Phát triển phần mềm nâng cao cho tính toán khoa học
Lập trình cơ bản

Vũ Tiến Dũng
Khoa Toán - Cơ - Tin học
Trường ĐH Khoa học Tự Nhiên Hà Nội

Nội dung

Gọi hàm

- Thư viện chuẩn của Python cung cấp nhiều hàm để thực hiện các chức năng thông dụng trong việc lập trình như hàm tính mũ, tính giá trị tuyệt đối, tính căn,...
- Ví dụ dưới đây là gọi đến lệnh sqrt trong thư viện (module) math.
- Python đã khẳng định được những ưu điểm của mình như dễ học, dễ đọc, dễ bảo trì, cùng với hệ thống thư viện phong phú, Python dần trở thành một trong những ngôn ngữ phổ biến nhất hiện nay.
- Một người mới học lập trình có thể dễ dàng tiếp cận và sử dụng Python để giải quyết các bài toán một cách nhanh chóng so với các ngôn ngữ phức tạp khác.


```
>>> from math import sqrt
>>> x = 9.0
>>> sqrt(x)
3.0
>>>
```

Gọi hàm

- Danh sách dưới đây là một số (hàm) dựng sẵn (builtins) trong Python, các lệnh này được gọi trực tiếp mà không cần import module nào.

[abs(), all(), any(), ascii(), bin(), bool(), bytearray(), bytes(), callable(), chr(), classmethod(), compile(), complex(), delattr(), dict(), dir(), divmod(), enumerate(), eval(), exec(), filter(), float(), format(), frozenset(), getattr(), globals(), hasattr(), hash(), help(), hex(), id(), input(), int(), isinstance(), issubclass(), iter(), len(), list(), locals(), map(), max(), memoryview(), min(), next(), object(), oct(), open(), ord(), pow(), print(), property(), range(), repr(), reversed(), round(), set(), setattr(), slice(), sorted(), @staticmethod(), str(), sum(), super(), tuple(), type(), vars(), zip()]

Gọi hàm

- Một số hàm toán học cơ bản trong module math.

sqrt Computes the square root of a number: sqrt(x) = px
exp Computes e raised a power: exp(x) = ex
log Computes the natural logarithm of a number: log(x) = loge x = lnx
log10 Computes the common logarithm of a number: log(x) = log10 x
cos And other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyper bolic sine, and hyperbolic tangent
pow Raises one number to a power of another: pow(x;y) = xy
degrees Converts a value in radians to degrees: degrees(x) = 180 p x
radians Converts a value in degrees to radians: radians(x) = 180 p x
fabs Computes the absolute value of a number: fabs(x) = x

Xây dựng hàm

- Một chương trình sẽ dễ dàng tiếp cận, dễ đọc, dễ hiểu và dễ kiểm soát lỗi nếu được tổ chức thành các hàm chức năng một cách hợp lý.
- Việc viết các hàm thực hiện các công việc thành phần trong một chương trình giúp tái sử dụng hàm đó với cùng một công việc được thực hiện nhiều lần, dễ khoanh vùng xác định lỗi và sửa lỗi.

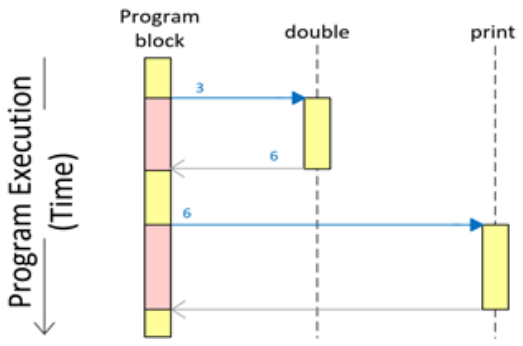
Xây dựng hàm

- Mỗi hàm trong Python có hai khía cạnh
  - Định nghĩa hàm **(Function definition)**: định nghĩa của một hàm chứa các dòng lệnh xác định các hành vi của hàm đó.
  - Lời gọi hàm **(Function invocation)**: Một hàm được sử dụng trong chương trình thông qua lời gọi hàm
- Mỗi hàm được định nghĩa một lần nhưng có thể gọi đến nhiều lần.
- Khuôn mẫu để định nghĩa một hàm như sau:

```
def name ( parameter list ):
    block
```

Xây dựng hàm

```
1 def double(a):
2     return 2*a;
3
4 x = 3
5 print(double(x))
```



- Trong ví dụ trên, từ khóa **def** đánh dấu việc bắt đầu định nghĩa hàm, tên hàm là **double**, đối số là **a**, nội dung hàm là trả lại giá trị bằng hai lần giá trị đối số **a**.
- Lời gọi hàm được thực hiện bằng tên hàm (double) cùng với đối số được truyền vào là x, kết quả lời gọi hàm double(x) được in ra màn hình bằng lệnh print. Luồng thực hiện được thể hiện trong hình minh họa dưới đây
- Từ khóa **return** được dùng để trả lại giá trị cho hàm. Một hàm có thể không có đối số, và có thể không cần trả lại giá trị gì

