnumeric() | Trả về **True** nếu chuỗi gốc string là chuỗi số, trả về **False** nếu không phải. Một chuỗi là số nếu tất cả các ký tự trong chuỗi là số và có ít nhất một ký tự trong chuỗi. | >>> numstr = '100'  >>> print(numstr.isnumeric())  True  >>> numstr = '1A'  >>> print(numstr.isnumeric())  False  >>> numstr = '1.1'  >>> print(numstr.isnumeric())  False |
| string.isprintable() | Trả về **True** nếu chuỗi gốc string có thể in được, nếu không thì trả về **False**. Một chuỗi có thể in được nếu tất cả các ký tự của nó được coi là có thể in được trong repr() hoặc nếu nó trống. | >>> mystr = 'Hello World'  >>> print(mystr.isprintable())  True  >>> mystr = 'Hello \nWorld'  >>> print(mystr.isprintable())  False |
| string.isspace() | Trả về **True** nếu chuỗi gốc string là chuỗi khoảng trắng, nếu không thì trả về **False**. Một chuỗi là khoảng trắng nếu tất cả các ký tự trong chuỗi là khoảng trắng và ở đó có nhất một ký tự khoảng trắng trong chuỗi. | >>> mystr = ' '  >>> print(mystr.isspace())  True  >>> mystr = '\t\t'  >>> print(mystr.isspace())  True  >>> mystr = 'Hello World'  >>> print(mystr.isspace())  False |
| string.istitle() | Trả về **True** nếu chuỗi gốc string là chuỗi tiêu đề, nếu không thì là **False**. Chuỗi tiêu đề là chuỗi mà kí tự đầu của mỗi từ là viết hoa, còn lại các kí tự khác viết thường. | >>> title ='Hello World'  >>> print(title.istitle())  True  >>> title ='Hello world'  >>> print(title.istitle())  False  >>> title = 'Welcome To Python!'  >>> print(title.istitle())  True |
| string.isupper() | Trả về **True** nếu chuỗi gốc string là chuỗi viết hoa, nếu không thì trả về **False**. Một chuỗi là chữ hoa nếu tất cả các ký tự trong chuỗi được viết hoa và có ít nhất một ký tự được viết trong chuỗi. | >>> mystr = 'HELLO WORLD'  >>> print(mystr.isupper())  True  >>> mystr = 'Hello world'  >>> print(mystr.isupper())  False |
| string.join(iterable) | Kết quả **trả về một chuỗi** mới sau khi nối dãy các phần tử chuỗi *iterable* bằng chuỗi gốc string. | >>> sep = '-'  >>> names = ['Steve', 'Bill', 'Ravi']  >>> print(sep.join(names))  'Steve-Bill-Ravi' |
| string.ljust(width, fillchar=' ') | Trả về **một chuỗi** mới căn trái với độ rộng *width*. Chuỗi mới được tạo bằng cách chèn đệm kí tự *fillchar* sau chuỗi gốc string, mặc định là khoảng trống. | >>> mystr = 'Hi'  >>> mystr.ljust(4)  'Hi '  >>> mystr.ljust(4, '-')  'Hi--' |
| string.lower() | Trả về **một chuỗi** bản sao của chuỗi gốc string được chuyển đổi thành chữ thường. | >>> mystr = 'HELLO WORLD'  >>> mystr.lower()  'hello world' |
| string.lstrip(chars=None) | Trả về **một chuỗi** bản sao của chuỗi gốc string sau khi đã xóa tất cả kí tự *chars* ở đầu, mặc định là khoảng trống. | >>> mystr = ' Hello World'  >>> mystr.lstrip()  'Hello World'  >>> mystr = '----Hello World----'  >>> mystr.lstrip('-')  'Hello World----' |
| string.maketrans(x [, y[, z]]) | Phương thức maketrans() trả về **một bảng ánh xạ** ánh xạ từng ký tự trong chuỗi đã cho với ký tự trong chuỗi thứ hai ở cùng một vị trí. Bảng ánh xạ này được sử dụng với phương thức translate(), sẽ thay thế các ký tự theo bảng ánh xạ. | >>> mystr = 'Most Mis'  >>> mappingtbl = mystr.maketrans('M','H')  >>> newstr = mystr.translate(mappingtbl)  >>> print(newstr)  'Host His' |
| string.partition(sep) | Phương thức này sẽ tìm kiếm dấu phân tách *sep* trong chuỗi, nếu tìm thấy sẽ tách chuỗi gốc string tại dấu phân tách *sep* đầu tiên và trả về **một bộ 3 chứa phần trước dấu phân tách, dấu phân tách, và phần sau nó**. Nếu không tìm thấy dấu phân tách, trả về một bộ 3 chứa chuỗi ban đầu và hai chuỗi trống. | >>> mystr = 'Hello World'  >>> print(mystr.partition(' '))  ('Hello', ' ', 'World')  >>> print(mystr.partition('s'))  ('Hello World', '', '') |
| string.replace(old, new, count=-1) | Trả về **một chuỗi** bản sao với tất cả các lần xuất hiện của chuỗi con cũ *old* được thay thế bằng chuỗi mới *new*. Tham số *count* với số lần xuất hiện tối đa chuỗi cũ *old* để thay thế, mặc định là -1 có nghĩa là thay thế tất cả các lần xuất hiện. | >>> mystr = 'Hello World!'  >>> print(mystr.replace('Hello','Hi'))  'Hi World!' |
| string.rfind(sub[, start[, end]]) | Trả về **chỉ số lớn nhất** trong chuỗi gốc string nơi tìm thấy chuỗi con *sub* trong khoảng từ chỉ số *start* đến chỉ số *end*. Chỉ số *start* và *end* là không bắt buộc và được hiểu như trong ký hiệu lát cắt. Trả về -1 khi không tìm thấy. | >>> mystr='Hello World'  >>> print(mystr.rfind('H'))  0  >>> print(mystr.rfind('o'))  7 |
| string.rindex(sub[, start[, end]]) | Trả về **chỉ số lớn nhất** trong chuỗi gốc string nơi tìm thấy chuỗi con *sub* trong khoảng từ chỉ số *start* đến chỉ số *end*. Chỉ số *start* và *end* là không bắt buộc và được hiểu như trong ký hiệu lát cắt.  Khi không tìm thấy thì phát sinh lỗi **ValueError**. | >>> mystr='Hello World'  >>> print(mystr.rindex('o'))  7  >>> print(mystr.rindex('h'))  ValueError: substring not found |
| string.rjust(width, fillchar=' ') | Trả về **một chuỗi** mới căn phải với độ rộng *width*. Chuỗi mới được tạo bằng cách chèn đệm kí tự *fillchar* trước chuỗi gốc string, mặc định là khoảng trống. | >>> mystr = 'Hi'  >>> mystr.rjust(4)  ' Hi'  >>> mystr.rjust(4, '-')  '--Hi' |
| string.rpartition(sep) | Phương thức này sẽ tìm kiếm dấu phân tách *sep* trong chuỗi, nếu tìm thấy sẽ tách chuỗi gốc string tại dấu phân tách *sep* sau cùng và trả về **một bộ 3 chứa phần trước dấu phân tách, dấu phân tách, và phần sau nó**. Nếu không tìm thấy dấu phân tách, trả về một bộ 3 chứa hai chuỗi trống và chuỗi ban đầu. | >>> mystr = 'Hello Hello World'  >>> print(mystr.partition(' '))  ('Hello', ' ', 'Hello World')  >>> print(mystr.rpartition(' '))  ('Hello Hello', ' ', 'World')  >>> print(mystr.rpartition('s'))  ('', '', 'Hello Hello World') |
| string.rsplit(sep=None, maxsplit=-1) | Trả về **danh sách các từ trong chuỗi** sau khi phân tách chuỗi tại dấu phân tách *sep* (mặc định là None có nghĩa là phân tách bằng khoảng trắng) với số lần phân tách *maxsplit* (mặc định là –1 có nghĩa là phân tách hết chuỗi)*.* Việc phân tách chuỗi được bắt đầu từ phía cuối chuỗi trước. | >>> mystr = 'Hello Hello World'  >>> print(mystr.rsplit())  ['Hello', 'Hello', 'World']  >>> langs = 'C,Python,Java,SQL'  >>> print(langs.rsplit(',', 2))  ['C,Python', 'Java', 'SQL'] |
| string.rstrip(chars=None) | Trả về **một chuỗi** bản sao của chuỗi gốc string sau khi đã xóa tất cả kí tự *chars* ở cuối, mặc định là khoảng trống. | >>> mystr = 'Hello World '  >>> print(mystr.rstrip())  'Hello World'  >>> mystr = '--Hello World--'  >>> print(mystr.rstrip('-'))  '--Hello World' |
| string.split(sep=None, maxsplit=-1) | Trả về **danh sách các từ trong chuỗi** sau khi phân tách chuỗi tại dấu phân tách *sep* (mặc định là None có nghĩa là phân tách bằng khoảng trắng) với số lần phân tách *maxsplit* (mặc định là –1 có nghĩa là phân tách hết chuỗi)*.* Việc phân tách chuỗi được bắt đầu từ phía đầu chuỗi trước. | >>> mystr = 'Hello World'  >>> print(mystr.split())  ['Hello', 'World']  >>> langs = 'C,Python,Java,SQL'  >>> print(langs.split(',', 2))  ['C', 'Python', 'Java,SQL'] |
| string.splitlines(keepends=False) | Trả về **danh sách các dòng** trong chuỗi, ngắt ở ranh giới dòng. Ngắt dòng không được bao gồm trong danh sách kết quả trừ khi tham số *keepends* có giá trị là True | >>> mystr = 'Welcome to Python!'  >>> print(mystr.splitlines())  ['Welcome to Python!']  >>> langs = 'C#\rPython\rJava\rSQL'  >>> print(langs.splitlines())  ['C#', 'Python', 'Java', 'SQL'] |
| string.startswith(prefix[, start[, end]]) | Trả về **True** nếu chuỗi gốc string bắt đầu bằng *prefix*, nếu không thì trả về False, với tùy chọn kiểm tra bắt đầu tại vị trí *start* và kết thúc tại vị trí *end*. | >>> mystr = 'Hello World'  >>> print(mystr.startswith('Hello'))  True  >>> print(mystr.startswith('hello'))  False |
| string.strip(chars=None) | Trả về **một chuỗi** bản sao của chuỗi gốc string sau khi đã xóa tất cả kí tự *chars* ở đầu và cuối, mặc định là khoảng trống. | >>> mystr = ' Hello World '  >>> print(mystr.strip())  'Hello World'  >>> mystr = '--Hello World--'  >>> print(mystr.strip('-'))  'Hello World' |
| string.swapcase() | Trả về **một chuỗi** bản sao của chuỗi gốc string sau khi chuyển đổi các ký tự viết hoa thành chữ thường và các ký tự viết thường thành chữ hoa. | >>> 'hello world'.swapcase()  'HELLO WORLD'  >>> 'Hello World'.swapcase()  'hELLO wORLD' |
| string.title() | Trả về **một chuỗi** phiên bản của chuỗi gốc string sau khi chuyển đổi thành chuỗi tiêu đề. Chuỗi tiêu đề là chuỗi mà các từ bắt đầu bằng ký tự viết hoa và tất cả kí tự còn lại viết thường. | >>> 'hello world'.title()  'Hello World'  >>> 'hELLo wORld'.title()  'Hello World' |
| string.translate(table) | Trả về **một chuỗi** trong đó mỗi ký tự được ánh xạ tới ký tự tương ứng của nó trong bảng dịch table. Bảng dịch được tạo bởi phương thức maketrans (). | >>> oldstr = 'hello word'  >>> mydict = {'h':'H','w':'W'}  >>> mytable = mystr.maketrans(mydict)  >>> newstr = mystr.translate(mytable)  >>> print(newstr)  Hello Word |
| string.upper() | Trả về **một chuỗi** bản sao của chuỗi gốc string được chuyển đổi thành chữ hoa. | >>> mystr = 'Hello World'  >>> mystr.upper()  'HELLO WORLD'  >>> 'heLLo wORld'.upper()  'HELLO WORLD' |
| string.zfill(width) | Trả về **một chuỗi** bản sao của chuỗi với các ký tự '0' được chèn ở bên trái. Nó thêm các số không (0) vào đầu chuỗi cho đến khi chiều dài của chuỗi bằng tham số chiều rộng *width* được chỉ định. | >>> num = '100'  >>> print(num.zfill(6))  '000100' |

