LẬP TRÌNH PYTHON TỪ CƠ BẢN ĐẾN NÂNG CAO

Thực hiện: TS.Trần Bình Long

CHƯƠNG 1 LẬP TRÌNH PYTHON

1. Giới thiệu Python
2. Từ khóa và định danh
3. Cách viết câu lệnh, thụt lề và chú thích
4. Kiểu dữ liệu

Lịch sử của Python

Python là ngôn ngữ lập trình được tạo bởi Guido Van Rossum vào cuối những năm 1980, được phát hành lần đầu tiên vào tháng 2 năm 1991.

Python không phải tên rắn thần Python trong thần thoại Hy Lạp. Python được đặt tên trong sê-ri "Monty Python's Flying Circus", chương trình hài cuối những năm 1970

Đặt tính của Python

Ngôn ngữ lập trình đơn giản, dễ học: Python có cú pháp rất đơn giản, rõ ràng, dễ đọc và viết hơn so với C++, Java, C#.

Miễn phí, mã nguồn mở: tự do sử dụng và phân phối Python.

Khả năng di chuyển: chương trình Python có thể chạy trên các nền tảng khác nhau như Windows, macOS, Linux.

Khả năng mở rộng và có thể nhúng: có thể kết hợp code C, C++ và những ngôn ngữ khác vào code Python, khả năng scripting.

Đặt tính của Python

Ngôn ngữ thông dịch cấp cao: với Python không cần quản lý bộ nhớ, thu dọn dữ liệu,... Python tự động chuyển đổi code sang ngôn ngữ máy tính.

Thư viện tiêu chuẩn lớn để giải quyết những tác vụ phổ biến: Python có số lượng thư viện lớn giúp việc lập trình dễ dàng, không cần viết nhiều code.

Ví dụ: cần kết nối cơ sở dữ liệu MySQL, nhập thư viện MySQLdb.

Hướng đối tượng: Python là hướng đối tượng (OOP) giúp giải quyết những vấn đề phức tạp một cách trực quan.

Ứng dụng của Python

Lập trình ứng dụng web: có thể tạo web app: Django, Flask, Pyramid, Plone, Django CMS. Các trang như Mozilla, Reddit, Instagram và PBS đều viết bằng Python.

Khoa học và tính toán: nhiều thư viện cho khoa học và tính toán số liệu: SciPy và NumPy.

EarthPy cho khoa học trái đất, AstroPy cho Thiên văn học,... Ngoài ra, còn machine learning, khai thác dữ liệu và deep learning.

Ứng dụng của Python

Tạo nguyên mẫu phần mềm: Python chậm hơn so với C++ và Java. Nhưng Python là ngôn ngữ để tạo những nguyên mẫu (bản chạy thử - prototype).

Ví dụ: dùng Pygame (thư viện viết game) tạo nguyên mẫu game trước. Sau đó dùng C++ để viết game.

Ngôn ngữ tốt để dạy lập trình: Python được nhiều công ty, trường học sử dụng do cú pháp đơn giản và dễ dùng.

Python trở nên phổ biến

Cú pháp đơn giản:
Lập trình bằng Python dễ dàng, dễ hiểu. Cú pháp của Python giống với ngôn ngữ tự nhiên,

ví dụ

```
a = 2
b = 3
tong = a + b
print(tong)
```

Có thể đoán được đoạn code này: cho hai số a, b, tính tổng và in ra tổng.

Python trở nên phổ biến

Không quá khắt khe:
Không cần xác định kiểu của biến, không cần dấu chấm phẩy vào cuối câu lệnh. Python yêu cầu tuân theo những bài tập có sẵn, giúp việc học Python dễ dàng.

Viết code ít hơn:
Python cho phép viết chương trình có nhiều chức năng với ít dòng code hơn.

Cộng đồng lớn, hỗ trợ tốt:
Có cộng đồng hỗ trợ rộng lớn, nhiều diễn đàn hoạt động trực tuyến trợ giúp khi gặp vấn đề trong Python

Cài đặt Python trên Windows

Tải Python: https://www.python.org/downloads/, chọn phiên bản cần.
Nhấp đúp vào file vừa tải về để cài đặt. Có 2 tùy chọn, chọn một để cài.

- **Install Now:** Mặc định cài Python vào ổ C, cài sẵn IDLE (cung cấp giao diện đồ họa để làm việc với Python), pip và tài liệu, tạo shortchut,...
- **Customize installation:** chọn vị trí cài và tính năng cần thiết.

Cài đặt Python trên Windows

Mở IDLE, tạo file mới trong IDLE, click File\New Windows hoặc nhấn tổ hợp phím Ctrl + N.
Sao chép đoạn code:

```
print("Xin chào!")
```

Vào file vừa tạo, nhấn Ctrl + S để lưu, đặt tên, file sẽ có đuôi là .py, ví dụ: test_python.py.

Nhấn Run\Run module hoặc F5 để xem kết quả

BÀI TẬP

1. print "Hello World!"
2. print "Hello Again"
3. print "I like typing this."
4. print "This is fun."
5. print 'Yay! Printing.'
6. print "I'd much rather you 'not'."
7. print 'I "said" do not touch this.'

Từ khóa và định danh

Python có từ khóa (keyword) và định danh (identifier) như các ngôn ngữ lập trình khác.

Những quy tắc cơ bản về keyword và định danh.

- **Keyword của Python**

Không dùng keyword để đặt tên cho biến, hằng hay định danh.

Ngoại trừ True, False và None được viết hoa, các keyword khác viết chữ thường.

Danh sách các từ khóa trong Python: (Python 3.0, các phiên bản mới hơn có thêm keyword bổ sung)

Từ khóa (Keyword)

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

Định danh

Định danh là tên đặt cho các thực thể như class, function, biến,..., giúp phân biệt các thực thể với nhau.

Quy tắc viết định danh:

Định danh kết hợp chữ cái viết thường (a - z), viết hoa (A - Z), số (0 - 9), dấu gạch dưới (_).

Ví dụ: bien_1, tinh_tong_0_9, firstClass.

Định danh không bắt đầu bằng số.

Ví dụ: 1bien là sai, bien1 thì đúng.

Định danh

Định danh phải khác các keyword.

Ví dụ: and = 1 chương trình sẽ báo lỗi "SyntaxError: invalid syntax". (and là keyword)

Định danh không hỗ trợ các ký tự đặc biệt: !, @, #, $, %,.... Nếu gán sẽ nhận báo lỗi "SyntaxError: invalid syntax" hoặc "NameError:...“

Định danh có độ dài tùy ý.

Định danh lớp (class) thường bắt đầu với chữ cái hoa. Các định danh khác bắt đầu với chữ cái thường.

Định danh

Định danh bắt đầu với một dấu dạch dưới _ là định danh private.

Định danh bắt đầu với 2 dấu gạch dưới __ thì mức độ private cao hơn.

Định danh bắt đầu và kết thúc bằng 2 dấu gạch dưới (**__init__**) là định danh đặc biệt được Python định nghĩa.

Nên đặt tên định danh có nghĩa.

Ví dụ: c = 10 vô nghĩa, nhưng count = 10 rõ nghĩa và dễ hiểu hơn.

Định danh

Định danh có phân biệt chữ hoa, chữ thường.

Ví dụ: bien và Bien là khác nhau.

Định danh nhiều từ, dùng dấu gạch dưới giữa các từ, vd: day_la_mot_vi_du.

Hoặc viết hoa từng từ DayLaMotViDu để phân biệt các từ trong trường hợp tên định danh dài.

Cách viết lệnh

Câu lệnh trong Python

Các dòng hướng dẫn để trình thông dịch Python thực hiện được gọi là các câu lệnh. Ví dụ, count = 0 là lệnh gán. Hay các dòng khác như: lệnh if, lệnh for, lệnh while,...

Viết câu lệnh trên nhiều dòng

Câu lệnh được kết thúc khi xuống dòng. Muốn viết câu lệnh trên nhiều dòng dùng ký tự (\). Ví dụ

```
sum = 1 + 3 + 5 + \
      7 + 9 + 11 + \
      13 + 15 + 17
```

Cách viết lệnh

Hoặc dùng dấu ngoặc đơn (), ngoặc vuông [ ] hay ngoặc nhọn { }. Ví dụ:

```
sum = (1 + 3 + 5 +
       7 + 9 + 11 +
       13 + 15 + 17)
mau_sac = {"vàng",
           "xanh",
           "cam"}
```

Có thể viết nhiều lệnh trên cùng một dòng, phân cách nhau bởi dấu chấm phẩy ; ví dụ:

```
a = 1; b = 2; c = 3
```

Thụt lề

Thụt lề trong Python

- Trong C, C++ hay Java sử dụng cặp { } để xác định các khối lệnh.
- Trong Python, khối lệnh được xác định thông qua thụt lề.
- Trong Python, nếu thụt hoặc thò nhầm sẽ bị báo lỗi.
- Khối lệnh (của hàm, vòng lặp,...) bắt đầu với thụt lề và kết thúc với dòng không thụt lề.
- Thụt lề phải nhất quán trong suốt khối lệnh, các lệnh trong cùng một khối phải có độ thụt lề bằng nhau.

Thụt lề

Thụt lề trong Python

4 lần phím cách để thụt lề được hay dùng hơn phím tab, ví dụ:

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

Khối lệnh của for gồm print(i), if được viết thụt lề bằng nhau, break là lệnh trong if, được thụt lề thêm đoạn nữa.

Thụt lề

Nếu lệnh trên được viết:

```
for i in range(1,11):
print(i)
    if i == 5:
        break
```

Sẽ nhận được thông báo lỗi, và lỗi sẽ hiện trước lệnh print(i).