Xây dựng hàm

```
1 def double(a):
2     a = a*2
3     return a;
4
5 x = 3
6 print(double(x))
7 print(x)
```

- Khi một hàm được gọi tới với tham số được truyền vào, khi đó biến tham số sẽ tham chiếu đến biến hoặc giá trị được truyền làm tham số
- Các kiểu dữ liệu như integer, float-point và string được xem là các đối tượng bất biến (**immutable**), giá trị của các đối tượng bất biến không bị thay đổi khi được truyền là tham số của các hàm
- Trong ví dụ trên, kết quả của double(x) là 6, nhưng x vẫn có giá trị bằng 3

Xây dựng hàm

- Tham số của hàm có thể có giá trị mặc định, giá trị này có thể bỏ qua trong lời gọi hàm, các tham số có giá trị mặc định phải đứng sau các tham số không có giá trị mặc định.
- Trong ví dụ dưới đây, tham số a của hàm grow có giá trị mặc định là 2, khi gọi hàm grow ta có thể bỏ qua tham số a, khi đó a sẽ nhận giá trị mặc định là 2, ta cũng có thể thay giá trị mặc định của a bằng giá trị khác.

```
>>> def grow(b, a= 2):
...     return a*b;
... x = 3
>>> print(grow(x))
6
>>> print(x)
3
>>> print(grow(x,3))
9
>>>
```

Phạm vi biến

- Biến được định nghĩa bên trong hàm được gọi là biến cục bộ (local variable). Biến cục bộ được lưu trong bộ nhớ khi hàm được gọi đến và thực thi, khi kết thúc hàm hoặc biến không còn được sử dụng tới trong hàm thì bộ nhớ dùng để lưu trữ biến sẽ được giải phóng. Cùng một tên biến có thể sử dụng trong các hàm khác nhau mà không gây xung đột.
- Trái ngược với biến cục bộ là biến toàn cục (global variable), biến toàn cục có phạm vi trên toàn bộ chương trình, trên tất cả các hàm, điều này có nghĩa là tất cả các hàm đều có thể truy cập và thay đổi biến toàn cục. Để khai báo sử dụng biến toàn cục ta sử dụng từ khóa global, việc sử dụng từ khóa global chỉ cần thiết khi ta gán, hoặc thay đổi giá trị của biến toàn cục.

Phạm vi biến

Hình 1: Ví dụ về biến toàn cục và biến cục bộ

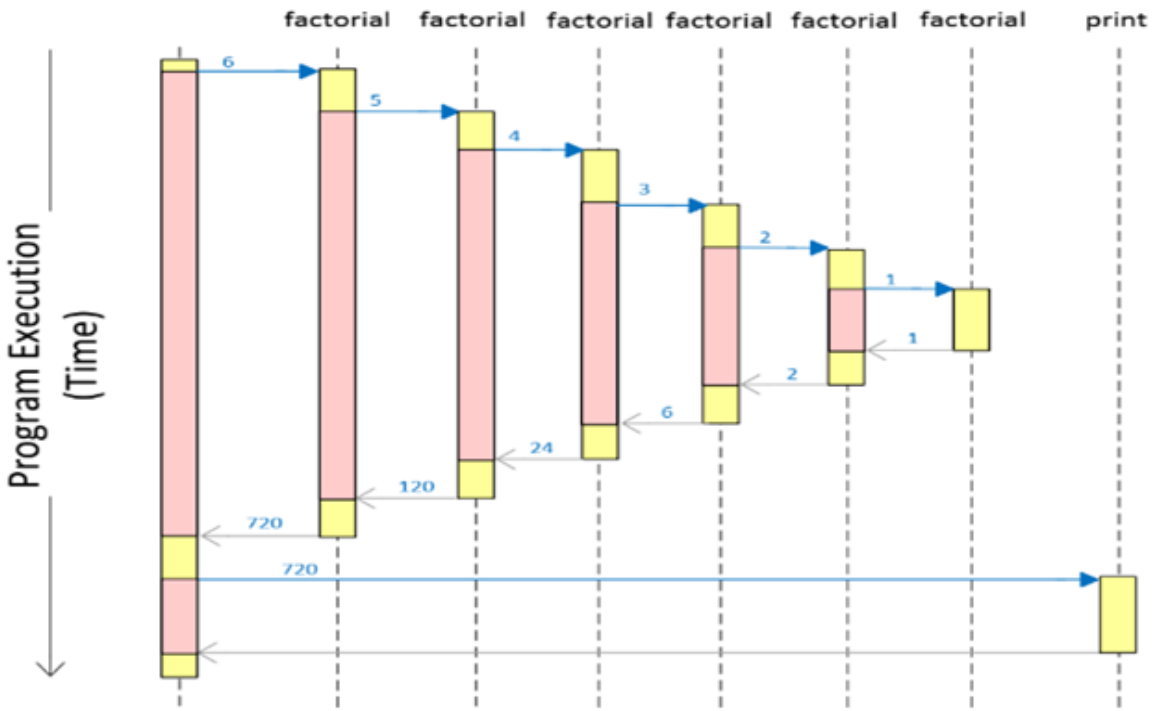
```
1 result = 0
2 x, y, z = 5, 6, 3
3
4 def maxInTriple():
5     return max(max(x,y),z);
6
7 def doubleMax():
8     global result
9     result = maxInTriple() * 2;
10
11 print(maxInTriple())
12 doubleMax()
13 print(result)
```