**Kiểu List**

Trong Python, danh sách là kiểu dãy có thể thay đổi. Đối tượng danh sách chứa một hoặc nhiều mục thuộc các kiểu dữ liệu khác nhau trong dấu ngoặc vuông [] được phân tách bằng dấu phẩy.

Ví dụ:

>>> mylist=[] # empty list

>>> names=["Jeff", "Bill", "Steve", "Mohan"] # string list

>>> item=[1, "Jeff", "Computer", 75.50, True] # list with heterogeneous data

**List Initialization – Khởi tạo danh sách**

Bạn có thể tạo danh sách bằng kí hiệu [] hoặc bằng hàm tạo khởi tạo danh sách list() từ dãy phần tử của các đối tượng khác như List, Tuple, Set, Dictionary

Ví dụ:

>>> mylist = []; mylist

[] # Empty list

>>> mylist = list(); mylist

[] # Empty list

>>> nums=[1,2,3,4]

>>> nums

[1, 2, 3, 4]

>>> names=list(("Jeff", "Bill", "Steve", "Mohan"))

>>> names

['Jeff', 'Bill', 'Steve', 'Mohan']

**List Comprehension – Danh sách phức tạp**

List Comprehension trong Python là một cú pháp dễ dàng và nhỏ gọn để tạo danh sách từ một chuỗi hoặc một danh sách khác. Đó là một cách rất ngắn gọn để tạo một danh sách mới bằng cách thực hiện một thao tác trên từng phần tử trong danh sách hiện có.

Cú pháp:

[expression for element in iterable if condition]

Cú pháp gồm ba phần: một biểu thức, một hoặc nhiều vòng lặp for và một hoặc nhiều điều kiện tùy chọn if. Phần hiểu danh sách phải nằm trong dấu ngoặc vuông []. Kết quả của biểu thức đầu tiên sẽ được lưu trong danh sách mới.

Ví dụ:

>>> squares = [x\*x for x in range(11) if x % 2 == 0]

>>> squares

[0, 4, 16, 36, 64, 100]

* Sử dụng List Comprehension trong vòng lặp

>>> for i in [x\*x for x in range(11) if x % 2 == 0]:

print(i, sep = ' ', end = ' ')

Output:

0 4 16 36 64 100

**List Class – Lớp List**

Tất cả các đối tượng danh sách là các đối tượng của lớp danh sách trong Python. Sử dụng hàm type () để lấy lớp của đối tượng danh sách.

Ví dụ:

>>> nums=[1,2,3,4]

>>> print(type(nums))

<class 'list'>

**Iterate List – Trình lặp List**

Một danh sách các phần tử có thể được lặp lại bằng cách sử dụng vòng lặp for.

Ví dụ:

>>> names=["Jeff", "Bill", "Steve", "Mohan"]

>>> for name in names:

print(name)

Kết quả:

Jeff

Bill

Steve

Mohan

**Update List – Cập nhật danh sách**

Danh sách có thể thay đổi. Bạn có thể thêm các phần tử mới vào danh sách bằng phương thức append() hoặc insert() và cập nhật các phần tử bằng cách sử dụng chỉ mục.

Ví dụ:

>>> names=["Jeff", "Bill", "Steve", "Mohan"]

>>> names[0]="Newton" # update 1st item at index 0

>>> names[1]="Ram" # update 2nd item at index 1

>>> names.append("Abdul") # adds new item at the end

>>> print(names)

['Newton', 'Ram', 'Steve', 'Mohan', 'Abdul']

**Remove Items – Loại bỏ các phân tử**

Sử dụng các phương thức remove(), pop() hoặc từ khóa del để xóa phần tử trong danh sách hoặc xóa toàn bộ danh sách.

Ví dụ:

>>> names=["Jeff", "Bill", "Steve", "Mohan"]

>>> del names[0] # removes item at index 0

>>> names

['Bill', 'Steve', 'Mohan']

>>> names.remove("Bill") # removes "Bill"

>>> names

['Steve', 'Mohan']

>>> print(names.pop(0)) # return and removes item at index 0

Steve

>>> names

['Mohan']

**List Operators – Các toán tử List**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Trả về **một danh sách** chứa tất cả các phần tử của danh sách thứ nhất và thứ hai. | >>> list1 = [1, 2]  >>> list2 = [3, 4]  >>> list1 + list2  [1, 2, 3, 4] |
| \* | Trả về **một danh sách** sau khi nối nhiều bản sao của chính nó. | >>> list1 = [1, 2]  >>> list1 \* 3  [1, 2, 1, 2, 1, 2] |
| [] | Trả về **phần tử** tại chỉ mục được chỉ định. Chỉ mục âm tính từ vị trí phía bên phải. | >>> list1 = [1, 2, 3]  >>> list1[0]  1  >>> list1[1]  2  >>> list1[2]  3  >>> list1[3]  IndexError: list index out of range |
| **[From : To - 1]** | Trả về **danh sách các phần tử** trong phạm vi được chỉ định bởi hai toán hạng From và To - 1. Nếu toán hạng đầu tiên bị bỏ qua, phạm vi bắt đầu từ chỉ số 0. Nếu toán hạng thứ hai bị bỏ qua, phạm vi sẽ lên đến cuối danh sách. | >>> list1 = [1, 2, 3, 4, 5, 6]  >>> list1[ : 3]  [1, 2, 3]  >>> list1[2 : ]  [3, 4, 5, 6]  >>> list1[1 : 4]  [2, 3, 4] |
| in | Trả về **True** nếu phần tử tồn tại trong danh sách đã cho, ngược lại trả về **False**. | >>> list1 = [1, 2, 3, 4, 5, 6]  >>> 2 in list1  True  >>> 7 in list1  False |
| not in | Trả về **True** nếu phần tử không tồn tại trong danh sách đã cho, ngược lại trả về **False**. | >>> list1 = [1, 2, 3, 4, 5, 6]  >>> 7 not in list1  True  >>> 2 not in list1  False |

**List Methods – Các phương thức List**

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Mô tả** | **Ví dụ** |
| list.append(object) | Nối đối tượng *object* vào cuối danh sách. | >>> cities = ['Mumbai', 'London']  >>> cities.append('Paris')  >>> cities  ['Mumbai', 'London', 'Paris'] |
| list.clear() | Gỡ bỏ tất cả các phần tử có trong danh sách | >>> list1 = [1, 2, 3]  >>> list1.clear()  >>> list1  [] |
| list.copy() | Trả về **bản sao của một danh sách** | >>> oldlist = [1, 2, 3]  >>> newlist = oldlist.copy()  >>> oldlist  [1, 2, 3]  >>> newlist  [1, 2, 3] |
| list.count(value) | Trả về **số lần** phần tử *value* xuất hiện trong danh sách. Nếu giá trị *value* không tồn tại thì trả về 0. | >>> list1 = [1, 2, 3, 2, 4]  >>> count = list1.count(2)  >>> count  2  >>> count = list1.count(5)  >>> count  0 |
| list.extend(iterable) | Thêm tất cả các phần tử từ đối tượng lặp *iterable* (danh sách, tuple, tập hợp, từ điển, chuỗi) vào cuối danh sách.  Lưu ý: Phương thức Extend() khác với thức append () là thêm toàn bộ đối tượng lặp *iterable* dưới dạng một phần tử ở cuối danh sách. | >>> cities = ['Mumbai', 'London']  >>> bigcities = ['Washington DC', 'Banglore']  >>> favcities = ['Paris', 'New York']  >>> cities.extend(favcities)  >>> cities  ['Mumbai', 'London', 'Paris', 'New York']  >>> bigcities.append(favcities)  >>> bigcities  ['Washington DC', 'Banglore', ['Paris', 'New York']] |
| list.index(value, start=0, stop=2147483647) | Trả về **chỉ mục** đầu tiên của giá trị *value* bắt đầu và kết thúc trong phạm vi chỉ mục *start* và *stop* (mặc định *star* là 0 là từ đầu chuỗi, *stop* lớn nhất là 2147483647).  Phát sinh lỗi **ValueError** nếu giá trị không tồn tại trong danh sách. | >>> cities = ['Mumbai', 'London', 'Paris', 'New York']  >>> pos = cities.index('London')  >>> pos  1  >>> pos = cities.index('Delhi')  ValueError: 'Delhi' is not in list |
| list.insert(index, object) | Chèn đối tượng *object* tại chỉ mục *index*. | >>> cities = ['Mumbai', 'London', 'Paris']  >>> cities.insert(0, 'New Delhi')  >>> cities  ['New Delhi', 'Mumbai', 'London', 'Paris'] |
| list.pop(index=-1) | Loại bỏ và trả về **phần tử** tại chỉ mục *index* (mặc định -1 là phần tử cuối cùng).  Phát sinh lỗi **IndexError** nếu danh sách trống hoặc chỉ mục nằm ngoài phạm vi. | >>> cities = ['Mumbai', 'London', 'New York']  >>> cities.pop()  'New York'  >>> cities  ['Mumbai', 'London']  >>> cities.pop(1)  'London'  >>> cities  ['Mumbai']  >>> cities.pop(1)  IndexError: pop index out of range |
| list.remove(value) | Loại bỏ giá trị *value* ra khỏi danh sách.  Phát sinh lỗi **ValueError** nếu giá trị không tồn tại trong danh sách. | >>> cities = ['Mumbai', 'London', 'Paris']  >>> cities.remove('Mumbai')  >>> cities  ['London', 'Paris']  >>> cities.remove('Tokio')  Traceback (most recent call last):  File "<pyshell#3>", line 1, in <module>  cities.remove('Tokio')  ValueError: list.remove(x): x not in list |
| list.reverse() | Đảo ngược vị trí của các phần tử trong danh sách. | >>> list1 = [1, 2, 3, 4, 5]  >>> list1.reverse()  >>> list1  [5, 4, 3, 2, 1] |
| list.sort(key=None, reverse=False) | Sắp xếp các phần tử của danh sách theo thứ tự tăng dần (mặc định reverse = False) hoặc giảm dần (reverse = True).  Sử dụng tham số khóa *key*=*function* để chuyển tên hàm được sử dụng để so sánh thay vì toán tử so sánh mặc định. | >>> list1 = [1, 2, 3, 4, 5]  >>> list1.sort(reverse = True)  >>> list1  [5, 4, 3, 2, 1]  # use len function as key parameter  # ascending sort  >>> cities.sort(key = len)  >>> cities  ['Paris', 'Mumbai', 'New York']  # sum of digits of an integer  >>> def totalDigits(n):  total = 0;  while (n > 0):  total = total + n % 10;  n = int(n / 10);  return total;  # sort ascending by sum of digits  >>> mylist = [564, 1234, 989]  >>> mylist.sort(key = totalDigits)  >>> mylist  [1234, 564, 989] |