Thụt lề

Câu lệnh viết trên một dòng không cần thụt lề. Tuy nhiên, viết dạng thụt lề nhìn dễ hiểu hơn. Ví dụ:

```
if True:
    print('Xin chào!')
    q = 10
```

và

```
if True: print('Xin chào!'); q = 10
```

Cả hai cách viết trên đều đúng và thực hiện công việc như nhau, nhưng cách viết đầu nhìn dễ hiểu hơn.

Chú thích

Chú thích, bình luận trong Python

- Chú thích để người viết code giao tiếp với người đọc, là phần không thể thiếu trong chương trình, mô tả điều đang xảy ra trong chương trình để người đọc không tốn nhiều thời gian tìm hiểu, suy đoán.
- Khi nhiều chương trình được viết cùng lúc, đan xen nhau, sự nhầm lẫn hoặc quên chi tiết quan trọng của đoạn code đã viết là không tránh khỏi. Do đó những chú thích, bình luận rất quan trọng.

Chú thích

Trong Python, ký tự # để bắt đầu một chú thích. Khi thông dịch, Python sẽ bỏ qua những chú thích này.

```
#Đây là chú thích
#In dòng chữ DaiHocLacHong
print('DaiHocLacHong')
```

Chú thích viết trên cùng một dòng với lệnh hoặc biểu thức.

```
print('DaiHocLacHong') #Đây là chú thích In dòng chữ DaiHocLacHong
```

Chú thích

Chú thích trên nhiều dòng, thêm # vào trước mỗi dòng

```
# Đây là chú thích
# trên nhiều dòng
# In dòng chữ DaiHocLacHong
# trong Python
print(‘DaiHocLacHong')
```

Chú thích

Dùng 3 dấu nháy đơn ' ' ' hay nháy kép " " ". Cách này được sử dụng để viết chú thích trên nhiều dòng.
Lưu ý: câu lệnh Docstring cũng dùng ba dấu nháy kép.

```
"""Đây là chú thích
trên nhiều dòng
In dòng chữ DaiHocLacHong
trong Python"""
```

Docstring

Docstring (Documentation string): câu lệnh đầu tiên trong mô đun, hàm, định nghĩa method, mô tả công việc mà hàm, lớp sẽ làm.
Ba dấu nháy kép được dùng để viết docstring
ví dụ

```
def nhandoi(num):
    """Hàm nhân đôi giá trị"""
    return 2*num
```

Docstring

Docstring xuất hiện dưới dạng thuộc tính **__doc__** của hàm. Để xem docstring sử dụng lệnh print() như sau

```
print(nhandoi.__doc__)
```

Kết quả:

```
Hàm nhân đôi giá trị
```

Biến

Biến là vị trí trong bộ nhớ được dùng để lưu trữ dữ liệu (giá trị). Biến được đặt tên duy nhất để phân biệt với các biến khác.

Quy tắc viết tên biến giống như quy tắc viết định danh.

Không cần khai báo biến trước khi sử dụng, chỉ cần gán cho biến một giá trị và nó được xác định.

Không cần khai báo kiểu, kiểu được nhận tự động dựa vào giá trị đã gán cho biến.

Biến

Gán giá trị cho biến:

- Để gán giá trị cho biến dùng toán tử = .Bất kỳ giá trị nào cũng có thể gán cho biến.
  Ví dụ

```
truong =  ‘Lạc Hồng’
coso = 1
lau = 3.5
```

Trong 3 lệnh gán, “Lạc Hồng" là chuỗi ký tự, được gán cho biến truong, 1 là số nguyên, được gán cho coso, 3.5 là số thập phân được gán cho lau.

Biến

Gán nhiều giá trị:

- Có thể gán nhiều giá trị trong một lệnh như sau:

```
truong, coso, lau = ‘’Lac Hồng", 1, 3.5
```

- Gán giá trị giống nhau cho nhiều biến:

```
truong, coso, lau = 1
```

Lệnh trên sẽ gán giá trị 1 cho cả 3 biến là truong, coso và lau.

Kiểu số

Các kiểu dữ liệu số trong Python

- Số nguyên, số thực và số phức, lần lượt được định nghĩa **int**, **float**, **complex**.
- Số nguyên và số thực phân biệt bằng dấu thập phân. Ví dụ: 9 là số nguyên, 9.0 là số thực.
- Số phức sử dụng hậu tố j hoặc J để chỉ phần ảo. Ví dụ: 1+4j . Ngoài **int** và **float**, Python hỗ trợ thêm 2 loại số nữa là **Decimal** và **Fraction**.
- Hàm **type()** kiểm tra biến hoặc giá trị thuộc lớp số nào và hàm **isinstance()** kiểm tra chúng thuộc class cụ thể nào.

Kiểu số

```
>>> a = 5
    print(type(a))
    # Output: <class 'int'>
    print(type(5.0))
    # Output: <class 'float'>
>>> b = 8 + 2j
    print(b + 2)
    # Output: (10+2j)
    # Kiểm tra b có phải là số phức không
    print(isinstance(b, complex))
    # Output: True
```

Kiểu số

Số nguyên trong Python không giới hạn độ dài,

Số thực có giới hạn 15 số sau dấu thập phân.

Ngoài hệ thập phân còn có hệ thống nhị phân, bát phân và thập lục phân.

Để biểu diễn những hệ số này cần đặt một tiền tố trước số đó.

Tiền tố hệ số cho các số Python

Hệ thống số	Tiền tố
Hệ nhị phân	'0b' hoặc '0B'
Hệ bát phân	'0o' hoặc '0O'
Hệ thập lục phân	'0x' hoặc '0X'

Kiểu số

Ví dụ dùng các tiền tố hệ số trong Python, hàm print() in ra màn hình số tương ứng trong hệ số 10.

```
>>> print(0b10111011)
    # Output: 187
    print(0xFA + 0b111)
    # Output: 257 (250 + 7)
    print(0o17)
    # Output: 15
```

Chuyển đổi giữa các kiểu số

Các phép toán cộng, trừ sẽ ngầm chuyển đổi số nguyên thành số thực (đổi tự động) nếu có một toán tử trong phép toán là số thực.

Ví dụ: cộng số nguyên 5 và số thực 1.0, kết quả trả về sẽ là số thực 6.0

```
>>> 5 + 1.0
    6.0
```

Chuyển đổi giữa các kiểu số

Dùng các hàm tích hợp sẵn: **int()**, **float()**, **complex()** để chuyển đổi giữa các kiểu số.

```
>>> int(3.6)
    3
>>> int(-1.2)
    -1
>>> float(7)
    7.0
>>> complex('2+8j')
    (2+8j)
```

Khi chuyển từ số thực sang số nguyên, chỉ lấy phần nguyên bỏ phần thập phân.

Mô-đun Decimal

Class float trong Python.

Nếu tính tổng 3.4 + 4.3 và gán kết quả là 7.7. Sẽ nhận kết quả là False.

```
>>> (3.4 + 4.3) == 7.7
    False
```

Do giới hạn của máy tính.

Phép cộng trên cho kết quả:

```
>>> 3.4+4.3
    7.699999999999999
```

Mô-đun Decimal

Để khắc phục vấn đề này, có thể sử dụng mô-đun **Decimal**. **float** lấy 15 số sau dấu thập phân thì mô-đun **Decimal** cho phép độ dài của số tùy ý.

```
>>> print(0.1)
    # Output: 0.1
    import decimal
    print(decimal.Decimal(0.1))
#Output:0.1000000000000000055511151231257827021181583404
```

Mô-đun Decimal

Mô-đun này được sử dụng khi thực hiện các phép toán ở hệ số thực.
Ví dụ:

```
>>> from decimal import Decimal
    print(Decimal('1.1') + Decimal('2.2'))
    # Output: 3.3
    print(Decimal('4.0') * Decimal('2.50'))
    # Output: 10.000
```

Mô-đun Decimal

Để ngắn gọn hơn, nhập mô-đun Decimal và đổi tên thành D bằng từ khóa **as**

```
>>> from decimal import Decimal as D
    print(D('1.1') + D('2.2'))
    # Output: 3.3
    print(D('4.0') * D('2.50'))
    # Output: 10.000
```

Trong phép nhân, thêm số 0 vào sau 4 và 2.5 sẽ nhận được tổng các số thập phân.

Các phép toán với **float** được hiện nhanh hơn các phép toán **Decimal**

Mô-đun Decimal

Khi nào nên sử dụng Decimal thay cho float

Sử dụng Decimal trong các trường hợp sau:

Khi tạo ứng dụng tài chính, cần biểu diễn phần thập phân chính xác.

Khi muốn kiểm soát mức độ chính xác của số.

Khi muốn thực hiện các phép toán cho kết quả như qui định.