Hàm đệ quy

- Hàm đệ quy là các hàm có chứa lời gọi hàm đến chính nó. Hàm đệ quy thường được sử dụng để giải các bài toán có định nghĩa đệ quy hoặc quy nạp
- Nhìn chung, với cùng một bài toán nếu có hàm đệ quy và hàm không đệ quy để giải bài toán đó thì thường hàm đệ quy có lời giải ngắn gọn hơn, tuy nhiên trong một số trường hợp hàm đệ quy sẽ tốn bộ nhớ và chi phí tính toán hơn.
- Khi xây dựng hàm đệ quy, cần xác định trường hợp cơ sở (**base case**) để đảm bảo hàm đệ quy có điểm dừng.

Hàm đệ quy

Hình 2: Ví dụ về hàm đệ quy




```
>>> def factorial(n):
...     if(n == 0):
...         return 1
...     else:
...         return n*factorial(n-
...
...
...
...
>>> factorial(6)
720
```

Kiểu dữ liệu của hàm

- Trong Python, một hàm là một đối tượng đặc biệt, hàm cũng xác định kiểu giống như số nguyên hay xâu ký tự. Ví dụ dưới đây, kiểu của hàm sqrt trong module math được xác định bằng hàm type.
- Trong ví dụ này, x được gán bằng sqrt, khi đó định danh x tham chiếu tới hàm sqrt và thực hiện chức năng tương tự, kết quả của x(9) là 3

```
>>> from math import sqrt
>>> type(sqrt)
<class 'builtin_function_or_method'>
>>> x = sqrt
>>> x(9)
3.0
```

Kiểu dữ liệu của hàm

Hình 3: Đối số của một hàm có thể là một hàm số khác

```
1 def add(a,b):
2     return a+b;
3
4 def sub(a,b):
5     return a-b;
6 def run(f,a,b):
7     return f(a,b)
8
9 a = 10
10 b = 21
11 print(run(add,a,b))
12 print(run(sub,a,b))
```

Biểu thức lambda

- Python hỗ trợ việc định nghĩa hàm thông qua biểu thức lambda.
- Biểu thức lambda giúp việc định nghĩa hàm dễ hơn và nhanh gọn hơn, ngoài ra nó còn cung cấp một cách để định nghĩa các hàm không định danh.


```
>>> f = lambda x, y: 10 if x == y else 2
>>> f(2,5)
2
>>> f(5,5)
10
>>> type(f)
<class 'function'>
>>>
```

Bộ sinh - Generator

- Một chương trình sinh, hay bộ sinh tạo ra một chuỗi các giá trị. Từ khóa **return** dùng để trả lại giá trị cho một hàm, đồng thời nó cũng kết thúc hàm ngay sau khi trả lại giá trị.
- Thay vì sử dụng từ khóa return, chúng ta sử dụng từ khóa yield để tạo ra bộ sinh. Từ khóa yield thực hiện trả lại một giá trị nhưng không kết thúc hàm.


```
>>> f = lambda x, y: 10 if x == y else 2
>>> f(2,5)
2
>>> f(5,5)
10
>>> type(f)
<class 'function'>
>>>
```

Bộ sinh - Generator

Hình 4: Ví dụ về bộ sinh

```
>>> def gen():
...     yield 1
...     yield 'aaa'
...     yield 3.5
...
...
>>> type(gen)
<class 'function'>
>>> f = gen()
>>> type(f)
<class 'generator'>
>>> next(f)
1
>>> next(f)
'aaa'
>>> next(f)
3.5
>>> next(f)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
>>>
>>> for i in gen():
...     print(i)
...
1
aaa
3.5
>>>
```

Biểu thức điều kiện

- Biểu thức điều kiện là biểu thức nhận giá trị logic hoặc là đúng hoặc là sai (True hoặc False). Python cung cấp một kiểu dữ liệu để lưu trữ giá trị logic là kiểu **bool**.
- Chú ý, vì Python là ngôn ngữ phân biệt hoa thường, từ khóa cho giá trị logic đúng là **True** (không phải *true* hay *TRUE*,...), tương tự từ khóa cho giá trị logic sai là **False** (không phải *false* hay *FALSE*). Trong ví dụ trên nếu gán a true, trình thông dịch sẽ báo lỗi, kiểu của giá biến lưu giá trị logic là **bool**.

Biểu thức điều kiện

Hình 5: Minh họa về giá trị logic - bool trong Python

```
>>> a = true
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'true' is not defined
>>> a = True
>>> print(a)
True
>>> type(a)
<class 'bool'>
```

Biểu thức điều kiện

Các biểu thức logic hay biểu thức điều kiện được tạo thành từ các biến và các phép toán quan hệ (phép so sánh). Bảng dưới đây mô tả các phép toán quan hệ logic trong Python.