**Kiểu Tuple**

Tuple là một tập hợp bất biến (không thể thay đổi) của các phần tử thuộc các kiểu dữ liệu khác nhau. Nó là một tập hợp có thứ tự, vì vậy nó bảo toàn thứ tự của các phần tử mà chúng đã được xác định. Bạn không thể thay đổi giá trị các phần tử của Tuple, nhưng bạn có thể xóa toàn bộ Tuple bằng từ khóa **del**.

Tuple được định nghĩa bằng cách đặt các phần tử trong dấu ngoặc đơn (), phân tách bằng dấu phẩy. Bạn sử dụng hàm type () để kiểm tra kiểu biến Tuple.

Ví dụ:

>>> tpl=(); tpl # empty tuple

()

>>> names = ('Jeff', 'Bill', 'Steve', 'Yash'); names # string tuple

('Jeff', 'Bill', 'Steve', 'Yash')

>>> nums = (1, 2, 3, 4, 5); nums # int tuple

(1, 2, 3, 4, 5)

>>> del nums

>>> nums

NameError: name 'nums' is not defined

>>> employee=(1, 'Steve', True, 25, 12000); employee # heterogeneous data tuple

(1, 'Steve', True, 25, 12000)

>>> type(employee)

<class 'tuple'>

**Tuple Initialization – Khởi tạo Tuple**

Bạn có thể sử dụng cặp dấu ngoặc đơn () hoặc hàm tuple () để khởi tạo biến tuple bằng dãy các phần tử khác (list, set, dictionary)

Ví dụ:

>>> nums = (1, 2, 3, 4, 5);

>>> nums

(1, 2, 3, 4, 5)

>>> names = tuple(['Jeff', 'Bill', 'Steve', 'Yash'])

>>> names

('Jeff', 'Bill', 'Steve', 'Yash')

>>> myset = tuple({'Mot', 'Hai'})

>>> myset

('Hai', 'Mot')

>>> mydict = tuple({1: 'Mot', 2: 'Hai'})

>>> mydict

(1, 2)

**Access Tuple Elements – Truy cập các phần tử Tuple**

Mỗi phần tử trong bộ tuple được truy cập bởi chỉ mục trong dấu ngoặc vuông []. Một chỉ mục bắt đầu bằng 0 và kết thúc bằng (số phần tử - 1). Bạn có thể sử dụng chỉ mục âm. Nếu phần tử tại chỉ mục được chỉ định không tồn tại, thì sẽ phát sinh lỗi "index out of range".

Ví dụ:

>>> nums = (1, 2, 3)

>>> nums[0]

1

>>> nums[1]

2

>>> nums[2]

3

>>> nums[-3]

1

>>> nums[-2]

2

>>> nums[-1]

3

>>> nums[3]

IndexError: tuple index out of range

* Các phần tử Tuple có thể được giải nén và gán cho các biến. Tuy nhiên, số lượng biến phải khớp với số phần tử trong một bộ giá trị; nếu không sẽ phát sinh lỗi.

Ví dụ:

>>> names = ('Jeff', 'Bill', 'Steve', 'Yash')

>>> a, b, c, d = names # unpack tuple

>>> a, b, c, d

('Jeff', 'Bill', 'Steve', 'Yash')

**Tuple Operations – Các toán tử Tuple**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Trả về **một bộ Tuple** chứa tất cả các phần tử của bộ thứ nhất và bộ thứ hai. | >>> tuple1 = (1, 2)  >>> tuple2 = (3, 4)  >>> tuple1+tuple2  (1, 2, 3, 4) |
| \* | Trả về **một bộ Tuple** sau khi nối nhiều bản sao của chính nó. | >>> tuple1 = (1, 2)  >>> tuple1 \* 3  (1, 2, 1, 2, 1, 2) |
| [] | Trả về **phần tử** tại chỉ mục được chỉ định. Chỉ mục âm được tính từ vị trí phía bên phải. | >>> tuple1 = (1, 2, 3)  >>> tuple1[0]  1  >>> tuple1[1]  2  >>> tuple1[2]  3  >>> tuple1[3]  IndexError: tuple index out of range |
| **[From : To - 1]** | Trả về **một bộ** **các phần tử** trong phạm vi được chỉ định bởi hai toán hạng From và To - 1. Nếu toán hạng đầu tiên bị bỏ qua, phạm vi bắt đầu từ chỉ số 0. Nếu toán hạng thứ hai bị bỏ qua, phạm vi sẽ lên đến cuối danh sách. | >>> tuple1 = (1, 2, 3, 4, 5, 6)  >>> tuple1[ : 3]  (1, 2, 3)  >>> tuple1[2 : ]  (3, 4, 5, 6)  >>> tuple1[1 : 4]  (2, 3, 4) |
| in | Trả về **True** nếu phần tử tồn tại trong bộ đã cho, ngược lại trả về **False**. | >>> tuple1 = (1, 2, 3, 4, 5, 6)  >>> 2 in tuple1  True  >>> 7 in tuple1  False |
| not in | Trả về **True** nếu phần tử không tồn tại trong bộ đã cho, ngược lại trả về **False**. | >>> tuple1 = (1, 2, 3, 4, 5, 6)  >>> 7 not in tuple1  True  >>> 2 not in tuple1  False |

**Tuple Methods – Các phương thức Tuple**

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Mô tả** | **Ví dụ** |
| tuple.count(value) | Trả về số lần xuất hiện của giá trị *value*. Nếu giá trị *value* không tồn tại thì trả về 0. | >>> tuple1 = (1, 2, 3, 2, 5, 6, 3)  >>> tuple1.count(2)  2  >>> tuple1.count(7)  0 |
| tuple.index(value, start=0, stop=2147483647) | Trả về **chỉ mục** đầu tiên của giá trị *value* bắt đầu và kết thúc trong phạm vi chỉ mục *start* và *stop* (mặc định *star* là 0 là từ đầu chuỗi, *stop* lớn nhất là 2147483647).  Phát sinh lỗi **ValueError** nếu giá trị không tồn tại trong bộ. | >>> tuple1 = (1, 2, 3, 2, 5, 6, 3)  >>> tuple1.index(2)  1  >>> tuple1.index(4, 2, 5)  ValueError: tuple.index(x): x not in tuple |

* **Kiểu tập hợp: Set, frozenset**

**Kiểu Set**

Một tập hợp Set là một tập có thể thay đổi của các phần tử riêng biệt, giống như danh sách List và bộ Tuple. Nó là một tập hợp các đối tượng không có thứ tự, có nghĩa là nó không ghi lại vị trí phần tử hoặc thứ tự chèn và do đó không thể truy cập các phần tử bằng cách sử dụng chỉ mục.

* Tập hợp trong Python là sự triển khai tập hợp trong Toán học. Một đối tượng tập hợp có các phương thức phù hợp để thực hiện các phép toán tập hợp toán học như hợp, giao, khác, v.v.
* Một đối tượng tập hợp chứa một hoặc nhiều phần tử, không nhất thiết phải cùng loại, được phân tách bằng dấu phẩy và được đặt trong dấu ngoặc nhọn {}.

Ví dụ:

>>> nums = {1, 2, 2, 3, 4, 4, 5, 5} # set of numbers

>>> nums

{1, 2, 3, 4, 5}

>>> emp = {1, 'Steve', 10.5, True} # set of different objects

>>> emp

{1, 10.5, 'Steve'}

>>> myset = {(5,8), 10, 20} # valid set

>>> myset

{10, 20, (5, 8)}

* Bạn sử dụng hàm type () để lấy kiểu của đối tượng tập hợp.

>>> myset = {10, 20, 30}

>>> type(myset)

<class 'set'>

**Set Initialization – Khởi tạo Set**

Bạn sử dụng cặp dấu ngoặc {} để khởi tạo một tập hợp hoặc dùng hàm set () để chuyển đổi đối tượng String, Tuple hoặc Dictonary thành một đối tượng tập hợp, như được hiển thị bên dưới.

Ví dụ:

>>> nums = {1, 2, 3, 4, 5}; nums

{1, 2, 3, 4, 5}

>>> s = set('Hello') # converts string to set

>>> s

{'e', 'l', 'H', 'o'}

>>> t = set((1,2,3,4,5)) # converts tuple to set

>>> t

{1, 2, 3, 4, 5}

>>> d = set({1:'One', 2: 'Two'})# converts tuple to set

>>> d

{1, 2}

**Modify Set Elements – Cập nhật các phần tử Set**

Sử dụng các hàm tập hợp có sẵn các phương thức add (), remove () hoặc update () để cập nhật tập hợp.

