Lập trình Python cơ bản

An Duy Le

Tháng 6, 2020

Mục lục

1 Biến

Biến được sử dụng để tham chiếm đến vị trí bộ nhớ. Biến trong Python còn được gọi là định danh và được sử dụng để lưu giá trị. Trong Python, người lập trình không cần chỉ định kiểu dữ liệu của biến biến vì Python tự động chọn kiểu dữ liệu cho phù hợp.

Tên biến gồm cả chữ cái và chữ số, nhưng phải bắt đầu bằng một chữ cái hoặc dấu gạch dưới. Nên sử dụng chữ cái viết thường để đặt tên cho biến. Trong Python, tên biến phân biệt chữ hoa và chữ thường.

1.1 Quy cách đặt tên biến

- Kí tự đầu tiên của biến bắt buộc phải là chữ cái hoặc dấu gạch dưới.
- Các kí tự đằng sau có thể là chữ thường, chữ hoa hoặc kí tự số.
- Tên biến không được chứa dấu cách hoặc các kí tự đặc biệt như *, !, @, #, ∧, &.
- Tên biến không được trùng với từ khoá. Xem danh sách các từ khoá ở mục Các từ khoá.

Lập trình viên có thể đặt tên biến theo 3 kiểu sau đây:

- Kiểu lạc đà: kí tự đầu tiên viết thường, kí tự bắt đầu mỗi từ đằng sau viết hoa. Ví dụ: className,...
- Kiểu Pascal: kí tự đầu tiên của mỗi từ được viết hoa. Ví dụ: Weigth, NameOfStudent,..
- Kiểu con rắn: tên biến viết thường, mỗi từ phân cách nhau bằng dấu gạch dưới. Ví dụ: check_prime_number,...

1.2 Khai báo biến

Python không bắt buộc lập trình viên phải khai báo biến trước khi sử dụng trong một ứng dụng, nó cho phép chúng ta khai báo bất cứ lúc nào người lập trình cần.

Người lập trình cũng không cần khai báo kiểu dữ liệu cho biến. Khi chúng ta gán dữ liệu cho nó bằng dấu "=", Python tự nhận dạng kiểu dữ liệu.

Ví dụ: `a = 10, string = "Hello world!",...`

Ngoài cách khai báo trên, người lập trình có thể khai báo và gán giá trị một lúc nhiều biến khác nhau. Các biến này có thể có cùng giá trị hoặc không.

Ví dụ:

```
1  a = b = 5
2  x, y, z = 1, 2, 3 # x = 1, y = 2, z = 3
3  print(a, b)
4  print(x, y, z)
```

Kết quả cho ra ở Console:

```
1  5 5
2  1 2 3
```

1.3 Tham chiếu đối tượng

Người lập trình cần phải hiểu cách trình biên dịch Python hoạt động khi khai báo biến. Quá trình xử lý này có phần khác với nhiều ngôn ngữ lập trình.

Python là ngôn ngữ lập trình mang tính hướng đối tượng cao, đó là lý do mà mọi kiểu dữ liệu thuộc về một lớp cụ thể. Sử dụng từ khoá `"type"` để hiển thị kiểu dữ liệu của giá trị.

Ví dụ:

```
1 print(type("Hello world!"))
2 print(type(123))
```

Kết quả cho ra ở Console:

```
1 <class 'str'>
2 <class 'int'>
```

Trong Python, các biến có một tên tượng trưng tham chiếu hoặc trỏ tới một đối tượng nào đó.

Ví dụ

```
a = 50
```

Khi khai báo `a = 50`, biến a tham chiếu tới một đối tượng số nguyên có giá trị là 50.

Hình 1.1: Mô tả tham chiếu của biến a

Khi lập trình viên gán giá trị của a cho một biến b:

```
a = 50
b = a
```

Biến b sẽ tham chiếu tới đối tượng mà biến a đang tham chiếu tới do Python không tạo ra đối tượng mới trong trường hợp này.

Hình 1.2: Mô tả tham chiếu của biến a và b

Python thể hiện tính hiệu quả trong việc quản lý dữ liệu khi người lập trình gán cùng dữ liệu cho nhiều biến khác nhau.

1.4 Danh tính đối tượng

Trong Python, mọi đối tượng được tạo ra đều là duy nhất. Python đảm bảo rằng không có hai đối tượng có cùng định danh. Sử dụng hàm `id()` để kiểm tra định danh của đối tượng.

Ví dụ:

```
a = b = 1
c = 128
print(id(a))
print(id(b))
print(id(c))
```

Kết quả cho ra ở Console:

```
1824716720
1824716720
1824718752
```

Biến a và b có cùng giá trị, do đó, chúng cùng tham chiếu đến một đối tượng nên có số `id()` như nhau. Biến c có giá trị khác a và b nên số `id()` cũng khác nhau.

2 Các kiểu dữ liệu

2.1 Kiểu chuỗi

Người lập trình có thể tạo ra một chuỗi bằng cách đặt nội dung vào dấu ngoặc đơn hoặc ngoặc kép.