Biểu thức	Ý nghĩa	Ví dụ
x == y	Biểu thức nhận giá trị đúng (True) nếu như x = y (phép so sánh bằng, không phải phép gán); ngược lại biểu thức nhận giá trị sai (False)	5 == 5 => True 5 == 4 => False
x < y	Biểu thức nhận giá trị đúng (True) nếu x < y; ngược lại nhận giá trị sai (False)	3 < 5 => True 4 < 4 => False
x <= y	Biểu thức nhận giá trị đúng (True) nếu x ≤ y; ngược lại nhận giá trị sai (False)	5 <= 5 => True 5 <= 4 => False
x > y	Biểu thức nhận giá trị đúng (True) nếu x > y; ngược lại nhận giá trị sai (False)	6 > 5 => True 5 > 5 => False
x >= y	Biểu thức nhận giá trị đúng (True) nếu x ≥ y ngược lại nhận giá trị sai (False)	5 >= 5 => True 5 >= 6 => False
x != y	Biểu thức nhận giá trị đúng (True) nếu x ≠ y; ngược lại nhận giá trị sai (False)	5 != 6 => True 5 != 5 => False

Biểu thức điều kiện

Các phép toán quan hệ, kết hợp với các phép toán logic sẽ tạo ra các biểu thức logic phức tạp hơn, nhằm đạt được các biểu thức điều kiện phù hợp với các bài toán thực tế. Bảng dưới đây mô tả các phép toán logic trong Python

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Biểu thức điều kiện

Cũng giống như các phép toán số học, các phép toán logic cũng có thứ tự ưu tiên. Bảng dưới đây mô tả thứ tự ưu tiên của các phép toán logic.

Số ngôi	Toán tử	Thứ tự
Hai ngôi	>, <, >=, <=, ==, !=	Trái sang phải
Một ngôi	not	
Hai ngôi	and	Trái sang phải
Hai ngôi	or	Trái sang phải

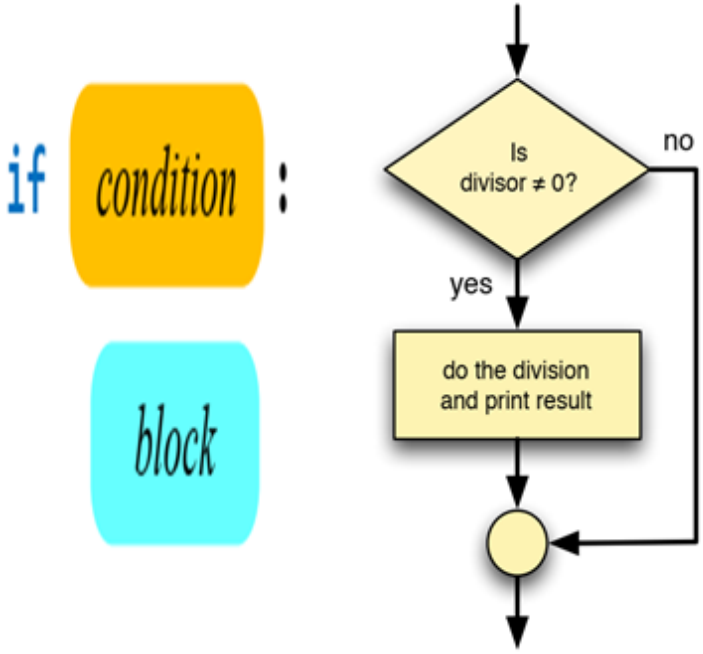
Biểu thức điều kiện

Các phép toán logic thì có độ ưu tiên thấp hơn so với các phép toán số học.

```
>>> x = 3
>>> x * 3 + 2 < 6 or x*2 + 3 > 8
True
```

Cấu trúc điều kiện if

- Lệnh if thực hiện tính giá trị của biểu thức điều kiện, nếu biểu thức nhận giá trị đúng True, các lệnh trong khối block sẽ được gọi tới, nếu giá trị của biểu thức điều kiện nhận giá trị sai thì các lệnh trong khối block sẽ không được gọi tới.



- Chú ý đến dấu ':' sau biểu thức điều kiện, và khối lệnh block cần được thụt lề so với lệnh if (điều này là bắt buộc) vì Python không gióng hàng lệnh bằng dấu .

Cấu trúc điều kiện if

- Trong Python, giá trị của biểu thức điều kiện có thể là giá trị số, một giá trị số có giá trị khác 0, khác rỗng sẽ nhận giá trị logic đúng True, và giá trị số bằng 0 hoặc 0.0 sẽ nhận giá trị logic sai False.

```
>>> if 2:
...     print('Giá trị khác 0 tương đương với giá trị logic đúng True')
...
Giá trị khác 0 tương đương với giá trị logic đúng True
>>> if 0:
...     print('Giá trị bằng 0 tương đương với giá trị logic sai False')
...
>>>
```

- Lệnh if có thể phát biểu như một mệnh đề điều kiện, nếu ... thì ...