Phân số

Mô-đun **fractions** cung cấp các phép toán liên quan đến phân số. Phân số có tử số và mẫu số là số nguyên. Tạo phân số (Fraction) theo nhiều cách:

```
>>> import fractions
    # Tạo phân số từ số thập phân
    print(fractions.Fraction(4.5))
    # Output: 9/2
    # Tạo phân số từ số nguyên
    print(fractions.Fraction(9))
    # Output: 9
    # Tạo phân số bằng cách khai báo tử số, mẫu số
    print(fractions.Fraction(2,5))
    # Output: 2/5
```

Phân số

Tạo phân số từ float, kết quả không như mong muốn, do hạn chế của máy tính. Nên thường khởi tạo phân số từ string.

```
>>> import fractions
    # Khởi tạo phân số từ float
    print(fractions.Fraction(0.1))
  # Output: 3602879701896397/36028797018963968
    # Khởi tạo phân số từ string
    print(fractions.Fraction('0.1'))
    # Output: 1/10
```

Phân số

Kiểu dữ liệu phân số hỗ trợ đầy đủ các phép toán cơ bản: cộng, trừ, nhân, chia, logic:

```
>>> from fractions import Fraction as F
    print(F(2,5) + F(3,5))
    # Output: 1
    print(F(2,5) + F(1,5))
    # Output: 3/5
    print(1 / F(3,7))
    # Output: 7/3
    print(F(-2,9) > 0)
    # Output: False
    print(F(-2,9) < 0)
    # Output: True
```

Phép Toán

Python cung cấp mô-đun **math** và **random** để giải quyết các vấn đề toán học: lượng giác, logarit, xác suất và thống kê, v.v... mô-đun **math** có nhiều hàm và thuộc tính.

Ví dụ về **math**

```
>>> import math
    print(math.pi)
    # Output: 3.141592653589793
```

Phép Toán

```
>>> print(math.cos(math.pi))
    # Output: -1.0
    print(math.exp(10))
    # Output: 22026.465794806718
    print(math.log2(4))
    # Output: 2.0
    print(math.sinh(1))
    # Output: 1.1752011936438014
    print(math.factorial(8))
    # Output: 40320
```

Phép Toán

Các phép toán:

Nhập vào một phép tính sẽ cho ra kết quả.

Cú pháp đơn giản, dùng các toán tử +, -, * , /, dấu ngoặc đơn () được dùng để gom nhóm. Ví dụ:

```
 >>> 2 + 2
     4
>>> 50 - 5*6
     20
>>> (50 - 5*6) / 4
     5.0
>>> 8 / 5 # phép chia luôn trả về một số dạng thập phân
    1.6
```

Phép Toán

Số nguyên (ví dụ 2 , 4 , 20) có kiểu int , số thực (vd: 5.0 , 1.6) có kiểu float .
Phép chia (/) luôn luôn trả về kiểu float.
Phép chia lấy phần nguyên dùng toán tử //
Lấy phần dư dùng toán tử % ví dụ:

```
>>> 17 / 3 # phép chia thường trả về số thập phân
    5.666666666666667
>>> 17 // 3 # phép chia lấy phần nguyên
    5
>>> 17 % 3 # phép chia lấy phần dư
    2
>>> 5 * 3 + 2 # thương số * số chia + số dư
    17
```

Phép Toán

Toán tử ** để tính số mũ:

```
>>> 5 ** 2 # 5 bình phương
    25
>>> 2 ** 7 # 2 mũ 7
    128
```

Dấu = dùng để gán giá trị cho biến.

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
    900
```

Phép Toán

Nếu biến chưa được định nghĩa (gán giá trị), sử dụng biến đó sẽ nhận được lỗi sau:

```
>>> n # biến n chưa gán giá trị gì
    Traceback (most recent call last):
      File "<pyshell#8>", line 1, in <module>
        n
    NameError: name 'n' is not defined
```

Python hỗ trợ dấu chấm động, phép tính có số nguyên và số thập phân, kết quả trả về số thập phân.

```
>>> 4 * 3.75 - 1
    14.0
```

Phép Toán

Trong chế độ tương tác, biểu thức được in ra cuối cùng sẽ được gán cho biến _ , giúp dễ dàng thực hiện các phép tính tiếp theo. Ví dụ:

```
>>> muc_thue = 12.5 / 100
>>> tri_gia = 100.50
>>> tri_gia * muc_thue
    12.5625
>>> tri_gia + _
    113.0625
>>> round(_, 2)
    113.06
```

Biến _ là read-only, không gán giá trị cho biến.

Chuỗi (string)

String là một dãy các ký tự. Việc chuyển đổi ký tự thành số được gọi là mã hóa và ngược lại được gọi là giải mã.
ASCII và Unicode là 2 mã phổ biến thường được sử dụng.

Trong Python, string là dãy các ký tự Unicode.

Unicode bao gồm mọi ký tự trong các ngôn ngữ và mang lại tính đồng nhất trong mã hóa.

Tạo string

Chuỗi được đặt trong dấu nháy đơn '...' hoặc kép "..." .
Dấu \' được dùng để thay dấu nháy ‘

```
>>> ‘tin hoc‘ # dấu nháy đơn
     ‘tin hoc’
>>> ‘doesn\‘t’ # sử  dụng \’ để viết dấu nháy đơn...
     “doesn’t”
>>> “doesn’t” # sử dụng dấu nháy kép
     “doesn’t”
>>> ‘”Yes,” he said.’
     ‘”Yes,” he said.’
>>> “\”Yes,\” he said.”
     ‘”Yes,” he said.’
>>> ‘”Isn\’t,” she said.’
     ‘”Isn\’t,” she said.’
```

Tạo string

Chuỗi kết quả bao gồm phần trong dấu ngoặc và ký tự \. Hàm print() cho kết quả dễ đọc hơn:

```
>>> '"Isn\'t," she said.'
    '"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
    "Isn't," she said.
>>> s = 'First line.\nSecond line.' # \n xuống dòng
>>> s # không có print(), \n sẽ được viết trong kết quả
    'First line.\nSecond line.'
>>> print(s) # có print(), \n sẽ tạo ra dòng mới
    First line.
    Second line
```

Tạo string

Để tránh nhầm lẫn giữa \n là xuống dòng và \n là chuỗi thông thường thêm r vào trước dấu nháy đầu tiên:

```
>>> print('C:\some\name') # ở đây \n là xuống dòng
    C:\some
    ame
>>> print(r'C:\some\name') # thêm r trước dấu nháy
    C:\some\name
```

Tạo string

Chuỗi viết trên nhiều dòng bằng cách sử dụng 3 dấu nháy: “””...””” hoặc ‘’’...’’’ . Ví dụ:

```
>>> print(“””
Usage: thingy [OPTIONS]
        -h                      Display this usage message
        -H                      hostname Hostname to
connect to
“””)
```

Kết quả:

```
>>> Usage: thingy [OPTIONS]
        -h                      Display this usage message
        -H                      hostname Hostname to
connect to
```

Danh sách ký tự cho chuỗi

Escape Sequence	Mô tả
\newline	Dấu gạch chéo ngược và dòng mới bị bỏ qua
\\	Dấu gạch chéo ngược
\'	Dấu nháy đơn
\"	Dấu nháy kép
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Ký tự có giá trị bát phân là ooo
\xHH	Ký tự có giá trị thập lục phân là HH

Danh sách ký tự cho chuỗi

Ví dụ:

```
>>> print(“C:\\Python32\\Tinhoc”)
    C:\Python32\Tinhoc
>>> print("In dòng này\nthành 2 dòng")
    In dòng này
    thành 2 dòng
>>> print("In giá trị \x48\x45\x58")
    In giá trị HEX
>>>
```

Truy cập vào phần tử của chuỗi

Cách truy cập vào phần tử của chuỗi

Chuỗi được lập chỉ mục với ký tự đầu tiên đánh số 0.
Mỗi ký tự là một số:

```
>>> text = 'Python'
>>> text[0] # ký tự ở vị trí số 0
    'P'
>>> text[5] # ký tự ở vị trí số 5
    'n'
```

Truy cập vào phần tử của chuỗi

Chỉ số là số âm, bắt đầu đếm từ bên phải:

```
>>> text[-1] # ký tự cuối
    'n'
>>> text[-2] # ký tự thứ hai từ bên phải
    'o'
>>> text[-6]
    'P'
```

Lưu ý số âm bắt đầu từ -1

Cắt chuỗi: index được sử dụng để cắt lấy chuỗi con

```
>>> text[0:2] # các ký tự từ vị trí 0 đến 2 (trừ 2)
    'Py'
>>> text[2:5] # các ký tự từ vị trí 2 đến 5 (trừ 5)
    'tho'
```

Truy cập vào phần tử của chuỗi

Chuỗi con s[:i] + s[i:] bằng chuỗi s

```
>>> text[:2] + text[2:]
    'Python'
>>> text[:4] + text[4:]
    'Python'
>>> text[:2] # ký tự từ đầu đến vị trí thứ 2 (loại bỏ 2)
    'Py'
>>> text[4:] # ký tự từ vị trí thứ 4 đến hết
    'on'
>>> text[-2:] # ký tự thứ hai từ cuối đến hết
    'on'
```

Truy cập vào phần tử của chuỗi

Truy cập phần tử quá lớn sẽ trả về kết quả lỗi

```
>>> text[10] # chỉ có 6 ký tự
    Traceback (most recent call last):
      File "<pyshell#50>", line 1, in <module>
        text[10]
    IndexError: string index out of range
```

Tuy nhiên, nếu cắt chuỗi thì vẫn được khi phần tử quá lớn:

```
>>> text[4:10] # cắt ký tự từ vị trí thứ 4 đến 10
    'on'
>>> text[10:] # cắt ký tự sau vị trí 10
    ''
```

Thay đổi hoặc xóa chuỗi

Chuỗi không thể thay đổi, nếu gán một ký tự nào đó cho vị trí đã lập chỉ mục trong chuỗi sẽ bị báo lỗi:

```
>>> text[0] = 'J'
    . . .
    TypeError: 'str' object does not support item assignment
>>> text[2:] = 'py'
    ...
    TypeError: 'str' object does not support item assignment
```

Thay đổi hoặc xóa chuỗi

Tốt nhất là tạo mới:

```
>>> 'J' + text[1:]
    'Jython'
>>> text[:2] + 'py'
    'Pypy'
```

Không thể xóa hay loại bỏ ký tự khỏi chuỗi, chỉ có thể xóa toàn bộ chuỗi, bằng từ khóa del:

```
>>> del text
```

Ghép chuỗi

Các chuỗi được nối với nhau bằng toán tử + và tự nhân bằng toán tử *

```
>>> # thêm 3 chử 'hoc' vào trước 'mãi'
>>> 3 * 'hoc' + 'mai'
    'hochochocmai'
```

Các ký tự dạng chuỗi (đặt trong dấu nháy) cạnh nhau được nối tự động

```
>>> 'Hoc''mai'
    'Hocmai'
```

Ghép chuỗi

Không ghép chuỗi với biến hay biểu thức:

```
>>> bien = ‘Hoc’
>>> bien ‘mai’ # không thể ghép biến với chuỗi
      SyntaxError: invalid syntax
>>> (3*‘hoc’) ‘mai’
      SyntaxError: invalid syntax
```

Muốn ghép biến với biến hoặc biến với chuỗi sử dụng toán tử +

```
>>>bien+’mai’
     ‘Hocmai’
>>>bien=(‘Xin ’
      ‘chao!’) # ghép chuỗi trên nhiều dòng dùng dấu ( )
      ‘Xin chao!’
```

Dùng vòng Lặp

Dùng vòng lặp for cho chuỗi, ví dụ đếm số ký tự “o” trong chuỗi:

```
>>> dem = 0
      for i in ‘lachong’:
         if (i == ‘o’):
            dem += 1
      print(‘Có’, count, ‘ chữ o được tìm thấy’)
      # Output: Có 1 chữ o được tìm thấy
```

Kiểm tra chuỗi con có trong chuỗi, dùng từ khóa in:

```
>>> ‘on’ in ‘lachong’
      True
>>> ‘co’ in ‘lachong’
      False
```

Hàm làm việc với chuỗi

2 hàm thường dùng với chuỗi là **enumerate()** và **len()**.
Hàm len() trả về độ dài của chuỗi:

```
>>> s = 'lachong'
>>> len(s)
    7
```

Hàm enumerate() trả về đối tượng liệt kê, chứa cặp index và giá trị của phần tử trong chuỗi.

```
>>> s = 'lachong'
    s_enu = list(enumerate(s))
    print('list(enumerate(s) = ', s_enu)
    # Output: list(enumerate(s) = [(0, 'l'), (1, 'a'), (2, 'c'), (3, 'h'), (4, 'o'), (5, 'n'), (6, 'g')]
```

Phương thức format()

Phương thức **format()** dùng để định dạng chuỗi, cặp dấu { } nhận giá trị thay thế, có thể dùng đối số vị trí hoặc từ khóa để thay đổi thứ tự

```
>>> # thứ tự mặc định
    thu_tu_mac_dinh = "{},{} và {}".format ('hoc','hoc nữa','hoc mãi')
    print('\n--- Thứ tự mặc định ---')
    print(thu_tu_mac_dinh)
    # Output: ---Thứ tự mặc định ---
    # Output: hoc, hoc nữa và hoc mãi
```

Phương thức format()

```
>>> # dùng đối số vị trí để sắp xếp thứ tự
    thu_tu_vi_tri= “{1}, {0} và {2}”.format(‘hoc’,’hoc nữa’, ‘hoc mãi’)
    print('\n--- Thứ tự theo vị trí ---')
    print(thu_tu_vi_tri)
    # Output: --- Thứ tự theo vị trí ---
    # Output: hoc nữa, hoc và hoc mãi
>>> # dùng từ khóa để sắp xếp thứ tự
    tu_khoa_thu_tu = "{c}, {b} và {a}".format(a=‘hoc’, b=‘hoc nữa’, c=‘hoc mãi’)
    print('\n--- Thứ tự theo từ khóa ---')
    print(thu_tu_tu_khoa)
    # Output: --- Thứ tự theo từ khóa ---
    # Output: hoc mãi, hoc nữa và hoc
```

Phương thức format()

Định dạng chuỗi: canh trái <, canh phải >, canh giữa ^.

Định dạng số nguyên, số nhị phân, thập lục phân. Tính phân số, hiển thị dạng số mũ.

```
>>> # Căn lề chuỗi
>>>“|{:<10}|{:^10}|{:>10}|”.format(‘Trường’,‘Lạc’,’Hồng’)
    ‘|Trường   |   Lạc   |     Hồng|’
>>> # Định dạng số nhị phân
>>> “Chuyển số {0} sang nhị phân: {0:b}”.format(12)
    ‘Chuyển số 12 sang nhị phân: 1100’
```

Phương thức format()

```
>>> # Hiển thị dạng số mũ
>>> “Số thập phân {0} dạng số mũ: {0:e}”.format(15.35)
      ‘Số thập phân 15.35 dạng số mũ: 1.535000e+01’
>>> # Tính phân số
>>> “1 phần 3  bằng: {0:.3f}”.format(1/3)
      ‘1 phần 3 bằng: 0.333’
```

Định dạng chuỗi

Định dạng chuỗi theo ngôn ngữ C bằng toán tử %

```
>>> x=25.7356
>>>print('Giá trị của x là %3.2f' %x)
      Giá trị của x là 25.74
>>>print('Giá trị của x là %3.4f' %x)
      Giá trị của x là 25.7356
>>>print('Giá trị của x là %1.4f' %x)
      Giá trị của x là 25.7356
>>>print('Giá trị của x là %0.10f' %x)
      Giá trị của x là 25.7356000000
```

Chuỗi định dạng	Kiểu ký tự	Ý nghĩa
%c	Char	Ký tự
%s	String	Chuỗi
%d	Int, short	Số nguyên
%u	Unsigned int, unsigned short	Số nguyên không dấu
%x	Int, short,	Thập lục phân
%o	Int, short	Bát phân
%f	Float	Số thực
%e	Float	Số thực dạng mũ
%g	Float	Số thực
%ld	Long	Số nguyên dài
%lu	Unsigned long	Số nguyên dài không dấu
%lo	Long, unsigned long	Số nguyên dài bát phân
%lx	Long, unsigned long	Số nguyên dài thập lục phân
%lf	Double, unsigned long	Số thực double
%a	Double	Số thực double thập lục phân

Phương thức sử dụng trong chuỗi

Các phương thức làm việc với chuỗi: format(), lower(), upper(), join(), split(), find(), replace(), v.v....

```
>>> “TruongLacHong”.lower()
      ‘truonglachong’
>>> “TruongLacHong”.upper()
       ‘TRUONGLACHONG’
>>> “Truong Lac Hong”.split()
      [‘Truong’, ‘Lac’, ‘Hong’]
>>> ‘ ’.join([‘Truong’, ‘Lac’, ‘Hong’])
       ‘Truong Lac Hong’
>>> “Truong Lac Hong”.find(‘La')
       7
>>> “Truong Lac Hong”.replace(‘Truong’,’Dai hoc’)
       ‘Dai Hoc Lac Hong’
```

Danh sách (list)

Danh sách được biểu diễn bằng dãy các giá trị, phân cách nhau bằng dấu phẩy, nằm trong dấu [ ]. Danh sách có thể chứa nhiều mục với kiểu dữ liệu khác nhau.

```
>>> list1 = [ ] # list rỗng
>>> list2 = [1, 2, 2] # list số nguyên
>>> list3 = [1, "Hello", 3.4] # list với nhiều kiểu dữ liệu

>>> day_so = [10, 5, 33, 8, 16, 21]
>>> day_so
    [10, 5, 33, 8, 16, 21]
```

Tạo list

Có thể tạo các list lồng nhau (danh sách trong danh sách), ví dụ

```
>>> a = ['a', 'b', 'c']
    n = [1, 2, 3]
    x = [a, n]
    print (x) # Output: [['a', 'b', 'c'], [1, 2, 3]]
    print (x[0]) # Output: ['a', 'b', 'c']
    print(x[0][1]) # Output: b
```

Hoặc khai báo list lồng nhau từ đầu:

```
>>> danh_sach = [“list”, [8, 4, 6], [‘a’]]
```

Truy cập phần tử của list

Index (chỉ mục) của list:
Sử dụng index [ ] để truy cập phần tử của list. Index bắt đầu từ 0 đến hết. Index có kiểu số nguyên.

```
>>>list_t = [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
    print(list_t[1.0])
TypeError: list indices must be integers or slices, not float
>>> print(list_t[0])
    # Output: L
>>> print(list_t[2])
    # Output: c
>>> print(list_t[4])
    # Output: o
```

Truy cập phần tử của list

List lồng nhau truy cập bằng index lồng nhau:

```
    # List lồng nhau
>>> ln_list = ["Happy", [1,3,5,9]]
    # Index lồng nhau
>>> print(ln_list[0][1])
    # Output: a
>>> print(ln_list[1][3])
    # Output: 9
```

Truy cập phần tử của list

Index âm:
Dùng đếm phần tử của list ngược từ cuối lên đầu, bắt đầu -1.

```
>>> list_am = [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
>>> print(list_am[-1])
    # Output: g
>>> print(list_am[-5])
    # Output: c
```

Toán tử Slice (:)

Truy cập nhiêu phần tử của list bằng toán tử : trả về list mới chứa các phần tử yêu cầu.

