THỰC HÀNH BUỔI 1

A - Giới thiệu về Python

1. Từ khóa và định danh trong Python

1.1. Từ khóa

Chúng ta không thể sử dụng những keyword này để đặt tên cho biến, hay sử dụng chúng như các hằng, tên định danh. Trong Python, ngoại trừ True, False và None được viết hoa ra thì các keyword khác đều được viết dưới dạng chữ thường, đây là điều bắt buộc. Dưới đây là danh sách các từ khóa trong ngôn ngữ lập trình Python:

- 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

1.2. Định danh trong Python

Định danh là tên được đặt cho các thực thể như class, function, biến,... trong Python. Nó giúp phân biệt thực thể này với thực thể khác.

Quy tắc viết định danh và vài tips cho định danh trong Python:

- Định danh có thể là sự kết hợp giữa các chữ cái viết thường (từ a đến z) hoặc viết hoa (A đến Z) hoặc các số (từ 0 đến 9) hoặc dấu gạch dưới (_). Định danh hợp lệ sẽ giống như thế này: bien_1, tinh_tong_0_9, firstClass.
- Định danh không thể bắt đầu bằng một chữ số, ví dụ 1bien là không hợp lệ, nhưng bien1 thì đúng.
- Định danh phải khác các keyword. Bạn thử nhập and = 1 rồi chạy sẽ xuất hiện thông báo lỗi "SyntaxError: invalid syntax".
- Python không hỗ trợ các ký tự đặc biệt như !, @, #, $, %,... trong định danh. Nếu cố tình gán các ký tự này trong định danh bạn cũng sẽ nhận được thông báo lỗi "SyntaxError: invalid syntax" hoặc "NameError:..."
- Định danh có thể dài bao nhiêu tùy ý.
- Tên lớp thường bắt đầu với một chữ cái hoa. Tất cả các định danh khác bắt đầu với chữ cái thường.
- Định danh bắt đầu với một dấu dạch dưới _ thì là định danh private.
- Định danh bắt đầu với 2 dấu gạch dưới __ thì mức độ private cao hơn.
- Nếu định danh bắt đầu và kết thúc bằng 2 dấu gạch dưới (__init__ chẳng hạn) thì định danh đó là tên đặc biệt được ngôn ngữ định nghĩa.
- Nên đặt tên định danh có nghĩa. Dù c = 10 vẫn đúng, nhưng viết count = 10 sẽ rõ nghĩa hơn và dễ hiểu nó là gì hơn dù bạn đã xem code một đoạn dài sau đó.
- Python là ngôn ngữ lập trình phân biệt chữ hoa, chữ thường, nghĩa là bien và Bien là không giống nhau.
- Khi đặt định danh nhiều từ bạn có thể dùng dấu gạch dưới giữa các từ, day_la_mot_bien_dai, kiểu vậy.
- Bạn có thể viết theo phong cách HoaRoiThuong như thế này để phân biệt các từ trong trường hợp biến dài.

2. Cách viết lệnh, canh lề và chú thích trong Python

Ghi chú trong Python

Ghi chú dòng đơn

```
#This is a comment
#written in
#more than just one line
```

Ghi chú nhiều dòng

```
"""
This is a comment
written in
more than just one line
"""
```

Câu lệnh trong Python

Những hướng dẫn mà trình thông dịch Python có thể thực hiện được gọi là các câu lệnh. Ví dụ, count = 0 là một lệnh gán. Lệnh if, lệnh for, lệnh while,... là những loại lệnh khác mà chúng ta sẽ được tìm hiểu chi tiết trong các bài sau.

Viết câu lệnh trên nhiều dòng

Trong Python, một câu lệnh được kết thúc bằng ký tự dòng mới, nghĩa là một câu lệnh sẽ kết thúc khi bạn xuống dòng. Nhưng chúng ta có thể mở rộng câu lệnh trên nhiều dòng với ký tự tiếp tục dòng lệnh (\). Ví dụ:

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

Đây là trường hợp viết tiếp câu lệnh rất minh bạch. Trong Python, viết tiếp câu lệnh thường sử dụng dấu ngoặc đơn (), ngoặc vuông [] hoặc ngoặc nhọn {}. Ví dụ trên có thể viết lại như sau:

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

Dấu () ở đây ngầm ý tiếp tục dòng lệnh, 2 dấu ngoặc còn lại cũng có chức năng tương tự. Ví dụ:

```
mau_sac = {"vàng",
 "xanh",
        "cam"}
```

Bạn cũng có thể đặt nhiều lệnh trên cùng một dòng, phân cách nhau bởi dấu chấm phẩy ; như thế này:

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

Canh lề trong Python

Trong Python thì khác, những khối lệnh sẽ được nhận biết thông qua canh lề. Đó là lý do vì sao canh lề trong Python vô cùng quan trọng, nếu bạn canh lề nhầm là sẽ bị báo lỗi ngay.