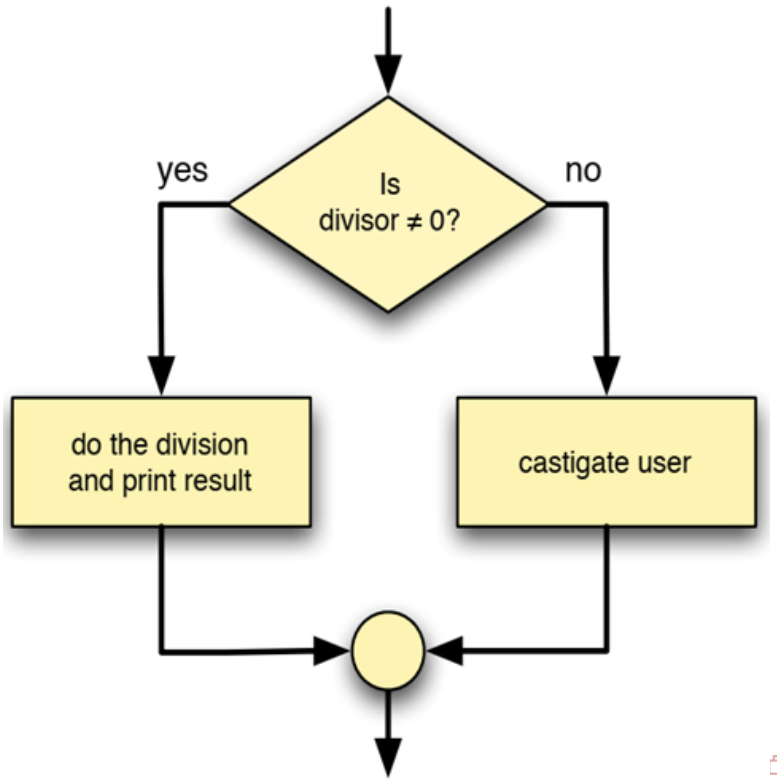
Cấu trúc điều kiện if

Hình 6: Ví dụ với cấu trúc điều kiện if

```
>>> if 2:
...     print('Giá trị khác 0 tương đương với giá trị logic đúng True')
...
Giá trị khác 0 tương đương với giá trị logic đúng True
>>> if 0:
...     print('Giá trị bằng 0 tương đương với giá trị logic sai False')
...
>>>
```

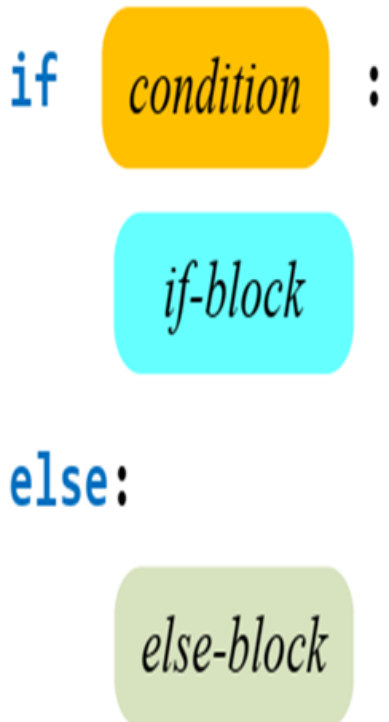
Cấu trúc điều kiện if/else

- Lệnh if sẽ thực hiện các lệnh trong khối block nếu biểu thức điều kiện nhận giá trị đúng True, và nếu chúng ta muốn thực hiện các lệnh khi biểu thức nhận giá trị sai ta cần viết thêm một lệnh if nữa, sẽ tốt hơn nếu ta có lệnh để tránh việc này.
- Python cung cấp khối lệnh if / else để việc viết các khối lệnh với biểu thức điều kiện linh hoạt hơn.



Cấu trúc điều kiện if/else

- Chú ý đến dấu ':' sau từ khóa else và việc thụt lề của khối lệnh trong else-block là bắt buộc



Cấu trúc điều kiện if/else

Hình 7: Ví dụ với cấu trúc điều kiện if/else

```
>>> x =  10
>>> y = int(input('Nhập giá trị  y = '))
Nhập giá trị  y = 5
>>> if(y == 0):
...     print('Không thực hiện được phép chia cho 0')
... else:
...     print(x,'/',y,' = ',x/y)
...
10 / 5  =  2.0
```

Lệnh pass

- Python cung cấp lệnh pass để thực hiện lệnh đặc biệt là "không thực hiện gì".

```
1 x = 5
2 if(x < 0):
3     #Không thực hiện gì
4 else:
5     print('x = ',x)
```

- Khối lệnh trong ví dụ trên là không hợp lệ trong Python, bên trong khối lệnh if cần có ít nhất một câu lệnh, để thực hiện yêu cầu như ví dụ trên ta có thể sử dụng lệnh pass.

```
1 x = 5
2 if(x < 0):
3     pass #Không thực hiện gì
4 else:
5     print('x = ',x)
```