```
>>> slice_list = [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
>>> print(slice_list[1:5])
    # Output: [‘a’, ‘c’, ‘H’, ‘o’]
>>> print(slice_list[:-6])
    # Output: [‘L’]
>>> print(slice_list[3:])
    # Output: [‘H’, ‘o’, ‘n’, ‘g’]
```

Toán tử Slice (:)

Dấu : giữa 2 index cần lấy các phần tử. Vd [1:5] sẽ lấy phần tử 1 đến 4, [:-6] lấy từ 0 đến -7,...

[:] trả về list mới là bản sao của list ban đầu:

```
>>> slice_list = [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
>>> print(slice_list[:])
      # Output: [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
```

Thay đổi hoặc thêm phần tử vào list

Ghép list:

```
>>> day_so + [25, 16, 33, 47, 50]
      [10, 5, 33, 8, 16, 21, 25, 16, 33, 47, 50]
```

Thay đổi phần tử

```
>>> lap_phuong = [1, 8, 27, 65, 125] # [1^3,2^3,3^3,4^3,5^3]
>>> 4 ** 3 # 4^3 là 64, không phải 65
      64
>>> lap_phuong[3] = 64 # thay thế giá trị sai
>>> lap_phuong
      [1, 8, 27, 64, 125]
```

Thay đổi hoặc thêm phần tử vào list

Thêm mục mới vào cuối list bằng phương thức **append()**:

```
>>> lap_phuong.append(216) # thêm lập phương của 6
>>> lap_phuong.append(7 ** 3) # lập phương của 7
>>> lap_phuong
    [1, 8, 27, 64, 125, 216, 343]
```

Thay đổi list, xóa list

```
>>> ky_tu = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’]
>>> ky_tu
    ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # thay đổi giá trị
>>> ky_tu[2:5] = ['C', 'D', 'E']
>>> ky_tu
    ['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # xóa phần tử
>>> ky_tu[2:5] = [ ]
>>> ky_tu
    ['a', 'b', 'f', 'g']
>>> # xóa list bằng cách thay thế bằng list rỗng
>>> ky_tu[:] = [ ]
>>> ky_tu
    [ ]
```

Thay đổi list, xóa list

remove() loại bỏ phần tử đã cho

pop() loại bỏ phần tử tại index chỉ định nếu pop() không chỉ định index sẽ loại bỏ phần tử cuối

clear() xóa tất cả các phần tử trong list

Thay đổi list, xóa list

```
>>> list_f=['L', 'a', 'c', 'H', 'o', 'n', 'g']
>>> list_f.remove('o')
      print(list_f)
      #Output ['L', 'a', 'c', 'H', 'n', 'g']
>>> list_f.pop(1)
       print(list_f)
       #Output ['L', 'c', 'H', 'n', 'g']
>>> list_f.pop() # xóa phần tử cuối
       print(list_f)
       #Output ['L', 'c', 'H', 'n']
>>> list_f.clear()
       print(list_f)
       #Output [ ]
```

Len()

Hàm len() áp dụng với list:

```
>>> ky_tu = ['a', 'b', 'c', 'd']
>>> len(ky_tu)
      4
```

Lệnh del

```
>>> list = [‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’]
      # xóa phần tử index 2
>>> del list[2]
      print(list)
      # Output: [‘L’, ‘a’, ‘H’, ‘o’, ‘n’, ‘g’]
      # xóa phần tử index từ 1 đến 7
>>> del list[1:4]
      print(list)
      # Output: [‘L’, ‘n’, ‘g’]
      # xóa toàn bộ list my_list
>>> del list
      print(list)
      # Error: NameError: name 'list' is not defined
```

Phương thức của list

Những phương thức có sẵn cho list trong Python:
append(): Thêm phần tử vào cuối list.
extend(): Thêm tất cả phần tử của list hiện tại vào list khác.
insert(): Chèn một phần tử vào index cho trước.
remove(): Xóa phần tử khỏi list.
pop(): Xóa phần tử khỏi list, trả về phần tử tại index đã cho.
clear(): Xóa tất cả phần tử của list.
index(): Trả về index của phần tử phù hợp đầu tiên.
count(): Trả về số lượng phần tử đã đếm được trong list như một đối số.
sort(): Sắp xếp các phần tử trong list theo thứ tự tăng dần.
reverse(): Đảo ngược thứ tự các phần tử trong list.
copy(): Trả về bản sao của list

Phương thức của list

Ví dụ

```
>>> day_so = [9,8,7,6,8,5,8]
>>> print(day_so.index(7))
      # Output: 2
>>> print(day_so.count(8))
      # Output: 3
>>> print(day_so.sort())
       # Output: [5, 6, 7, 8, 8, 8, 9]
>>> day_so.reverse()
       print(day_so)
       # Output: [9, 8, 8, 8, 7, 6, 5]
```

Phương thức của list

```
>>> list_f=['L', 'a', 'c', 'H', 'o', 'n', 'g']
>>> print(list_f.index('H'))
    #Output: 3
>>> print(list_f.count('o'))
    #Output: 1
>>> list_f.sort()
    list_f
    #Output ['H', 'L', 'a', 'c', 'g', 'n', 'o']
>>> list_f.reverse()
    list_f
    #Output ['o', 'n', 'g', 'c', 'a', 'L', 'H']
```

List comprehension

List comprehension là một biểu thức đi kèm với lệnh for được đặt trong cặp dấu ngoặc vuông [ ].
Ví dụ

```
>>> boi_3 = [3 ** x for x in range(9)]
    print(boi_3)
    # Output: [1, 3, 9, 27, 81, 243, 729, 2187, 6561]
```

Code trên tương đương với

```
>>> boi_3 = [ ]
    for x in range (9):
        boi_3.append(3**x)
    print(boi_3)
```

List comprehension

Ngoài for, if cũng được dùng trong list comprehension. Lệnh if lọc các phần tử trong list hiện tại để tạo thành list mới. Ví dụ:

```
>>> boi_3 = [3 ** x for x in range(9) if x > 4]
    print(boi_3)
    # Output: [243, 729, 2187, 6561]
 >>> so_le = [x for x in range (18) if x % 2 == 1]
    print(so_le)
    # Output: [1, 3, 5, 7, 9, 11, 13, 15, 17]
>>> noi_list = [x+y for x in ['Ngôn ngữ ','Lập trình '] for y in ['‘Python’, ‘C++’]]
    print(noi_list)
    # Output: ['Ngôn ngữ Python', 'Ngôn ngữ C++', 'Lập trình Python’, Lập trình C++]
```

Kiểm tra phần tử có trong list

Kiểm tra phần tử có trong list. Nếu phần tử đã tồn tại, kết quả trả về là True, ngược lại là False

```
>>> list_f=['L', 'a', 'c', 'H', 'o', 'n', 'g']
    print(‘a’ in list_f)
    # Output: True
    print(‘g’ in list_f)
    # Output: True
    print('z' in list_f)
    # Output: False
```

Vòng lặp for trong list

Dùng vòng lặp for để duyệt qua các phần tử trong list:

```
>>> for x in ['Python','Java','C']:
        print('Tôi thích lập trình ',x)
```

Kết quả trả về sẽ như sau:

```
Tôi thích lập trình Python
Tôi thích lập trình Java
Tôi thích lập trình C
```

Các hàm tích hợp với list

Các hàm Python tích hợp sẵn: all(), any(), enumerate(), len(), max(), min(), list(), sorted(),...

all(): Trả về giá trị True nếu tất cả các phần tử của list đều là true hoặc list rỗng.

any(): Trả về True khi bất kỳ phần tử nào trong list là true. Nếu list rỗng hàm trả về giá trị False.

enumerate(): Trả về đối tượng enumerate, chứa index và giá trị của tất cả các phần tử của list dạng tuple.

len(): Trả về độ dài (số lượng phần tử) của list.

Các hàm tích hợp với list

list(): Chuyển đổi một đối tượng (tuple, string, set, dictionary) thành list.
max(): Trả về phần tử lớn nhất trong list.
min(): Trả về phần tử nhỏ nhất trong list.
sorted(): Trả về list mới đã được sắp xếp.
sum(): Trả về tổng của tất cả các phần tử trong list

Tuple

Tuple là chuỗi các phần tử như list. Trong tuple các phần tử không thể thay đổi, khác với list các phần tử có thể thay đổi.

Tuple được sử dụng với dữ liệu không cho phép sửa đổi và thực hiện nhanh hơn list vì không thể thay đổi.

Tuple được định nghĩa bằng dấu ngoặc đơn (), các phần tử trong tuple cách nhau bằng dấu phẩy (,).

Tuple

Ví dụ:

```
>>> t = (10, 'Lac Hong', 2j)
```

Sử dụng toán tử [ ] để trích xuất phần tử trong tuple nhưng không thể thay đổi giá trị của phần tử

```
t[0:2]
#Output (10, 'Lac Hong')
```

So sánh Tuple và List

Tuple và List gần giống nhau, sử dụng tương tự nhau. Tuy nhiên, tuple có những điểm khác list như:

- Tuple được sử dụng cho kiểu dữ liệu không đồng nhất (khác nhau), list sử dụng cho kiểu dữ liệu đồng nhất (giống nhau).
- Vì tuple không thể thay đổi, việc duyệt qua các phần tử của tuple nhanh hơn so với list.
- Tuple chứa phần tử không đổi, nên được sử dụng như key cho dictionary.
- Khi có dữ liệu không đổi việc sử dụng tuple sẽ đảm bảo dữ liệu được bảo vệ chống ghi (writeprotected)

Tạo Tuple

Các phần tử của Tuple đặt trong dấu ngoặc đơn (), phân cách bằng dấu phẩy.

Tuple không giới hạn số lượng phần tử và kiểu dữ liệu (số nguyên, số thực, list, string,...)