Một khối code (thường là khối lệnh của hàm, vòng lặp,...) bắt đầu với canh lề và kết thúc với dòng đầu tiên không canh lề. Canh lề bao nhiêu là tùy thuộc ở bạn nhưng chúng phải nhất quán trong suốt khối code đó, tức là các lệnh trong cùng một khối thì phải có độ canh lề bằng nhau.

Thông thường, sẽ dùng 4 lần phím cách để canh lề, chúng được các lập trình viên yêu thích hơn là phím tab, giống như ví dụ dưới đây:

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

Khối lệnh của for gồm có print(i) và if, được viết canh lề bằng nhau, break là lệnh trong if, nên lại được canh lề thêm một đoạn nữa. Bạn chưa cần hiểu cụ thể đoạn code trên nói cái gì, chỉ cần ghi nhớ trong đầu về cách canh lề của Python thôi, dần dần chúng ta sẽ tìm hiểu chi tiết, từng thứ một.
Nếu lệnh trên được viết thành:

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

Bạn sẽ nhận được thông báo lỗi ngay lập tức, và lỗi sẽ hiện trước lệnh print(i).
Nhờ canh đầu dòng mà code trong Python trông gọn gàng và rõ ràng hơn. Bạn có thể bỏ qua canh đầu dòng với những câu lệnh viết trên nhiều dòng, nhưng những nhà lập trình nhiều kinh nghiệm khuyên rằng: ***Hãy luôn canh lề, điều đó làm cho code dễ đọc hơn.***
Ví dụ:

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

Và:

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

Chú thích, bình luận trong Python

Trong Python, chúng ta sử dụng ký tự # để bắt đầu một chú thích. Chú thích bắt đầu sau dấu # cho đến khi bắt đầu một dòng mới. Khi thông dịch, Python sẽ bỏ qua những chú thích này.

```
#Đây là chú thích
```

Nếu bạn muốn viết chú thích trên nhiều dòng, rất đơn giản, chỉ cần thêm # vào trước mỗi dòng, như thế này:

```
#Đây là chú thích
#trên nhiều dòng #In dòng chữ HelloWorld
#trong Python  print(Hello World!')
```

Có cách khác để viết chú thích là sử dụng 3 dấu nháy đơn ' ' ' hoặc nháy kép " " ". Những dấu nháy này thường được sử dụng cho các chuỗi nhiều dòng. Nhưng chúng cũng có thể được sử dụng để viết chú thích trên nhiều dòng.

3. Kiểu dữ liệu trong Python: chuỗi, số, list, tuple, set và dictionary

a. Biến trong Python

Biến là một vị trí trong bộ nhớ được sử dụng để lưu trữ dữ liệu (giá trị). Biến được đặt tên duy nhất để phân biệt giữa các vị trí bộ nhớ khác nhau. Các quy tắc để viết tên một biến giống như quy tắc viết các định danh trong Python.
Trong Python, chúng ta không cần khai báo biến trước khi sử dụng, chỉ cần gán cho biến một giá trị và nó sẽ tồn tại. Cũng không cần phải khai báo kiểu biến, kiểu biến sẽ được nhận tự động dựa vào giá trị mà bạn đã gán cho biến.

Gán giá trị cho biến:

Để gán giá trị cho biến ta sử dụng toán tử =. Bất kỳ loại giá trị nào cũng có thể gán cho biến hợp lệ.
Ví dụ:

```
hoa = "Hồng"
 la = 3
 canh = 5.5
```

Phía trên là 3 câu lệnh gán, "Hồng" là một chuỗi ký tự, được gán cho biến hoa, 3 là số nguyên và được gán cho la, 5.5 là số thập phân và gán cho canh.

Gán nhiều giá trị:

Trong Python chúng ta có thể thực hiện gán nhiều giá trị trong một lệnh như sau:

```
hoa, la, canh = "Hồng", 3, 5.5
```

Nếu muốn gán giá trị giống nhau cho nhiều biến thì có thể viết lệnh như sau:

```
hoa, la, canh = 3
```

Lệnh trên sẽ gán giá trị 3 cho cả 3 biến là hoa, la và canh.

b. Kiểu dữ liệu số trong Python

Python hỗ trợ số nguyên, số thập phân và số phức, chúng lần lượt được định nghĩa là các lớp int, float, complex trong Python. Số nguyên và số thập phân được phân biệt bằng sự có mặt hoặc vắng mặt của dấu thập phân. Ví dụ: 5 là số nguyên, 5.0 là số thập phân. Python cũng hỗ trợ số phức và sử dụng hậu tố j hoặc J để chỉ phần ảo. Ví dụ: 3+5j. Ngoài int và float, Python hỗ trợ thêm 2 loại số nữa là Decimal và Fraction.