Sai số với phép toán trên số thực

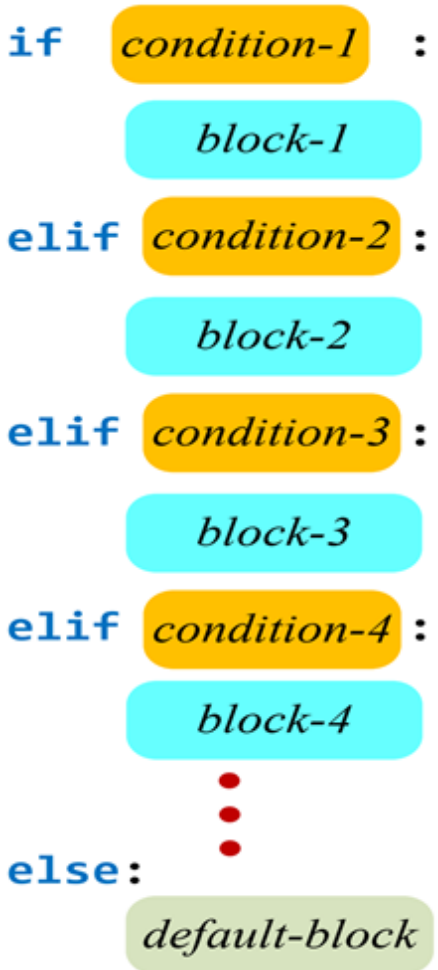
- Chú ý đến sai số trong các phép so sánh với giá trị số thực.

```
>>> x = 1.11 - 1.1
>>> y = 2.11 - 2.1
>>> print('x = ',x,'; y = ',y)
x =  0.010000000000000009 ; y =  0.009999999999999787
>>> x == y
False
>>>
```

Cấu trúc điều kiện if/elif

- Các khối lệnh if/else có thể lồng nhau để mô tả các biểu thức hoặc chương trình phức tạp, một lệnh if có thể đứng độc lập, tuy vậy một lệnh else thì cần đi cùng với 1 lệnh if, lệnh else sẽ bắt cặp với lệnh if gần nhất không chứa else.
- Python không có cấu trúc switch/case như các ngôn ngữ Java, C, C++,... Tuy nhiên Python có cặp lệnh if/elif/else thực hiện chức năng tương tự và linh hoạt hơn.

Cấu trúc điều kiện if/elif




```
1  month = int(input('Nhập vào tháng 1 - 12'))
2  if month == 1:
3      print('tháng 1')
4  elif month == 2:
5      print('tháng 2')
6  elif month == 3:
7      print('tháng 3')
8  elif month == 4:
9      print('tháng 4')
10 elif month == 5:
11     print('tháng 5')
12 elif month == 6:
13     print('tháng 6')
14 elif month == 7:
15     print('tháng 7')
16 elif month == 8:
17     print('tháng 8')
18 elif month == 9:
19     print('tháng 9')
20 elif month == 10:
21     print('tháng 10')
22 elif month == 11:
23     print('tháng 11')
24 elif month == 12:
25     print('tháng 12')
26 else:
27     print('Giá trị nhập không đúng (1 - 12)')
```

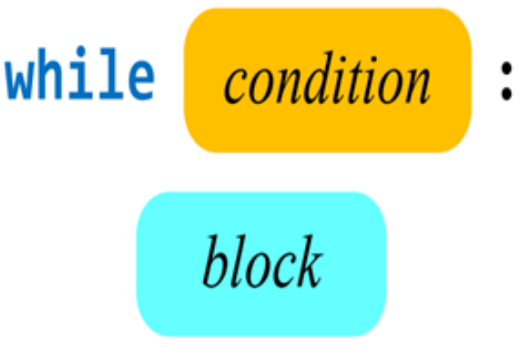
Cấu trúc điều kiện if/elif

Ngoài ra, biểu thức điều kiện có thể viết dưới dạng như sau:

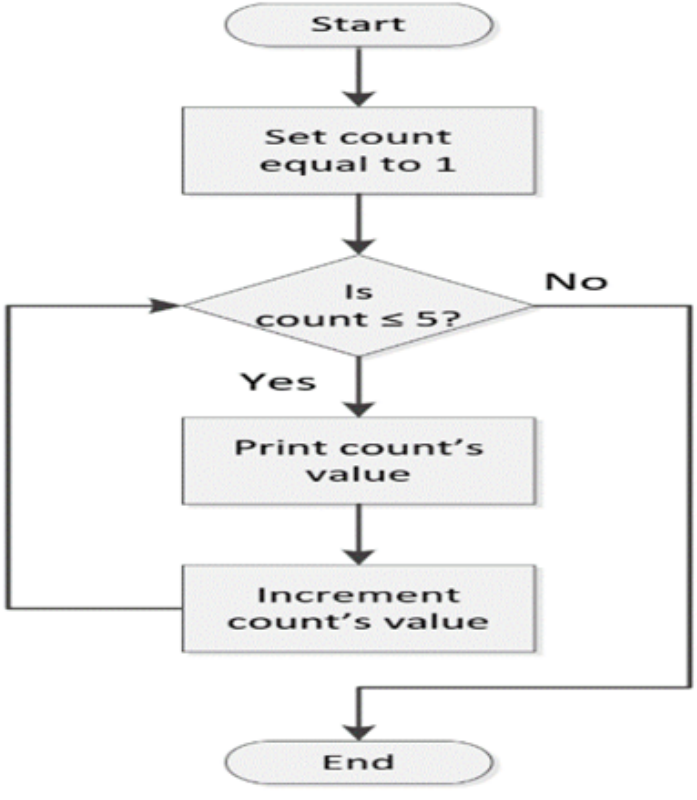

```
1 số_bị_chia = int(input('nhập vào số chia = '))
2 số_chia = int(input('nhập vào số chia = '))
3
4 thương = số_bị_chia / số_chia if số_chia != 0 else 'Lỗi, không chia được cho 0'
5 print(thương)
```