```
>>># Tuple rỗng
    my_tuple = ()
    print(my_tuple)
    # Output: ()
>>># tuple số nguyên
    my_tuple = (2, 4, 16, 256)
    print(my_tuple)
    # Output: (2, 4, 16, 256)
```

Tạo Tuple

```
>>> # tuple có nhiều kiểu dữ liệu
    my_tuple = (10, “LacHong", 3.5)
    print(my_tuple)
    # Output: (10, “LacHong", 3.5)
>>> # tuple lồng nhau
    my_tuple = (“KTLT", [2, 4, 6], (3, 5, 7))
    print(my_tuple)
    # Output: (“KTLT", [2, 4, 6], (3, 5, 7))
>>> # tuple có thể được tạo mà không cần dấu ()
    my_tuple = 10, “LacHong", 3.5
    # Output: (10, “LacHong", 3.5)
```

Tạo Tuple

Tuple có một phần tử cần thêm dấu phẩy sau phần tử đó để phân biệt với chuỗi

```
>>> # tạo tuple không có dấu phẩy sau phần tử
    my_tuple = (“LacHong")
    print(type(my_tuple))
    # Output: <class 'str'>
>>> # thêm dấu phẩy vào cuối
    my_tuple = (“LacHong",)
    print(type(my_tuple))
    # Output: <class 'tuple'>
```

Truy cập vào phần tử của tuple

Truy cập vào phần tử của tuple giống với list.

Index: dùng index [ ] bắt đầu từ 0. Tuple lồng nhau được truy cập bằng cách sử dụng index lồng nhau:

```
>>># tuple lồng nhau
    n_tuple = (“LacHong”, [2, 6, 8], (1, 2, 3))
    # index lồng nhau
    print(n_tuple[0][5])
    # Output: ‘n'
    # index lồng nhau
    print(n_tuple[1][2])
    # Output: 8
```

Index âm: Index -1 phần tử cuối cùng.

Truy cập nhiều phần tử: dùng toán tử : (dấu 2 chấm)

Thay đổi tuple

Các phần tử của tuple không thể thay đổi, nhưng nếu phần tử có kiểu dữ liệu thay đổi được (như list) thì các phần tử (list) thay đổi được.

Có thể gán giá trị khác cho tuple (gọi là gán lại - reassignment)

```
>>> my_tuple = (1, 3, 5, [7, 9])
    my_tuple[1] = 9
    …..
    TypeError: 'tuple' object does not support item assigment
    # không thể thay đổi phần tử của tuple
```

Thay đổi tuple

```
>>> # phần tử có index 3 trong tuple là list
    # list thay đổi được, nên phần tử đó thay đổi được
    my_tuple[3][0] = 8
    print(my_tuple)
    # Output: (1, 3, 5, [8, 9])

>>> # Gán lại giá trị cho tuple
    my_tuple = ('L', 'a', 'c', 'H', 'o', 'n', 'g')
    print(my_tuple)
    # Output: ('L', 'a', 'c', 'H', 'o', 'n', 'g')
```

Ghép Tuple

Dùng toán tử + để nối 2 tuple, toán tử * để lặp lại tuple theo số lần đã cho. Kết quả được một tuple mới

```
>>> # Nối 2 tuple
    print((2, 4, 6) + (3, 5, 7))
    # Output: (2, 4, 6, 3, 5, 7)
    # Lặp lại tuple
    print((‘LacHong’,) * 3)
    # Output: (‘LacHong’, ‘LacHong’, ‘LacHong’)
```

Xóa tuple

Không thể xóa phần tử trong tuple.

Xóa toàn bộ tuple thực hiện với từ khóa del:

```
>>> my_tuple = (‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’)
    del my_tuple[3]
    TypeError: 'tuple' object doesn't support item
deletion
    # Không thể xóa phần tử của tuple
    # Có thể xóa toàn bộ tuple
    del my_tuple
    print (my_tuple)
    # NameError: name ‘my_tuple' is not defined
```

Phương thức và hàm dùng với Tuple

Có 2 phương thức:
count(x): Đếm số phần tử x trong tuple.
index(x): Trả về giá trị index của phần tử x đầu tiên gặp trong tuple

```
>>> my_tuple = (‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’)
    # Count
    print(my_tuple.count(‘n'))
    # Output: 1
    # Index
    print(my_tuple.index('n'))
    # Output: 5
```

Phương thức và hàm dùng với Tuple

Các hàm dùng trong tuple:
all(): Trả về giá trị True nếu tất cả các phần tử của tuple là true hoặc tuple rỗng.
any(): Trả về True nếu bất kỳ phần tử nào của tuple là true, nếu tuple rỗng trả về False.
enumerated(): Trả về đối tượng enumerate (liệt kê), chứa cặp index và giá trị của tất cả phần tử của tuple.
len(): Trả về độ dài (số phần tử) của tuple.
max(): Trả về phần tử lớn nhất của tuple.
min(): Trả về phần tử nhỏ nhất của tuple.
sorted(): Lấy phần tử trong tuple và trả về list mới được sắp xếp (tuple không sắp xếp được).
sum(): Trả về tổng tất cả các phần tử trong tuple.
tuple(): Chuyển đổi những đối tượng có thể lặp (list, string, set, dictionary) thành tuple

Kiểm tra phần tử trong tuple

Kiểm tra một phần tử có trong tuple với từ khóa in

```
>>> my_tuple = (‘L’, ‘a’, ‘c’, ‘H’, ‘o’, ‘n’, ‘g’)
    # Kiểm tra phần tử
    print('a' in my_tuple)
    # Output: True
    print('b' in my_tuple)
    # Output: False
    # Dùng từ khóa Not in
    print('g' not in my_tuple)
    # Output: False
```

Vòng lặp for trong Tuple

Sử dụng vòng for để lặp các phần tử trong tuple

```
>>> for x in ('Python','C++','Web'):
        print("Tôi thích lập trình ", x)
    # Kết quả trả về:
    Tôi thích lập trình Python
    Tôi thích lập trình C++
    Tôi thích lập trình Web
```

Set

Set là tập hợp các phần tử duy nhất, không có thứ tự.

Các phần tử trong set phân cách nhau bằng dấu phẩy và nằm trong dấu ngoặc nhọn { }.

Phần tử trong set có thể thay đổi, có thể thêm hoặc xóa phần tử của set.

Set được sử dụng thực hiện các phép toán tập hợp, giao, ...

Tạo Set

Set không giới hạn số phần tử, chứa nhiều kiểu dữ liệu khác nhau, nhưng không chứa phần tử có thể thay đổi được như list, set hay dictionary

Ví dụ:

```
>>> a = {5,2,3,1,4}
    print("a=", a)
    # Kết quả:
    a = {1, 2, 3, 4, 5}
    # Set với nhiều kiểu dữ liệu
    my_set = {1.0, "Xin chào", (1, 2, 3)}
    print(“my_set= ",my_set)
    #Output: my_set= {1.0, (1, 2, 3), 'Xin chào'}
```

Tạo Set

Tạo set rỗng bằng cặp dấu { } sẽ tạo ra dictionary. Để tạo set rỗng sử dụng hàm set() không đối số.

```
>>> # Tạo set bằng { }
    set_t = { }
    print(type(set_t))
    # Kết quả
    # Output: <class 'dict'>
    # Tạo bằng hàm set()
    set_t = set()
    # Kết quả
    print(type(set_t))
    # Output: <class 'set'>
```

Thay đổi Set

Set không có chỉ mục (index), nên toán tử [ ] không hoạt động.
Nếu sử dụng, sẽ nhận được thông báo lỗi.

```
>>> a[1]
    Traceback (most recent call last):
     File "<pyshell#2>", line 1, in <module>
     a[1]
     TypeError: 'set' object does not support indexing
```

Thay đổi Set

Thêm 1 phần tử sử dụng **add()**, thêm nhiều phần tử dùng **update()**.

Update() nhận tuple, list, strring và set làm đối số. Trong mọi trường hợp, Set có giá trị duy nhất, các giá trị trùng sẽ tự động bị loại bỏ.

```
>>> # Khởi tạo my_set
    my_set = {1,3}
    print(my_set)
    {1, 3}
    my_set[0]
    TypeError: 'set' object does not support indexing
```

Thay đổi Set

```
>>> # Thêm phần tử
    my_set.add(2)
    print(my_set)
    # Output: {1, 2, 3}
>>> # Thêm nhiều phần tử vào set
    my_set.update([2,3,4])
    print(my_set)
    # Output: {1, 2, 3, 4}
>>> # Thêm list và set
     my_set.update([4,5], {1,6,8})
     print(my_set)
     # Output: {1, 2, 3, 4, 5, 6, 8}
```

Xóa phần tử khỏi set

Dùng **discard()** và **remove()** để xóa phần tử khỏi set. Khi phần tử cần xóa không có trong set thì **discard()** không báo lỗi, **remove()** sẽ báo lỗi

```
>>> # Khởi tạo my_set
    my_set = {1, 3, 4, 5, 6}
>>> # Xóa phần tử bằng discard()
    my_set.discard(4)
    print(my_set)
    # Output: {1, 3, 5, 6}
>>> # Xóa bằng remove()
    my_set.remove(6)
    print(my_set)
    # Output: {1, 3, 5}
```

Xóa phần tử khỏi set

```
>>> # Xóa phần tử không có
    # trong set bằng discard()
    my_set.discard(2)
    print(my_set)
    # Output: {1, 3, 5}
>>> # Xóa phần tử không có
    # trong set bằng remove()
    my_set.remove(2)
    # nhận được lỗi.
    # Output: KeyError: 2
```

Xóa phần tử khỏi set

Xóa phần tử bằng phương thức **pop()**, phần tử bị xóa ngẫu nhiên.

Xóa toàn bộ set thực hiện bằng lệnh **clear()**