Dùng hàm type() để kiểm tra xem biến hoặc giá trị thuộc lớp số nào và hàm isinstance() để kiểm tra xem chúng có thuộc về một class cụ thể nào không.

```
a = 9

# Output: <class 'int'>
print(type(a))

# Output: <class 'float'>
print(type(5.0))

# Output: (10+2j)
b = 8 + 2j
print(b + 2)

# Kiểm tra xem b có phải là số phức không
# Output: True
print(isinstance(b, complex))
```

Chuyển đổi giữa các kiểu số trong Python

Sử dụng các hàm Python tích hợp sẵn như int(), float() và complex() để chuyển đổi giữa các kiểu số một cách rõ ràng. Những hàm này thậm chí có thể chuyển đổi từ các chuỗi.

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

Khi nào nên sử dụng Decimal thay cho float?

Ta thường 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 giống như đã học ở trường.

Phân số trong Python

Python cung cấp các phép toán liên quan đến phân số thông qua mô-đun fractions. Một phân số có tử số và mẫu số, cả hai đều là số nguyên. Ta có thể tạo đối tượng phân số (Fraction) theo nhiều cách khác nhau:

```
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, mẫu số
 print(fractions.Fraction(2,5))
 # Output: 2/5
```

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

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

Toán học trong Python

Python cung cấp các mô-đun math và random để giải quyết các vấn đề toán học khác như lượng giác, logarit, xác suất và thống kê, v.v...

```
from fractions import Fraction as F
 import math

 # Output: 3.141592653589793
 print(math.pi)
 # Output: -1.0
 print(math.cos(math.pi))
 # Output: 22026.465794806718
 print(math.exp(10))
 # Output: 2.0
 print(math.log2(4))
 # Output: 1.1752011936438014
 print(math.sinh(1))
 # Output: 40320
 print(math.factorial(8))
```

c. Chuỗi (string)

String trong Python là một dãy các ký tự. Máy tính không xử lý các ký tự, chúng chỉ làm việc với số nhị phân. Dù bạn có thể nhìn thấy các ký tự trên màn hình, nhưng chúng được lưu trữ và xử lý nội bộ dưới dạng kết hợp của số 0 và 1. Việc chuyển đổi ký tự thành số được gọi là mã hóa

và quá trình ngược lại được gọi là giải mã. ASCII và Unicode là 2 trong số những mã hóa phổ biến thường được sử dụng.
Trong Python, string là một dãy các ký tự Unicode. Unicode bao gồm mọi ký tự trong tất cả các ngôn ngữ và mang lại tính đồng nhất trong mã hóa.

Cách tạo string trong Python

Bên cạnh số, Python cũng có thể thao tác với chuỗi, được biểu diễn bằng nhiều cách. Chúng có thể được để trong dấu nháy đơn ('...') hoặc kép ("...") với cùng một kết quả. \ được sử dụng để "trốn (escape)" 2 dấu nháy này.

```
>>> 'spam eggs' # dấu nháy đơn
'spam eggs'
>>> 'doesn\'t' # sử dụng \' để viết dấu nháy đơn...
"doesn't"
>>> "doesn't" # ...hoặc 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.'
```

Trong trình thông dịch tương tác, chuỗi kết quả bao gồm phần trong dấu ngoặc kép và các ký tự đặc biệt "trốn" được nhờ sử dụng \. Dù đầu ra trông có vẻ hơi khác với đầu vào (dấu nháy kèm theo có thể thay đổi) nhưng hai chuỗi này là tương đương. Chuỗi được viết trong dấu ngoặc kép khi chuỗi chứa dấu nháy đơn và không có dấu nháy kép), ngược lại nó sẽ được viết trong dấu nháy đơn. Hàm print() tạo chuỗi đầu ra dễ đọc hơn, bằng cách bỏ qua dấu nháy kèm theo và in các ký tự đặc biệt, đã "trốn" được dấu nháy:

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

Nếu không muốn các ký tự được thêm vào bởi \ được trình thông dịch hiểu là ký tự đặc biệt thì sử dụng chuỗi raw bằng cách thêm r vào trước dấu nháy đầu tiên:

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

Chuỗi ký tự dạng chuỗi có thể viết trên nhiều dòng bằng cách sử dụng 3 dấu nháy: """..."""" hoặc '''...'''. Kết thúc dòng tự động bao gồm trong chuỗi, nhưng có thể ngăn chặn điều này bằng cách thêm \ vào cuối dòng. Ví dụ:

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

Đây là danh sách tất cả các ký tự thoát (escape sequence) được Python hỗ trợ:

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

Ví dụ: Hãy chạy từng lệnh riêng lẻ ngay trong trình biên dịch để thấy kết quả bạn nhé.

```
>>> print("C:\\Python32\\abc.com")
C:\Python32\abc.com
>>> print("In dòng này\nthành 2 dòng")
In dòng này
thành 2 dòng
>>> print("In giá trị \x48\x45\x58")
In giá trị HEX
>>>
```

Cách truy cập vào phần tử của chuỗi

Các chuỗi có thể được lập chỉ mục với ký tự đầu tiên được đánh số 0. Không có kiểu ký tự riêng biệt, mỗi ký tự đơn giản là một con số:

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

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

```
>>> word[-1] # last character
'n'
>>> word[-2] # second-last character
'o'
>>> word[-6]
'P'
```

Phương thức format() để định dạng chuỗi

Phương thức format() rất linh hoạt và đầy sức mạnh khi dùng để định dạng chuỗi. Định dạng chuỗi chứa dấu {} làm trình giữ chỗ hoặc trường thay thế để nhận giá trị thay thế. Bạn cũng có thể sử dụng đối số vị trí hoặc từ khóa để chỉ định thứ tự.

```
# default(implicit) order
thu_tu_mac_dinh = "{}, {} và {}".format('Quản','Trị','Mạng')
print('\n--- Thứ tự mặc định ---')
print(thu_tu_mac_dinh)

# sử dụng đối số vị trí để sắp xếp thứ tự
vi_tri_thu_tu= "{1}, {0} và {2}".format('Quản','Trị','Mạng')
print('\n--- Thứ tự theo vị trí ---')
print(vi_tri_thu_tu)

# sử dụng từ khóa để sắp xếp thứ tự
tu_khoa_thu_tu = "{s}, {b} và {j}".format(j='Quản',b='Trị',s='Mạng')
print('\n--- Thứ tự theo từ khóa ---')
print(tu_khoa_thu_tu)
```

Ta có kết quả khi chạy code trên như sau:

```
--- Thứ tự mặc định ---
Quản, Trị và Mạng

--- Thứ tự theo vị trí ---
Trị, Quản và Mạng

--- Thứ tự theo từ khóa ---
Mạng, Trị và Quản
```

Phương thức thường được sử dụng trong string

Có rất nhiều phương thức được tích hợp sẵn trong Python để làm việc với string. Ngoài format() được đề cập bên trên còn có lower(), upper(), join(), split(), find(), replace(), v.v....

```
>>> "Toi Di Hoc".lower()
'toi di hoc'
>>> " Toi Di Hoc ".upper()
'TOI DI HOC'
>>> "Toi Di Hoc".split()
['Toi', 'Di', 'Hoc']
>>> ' '.join(['Toi', 'Di', 'Hoc'])
'Toi Di Hoc'
>>> 'Toi di hoc'.find('Toi')
0
>>> 'Toi di hoc'.replace('di','nghi')
'Toi nghi hoc'
```

d. Danh sách (list)

Python cung cấp một loạt các dữ liệu phức hợp, thường được gọi là các chuỗi (sequence), sử dụng để nhóm các giá trị khác nhau. Đa năng nhất là danh sách (list).

Cách tạo list trong Python

Trong Python, list được biểu diễn bằng dãy các giá trị, được phân tách nhau bằng dấu phẩy, nằm trong dấu []. Các danh sách có thể chứa nhiều mục với kiểu khác nhau, nhưng thông thường là các mục có cùng kiểu.

```
>>> squares = [1, 4, 9, 16, 25]
 >>> squares
 [1, 4, 9, 16, 25]
```

List không giới hạn số lượng mục, bạn có thể có nhiều kiểu dữ liệu khác nhau trong cùng một list, như chuỗi, số nguyên, số thập phân,...

```
list1 = [] # list rỗng
 list2 = [1, 2, 3] # list số nguyên
 list3 = [1, "Hello", 3.4] # list với kiểu dữ liệu hỗn hợp
```

Bạn cũng có thể tạo các list lồng nhau (danh sách chứa 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:

```
list4 = [mouse", [8, 4, 6], ['a']]
```

Truy cập vào phần tử của list

Có nhiều cách khác nhau để truy cập vào phần tử của một danh sách:

Index (chỉ mục) của list:

Sử dụng toán tử index [] để truy cập vào một phần tử của list. Index bắt đầu từ 0, nên một list có 5 phần tử sẽ có index từ 0 đến 4. Truy cập vào phần tử có index khác index của list sẽ làm phát sinh lỗi IndexError. Index phải là một số nguyên, không thể sử dụng float, hay kiểu dữ liệu khác, sẽ tạo lỗi TypeError.

```
qtm_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m']
 # Output: q
 print(qtm_list[0])

 # Output: a
 print(qtm_list[2])

 # Output: t
 print(qtm_list[4])

 # List lồng nhau
 ln_list = ["Happy", [1,3,5,9]]

 # Index lồng nhau

 # Output: a
 print(ln_list[0][1])

 # Output: 9
 print(ln_list[1][3])
```

Python cho phép lập chỉ mục âm cho các chuỗi. Index -1 là phần tử cuối cùng, -2 là phần tử thứ 2 từ cuối cùng lên. Nói đơn giản là index âm dùng khi bạn đếm phần tử của chuỗi ngược từ cuối lên đầu.

```
qtm_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m']

 # Output: m
print(qtm_list[-1])
# Output: i
print(qtm_list[-9])
```

Phương thức list trong Python

Những phương thức có sẵn cho list trong Python gồm:

- **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 và 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.

```
my_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m']
 my_list.remove('.')

 # Output: ['q', 'u', 'a', 'n', 't', 'r', 'i', 'm', 'a', 'n', 'g', 'c', 'o', 'm']
 print(my_list)

 # Output: n
 print(my_list.pop(3))

 # Output: ['q', 'u', 'a', 't', 'r', 'i', 'm', 'a', 'n', 'g', 'c', 'o', 'm']
 print(my_list)

 # Output: m
 print(my_list.pop())

 # Output: ['q', 'u', 'a', 't', 'r', 'i', 'm', 'a', 'n', 'g', 'c', 'o']
 print(my_list)

 my_list.clear()

 # Output: [] (list rỗng)
 print(my_list)
```

Các hàm Python tích hợp với list

Các hàm Python tích hợp sẵn như all(), any(), enumerate(), len(), max(), min(), list(), sorted(),... thường được sử dụng với list để thực hiện những nhiệm vụ khác nhau.

- **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 dưới dạng tuple.
- **len()**: Trả về độ dài (số lượng phần tử) của list.
- **list()**: Chuyển đổi một đối tượng có thể lặp (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.

4. Chương trình Python đầu tiên

```
print("Hello, World!")
# Them hai so va in tong
num1 = 6
num2 = 9
sum = num1+num2
print(sum)
```

5. Mảng trong Python

```
import array as arr
a = arr.array('d',[1.1, 3.5, 4.5])
print(a)
```

Code trên tạo mảng có kiểu float. Chữ 'd' là mã kiểu, quyết định kiểu của mảng trong quá trình tạo. Dưới đây là những mã kiểu thường dùng:

Mã kiểu	C Type	Python Type	Kích thước tối thiểu tính theo byte
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

Làm sao để truy cập vào các phần tử của mảng?

Chúng ta sử dụng index để truy cập đến các phần tử của mảng. Index cũng bắt đầu từ 0, tương tự như trong list Python.

```
import array as arr a = arr.array('i', [2, 4, 6, 8])
print("Phần tử đầu tiên:", a[0])
print("Phần tử thứ 2:", a[1])
print("Phần tử cuối cùng:", a[-1])
```

Chạy chương trình trên ta được:

```
Phần tử đầu tiên: 2
Phần tử thứ 2: 4
Phần tử cuối cùng: 8
```

Bạn có thể truy cập vào một dải phần tử trong mảng, sử dụng toán tử cắt lát **:**.

```
import array as arr
numbers_list = [5, 85, 65, 15, 95, 52, 36, 25]
numbers_array = arr.array('i', numbers_list)
print(numbers_array[2:5])
# Phần tử thứ 3 đến 5
print(numbers_array[:-5])
# Phần tử đầu tiên đến 4
print(numbers_array[5:])
# Phần tử thứ 6 đến hết
print(numbers_array[:])
# Phần tử đầu tiên đến cuối cùng
```

Khi bạn chạy code trên sẽ nhận được output là:

```
array('i', [65, 15, 95])
array('i', [5, 85, 65])
array('i', [52, 36, 25])
array('i', [5, 85, 65, 15, 95, 52, 36, 25])
```