>>> nums = {2, 5, 4}

>>> nums.add(6); nums

{2, 4, 5, 6}

>>> update\_nums = {5, 6, 7, 8}

>>> nums.update(update\_nums)

>>> nums

{2, 4, 5, 6, 7, 8}

>>> nums.remove(4)

>>> nums

{2, 5, 6, 7, 8}

**Set Operations – Toán tử tập hợp**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| Phép hợp (Union)  | | Trả về **một tập hợp** **mới** với các phần tử từ cả hai tập hợp. | >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1|s2  {1, 2, 3, 4, 5, 6, 7, 8} |
| Phép giao (Intersection)  & | Trả về **một tập hợp mới** chứa các phần tử chung của cả hai tập hợp. | >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1&s2  {4, 5} |
| Phép khác (Difference)  - | Trả về **một tập hợp** chỉ chứa các phần tử trong tập hợp đầu tiên, và không chứa trong tập hợp thứ hai. | >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1- s2  {1, 2, 3}  >>> s2-s1  {8, 6, 7} |
| Phép khác đối xứng (Symmetric Difference)  ^ | Trả về **một tập hợp** bao gồm các phần tử trong cả hai tập hợp, không bao gồm các phần tử chung. | >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1^s2  {1, 2, 3, 6, 7, 8} |

**Set Methods – Các phương thức tập hợp**

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Mô tả** | **Ví dụ** |
| set.add(element) | Thêm một phần tử *element* vào tập hợp. Nếu một phần tử đã tồn tại trong tập hợp, thì nó không thêm phần tử đó vào. | >>> lang = {'Python', 'C++', 'Java'}  >>> lang.add('Dart')  >>> lang  {'C++', 'Java', 'Dart', 'Python'} |
| set.clear() | Loại bỏ tất cả các phần tử khỏi tập hợp. | >>> lang = {'Python', 'C++', 'Java'}  >>> lang.clear()  >>> lang  set() |
| set.difference(\*other\_set) | Trả về **một** **tập hợp mới** với các phần tử duy nhất không có trong các tập hợp khác được truyền dưới dạng tham số. | >>> nums1 = {1, 2, 2, 3, 4, 5}  >>> nums2 = {4, 5, 6, 7, 8, 8}  >>> nums1.difference(nums2)  {1, 2, 3} |
| set.difference\_update(another\_set) | Xóa tất cả các phần tử có trong tập hợp khác ra khỏi tập hợp này. | >>> cities = {'Mumbai','Chicago','Hong Kong'}  >>> otherCities = {'Mumbai','Chicago'}  >>> cities.difference\_update(otherCities)  >>> cities  {'Hong Kong'}  >>> otherCities  {'Chicago', 'Mumbai'} |
| set.discard(element) | Loại bỏ một phần tử *element* khỏi tập hợp. | >>> cities = {'Mumbai','Chicago','Hong Kong'}  >>> cities.discard('Hong Kong')  >>> cities  {'Mumbai', 'Chicago'} |
| set.intersection(\*other\_sets) | Trả về **một tập hợp mới** với các phần tử tồn tại trong các tập hợp đã cho. | >>> nums = {1, 2, 3, 4, 5 }  >>> oddNums = {1, 3, 5, 7, 9}  >>> commonNums = nums.intersection(oddNums)  >>> commonNums  {1, 3, 5} |
| set.intersection\_update(\*other\_set) | Cập nhật tập hợp mà phương thức instersection\_update() được gọi, với các **phần tử chung** trong các tập hợp được chỉ định. | >>> nums = {1, 2, 3, 4, 5 }  >>> oddNums = {1, 3, 5, 7, 9}  >>> nums.intersection\_update(oddNums)  >>> nums  {1, 3, 5} |
| set.isdisjoint(other\_set) | Trả về **True** nếu các tập hợp đã cho không có phần tử chung. | >>> nums = {1, 2, 3, 4, 5 }  >>> oddNums = {1, 3, 5, 7, 9}  >>> primeNums = {7, 11, 13}  >>> nums.isdisjoint(oddNums)  False  >>> nums.isdisjoint(primeNums)  True |
| set.issubset(other\_set) | Trả về giá trị **True** nếu tập hợp (mà hàm issubset () được gọi) chứa mọi phần tử của tập hợp khác được truyền dưới dạng đối số. | >>> nums = {1, 2, 3, 4, 5 }  >>> oddNums = {1, 3, 5}  >>> primeNums = {1, 3, 5, 7}  >>> oddNums.issubset(nums)  True  >>> primeNums.issubset(nums)  False |
| set.pop() | Loại ra và **trả về một phần tử** ngẫu nhiên từ tập hợp. | >>> cities = {'Mumbai','Chicago','New York'}  >>> cities.pop()  'Mumbai' |
| set.remove(element) | Loại phần tử được chỉ định *element* ra khỏi tập hợp. Nếu phần tử được chỉ định không được tìm thấy sẽ phát sinh ra lỗi KeyError. | >>> cities = {'Mumbai','Chicago','New York'}  >>> cities.remove('Chicago')  >>> cities  {'Mumbai', 'New York'}  >>> cities.remove('mumbai')  KeyError: 'mumbai' |
| set.symmetric\_difference(other\_set) | Trả về **một tập hợp mới** với các phần tử riêng biệt được tìm thấy trong cả hai tập hợp. | >>> nums1 = {1,2,3,4,5}  >>> nums2 = {4,5,6,7,8}  >>> nums3 = nums1.symmetric\_difference(nums2)  >>> nums3  {1, 2, 3, 6, 7, 8} |
| set.symmetric\_difference\_update(other\_Set) | Cập nhật tập hợp mà phương thức instersection\_update() được gọi, với các phần tử chung giữa các tập hợp được chỉ định. | >>> nums = {1, 2, 3, 4, 5 }  >>> oddNums = {1, 3, 5, 7, 9}  >>> nums.intersection\_update(oddNums)  >>> nums  {1, 3, 5} |
| set.union(\*other\_sets) | Trả về **một tập hợp mới** với các phần tử khác biệt với tất cả các tập hợp đã cho. | >>> nums1 = {1,2,3,4,5}  >>> nums2 = {4,5,6,7,8}  >>> nums1.union(nums2)  {1, 2, 3, 4, 5, 6, 7, 8} |
| set.update(iterable) | Cập nhật tập hợp bằng cách thêm các phần tử khác biệt từ một *iterable*. | >>> nums = { 1, 2, 3 }  >>> evenNums = { 2, 4, 6 }  >>> primeNums = { 5, 7 }  >>> nums.update(evenNums, primeNums)  >>> nums  {1, 2, 3, 4, 5, 6, 7} |

**Kiểu frozenset**

Kiểu frozenset là một tập không có thứ tự, có tính bất biến của các phần tử duy nhất. Frozenset cũng giống như set nhưng các frozenset là bất biến, có nghĩa là các phần tử của frozenset không thể được thêm vào hoặc loại bỏ sau khi được tạo.

Bạn sử dụng hàm frozenset([iterable]) để chuyển một đối tượng Iterable thành đối tượng bất biến – làm đóng băng các phần tử.

Ví dụ:

>>> num = (1, 2, 3, 4, 5)

>>> fnum = frozenset(num)

>>> fnum

frozenset({1, 2, 3, 4, 5})

* **Kiểu ánh xạ: Dictionary**

**Kiểu Dictionary**

Từ điển là một tập hợp không có thứ tự chứa các cặp *<khóa>*: <*giá trị>* được phân tách bằng dấu phẩy bên trong dấu ngoặc nhọn {}. Từ điển là sự lựa chọn tối ưu hóa để truy xuất các giá trị khi biết được khóa.

Ví dụ:

>>> d = {} # empty dictionary

>>> d

{}

>>> numNames={1:"One", 2: "Two"} # int key, string value

>>> numNames

{1: 'One', 2: 'Two'}

>>> decNames={1.5:"One and Half", 2.5: "Two and Half"} # float key, string value

>>> decNames

{1.5: 'One and Half', 2.5: 'Two and Half'}

>>> capitals = {"France":"Paris", "India":"New Delhi"} # string key, string value

>>> capitals

{'France': 'Paris', 'India': 'New Delhi'}

* Trong dữ liệu kiểu từ điển, danh sách không được dùng làm khóa. Tuy nhiên, giá trị có thể có kiểu danh sách.

Ví dụ:

>>> dict\_obj = {["Mango","Banana"]:"Fruit", ["Blue", "Red"]:"Color"}

TypeError: unhashable type: 'list'

>>> dict\_obj = {"Fruit":["Mango","Banana"], "Color":["Blue", "Red"]}

>>> dict\_obj

{'Fruit': ['Mango', 'Banana'], 'Color': ['Blue', 'Red']}

* Giá trị khóa không xuất hiện nhiều hơn một lần trong từ điển (không trùng khóa).

>>> numNames = {1:"One", 2:"Two", 3:"Three", 2:"Hai", 1:"Mot"}

>>> numNames

{1: 'Mot', 2: 'Hai', 3: 'Three'}

**Dictionary Initialization – Khởi tạo từ điển**

Bạn có thể khởi tạo từ điển bằng cặp kí hiệu {} hoặc sử dụng hàm khởi tạo dict().

Ví dụ:

>>> mydict = {} #empty dict

>>> mydict

{}

>>> emptydict = dict()#empty dict

>>> emptydict

{}

>>> numNames = {1:"One", 2:"Two", 3:"Three"}

>>> numNames

{1: 'One', 2: 'Two', 3: 'Three'}

>>> numdict = dict(I='one', II='two', III='three')

>>> numdict

{'I': 'one', 'II': 'two', 'III': 'three'}

**Access Dictionary – Truy cập từ điển**

Từ điển là một tập hợp không có thứ tự, vì vậy không thể truy cập một giá trị bằng cách sử dụng một chỉ mục. Thay vào đó, một khóa được chỉ định trong dấu ngoặc vuông.

>>> capitals = {"USA":"Washington DC", "France":"Paris", "India":"New Delhi"}

>>> capitals['USA']

'Washington DC'

>>> capitals[0]

Traceback (most recent call last):

File "<pyshell#21>", line 1, in <module>

capitals[0]

KeyError: 0

* Sử dụng phương thức dictionary.get() để nhận giá trị của khóa. Nếu khóa sai, sẽ trả về None.