```
>>> # Khởi tạo set_text
    set_text = set("LacHong")
    print(set_text)
    # Output: {'a', 'L', 'n', 'c', 'o', 'g', 'H'}
>>> # xóa phần tử bằng pop()
    print(set_text.pop())
    # Output: phần tử bị xóa ngẫu nhiên
```

Xóa phần tử khỏi set

```
>>> # xóa phần tử khác bằng pop()
    set_text.pop()
    print(set_text)
    # Output: phần tử bị xóa ngẫu nhiên

>>> # clear set_text
    set_text.clear()
    print(set_text)
    #Output: set()
```

Các toán tử Set

Phương thức và toán tử để thực hiện những phép toán tập hợp: hợp, giao, hiệu, bù.
Hợp của A và B là tập hợp các phần tử của A và B.
Hợp dùng toán tử | hoặc phương thức **union()**

```
>>> # Khởi tạo A và B
    A = {1, 2, 3, 4, 5}
    B = {4, 5, 6, 7, 8}
>>> # sử dụng toán tử |
    print(A | B)
    # Output: {1, 2, 3, 4, 5, 6, 7, 8}
>>> # sử dụng hàm union()
    print(A.union(B)) # hoặc print(B.union(A))
    # Output: {1, 2, 3, 4, 5, 6, 7, 8}
```

Các toán tử Set

Giao của A và B là tập hợp phần tử chung của A và B. Giao dùng toán tử & hoặc **hàm intersection()**

```
>>> # khởi tạo A và B
    A = {1, 2, 3, 4, 5}
    B = {4, 5, 6, 7, 8}
>>> # sử dụng &
    print(A & B)
    # Output: {4, 5}
>>> # sử dụng intersection()
    print(A.intersection(B))
    print(B.intersection(A))
    # Output: {4, 5}
```

Các toán tử Set

Hiệu của Set: (A-B) là phần tử chỉ có trong A, (B-A) là phần tử chỉ có trong B. Hiệu dùng toán tử - hoặc **hàm difference()**

```
>>> # Khởi tạo A và B
    A = {1, 2, 3, 4, 5}
    B = {4, 5, 6, 7, 8}
>>> # (A – B) sử dụng toán tử –
    print(A - B)
    # Output: {1, 2, 3}
>>> # (A – B) sử dụng hàm difference()
    print(A.difference(B))
    # Output: {1, 2, 3}
```

Các toán tử Set

```
>>> # (B – A)  sử dụng toán tử –
    print(B - A)
    # Output: {8, 6, 7}
>>> # (B – A) sử dụng difference()
    print(B.difference(A))
    # Output: {8, 6, 7}
```

Các toán tử Set

Bù của A và B là phần tử có trong A và B nhưng không phải phần tử chung của A và B. Bù dùng toán tử ^ hoặc **hàm symmetric_difference()**.

```
>>> # Khởi tạo A và B
    A = {1, 2, 3, 4, 5}
    B = {4, 5, 6, 7, 8}
>>> # Sử dụng toán tử ^
    print(A ^ B)
    # Output: {1, 2, 3, 6, 7, 8}
>>> # Sử dụng symmetric_difference() trên A
    print(A.symmetric_difference(B))
    # Output: {1, 2, 3, 6, 7, 8}
```

Các phương thức dùng trên set

Phương thức	Mô tả
add()	Thêm một phần tử vào set.
clear()	Xóa tất cả phần tử của set.
copy()	Trả về bản sao chép của set.
difference()	Trả về set mới chứa những phần tử khác nhau của 2 hay nhiều set.
difference_update()	Xóa tất cả các phần tử của set khác từ set này.
discard()	Xóa phần tử nếu nó có mặt trong set.
intersection()	Trả về set mới chứa phần tử chung của 2 set.
intersection_update()	Cập nhật set với phần tử chung của chính nó và set khác.
isdisjoint()	Trả về True nếu 2 set không có phần tử chung.

Các phương thức dùng trên set

issubset()	Trả về True nếu set khác chứa set này.
issuperset()	Trả về True nếu set này chưa set khác.
pop()	Xóa và trả về phần tử ngẫu nhiên, báo lỗi KeyError nếu set rỗng.
remove()	Xóa phần tử từ set. Nếu phần tử đó không có trong set sẽ báo lỗi KeyError.
symmetric_difference()	Trả về set mới chứa những phần tử không phải là phần tử chung của 2 set.
symmetric_difference_update()	Cập nhật set với những phần tử khác nhau của chính nó và set khác.
union()	Trả về set mới là hợp của 2 set.
update()	Cập nhật set với hợp của chính nó và set khác.

Kiểm tra phần tử trong set

Kiểm tra một đối tượng có trong set dùng từ khóa in

```
>>> # Khởi tạo my_set
    my_set = set("LacHong")
>>> # Kiểm tra H có trong my_set không
    print('H' in my_set)
    # Output: True
>>> # Kiểm tra xem h có trong my_set không
    print('h' in my_set)
    # Output: False
```

Vòng lặp for trên set

Dùng vòng lặp for để duyệt qua các phần tử của set.

```
>>> for x in set('LacHong):
        print(x)
    # kết quả:
    L
    a
    c
    H
    o
    n
    g
```

Hàm thường dùng trên set

Hàm thường dùng trên set: **all(), any(), enumerate(), len(), max(), min(), sorted(), sum()**.

Sử dụng giống như khi sử dụng trên list, tuple.

Frozenset trong Python

Frozenset có đặc điểm của set, các phần tử không thay đổi sau khi gán.
Set thay đổi được nhưng không thể băm (hash) được, nên không dùng set làm key cho dictionary. Frozenset có thể băm được nên được dùng như key cho dictionary.
Frozenset được tạo bằng hàm frozenset() có các phương thức như **copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() và union()**. Do phần tử không thay đổi nên phương thức add() và remove() không sử dụng được trên frozenset.

Dictionary

Dictionary là tập hợp các cặp key - value không có thứ tự. Được sử dụng với dữ liệu lớn.

Dictionary được dùng để trích xuất dữ liệu với khóa đã biết để lấy giá trị.

Tạo dictionary trong Python
Dictionary: các phần tử trong dấu ngoặc nhọn { }, mỗi phần tử là một cặp key:value. Key-value có kiểu dữ liệu bất kỳ.

Có thể tạo dictionary bằng **hàm dict()**.
Ví dụ:

Tạo dictionary

```
>>> dict1 = { } #dictionary rỗng
      #Tạo dictionary với các khóa nguyên
      dict2 = {1: ‘LacHong', 2: ‘Tinhoc'}
      #Tạo dictionary với khóa hỗn hợp
      dict3 = {'tên': ‘KTLT', 1: [1, 3, 5]}
      #Tạo dictionary bằng dict()
      dict4 = dict({1:‘tinhoc', 2:‘ngoaingu'})
      #Tạo dictionary từ chuỗi với mỗi mục là một cặp
      dict5 = dict([(1,‘KTLT'), (2,‘TOAN')])
```

Kiểm tra kiểu dữ liệu:

```
>>> type(dict2)
      <class 'dict'>
```

Truy cập phần tử của dictionary

Dictionary sử dụng key để truy cập các giá trị . Key sử dụng cặp dấu ngoặc vuông hoặc **hàm get()**.

```
 >>> # Khởi tạo dict2
      dict2 = {1: ‘Lac Hong’, ‘saudaihoc’: ‘Tin Hoc‘}
      print(type(dict2)) # In kiểu dữ liệu của dict2
      # Truy cập dữ liệu bằng khóa
      print("dict2[1] = ", dict2[1])
      print("dict2[saudaihoc] = ",dict2[‘saudaihoc'])
```

Kết quả:

```
>>> <class 'dict'>
      dict2[1] = LacHong
      dict2[saudaihoc] = Tin Hoc
```

Thay đổi, thêm phần tử cho dictionary

Dictionary có thể thay đổi, thêm phần tử hoặc thay đổi giá trị của các phần tử bằng toán tử gán.
Nếu key đã có, giá trị sẽ được cập nhật, nếu là một cặp key: value mới thì sẽ được phần tử mới.

```
>>> dict2 = {1: 'Lac Hong', 'saudaihoc': 'Tin Hoc'}
    # Cập nhật giá trị
    dict2['saudaihoc'] = 'Anh văn'
    print(dict2)
    # Output: {1: 'Lac Hong', 'saudaihoc': 'Anh văn'}
    # Thêm phần tử mới
    dict2[2] = 'Lập trình'
    print(dict2)
    # Output: {1: 'Lac Hong', 'saudaihoc': 'Lập trình'}
```

Xóa phần tử từ dictionary

Xóa phần tử của dictionary bằng **pop()**, trả về giá trị.

Popitem() xóa phần tử và trả về dạng (key, value).

Xóa các phần tử trong dictionary bằng **clear()**.

Del cũng dùng để xóa phần tử hay toàn bộ dictionary

```
>>> # tạo dictionary
    b_phuong = {1:1, 2:4, 3:9, 4:16, 5:25}
    # xóa phần tử số 4
    print(b_phuong.pop(4))
    # Output: 16
    print(b_phuong)
    # Output: {1: 1, 2: 4, 3: 9, 5: 25}
```

Xóa phần tử từ dictionary