Thay đổi, thêm phần tử trong mảng Python

Mảng có thể thay đổi, các phần tử của nó có thể thay đổi theo cách tương tự như list.

```
import array as arr
numbers = arr.array('i', [1, 1, 2, 5, 7, 9])
# thay đổi phần tử đầu tiên
numbers[0] = 0
print(numbers)
# Output: array('i', [0, 1, 2, 5, 7, 9])
# thay phần tử thứ 3 đến thứ 5
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)
# Output: array('i', [0, 1, 4, 6, 8, 9])
```

Bạn có thêm một mục vào list sử dụng append() hoặc thêm vài mục sử dụng extend():

```
import array as arr
numbers = arr.array('i', [3, 5, 7])
numbers.append(4)
print(numbers)
# Output: array('i', [3, 5, 7, 4])
# extend() nối vào cuối mảng
numbers.extend([5, 6, 7])
print(numbers)
# Output: array('i', [3, 5, 7, 4, 5, 6, 7])
```

2 mảng cũng có thể nối lại thành một nhờ toán tử +:

```
import array as arr
mang_le = arr.array('i', [3, 5, 7])
mang_chan = arr.array('i', [2, 6, 8])
numbers = arr.array('i')
# tạo mảng trống numbers = mang_le + mang_chan
print(numbers)
# Output: array('i', [3, 5, 7, 2, 6, 8])
```

Xóa phần tử của mảng trong Python

Để xóa một hoặc nhiều phần tử của mảng ta sử dụng lệnh del.

```
import array as arr
number = arr.array('i', [1, 3, 3, 5, 7])
del number[2]
# xóa phần tử thứ 3 print(number)
# Output: array('i', [1, 3, 5, 7]) del number
# xóa toàn bộ mảng
print(number)
# Error: array 'number' is not defined
```

Có thể sử dụng remove() để xóa mục đã cho hoặc pop() để xóa mục với index cho trước:

```
import array as arr
numbers = arr.array('i', [1, 1, 3, 5, 9])
numbers.remove(1)
print(numbers)
# Output: array('i', [1, 3, 5, 9])
print(numbers.pop(2))
# Output: 12
print(numbers)
# Output: array('i', [1, 3, 9])
```

B – Các lệnh trong Python

1. Lệnh if, if...else, if...elif...else trong Python

a. Cấu trúc lệnh if trong Python

```
if điều kiện

   khối lệnh
```

Ví dụ 1:

```
# Nếu là số dương ta sẽ in một thông điệp thích hợp
num = 3
if num > 0:
 print(num, "là số dương.")
print("Thông điệp này luôn được in.")

num = -1
if num > 0:
 print(num, "là số dương.")
print("Thông điệp này cũng luôn được in.")
```

Kết quả đầu ra của chương trình trên:

```
3 là số dương.
Thông điệp này luôn được in.
Thông điệp này cũng luôn được in.
```

b. Lệnh if...else

Cấu trúc lệnh if...else

```
if điều kiện:
    Khối lệnh của if
else:
    Khối lệnh của else
```

Ví dụ 2:

```
# Kiem tra xem so am hay duong
# Va hien thi thong bao phu hop

num = 3

if num >= 0:
   print("So duong hoac bang 0")
else:
   print("So am")

num1=-1

if num1 >= 0:
   print("So duong hoac bang 0")
else:
   print("So am")
```

c. Cấu trúc lệnh if...elif...else

```
if điều kiện:
 Khối lệnh của if
elif test expression:
 Khối lệnh của elif
else:
 Khối lệnh của else
```

Ví dụ 3:

```
x = int(input("Nhap mot so: "))
if x < 0:
 x = 0
 print('So am')
elif x == 0:
 print('So 0')
elif x == 1:
 print('So 1')
else:
 print('So duong')
```

d. Lệnh if lồng nhau trong Python

Ví dụ 4:

```
# Trong code này, nhập vào một số
# Kiểm tra xem số âm hay dương
# hay bằng không và hiển thị
# thông báo thích hợp
# Sử dụng hàm if lồng nhau

num = float(input("Nhập một số: "))
if num >= 0:
   if num == 0:
       print("Số Không")
   else:
       print("Số dương")
else:
   print("Số âm")
```

2. Vòng lặp for trong Python

```
for bien_lap in chuoi_lap:
       Khối lệnh của for
```

Ví dụ 1:

```
#Lặp chữ cái trong toidihoc
for chu in 'toidihoc':
 print('Chữ cái hiện tại:', chu)
#Lặp từ trong chuỗi
chuoi = ['bố','mẹ']
for tu in chuoi:
    print('Anh yêu', tu)
```