Ví dụ:

>>> capitals = {"USA":"Washington DC", "France":"Paris", "India":"New Delhi"}

>>> capitals.get('France')

'Paris'

>>> print(capitals.get('france'))

None

**Access Dictionary using For Loop – Truy cập từ điển sử dụng vòng lặp for**

Sử dụng vòng lặp for để lặp qua các phần tử một từ điển trong tập lệnh Python.

>>> capitals = {"USA":"Washington D.C.", "France":"Paris", "India":"New Delhi"}

>>> for key in capitals:

print(key, capitals[key])

USA Washington D.C.

France Paris

India New Delhi

**Update Dictionary – Cập nhật từ điển**

* Gán một giá trị mới cho phần tử bằng cách truy cập đối tượng của từ điển thông qua khóa.

>>> captains = {"England":"Root", "Australia":"Smith"}

>>> captains['Australia'] = 'Paine'

>>> captains

{'England': 'Root', 'Australia': 'Paine'}

* Sử dụng khóa mới để thêm phần tử vào từ điển.

>>> captains

{'England': 'Root', 'Australia': 'Paine'}

>>> captains['SouthAfrica']='Plessis'

>>> captains

{'England': 'Root', 'Australia': 'Paine', 'SouthAfrica': 'Plessis'}

**Deleting Values from a Dictionary – Xóa giá trị từ từ điển**

Sử dụng các phương thức từ khóa **del**, pop () hoặc popitem () để xóa phần tử (một cặp <khóa>: <giá trị>) khỏi từ điển hoặc chính đối tượng từ điển. Để xóa một cặp, hãy sử dụng khóa của nó làm tham số. Để xóa một đối tượng từ điển, hãy sử dụng tên của nó.

Ví dụ:

>>> captains = {'England': 'Root', 'Australia': 'Paine', 'India': 'Virat'}

>>> del captains['India']

>>> captains

{'England': 'Root', 'Australia': 'Paine'}

>>> captains.pop('Australia')

'Paine'

>>> captains

{'England': 'Root'}

>>> del captains

>>> captains

NameError: name 'captains' is not defined

**Retrieve Dictionary Keys and Values – Truy xuất giá trị và khóa của từ điển**

Các phương thức khóa () và giá trị () trả về một đối tượng dạng xem có chứa khóa và giá trị tương ứng.

Ví dụ:

>>> d1 = {'name': 'Steve', 'age': 21, 'marks': 60, 'course': 'Computer Engg'}

>>> d1.keys()

dict\_keys(['name', 'age', 'marks', 'course'])

>>> type(d1.keys())

<class 'dict\_keys'>

>>> d1.values()

dict\_values(['Steve', 21, 60, 'Computer Engg'])

>>> type(d1.values())

<class 'dict\_values'>

**Check Dictionary Keys – Kiểm tra khóa từ điển**

Bạn sử dụng toán tử in hoặc not in để kiểm tra xem khóa có tồn tại trong từ điển hay không.

>>> captains = {'England': 'Root', 'Australia': 'Paine', 'India': 'Virat'}

>>> 'England' in captains

True

>>> 'USA' not in captains

True

**Built-in Dictionary Methods – Các phương thức sẳn có của từ điển**

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Mô tả** | **Ví dụ** |
| dict.clear() | Xóa tất cả các cặp khóa-giá trị khỏi từ điển. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.clear()  >>> numdict  {} |
| dict.copy() | Trả về **một bản sao** của từ điển. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.copy()  {1: 'one', 2: 'two', 3: 'three''} |
| dict.fromkeys(iterable, value=None) | Tạo **một từ điển** mới từ iterable (string, list, Tuple, set) dưới dạng các khóa và với giá trị được chỉ định. | >>> keys = {'Mumbai','Bangalore'}  >>> value = 'city'  >>> dictionary = dict.fromkeys(keys, value)  >>> dictionary  {'Mumbai': 'city', 'Bangalore': 'city'} |
| dict.get(key, default=None) | Trả về **giá trị** của khóa *key* được chỉ định. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.get(1)  'one' |
| dict.items() | Trả về **một đối tượng** dạng xem của từ điển dưới dạng danh sách các cặp khóa-giá trị. Đối tượng dạng xem này thay đổi khi từ điển thay đổi. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.items()  dict\_items([(1, 'one'), (2, 'two'), (3, 'three')])  >>> type(numdict.items())  <class 'dict\_items'> |
| dict.keys() | Trả về **một đối tượng** dạng xem có chứa danh sách các khóa của từ điển. | numdict = {1:'one',2:'two',3:'three'}  >>> numdict.keys()  dict\_keys([1, 2, 3])  >>> type(numdict.keys())  <class 'dict\_keys'> |
| dict.pop(key[, default]) | Loại bỏ khóa *key* và trả về **giá trị** của nó. Nếu một khóa không tồn tại trong từ điển, thì trả về giá trị mặc định *default* nếu có, nếu không sẽ phát sinh lỗi KeyError. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.pop(3)  'three'  >>> numdict.pop(3, 'Error value')  'Error value'  >>> numdict.pop(3)  KeyError: 3 |
| dict.popitem() | Loại bỏ và trả về **một bộ cặp (khóa, giá trị)** khỏi từ điển. Các cặp được trả lại theo thứ tự vào sau ra trước (Last In First Out-LIFO). Nếu từ điển rỗng thì phát sinh lỗi KeyError. | >>> numdict = {1:'one',2:'two',3:'three'}  >>> numdict.popitem()  (3, 'three')  >>> numdict  {1: 'one', 2: 'two'}  >>> mydict = {}  >>> mydict.popitem()  KeyError: 'popitem(): dictionary is empty' |
| dict.setdefault(key, default=None) | Trả về **giá trị** của khóa *key* được chỉ định trong từ điển. Nếu không tìm thấy khóa *key*, thì nó sẽ thêm khóa với giá trị mặc định *default* được chỉ định. Nếu tham số giá trị mặc định không được chỉ định, thì nó sẽ đặt None. | >>> romanNums = {'I':1, 'II':2, 'III':3 }  >>> value = romanNums.setdefault('VI', 4)  >>> value  4  >>> romanNums  {'I': 1, 'II': 2, 'III': 3, 'VI': 4} |
| dict.update(iterable) | Cập nhật từ điển với các cặp khóa-giá trị từ một từ điển khác. | >>> romanNums = {'I':1,'III':3}  >>> evenRomanNums = {'II':2,'IV':4}  >>> romanNums.update(evenRomanNums)  >>> romanNums  {'I': 1, 'III': 3, 'II': 2, 'IV': 4} |
| dict.values() | Trả về **đối tượng dạng xem** từ điển dùng để xem tất cả các giá trị trong từ điển. Đối tượng này thay đổi khi từ điển thay đổi. | >>> romanNums = {'I':1,'II':2,'III':3}  >>> romanNums.values()  dict\_values([1, 2, 3]) |

**Vấn đề 11. Python - Operators**

Toán tử là các ký hiệu đặc biệt thực hiện một số thao tác trên các toán hạng và trả về kết quả. Ví dụ, 5 + 6 là một biểu thức, trong đó + là một toán tử thực hiện phép toán cộng số học trên toán hạng bên trái số 5 và toán hạng bên phải 6 và trả về kết quả là tổng của hai toán hạng.

Python có **operator** module bao gồm các phương thức cơ bản cho mỗi toán tử. Ví dụ, toán tử + gọi phương thức operator.add (a, b).

Dưới đây là danh sách các toán tử trong Python:

## Arithmetic (Số học), Assignment (Toán tử gán), Comparison (So sánh), Logical (lôgic), Identity (Nhận dạng), Membership (Thành viên), Bitwise (Thao tác bit)

**Arithmetic Operators – Toán tử số học**

Toán tử số học thực hiện phép toán thông thường trên các toán hạng số.

Các toán tử số học trả về kiểu kết quả phụ thuộc vào kiểu toán hạng:

1. Nếu một trong hai toán hạng là số phức, kết quả được chuyển thành phức;
2. Nếu một trong hai toán hạng là số dấu phẩy động, kết quả được chuyển đổi thành dấu phẩy động;
3. Nếu cả hai toán hạng đều là số nguyên thì kết quả là số nguyên và không cần chuyển đổi.

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| Cộng: + | operator.add(a, b) | Cộng hai toán hạng | >>> x = 5  >>> y = 6  >>> x + y  11  >>> import operator as oper  >>> oper.add(5, 6)  11 |
| Trừ: - | operator.sub(a, b) | Toán hạng bên trái từ toán hạng bên phải | >>> x = 10  >>> y = 5  >>> x - y  5  >>> import operator as oper  >>> oper.sub(10, 5)  5 |
| Nhân: \* | operator.mul(a, b) | Nhân hai toán hạng | >>> x = 3  >>> y = 2  >>> x \* y  6  >>> import operator as oper  >>> oper.mul(3, 2)  6 |
| Lũy thừa: \*\* | operator.pow(a, b) | Toán hạng bên trái được nâng lên toán hạng bên phải lũy thừa | >>> x = 3  >>> y = 2  >>> x \*\* y  9  >>> import operator as oper  >>> oper.pow(3, 2)  9 |
| Chia: / | operator.truediv(a, b) | Chia toán hạng bên trái cho toán hạng bên phải | >>> x = 5  >>> y = 2  >>> x / y  2.5  >>> import operator as oper  >>> oper.truediv(5, 2)  2.5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Chia nguyên: //  Tương đương  math.floor(a, b) | operator.floordiv(a, b) | Lấy phần nguyên sau khi chia toán hạng bên trái cho toán hạng bên phải | >>> x = 20  >>> y = 6  >>> x // y  3  >>> import operator as oper  >>> oper.floordiv(20, 6)  3 |
| Chia dư: % | operator.mod(a, b) | Lấy phần còn lại (dư) sau khi chia toán hạng bên trái thành toán hạng bên phải phần. | >>> x = 20  >>> y = 6  >>> x % y  2  >>> import operator as oper  >>> oper.mod(20, 6)  2 |