```
>>> # xóa phần tử dùng Del
    del b_phuong[2]
    print(b_phuong)
    # output: {1: 1, 3: 9, 5: 25}
>>> # xóa phần tử bằng popitem
    print(b_phuong.popitem())
    # Output: (5, 25)
    print(b_phuong)
    # Output: {1: 1, 3: 9}
>>> # xóa tất cả phần tử
    b_phuong.clear()
```

Các phương thức và hàm cho dictionary

Method	Mô tả
clear()	Xóa tất cả phần tử của dictionary.
copy()	Trả về một bản sao shollow copy của dictionary.
fromkeys(seq[,v])	Trả về dictionary mới với key từ seq và value bằng v (default là None).
get(key[,d])	Trả về giá trị của key, nếu key không tồn tại, trả về d. (default là None).
items()	Trả lại kiểu xem mới của các phần tử trong dictionary (key, value).
keys()	Trả về kiểu xem mới của các key trong dictionary.
pop(key[,d])	Xóa phần tử bằng key và trả về giá trị hoặc d nếu key không tìm thấy. Nếu d không được cấp, key không tồn tại thì sẽ tạo lỗi KeyError .

Các phương thức và hàm cho dictionary

popitem()	Xóa và trả về phần tử bất kỳ ở dạng (key, value). Tạo lỗi KeyError nếu dictionary rỗng.
setdefault(key,[,d])	Nếy key tồn tại trả về value của nó, nếu không thêm key với value là d và trả về d (default là None).
update([other])	Cập nhật dictionary với cặp key/value từ other, ghi đè lên các key đã có.
values()	Trả về kiểu view mới của value trong dictionary

Các **hàm all(), any(), len(), cmp(), sorted(),...** cũng được sử dụng với dictionary.

Dictionary comprehension

Dictionary comprehension: tạo dictionary từ vòng lặp for. Câu lệnh gồm cặp (key:value) cùng câu lệnh for đặt trong dấu { }.

```
>>> lap_phuong = {x: x*x*x for x in range(6)}
    print(lap_phuong)
    # Output: {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

Dictionary comprehension

Chương trình trên tương đương với

```
>>> lap_phuong = { }
    for x in range(6):
        lap_phuong[x] = x*x*x
>>> print(lap_phuong)
```

Lệnh if trong dictionary comprehension. Lệnh if lọc những phần tử trong dictionary hiện có để tạo thành dictionary mới.

```
>>> lap_phuong_chan = {x: x*x*x for x in range (10) if x%2==0}
    print(lap_phuong_chan)
    # output: {0: 0, 2: 8, 4: 64, 6: 216, 8: 512}
```

Kiểm tra phần tử và dùng vòng for

Kiểm tra key có trong dictionary chưa dùng in.

```
>>> lap_phuong = {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
    print (2 in lap_phuong)
    # Output: True
    print (5 not in lap_phuong)
    # Output: False
```

Vòng lặp for duyệt key trong dictionary.

```
>>> lap_phuong = {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
    for i in lap_phuong
        print(lap_phuong[i])
    # các giá trị từng key sẽ được in ra màn hình theo thứ tự.
```

Chuyển đổi kiểu dữ liệu

Chuyển đổi kiểu dữ liệu dùng hàm **int()**,**float()**,**str()**,...

Ví dụ:

```
>>> float(11)
      11.0
>>> int(18.6)
      18
>>> set([2,4,6])
      {2,4,6}
>>> tuple({3,5,7})
       (3,5,7)
>>> list('lachong')
      ['l', 'a', 'c', 'h', 'o', 'n', 'g']
```

Chuyển đổi kiểu dữ liệu

Chuyển đổi sang dictionary, mỗi phần tử phải là một cặp key:value.

```
>>> dict([[2,4],[1,3]])
      {2: 4, 1: 3}
>>> dict([(3,9),(4,16)])
      {3: 9, 4: 16}
```

Đợi người dùng hành động

Dòng lệnh sau sẽ hiển thị dấu nhắc, câu lệnh yêu cầu "Nhấn phím Enter để thoát!" và đợi hành động của người dùng

```
input("\n\n Nhấn phím Enter để thoát!")
```

"\n\n" được dùng để tạo ra hai dòng mới trước khi hiển thị dòng thực tế. Khi người dùng nhấn Enter chương trình sẽ kết thúc. Đây là thủ thuật giữ cửa sổ chương trình hiển thị đến khi người dùng hoàn tất thao tác với ứng dụng.

BÀI TẬP

1. Nhập số nguyên n, hãy viết chương trình tạo dictionary chứa (i, i*i) phần tử với i từ 1 đến n (bao gồm cả 1 và n), in ra dictionary.
2. Viết chương trình nhập một chuỗi số, phân cách bằng dấu phẩy, in ra một list và một tuple chứa các số.
3. Viết chương trình và in chuỗi số giá trị theo công thức: Q = √([(2 * C * D)/H]) (ý nghĩa: Q bằng căn bậc hai của [(2 nhân C nhân D) chia H]. Cho C = 50, H = 30. D là chuỗi giá trị được nhập từ bàn phím phân cách bằng dấu phẩy.

Ví dụ: chuỗi D nhập vào là 100,150,180 thì kết quả là 18,22,24

BÀI TẬP

4. Viết chương trình tạo mảng 2 chiều với m, n nhập từ bàn phím. Giá trị phần tử hàng i, cột j của mảng là i*j

5. Viết chương trình nhập chuỗi từ từ bàn phím, phân cách bởi dấu phẩy và in những từ đó thành chuỗi theo thứ tự bảng chữ cái, phân cách nhau bằng dấu phẩy.

6. Viết chương trình nhập vào một chuỗi các từ phân cách bởi khoảng trắng, loại bỏ các từ trùng lặp, sắp xếp theo thứ tự bảng chữ cái, rồi in chúng.

7. Viết chương trình tính giá trị của a+aa+aaa+aaaa với a là số được nhập từ bàn phím.

8. Viết chương trình in danh sách các số lẻ từ danh sách được người dùng nhập vào.

BÀI TẬP

19. Với tuple (1,2,3,4,5,6,7,8,9,10) cho trước, viết chương trình in một nửa giá trị đầu tiên trong 1 dòng và 1 nửa còn lại trong 1 dòng

10. Viết chương trình để tạo tuple khác, chứa các giá trị là số chẵn trong tuple (1,2,3,4,5,6,7,8,9,10) cho trước

11. In chuỗi Unicode "Hello world"

12. Viết chương trình để đọc chuỗi ASCII và chuyển đổi nó sang chuỗi Unicode được mã hóa bằng UTF-8

13. Viết chương trình nhập biểu thức toán học cơ bản và in kết quả ra ngoài màn hình (dùng **hàm eval**)

BÀI TẬP

14. Tạo một số thập phân ngẫu nhiên, có giá trị nằm trong khoảng từ 10 đến 100 bằng cách sử dụng module math của Python (dùng **random.random()**)

15. Tạo một số thập phân ngẫu nhiên, có giá trị nằm trong khoảng 5 đến 95, sử dụng module math của Python.

16. Viết chương trình xuất ra một số chẵn ngẫu nhiên trong khoảng 0 đến 10 (bao gồm cả 0 và 10), sử dụng module random và list comprehension.

17. viết chương trình để xuất một số ngẫu nhiên, chia hết cho 5 và 7, từ 0 đến 200 (gồm cả 0 và 200), sử dụng module random và list comprehension

BÀI TẬP

18. viết chương trình để tạo một list với 5 số ngẫu nhiên từ 100 đến 200 (**random.sample()**)

19. Viết chương trình tạo ngẫu nhiên list gồm 5 số chẵn nằm trong đoạn [100;200]

20. Viết chương trình để tạo ngẫu nhiên một list gồm 5 số, chia hết cho 5 và 7, nằm trong đoạn [1;1000].

21. Viết chương trình để in một số nguyên ngẫu nhiên từ 7 đến 15 (**random.randrange()**)

22. Viết chương trình để nén và giải nén string ""hello world!hello world!hello world!hello world!" (**modul zlib zlib.compress() zlib.decompress()**)

BÀI TẬP

23. Bạn hãy viết một chương trình để in thời gian thực thi (running time of execution) phép tính "1+1" 100 lần. (**modul timeit timeit.Timer()**)

24. Viết chương trình để trộn và in list [3,6,7,8] (**random.shuffle()**)

25. Viết chương trình in list sau khi xóa các số chẵn trong [5,6,77,45,22,12,24]

26. Sử dụng list comprehension để viết chương trình in list sau khi đã loại bỏ các số chia hết cho 5 và 7 trong [12,24,35,70,88,120,155]

BÀI TẬP

27. Viết chương trình in list sau khi đã xóa số thứ 0, thứ 2, thứ 4, thứ 6 trong [12,24,35,70,88,120,155]. (**hàm enumerate()**)

28. Viết chương trình tạo mảng 3D 3*5*8 có mỗi phần tử là 0.

29. Viết chương trình in list sau khi đã xóa số ở vị trí thứ 0, thứ 4, thứ 5 trong [12,24,35,70,88,120,155].

30. Viết chương trình in list sau khi đã xóa giá trị 24 trong [12,24,35,24,88,120,155]

31. Với 2 list cho trước: [12,3,6,78,35,55,120] và [12,24,35,24,88,120,155], viết chương trình để tạo list có phần tử là giao của 2 list đã cho

BÀI TẬP

32. Viết chương trình in list từ list [12, 12, 15, 24, 35, 35, 24, 88, 120, 155, 88,120,155], sau khi đã xóa hết các giá trị trùng nhau

33. Viết chương trình đếm và in số ký tự của chuỗi do người dùng nhập vào (**dict.get()**)

34. Viết chương trình nhập chuỗi và in nó theo thứ tự ngược lại (**[::-1]**)

35. Viết chương trình nhập chuỗi và in các ký tự có chỉ số chẵn (**[::2]**)

36. Viết chương trình in tất cả các hoán vị của [1,2,3] (**modul itertools itertools.permutations()**)