Ta có kết quả đầu ra như sau:

```
Chữ cái hiện tại: t
Chữ cái hiện tại: o
Chữ cái hiện tại: i
Chữ cái hiện tại: d
Chữ cái hiện tại: i
Chữ cái hiện tại: h
Chữ cái hiện tại: o
Chữ cái hiện tại: c
Anh yêu bố
Anh yêu mẹ
```

Ví dụ 2:

```
# Tính tổng tất cả các số trong danh sách A
# Danh sách A
A = [1, 3, 5, 9, 11, 2, 6, 8, 10]
# Biến để lưu trữ tổng các số là tong, gán giá trị ban đầu bằng 0
tong = 0
# Vòng lặp for, a là biến lặp
for a in A:
     tong = tong+a
# Đầu ra: Tổng các số là 55
print("Tổng các số là", tong)
```

Khi chạy đoạn code trên, kết quả là:

```
Tổng các số là 55
```

3. Hàm range()

Bạn có thể sử dụng hàm range() để tạo ra một dãy số. Ví dụ, range(100) sẽ tạo một dãy số từ 0 đến 99 (100 số).

Hàm range(số bắt đầu, số kết thúc, khoảng cách giữa hai số)

```
#Lệnh 1
print(range(9))
#Lệnh 2
print(list(range(9)))
#Lệnh 3
print(list(range(2, 5)))
#Lệnh 4
print(list(range(0, 15, 5)))
```

Chúng ta sẽ có đầu ra như sau:

```
range(0, 9)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[2, 3, 4]
[0, 5, 10]
```

Ví dụ:

```
chuoi = ['bố','mẹ']

for tu in range(len(chuoi)):
    print("Anh yêu",chuoi[tu])
```

Ta có kết quả đầu ra giống như ví dụ 1 bên trên:

```
Anh yêu bố
Anh yêu mẹ
```

4. Vòng lặp while trong Python

Cú pháp:

```
while điều_kiện_kiểm_tra:
      Khối lệnh của while
```

Ví dụ 1:

```
count = 0
n = 0
while (count < 8):
      print ('Số thứ', n,' là:', count)
      n = n + 1
      count = count + 1
print ("Kết thúc!")
```

Với đoạn code này, ta sẽ tăng dần count và in giá trị của nó cho đến khi giá trị này không còn nhỏ hơn 8 nữa. Kết quả khi chạy lệnh trên ta có:

```
Số thứ 0 là: 0
Số thứ 1 là: 1
Số thứ 2 là: 2
Số thứ 3 là: 3
Số thứ 4 là: 4
Số thứ 5 là: 5
Số thứ 6 là: 6
Số thứ 7 là: 7
Kết thúc!
```

Ví dụ 2: Sử dụng while để tính tổng các số

```
n = int(input("Nhập n: ")) #Nhập số n tùy ý
tong = 0 #khai báo và gán giá trị cho tong
i = 1 #khai báo và gán giá trị cho biến đếm i

while i <= n:
    tong = tong + i
    i = i+1 # cập nhật biến đếm

print("Tổng là", tong)
```

Với khối lệnh trên ta có, nhập một số tự nhiên n bất kỳ và tính tổng các số từ 1 đến n, sau đó in tổng. Biến lưu trữ tổng là tong, biến đếm là i, cho đến khi i còn nhỏ hơn hoặc bằng n thì vòng lặp vẫn tiếp tục và tong vẫn tăng.

Sau khi chạy lệnh ta có kết quả:

```
Nhập n: 11
Tổng là 66
```

Ví dụ 3: Vòng lặp vô hạn

Lấy lại ví dụ trên, bạn chỉ cần bỏ đi dòng i=i+1

```
n = int(input("Nhập n: ")) #Nhập số n tùy ý
tong = 0 #khai báo và gán giá trị cho tong
i = 1 #khai báo và gán giá trị cho biến đếm i

while i <= n:
    tong = tong + i

print("Tổng là", tong)
```

Kết hợp while với else

Giống như vòng lặp for, bạn cũng có thể kết hợp else với while. Trong trường hợp này, khối lệnh của else sẽ được thực hiện khi điều kiện của while là False.

Ví dụ 4: Minh họa việc sử dụng while kết hợp với else

```
dem = 0
while dem < 3:
    print("Đang ở trong vòng lặp while")
    dem = dem + 1
else:
    print("Đang ở trong else")
```

Ở đây ta sử dụng biến dem để in chuỗi "Đang ở trong vòng lặp while" 3 lần. Đến lần lặp thứ 4, điều kiện của while trở thành False, nên phần lệnh của else được thực thi. Kết quả là:

```
Đang ở trong vòng lặp while
Đang ở trong vòng lặp while
Đang ở trong vòng lặp while
Đang ở trong else
```

5. Lệnh break trong Python

```
for var in sequence:

    #khối code bên trong vòng lặp for

    if dieu_kien:
       break
    #code khác bên trong vòng lặp for

#code bên ngoài vòng lặp for
```

Khi break được thực thi thì "#code khác bên trong vòng lặp for" sẽ bị bỏ qua và chuyển đến "#code bên ngoài vòng lặp for".