**Assignment Operators – Toán tử gán**

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| = |  | Gán bằng | >>> x = 5  >>> x  5 |
| += | operator.iadd(a, b) | Tương đương a = a + b | >>> x = 5  >>> y = 2  >>> x += y  >>> x  7  >>> import operator as oper  >>> oper.iadd(5, 2)  7 |
| -= | operator.isub(a, b) | Tương đương a = a – b | >>> x = 5  >>> y = 2  >>> x -= y  >>> x  3  >>> import operator as oper  >>> oper.isub(5, 2)  3 |
| \*= | operator.imul(a, b) | Tương đương a \*= b | >>> x = 5  >>> y = 2  >>> x \*= y  >>> x  10  >>> import operator as oper  >>> oper.imul(5, 2)  10 |
| /= | operator.itruediv(a, b) | Tương đương a = a / b | >>> x = 5  >>> y = 2  >>> x /= y  >>> x  2.5  >>> import operator as oper  >>> oper.itruediv(5, 2)  2.5 |
| //= | operator.ifloordiv(a, b) | Tương đương a = a // b | >>> x = 5  >>> y = 2  >>> x //= y  >>> x  2  >>> import operator as oper  >>> oper.ifloordiv(5, 2)  2 |
| %= | operator.imod(a, b) | Tương đương a = a % b | >>> x = 5  >>> y = 2  >>> x %= y  >>> x  1  >>> import operator as oper  >>> oper.imod(5, 2)  1 |
| &= | operator.iand(a, b) | Tương đương a = a and b | >>> x = 11  >>> x &= 3  >>> x  3  >>> import operator as oper  >>> oper.iand(11, 3)  3 |

|  |  |  |  |
| --- | --- | --- | --- |
| |= | operator.ior(a, b) | Tương đương a = a or b | >>> x = 5  >>> x |= 3  >>> x  7  >>> import operator as oper  >>> oper.ior(5, 3)  7 |
| >>= | operator.irshift(a, b) | Tương đương a = a >> b | >>> x = 5  >>> x >> 2  1  >>> import operator as oper  >>> oper.irshift(5, 2)  1 |
| <<= | operator.ilshift(a, b) | Tương đương a = a << b | >>> x = 5  >>> x << 2  20  >>> import operator as oper  >>> oper.ilshift(5, 2)  20 |

**Comparison Operators – Toán tử so sánh**

Các toán tử so sánh so sánh hai toán hạng và trả về giá trị lôgic True hoặc False.

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| > | operator.gt(a, b) | Lớn hơn | >>> 5 > 3  True  >>> import operator as oper  >>> oper.gt(5, 3)  True |
| < | operator.lt(a, b) | Bé hơn | >>> 3 < 5  True  >>> import operator as oper  >>> oper.lt(3, 5)  True |
| == | operator.eq(a, b) | Bằng | >>> 5 == 3  False  >>> import operator as oper  >>> oper.eq(5, 3)  False |
| != | operator.ne(a, b) | Khác | >>> 5 != 3  True  >>> import operator as oper  >>> oper.ne(5, 3)  True |
| >= | operator.ge(a, b) | Lớn hơn hoặc bằng | >>> 5 >= 3  True  >>> import operator as oper  >>> oper.ge(5, 3)  True |
| <= | operator.le(a, b) | Bé hơn hoặc bằng | >>> 3 <= 5  True  >>> import operator as oper  >>> oper.le(3, 5)  True |

**Logical Operators – Toán tử lôgic**

Các toán tử logic được sử dụng để kết hợp hai biểu thức boolean. Các phép toán logic thường được áp dụng kiểm tra tính đúng sai của đối tượng, kiểm tra nhận dạng và các hoạt động lôgic.

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| and | Trả về True nếu cả hai toán hạng đều là True. | >>> True and True  True  >>> True and False  False  >>> 5 > 3 and 2 == 4  False |
| or | Trả về True nếu có từ một toán hạng đúng | >>> True or True  True  >>> True or False  True  >>> 5 > 3 or 2 == 4  True |
| not | Đảo ngược giá trị toán hạng | >>> not True  False  >>> not False  True  >>> not 3 < 5  False |

**Identity Operators – Toán tử nhận dạng**

Các toán tử nhận dạng kiểm tra xem hai đối tượng có cùng giá trị id (cả hai đối tượng đều trỏ đến cùng một vị trí bộ nhớ).

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| is | operator.is\_(a,b) | Trả về True nếu cả hai cùng giá trị. | >>> x = 5; y = 6  >>> x is y  False  >>> import operator as oper  >>> oper.is\_(x, y)  False |
| is not | operator.is\_not(a,b) | Trả về True nếu cả hai không cùng giá trị. | >>> x = 5; y = 6  >>> y is not x  True  >>> import operator as oper  >>> oper.is\_not(x, y)  True |

**Membership Test Operators – Toán tử kiểm tra thành viên**

Toán tử thành viên kiểm tra dãy (sequence) có chứa phần tử hay không.

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| in | operator.contains(a,b) | Trả về True nếu dãy chứa phần tử được chỉ định, ngược lại trả về False. | >>> nums = [1,2,3,4,5]  >>> 1 in nums  True  >>> 10 in nums  False  >>> 'str' in 'string'  True  >>> import operator as oper  >>> oper.contains(nums, 2)  True |
| not in | not operator.contains(a,b) | Trả về True nếu dãy không chứa phần tử được chỉ định, ngược lại trả về False. | >>> nums = [1,2,3,4,5]  >>> 6 not in nums  True  >>> 'str' not in 'string'  False  >>> import operator as oper  >>> not oper.contains(nums, 2)  False |

**Bitwise Operators – Toán tử thao tác Bit**

Toán tử bitwise thực hiện các phép toán trên toán hạng nhị phân.

|  |  |  |  |
| --- | --- | --- | --- |
| **Toán tử** | **Hàm** | **Mô tả** | **Ví dụ** |
| & | operator.and\_(a, b) | Đặt mỗi bit thành 1 nếu cả hai bit đều là 1. | >>> x = 5; y = 10  >>> z = x & y  >>> z  0  >>> import operator as oper  >>> oper.and\_(x, y)  0 |
| | | operator.or\_(a, b) | Đặt mỗi bit thành 1 nếu có ít nhất một bit là 1. | >>> x = 5; y = 10  >>> z = x | y  >>> z  15  >>> import operator as oper  >>> oper.or\_(x, y)  15 |
| ^ | operator.xor(a, b) | Đặt mỗi bit thành 1 nếu chỉ một trong hai bit là 1. | >>> x = 5; y = 10  >>> z = x ^ y  >>> z  15  >>> import operator as oper  >>> oper.xor(x, y)  15 |
| ~ | operator.invert(a) | Đảo ngược tất cả các bit. | >>> x = 5; z = ~x  >>> z  -6  >>> import operator as oper  >>> oper.invert(x)  -6 |
| << | operator.lshift(a, b) | Di chuyển sang trái bằng cách đẩy các số không từ bên phải vào và để các bit ngoài cùng bên trái rơi ra. | >>> x = 5; z = x << 2  >>> z  20  >>> import operator as oper  >>> oper.lshift(x, 2)  20 |
| >> | operator.rshift(a, b) | Di chuyển sang phải bằng cách đẩy bit ngoài cùng bên trái vào từ bên trái *b* lần và để các bit ngoài cùng bên phải rơi ra. | >>> x = 5; x >> 2  1  >>> import operator as oper  >>> oper.rshift(x, 2)  1 |

**Operator Precedence – Độ ưu tiên toán tử**

Tất cả các toán tử mà ngôn ngữ hỗ trợ được chỉ định một mức độ ưu tiên. Trong một biểu thức, tất cả các toán tử có mức độ ưu tiên cao nhất được thực hiện đầu tiên. Sau khi thu được các kết quả đó, các toán tử có mức độ ưu tiên cao nhất tiếp theo sẽ được thực hiện. Vì vậy, nó tiếp tục, cho đến khi biểu thức được đánh giá đầy đủ. Bất kỳ toán tử nào có mức độ ưu tiên như nhau được thực hiện theo thứ tự từ trái sang phải.

Ví dụ biểu thức 20 + 4 \* 10 có kết quả là 60. Để thay đổi thứ tự ưu tiên, bạn đặt biểu thức có toán tử trong cặp dấu ngoặc đơn. Ví dụ (20 + 4) \* 10 có kết quả là 240. Dưới đây là bảng độ ưu tiên của các toán tử trong Python.

|  |  |  |
| --- | --- | --- |
| **Ưu tiên** | **Toán tử** | **Mô tả** |
| Cao nhất | \*\* | Lũy thừa |
|  | +x, -x, ~x | Toán tử khẳng định một lần, phủ định một lần, đảo ngược |
|  | \*, /, //, % | Toán tử nhân, chia, chia nguyên, chia dư |
|  | +, - | Toán tử cộng, trừ |
|  | <<, >> | Toán tử chuyển bit |
|  | & | Toán tử bitwise AND |
|  | ^, | | Toán tử bitwise XOR, OR |
|  | ==, !=, <=, <, >=, >, is, is not | Toán tử So sánh và nhận dạng |
|  | not | Toán tử boolean NOT |
|  | and | Toán tử boolean AND |
| Thấp nhất | or | Toán tử boolean OR |

* Trong Python, bạn có thể viết biểu thức với liên tiếp nhiều toán tử, như ví dụ dưới đây:

>>> x = 5

>>> 3 <= x <= 10 # similar 3 <= x and x <= 10

True

>>> True and False and Falsle

False

>>> True or False or Falsle

True

**Vấn đề 12. Python - if, elif, else**

**if Condition – Điều kiện if**

Theo mặc định, các câu lệnh trong script được thực thi tuần tự từ câu đầu tiên đến câu cuối cùng. Nếu xử lý logic được yêu cầu, luồng tuần tự có thể được thay đổi theo điều kiện theo sau if.

Cú pháp:

if boolean\_expression:

statement1

statement2

...

statementN

Theo sau từ khóa if là một biểu thức logic *boolean\_expression*. Python sử dụng dấu : và nhấn Enter sau biểu thức để bắt đầu một khối và thụt lề tự động. Một hoặc nhiều câu lệnh được viết với cùng một mức độ thụt lề sẽ được thực thi nếu biểu thức *boolean\_expression* được đánh giá là True. Để kết thúc khối, hãy giảm thụt lề tương ứng.

Ví dụ: bạn có thể viết nhiều câu lệnh điều kiện if liên tiếp như minh họa dưới đây

price = 100

if price > 100:

print("price is greater than 100")

if price == 100:

print("price is 100")

if price < 100:

print("price is less than 100")

Output:

price is 100

* Câu lệnh cùng mức thụt lề với if sẽ được thực hiện theo luồng và không phụ thuộc vào điều kiện của if nên không được xem xét trong khối của if trước nó.

Ví dụ:

price = 50

if price == 100:

print("price is less than 100")

print("Alway executed")# does not depend on if

Output:

Alway executed

**else Condition – Điều kiện else**

Cùng với câu lệnh if, điều kiện tùy chọn else có thể được sử dụng để xác định một khối câu lệnh thay thế sẽ được thực thi nếu biểu thức boolean trong điều kiện if được đánh giá là False.