Vòng lặp while

Khối lệnh block được thực hiện trong khi điều kiện condition còn nhận giá trị đúng True.




```
count = 1
while count <= 5:
    print(count)
    count +=1
```



Vòng lặp for

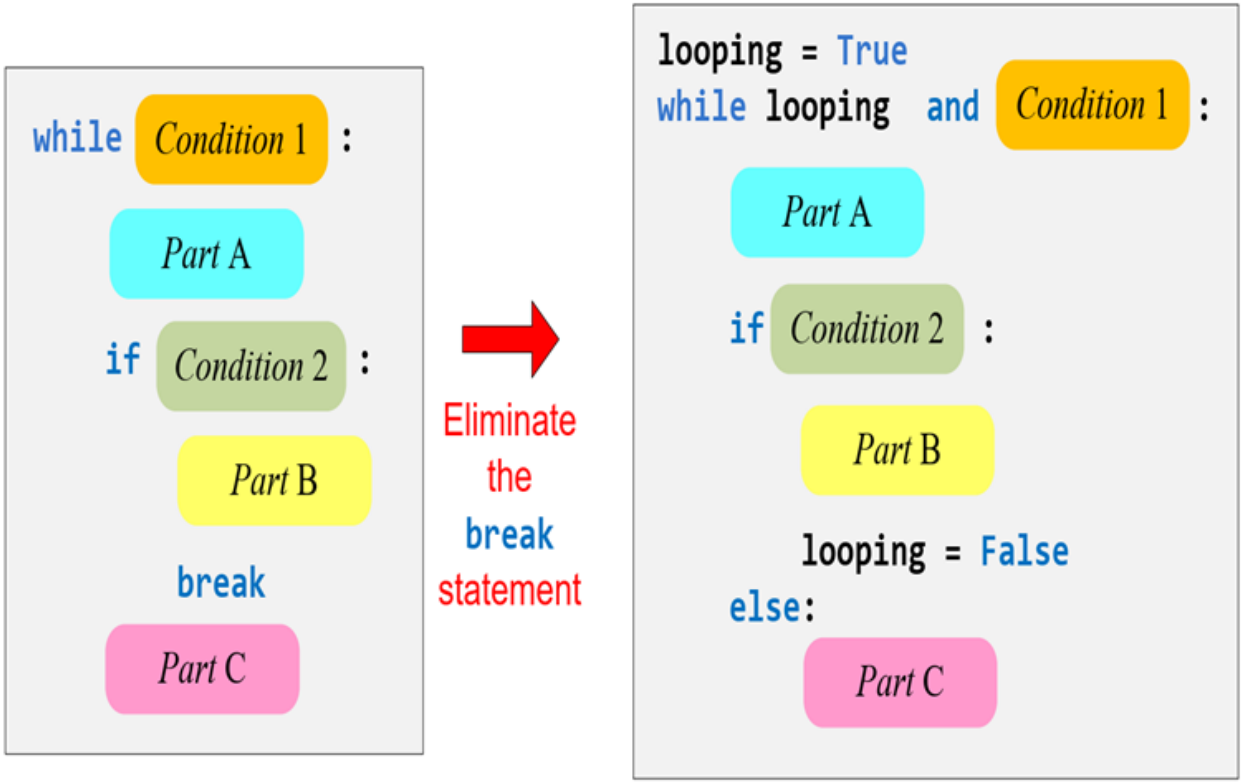
- Vòng lặp for thực hiện duyệt qua lần lượt các phần tử trong danh sách và gán cho biến variable, ứng với mỗi phần tử trong list khối lệnh block sẽ được thực hiện một lần.
- Các vòng lặp có thể được sử dụng lồng nhau để giải quyết các bài toán như in ma trận, duyệt các phần tử trên ma trận, hay khi làm việc với dữ liệu kiểu mảng nhiều chiều (danh sách của các danh sách).

```
for variable in list:
    block
```

```
for i in range(1, 10):
    print(i)
```

Lệnh break

- Các vòng lặp có thể kết thúc bằng lệnh **break**.
- Lệnh **break** thường được dùng để kết thúc vòng lặp khi đạt được điều kiện không, hoặc khó xác định bởi số lần lặp, đôi khi là vòng lặp vô hạn cho đến khi tìm thỏa mãn điều kiện thì dừng.