Nếu sử dụng break trong vòng lặp while Python sẽ như sau:

```
while dieu_kien_kiem_tra:

    #code bên trong vòng lặp while

    if dieu_kien:
       break

    #code khác bên trong vòng lặp while

#code bên ngoài vòng lặp while
```

Khi break được thực thi thì "#code khác bên trong vòng lặp while" sẽ bị bỏ qua và chuyển đến "#code bên ngoài vòng lặp while".

Ví dụ 1:

```
#Sử dụng break trong for

for val in "string":
```

```
    if val == "i":
        break
    print(val)

print("Kết thúc!")
```

Trong đoạn code trên, chúng ta lặp chuỗi "string", và kiểm tra điều kiện, nếu chữ cái là "i" thì sẽ thực thi lệnh break, nếu chữ cái khác "i" thì in ra màn hình. Chạy code trên ta được kết quả là các chữ cái trước "i" đã được in ra. Sau đó vòng lặp kết thúc, như kết quả dưới đây:

```
s t r Kết thúc!
```

6. Lệnh continue trong Python

Lệnh continue được sử dụng để bỏ qua phần còn lại của code bên trong vòng lặp, áp dụng cho lần lặp hiện tại. Nghĩa là vòng lặp không chấm dứt, nó sẽ tiếp tục với lần lặp kế tiếp.

Cấu trúc của continue:

```
continue
```

Lệnh continue trong vòng lặp for sẽ như sau:

```
for var in sequence:

    #khối code bên trong vòng lặp for

    if dieu_kien:

       continue

    #code khác bên trong vòng lặp for

#code bên ngoài vòng lặp for
```

Khi continue được thực thi thì "#code khác bên trong vòng lặp for" bị bỏ qua và quay trở lại "#Khối code bên trong vòng lặp for"

Lệnh continue trong vòng lặp while sẽ như sau:

```
while dieu_kien_kiem_tra:

    #code bên trong vòng lặp while

    if dieu_kien:

    continue

    #code khác bên trong vòng lặp while

#code bên ngoài vòng lặp while
```

Khi continue được thực thi "`#code khác bên trong vòng lặp while`" sẽ bị bỏ qua và quay trở lại "`#code bên trong vòng lặp while`"

Ví dụ 2:

```
# Sử dụng continue trong for

for val in "string":
    if val == "i":
       continue
    print(val)

print("Kết thúc!")
```

Code này giống hệt bên trên, chỉ thay lệnh break bằng continue. Tại đây, khi lặp chuỗi "string" đến chữ cái "i" thì sẽ bỏ qua lệnh in biến print(val) và quay trở lại lệnh if val=="i":, ta có kết quả:

```
s t r n g Kết thúc!
```

Ví dụ 3:

```
bien = 10
while bien > 0:
 bien = bien -1
 if bien == 5:
 continue
 print ('Giá trị biến hiện tại là: ', bien)
print ("OK!")
```

Nếu bien = 5 thì bỏ qua và thực hiện lần lặp tiếp theo, kết quả là:

```
Giá trị biến hiện tại là: 9
Giá trị biến hiện tại là: 8
Giá trị biến hiện tại là: 7
Giá trị biến hiện tại là: 6
Giá trị biến hiện tại là: 4
Giá trị biến hiện tại là: 3
Giá trị biến hiện tại là: 2
Giá trị biến hiện tại là: 1
Giá trị biến hiện tại là: 0
OK!
```

BÀI TẬP

1. Viết một chương trình Python in ra dãy số Fibonacci giữa 0 và 50.
2. Viết một chương trình Python đếm số các số chẵn và số các số lẻ của một dãy số
3. Viết một chương trình Python xây dựng cấu trúc sau, bằng việc sử dụng vòng lặp for lồng nhau.

   *
   * *
   * * *
   * * * *
   * * * * *
4. Viết một chương trình Python tính tổng tất cả các số trong một danh sách.
5. Viết một chương trình Python in ra các số chẵn trong một danh sách cho trước.
6. Viết một chương trình Python đảo ngược một chuỗi.
7. Viết một chương trình Python đầu vào là một số và kiểm tra xem số đó có phải là số nguyên tố hay không.
8. Viết một chương trình Python tìm các biến thực của phương trình $ax^2+bx+c=0$.