Cú pháp:

if boolean\_expression:

statement1

statement2

...

statementN

else:

statement1

statement2

...

statementN

Ví dụ:

price = 50

if price >= 100:

print("price is greater than 100")

else:

print("price is less than 100")

Output:

price is less than 100

**elif Condition – Điều kiện elif**

Điều kiện elif được sử dụng để bao gồm nhiều biểu thức điều kiện sau điều kiện if. Khối elif được thực thi nếu điều kiện sau nó được đánh giá là True.

Cú pháp:

if boolean\_expression:

statements

elif boolean\_expresion:

statements

elif boolean\_expresion:

statements

else:

statements

Ví dụ:

price = 100

if price > 100:

print("price is greater than 100")

elif price == 100:

print("price is 100")

elif price < 100:

print("price is less than 100")

Output:

price is 100

**Nested if, elif, else Conditions – Điều kiện if, elif, else lồng nhau**

Python hỗ trợ điều kiện if, elif và else lồng nhau. Điều kiện bên trong phải có thụt lề tăng lên so với điều kiện bên ngoài và tất cả các câu lệnh trong một khối phải có cùng một thụt lề.

Ví dụ:

price = 50

quantity = 5

amount = price\*quantity

if amount > 100:

if amount > 500:

print("Amount is greater than 500")

else:

if amount < 500 and amount > 400:

print("Amount is")

elif amount < 500 and amount > 300:

print("Amount is between 300 and 500")

else:

print("Amount is between 200 and 500")

elif amount == 100:

print("Amount is 100")

else:

print("Amount is less than 100")

Output:

Amount is between 200 and 500

**Vấn đề 13. Python - While Loop**

Python sử dụng từ khóa **while** để tạo thành một vòng lặp có điều kiện, trong đó việc thực thi lặp lại một khối câu lệnh được thực hiện trong khi biểu thức boolean được đánh giá là True.

Cú pháp:

while boolean\_expression:

statement1

statement2

...

statementN

Theo sau từ khóa while là một biểu thức điều kiện boolean\_expression. Theo sau biểu thức điều kiện là dấu : biểu tượng để bắt đầu một khối có thụt lề tăng lên. Khối này có các câu lệnh được thực thi lặp đi lặp lại. Một khối như vậy thường được gọi là phần thân của vòng lặp. Phần thân sẽ tiếp tục thực thi trong khi điều kiện được đánh giá là Đúng. Nếu khi nó trở thành Sai, chương trình sẽ thoát khỏi vòng lặp.

Ví dụ:

num =0

while num < 5:

num = num + 1

print('num = ', num)

Output:

num = 1

num = 2

num = 3

num = 4

num = 5

**Exit from the While Loop – Kết thúc vòng lặp While**

Từ khóa **break** được dùng để thoát khỏi vòng lặp while ngay lập tức. Thông thường break được dùng để thoát khỏi vòng lặp khi thỏa mãn điều kiện nào đó trong câu lệnh if.

Ví dụ:

num = 0

while num < 5:

num += 1

print('num = ', num)

if num == 3:

break

Output:

num = 1

num = 2

num = 3

**Continue Next Iteration – Tiếp tục lặp**

Sử dụng từ khóa **continue** để bắt đầu l3ần lặp tiếp theo và bỏ qua các câu lệnh sau câu lệnh continue trong một số điều kiện.

Ví dụ:

num = 0

while num < 5:

num += 1

if 3 <= num <=4:

continue

print('num = ', num)

Output:

num = 1

num = 2

num = 5

**While Loop with else Block – Vòng lặp While với khối else**

Khối **else** có thể sử dụng cùng với vòng lặp while. Khối else sẽ được thực thi khi biểu thức boolean của vòng lặp while được đánh giá là False.

Ví dụ:

num = 0

while num < 3:

num += 1

print('num = ', num)

else: print('else block executed')

Output:

num = 1

num = 2

num = 3

else block executed

Ví dụ sau đây sẽ nhận input từ là số từ người dùng, sau đó tính giá trị trung bình. Nhấn -1 để kết thúc nhập.

num=0

count=0

sum=0

while num>=0:

num = int(input('Enter any number .. -1 to exit: '))

if num >= 0:

count += 1

sum += num

avg = sum/count

print('Total numbers: ', count, ', Average: ', format(avg, '>6.2'))

**Vấn đề 14. Python - For Loop**

Trong Python, từ khóa **for** cung cấp một cơ chế toàn diện hơn để tạo thành một vòng lặp. Vòng lặp for được sử dụng với các kiểu trình tự như string, list, tuple, set, range, v.v.

Cú pháp 1:

for x in sequence:

statement1

statement2

...

StatementN

Để bắt đầu, một biến x trong câu lệnh for tham chiếu đến phần tử chỉ số 0 trong dãy sequence. Khối các câu lệnh có thụt lề đồng nhất sau biểu tượng: sẽ được thực thi. Sau đó biến x tham chiếu đến mục tiếp theo và lặp lại phần nội dung của vòng lặp cho đến khi hết dãy.

Cú pháp 2:

for var\_1, var\_2,… var\_n in sequence:

statement1

statement2

...

StatementN

Ở cấu trúc vòng lặp này, bạn có thể **for**bao nhiêu biến theo sau cũng được. Nhưng phải đảm bảm một điều rằng, nếu bạn dùng **for** với n biến thì mỗi phần tử trong sequence cũng phải bao gồm n (không lớn hơn hoặc nhỏ hơn) giá trị.

Ví dụ: vòng lặp với danh sách List

nums = [10, 20, 30, 40, 50]

for i in nums:

print(i)

Output:

10

20

30

40

50

Ví dụ: vòng lặp với bộ Tuple

nums = (10, 20, 30, 40, 50)

for i in nums:

print(i)

Output:

10

20

30

40

50

Ví dụ: vòng lặp với chuỗi String

for char in 'Hello':

print (char)

Output:

H

e

l

l

o

Ví dụ: vòng lặp với hàm range ()

for i in range(5):

print(i)

Output:

0

1

2

3

4

Ví dụ: vòng lặp với từ điển Dictionary

numNames = { 1:'One', 2: 'Two', 3: 'Three'}

for k,v in numNames.items():

print("key = ", k , ", value =", v)

Output:

key = 1 , value = One

key = 2 , value = Two

key = 3 , value = Three

Ví dụ: vòng lặp với danh sách lồng các đối tượng khác

mylist = [[1, 2, 3], (4, 5, 6), {7, 8, 9}]

for i, j, k in mylist:

print(i, j, k)

Output:

1 2 3

4 5 6

8 9 7

**Exit the For Loop – Kết thúc vòng lặp**

Việc thực hiện vòng lặp for có thể dừng và thoát bằng cách sử dụng từ khóa **break** với một số điều kiện.

for i in range(1, 5):

if i > 3:

break

print(i, sep = ' ', end = ' ')

Output:

1 2 3

**Continue Next Iteration – Lặp với phần tử tiếp theo**

Sử dụng từ khóa continue để bỏ qua quá trình thực thi hiện tại và tiếp tục trong lần lặp tiếp theo bằng cách sử dụng từ khóa continue với một số điều kiện.

for i in range(1, 5):

if 2 <= i <= 3:

continue

print(i, sep = ' ', end = ' ')

Output:

1 4

**For Loop with Else Block – Vòng lặp For với khối Else**

Khối else có thể theo sau vòng lặp for, khối này sẽ được thực thi khi vòng lặp for kết thúc.

Ví dụ:

for i in range(2):

print(i)

else:

print('End of for loop')

Output:

0

1

End of for loop

**Nested for Loop – Vòng lặp For lồng nhau**

Nếu vòng lặp bên ngoài được thiết kế để thực hiện m lần lặp và vòng lặp bên trong được thiết kế để thực hiện n lần lặp lại thì khối nội dung của vòng lặp bên trong sẽ được thực hiện m X n lần.

Ví dụ:

for x in range(1,3):

for y in range(1,3):

print('x = ', x, ', y = ', y)

Output:

x = 1 , y = 1

x = 1 , y = 2

x = 2 , y = 1

x = 2 , y = 2

**Python Generator Functions – Hàm trình tạo trong Python**

Python cung cấp một trình tạo để tạo hàm vòng lặp của riêng bạn. Một trình tạo là một loại hàm đặc biệt không trả về một giá trị duy nhất, thay vào đó, nó trả về một đối tượng trình lặp với một chuỗi giá trị. Trong một hàm trình tạo, câu lệnh **yield** được sử dụng thay vì câu lệnh return.

Hàm trình tạo không thể bao gồm từ khóa return. Nếu bạn bao gồm nó, thì nó sẽ kết thúc hàm. Sự khác biệt giữa yield và return là trong khi câu lệnh return trả về một giá trị, chấm dứt việc thực thi hàm và giải phóng bộ nhớ, còn yield trả về một giá trị và tạm dừng việc thực thi trong khi vẫn duy trì các trạng thái bên trong.

Ví dụ: Hàm trình tạo và việc sử dụng hàm next () để duyệt qua các phần tử của đối tượng trình lặp.

>>> def mygenerator():

print('First item')

yield 10

print('Second item')

yield 20

print('Last item')

yield 30

>>> gen = mygenerator()

>>> next(gen)

First item

10

>>> next(gen)

Second item

20

>>> next(gen)

Last item

30

>>> next(gen) # an error is raised after the last element

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

next(gen)

StopIteration

**Using for Loop with Generator Function – Sử dụng vòng lặp với hàm trình tạo**

Hàm trình tạo được sử dụng trong vòng lặp.

Ví dụ:

def square\_of\_sequence(x):

for i in range(x):

yield i\*i

for x in square\_of\_sequence(5):

print(x, sep = ' ', end = ' ')

Output:

0 1 4 9 16

**Generator Expression – Biểu thức trình tạo**

Python cũng cung cấp biểu thức trình tạo, đây là một cách ngắn hơn để xác định các hàm trình tạo đơn giản. Biểu thức trình tạo là một hàm tạo ẩn danh.

Ví dụ:

for i in (x \* x for x in range(5)):

print(i, sep = ' ', end = ' ')

Output:

0 1 4 9 16

**Vấn đề 15. Python - Functions**

Python bao gồm nhiều hàm tích hợp sẵn. Các hàm này thực hiện một nhiệm vụ được xác định trước và có thể được gọi trong bất kỳ chương trình nào. Tuy nhiên, nếu bạn không tìm thấy một hàm tích hợp sẵn phù hợp để phục vụ mục đích của mình, bạn có thể xây dựng một hàm mới.

**Defining a Function – Định nghĩa hàm**

Hàm là một khối câu lệnh có thể tái sử dụng, được thiết kế để thực hiện một tác vụ nhất định. Python cung cấp từ khóa **def** để bạn định nghĩa một hàm.