Lệnh break

Hình 8: Ví dụ sử dụng break trong vòng lặp

```
import math
a , b = 1, 10 # hai điểm đầu mút của đoạn [1, 10]
e = 0.000000001 # sai số
f = lambda x : 2*x-4
y = (a+b)/2

while math.fabs(f(y)) > e:
    y = (a+b)/2
    if(math.fabs(f(y)) < e):
        break
    elif f(y)*f(a) < 0:
        b = y
    else:
        a = y

print('nghiệm của phương trình là y = ',y)
```

Lệnh continue

- Khi chương trình gặp lệnh break bên trong một vòng lặp, các câu lệnh phía sau lệnh **break** trong vòng lặp sẽ bị bỏ qua (không thực hiện) và chương trình thoát khỏi vòng lặp.
- Trong vòng lặp, đôi khi với một điều kiện nào đó chúng ta muốn bỏ qua lượt lặp hiện tại, nhưng vẫn muốn thực hiện các vòng lặp tiếp theo ta có thể dùng lệnh **continue**.

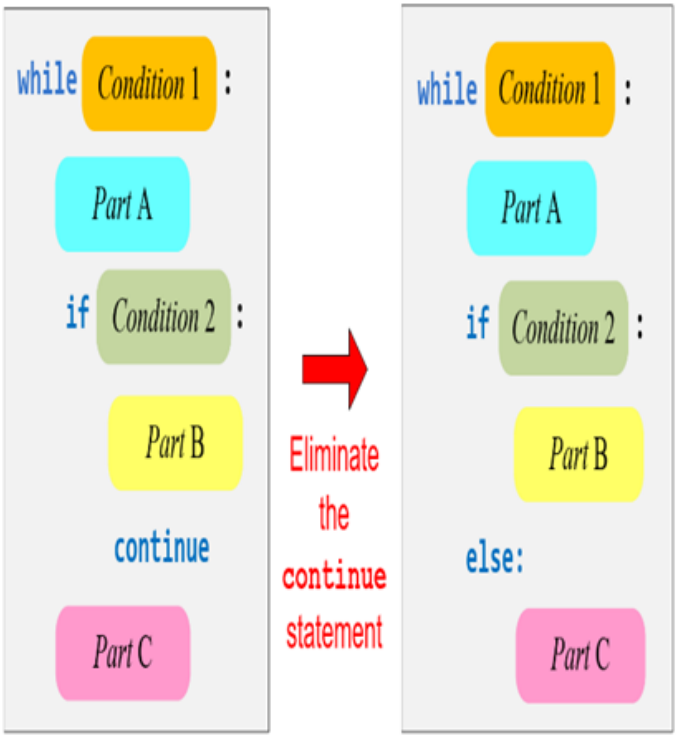
Lệnh continue

Hình 9: Ví dụ sử dụng continue trong vòng lặp

```
count = 1
sum = 0

while count < 1000: # Tính tổng các số chẵn nhỏ hơn 1000
    count += 1
    if(count % 2 == 1):
        continue
    sum += count

print('Tổng  = ', sum)
```



Lệnh while / else và for / else

- Python hỗ trợ cấu trúc while / else và for / else cho các vòng lặp while và for.
- Với cấu trúc này, khi vòng lặp kết thúc, các lệnh trong khối else sẽ được gọi tới và thực thi
- Chú ý, nếu vòng lặp bị kết thúc bởi lệnh break thì khối lệnh trong else sẽ không được gọi tới.

```
count = 1
sum = 0

while count < 1000: # Tính tổng các số chẵn nhỏ hơn 1000
    count += 1
    if(count % 2 == 1):
        continue
    sum += count
else:
    print('Tổng  = ', sum)
```

```
count = 1
sum = 0

while count < 1000: # Tính tổng các số chẵn nhỏ hơn 1000
    count += 1
    if(count % 2 == 1):
        continue
    sum += count
    if(sum > 2500):
        break
else:
    print('Tổng  = ', sum)
```

Ví dụ với vòng lặp

- Ví dụ dưới đây minh họa chương trình đếm ngược từ 10 về 0.

```
from time import sleep
for count in range(10, -1, -1): # Range 10, 9, 8, ..., 0
    print(count) # Hiển thị giá trị count hiện tại
    sleep(1) # Tạm dừng (ngủ) chương trình 1 giây
```

- Chương trình sinh số ngẫu nhiên với module random

```
from random import randrange, seed

seed(23) # Set random number seed
for i in range(0, 100): # Print 100 random numbers
    print(randrange(1, 101), end=' ') # Số ngẫu nhiên từ 1-100
print() # Print newine
```

Ví dụ với vòng lặp

- Chương trình sinh số ngẫu nhiên với module random với nhân (seed được khởi tạo theo thời gian)

```
from random import randrange, seed
from datetime import datetime
seed(datetime.now())
for i in range(0, 100): # Print 100 random numbers
    print(randrange(1, 101), end=' ') # Số ngẫu nhiên
từ 1 đến 100
print() # Print newine
```