Cú pháp:

def function\_name(parameters):

"""docstring"""

statement1

statement2

...

...

return [expr]

Theo sau từ khóa def là tên của hàm và cặp dấu ngoặc đơn. Bên trong cặp dấu ngoặc đơn có thể không có hoặc có một hay nhiều tham số parameters. Sau dấu ngoặc đơn là dấu : để bắt đầu các khối lệnh của hàm bằng việc tăng thụt lề.

Câu lệnh đầu tiên trong thân hàm có thể là một chuỗi, được gọi là docstring. Nó giải thích chức năng của hàm / lớp. Chuỗi docstring là không bắt buộc.

Phần thân hàm chứa một hoặc nhiều câu lệnh thực hiện một số hành động. Bạn cũng có thể sử dụng từ khóa **pass**.

Trong thân hàm có thể có một hoặc nhiều câu lệnh tùy chọn return, nó trả điều khiển thực thi về môi trường gọi nó. Theo sau return là một biểu thức trả về giá trị của hàm.

Ví dụ:

def greet():

"""This function displays 'Hello World!'"""

print('Hello World!')

**Function Parameters – Tham số hàm**

Có thể định nghĩa một hàm nhận một hoặc nhiều tham số (còn gọi là đối số) và sử dụng chúng để xử lý bên trong khối hàm. Các tham số/đối số này được gọi là tham số/đối số hình thức (formal arguments/parameters) và có thể được đặt tên phù hợp với mục đích sử dụng.

Các đối tượng thực sự được sử dụng trong khi gọi hàm được gọi là các đối số/tham số thực tế (actual arguments/parameters).

Ví dụ:

def greet(name1, name2, name3):

print ('Hello ', name1, ' , ', name2 , ', and ', name3)

greet('Steve', 'Bill', 'Yash') # calling function with string argument

Output:

Hello Steve , Bill , and Yash

**Function with Return Value – Hàm với giá trị trả về**

Trong hầu hết các trường hợp, chúng ta cần kết quả của hàm để sử dụng trong các quy trình tiếp theo. Do đó, khi một hàm trả về, nó cũng sẽ trả về một giá trị.

Một hàm do người dùng định nghĩa cũng có thể trả về một giá trị cho môi trường gọi bằng cách đặt một từ khóa return trước một biểu thức trả về. Để sử dụng giá trị trả về của hàm, ta thường gán hàm cho một biến hoặc cũng có thể sử dụng trực tiếp lời gọi hàm.

Ví dụ:

def sum(a, b):

return a + b

total = sum(2, 3)

print(total)

print(sum(5, 4))

Output:

5

9

**Parameter with Default Value – Tham số với giá trị mặc định**

Trong khi định nghĩa một hàm, các tham số của nó có thể được gán các giá trị mặc định. Giá trị mặc định này được thay thế nếu một đối số thực tế thích hợp được truyền khi hàm được gọi. Tuy nhiên, nếu đối số thực tế không được cung cấp, giá trị mặc định sẽ được sử dụng bên trong hàm.

Ví dụ:

def exponent(x, y = 2):

"""calculate x power y with default exponent of 2"""

return pow(x, y)

print(exponent(3))# call function with default parameters

print(exponent(3, 3))

Output:

9

27

**Unknown Number of Arguments – Không biết số lượng tham số**

Một hàm trong Python có thể có một số lượng đối số không xác định (tham số tùy biến) bằng cách đặt \* trước tham số nếu bạn không biết số lượng đối số mà người dùng sẽ chuyển.

Ví dụ:

def greet(\*name):

print(name)

greet('Steve', 'Bill', 'Yash')

Output:

('Steve', 'Bill', 'Yash')

* Do đối số \*name được truyền dưới dạng một bộ Tuple nên bạn có thể dùng vòng lặp for để duyệt qua các phần tử của đối tượng tham số.

Ví dụ:

def greet(\*name):

for i in name:

print(i, sep = ' ', end = ' ')

greet('Steve', 'Bill', 'Yash')

Output:

Steve Bill Yash

**Function with Keyword Arguments – Hàm với tham số khóa**

Để gọi một hàm có đối số, bạn phải cung cấp số đối theo trình tự tham số hình thức trong định nghĩa hàm. Tuy nhiên, một hàm có thể được gọi bằng cách chuyển các giá trị tham số bằng cách sử dụng các tên tham số theo bất kỳ thứ tự nào.

Ví dụ:

def greet(firstname, lastname):

print ('Hello', firstname, lastname)

greet(lastname='Jobs', firstname='Steve') # passing parameters in any order

Output:

Hello Steve Jobs

**Keyword Argument \*\*kwarg – Tham số khóa tùy biến**

Một hàm trong Python có thể có một số lượng tham số khóa không xác định (tham số khóa tùy biến) bằng cách đặt \*\* trước tham số nếu bạn không biết số lượng đối số khóa mà người dùng sẽ chuyển, như ví dụ dưới đây:

def greet(\*\*person):

print(person)

greet(lastname='Jobs', firstname='Steve')

Output:

{'lastname': 'Jobs', 'firstname': 'Steve'}

* Do đối số \*\*kwarg được truyền dưới dạng một từ điển Dictionary nên bạn có thể dùng vòng lặp for để duyệt qua các phần tử của đối tượng từ điển thông qua khóa.

Ví dụ:

def greet(\*\*person):

for key in person:

print(key, person[key])

greet(lastname='Jobs', firstname='Steve')

Output:

lastname Jobs

firstname Steve

**Lambda Functions in Python – Hàm Lambda trong Python**

Từ khóa **lambda** được sử dụng để xác định các hàm ẩn danh trong Python.

Cú pháp:

lambda [arguments] : expression

* Hàm lambda được gán cho một biến và bạn có thể sử dụng tên biến như tên hàm.

Ví dụ:

>>> square = lambda x : x \* x

>>> square(5)

25

* Bạn cũng có thể khai báo một hàm lambda và gọi nó như một hàm ẩn danh, mà không cần gán nó cho một biến.

Ví dụ:

>>> (lambda x: x\*x)(5)

25

* Thông thường bạn dùng hàm lambda trong tình huống sử dụng hàm một lần trong chương trình.

Ví dụ:

>>> mylist = list(map(lambda x : x \* x, [1, 2, 3, 4, 5]))

>>> mylist

[1, 4, 9, 16, 25]

**Vấn đề 16. Python - Variable Scope**

* Biến cục bộ (local variable) là biến chỉ được sử dụng trong khối mà nó được xác định.

Ví dụ:

>>> def greet():

name = 'Steve'

print('Hello ', name)

>>> greet()

Hello Steve

* Biến toàn cục (global variable) là biến được khai báo bên ngoài bất cứ khối nào. Gía trị của nó được sử dụng trong các khối theo sau nó.

Ví dụ:

>>> name='John'

>>> def greet():

print ("Hello ", name)

>>> greet()

Hello John

* Biến toàn cục và biến cục bộ có thể trùng tên. Trong một khối, khi trùng tên, ta sử dụng tên biến trùng là đang sử dụng biến cục bộ.

Ví dụ:

>>> name = 'Steve'

>>> def greet():

name = 'Bill'

print('Hello ', name)

>>> greet()

Hello Bill

* Bạn có thể gán lại giá trị cho biến toàn cục bên trong khối bằng cách sử dụng từ khóa **global**.

Ví dụ:

>>> name = 'Steve'

>>> def greet():

global name

name = 'Bill'

print('Hello ', name)

>>> greet()

Hello Bill

>>> name

'Bill'

**Vấn đề 17. Python - Modules**

Bất kỳ tệp văn bản nào có phần mở rộng .py chứa mã Python về cơ bản là một mô-đun. Các đối tượng Python khác nhau như hàm, lớp, biến, hằng, v.v., được xây dựng trong một mô-đun có thể được cung cấp cho trình thông dịch hoặc một tập lệnh Python. Các chức năng được xây dựng trong các mô-đun tích hợp sẵn cần phải được import trước khi sử dụng.

Nếu thuật toán phức tạp, yêu cầu xác định nhiều hàm và lớp, chúng cần được tổ chức hợp lý trong các mô-đun. Một mô-đun lưu trữ các lớp, chức năng và các tài nguyên khác có liên quan. Cấu trúc mã sử dụng các mô-đun như vậy giúp dễ hiểu, dễ sử dụng và dễ bảo trì.

**Creating a Module – Tạo một mô-đun**

Đoạn mã sau tạo các hàm tính tổng, tính trung bình, tính lũy thừa và lưu với tên calc.py như một mô-đun trong thư mục làm việc hiện tại.

# calc.py

def sum(x,y):

"""sum of x and y"""

return x + y

def average(x, y):

"""average of x and y"""

return (x + y)/2

def power(x, y):

"""exponentiation of x and y"""

return x\*\*y

**Importing a Module – Nhập mô-đun**

Sử dụng câu lệnh import module\_name để sử dụng mô\_đun.

Ví dụ:

import calc

total = calc.sum(3, 5)

print(total)

Output:

8

**Renaming the Imported Module – Đổi tên mô-đun đã nhập**

Sử dụng từ khóa as để đổi tên mô-đun đã nhập.

Ví dụ:

import math as m

total = m.sqrt(4)

print(total)

Output:

2.0

**from .. import statement – câu lệnh from..import**

Câu lệnh nhập ở trên sẽ tải tất cả các tài nguyên của mô-đun trong môi trường làm việc hiện tại (còn được gọi là không gian tên). Có thể nhập các đối tượng cụ thể từ một mô-đun bằng cách sử dụng cú pháp from .. import ..

Ví dụ:

from calc import sum

total = sum(3, 5)

print(total)

Output:

8

Bạn cũng có thể sử dụng tất cả các hàm của mô-đun bằng cú pháp impor..from \*

Ví dụ:

from calc import \*

avg = average(3, 5)

print(avg)

Output:

4.0

**Getting Help on Modules – Nhận giúp đỡ trong mô-đun**

Bạn có thể hiển thị giúp đỡ của một mô-đun bằng câu lệnh help (). Trước đó, bạn cần lưu tập tin mã nguồn và tập tin mô-đun cùng một thư mục để trình thông địch xác định thư mục đó là thư mục hiện tại.

Ví dụ:

>>> import calc

>>> help(calc)

Help on module calc:

NAME

calc

FUNCTIONS

average(x, y)

average of x and y

power(x, y)

exponentiation of x and y

sum(x, y)

sum of x and y

FILE

d:\tai lieu python\bt lap trinh python\calc.py

Xem hết trang 13