Ví dụ:

```
"Hello", 'world!'
```

Ngoài chuỗi trên cùng một hàng, Python hỗ trợ hai cách khởi tạo chuỗi có nội dung trên nhiều hàng:

Ví dụ:

```
str1 = 'Hello '\
    'World!'
str2 = """Hello
World!
"""
print(str1)
print()
print(str2)
```

Kết quả cho ra ở Console:

```
Hello World!

Hello
World!
```

2.2 Kiểu số

Tương tự như các ngôn ngữ lập trình khác, Python cũng có kiểu số nguyên và số thực. Ngoài ra, trong Python hỗ trợ thêm cho các lập trình viên kiểu số phức. Số nhị phân, bát phân, thập lục phân cũng được khai báo một cách dễ dàng.

Ví dụ:

```
a = 0b10100 # Binary Literals
b = 100 # Decimal Literal
c = 0o215 # Octal Literal
d = 0x12d # Hexadecimal Literal
f = 100.5
z = 2 + 3j
print(a, b, c, d)
print(f)
print(z, z.real, z.imag)
```

Kết quả cho ra ở Console:

```
20 100 141 301
100.5
(2+3j) 2.0 3.0
```

2.3 Kiểu logic

Kiểu logic chứa một trong hai giá trị là `True` và `False`. Trong kiểu Boolean, giá trị `True` được hiểu là 1, giá trị `False` được hiểu là 0.

Ví dụ:

```
print(True + 10)
print(False + 10)
print(True == 2)
```

Kết quả cho ra ở Console:

```
11
10
False
```

2.4 Kiểu None

Kiểu `None` là một kiểu đặc biệt trong Python, dùng để ám chỉ một biến chưa được khởi tạo hoặc làm giá trị kết thúc của list.

Ví dụ:

```
1 val1, val2 = 10, None
2 print(val1, val2)
```

Kết quả cho ra ở Console:

```
1 10 None
```

2.5 Các collections

Python cung cấp 4 loại collection: List, Tuple, Dict và Set.

2.5.1 List

List có thể chứa nhiều phần tử có kiểu dữ liệu khác nhau. List có thể thay đổi, sửa đổi các phần tử bên trong được. Các giá trị trong list phân tách nhau bởi dấu phẩy và được đặt trong cặp dấu ngoặc vuông.

Ví dụ:

```
ls = [1, 3, 5, 'Python']
print(ls)
ls[0] = 2
print(ls)
```

Kết quả cho ra ở Console:

```
[1, 3, 5, 'Python']
[2, 3, 5, 'Python']
```

2.5.2 Tuple

Tương tự như list, Tuple có thể chứa nhiều phần tử có kiểu dữ liệu khác nhau. Tuy nhiên, Tuple là kiểu dữ liệu immutable (không thể thay đổi các phần tử sau khi khởi tạo). Các giá trị trong Tuple phân cách nhau bởi dấu phẩy và được đặt trong cặp dấu ngoặc tròn.

Ví dụ:

```
tpl = (1, 3, [2, "Py"])
print(tpl)
tpl[1] = 2
```

Kết quả cho ra ở Console:

```
TypeError: 'tuple' object does not support item assignment
(1, 3, [2, 'Py'])
```

2.5.3 Dictionary

Dictionary lưu trữ dữ liệu bằng từng cặp key - value. Các giá trị trong Dict phân cách nhau bởi dấu phẩy và được đặt trong cặp dấu ngoặc nhọn. Cặp key - value phân cách nhau bởi dấu hai chấm.

Ví dụ:

```
1  dct = {'Name': 'An', 'Age': 21, 'School': 'ACT'}
2  print(dct)
```

Kết quả cho ra ở Console:

```
1  {'Name': 'An', 'Age': 21, 'School': 'ACT'}
```

2.5.4 Set

Set lưu trữ các giá trị không theo thứ tự. Các giá trị trong Set phân cách nhau bởi dấu phẩy và được đặt trong cặp dấu ngoặc nhọn.

Tương tự như List và Tuple, một Set cũng có thể lưu được nhiều kiểu dữ liệu khác nhau, tuy nhiên, Set không chấp nhận phần tử có kiểu dữ liệu là List.

Ví dụ:

```
1  st = {"C++", "Java", "Python", "Scala", "Ruby", 1}
2  print(st)
```

Kết quả cho ra ở Console:

```
1  {1, 'Ruby', 'Python', 'Scala', 'C++', 'Java'}
```

3 Các từ khoá

Từ khoá trong Python là những từ đặc biệt đối với trình biên dịch, mang một ý nghĩa nào đó. Những từ này không được dùng như tên biến.

Từ khoá	Ý nghĩa
`True`	trả về giá trị `True` nếu điều kiện đưa ra là đúng. Giá trị khác 0 được hiểu là `True`.
`False`	trả về giá trị `False` nếu điều kiện đưa ra là sai. Giá trị 0 được hiểu là `False`.
`None`	biểu thị cho giá trị `null`. Một list rỗng hay giá trị 0 không được xem là `None`.
`asset`	dùng để debug trong Python, in ra lỗi của một đoạn code.
`def`	dùng để khởi tạo một hàm.
`class`	dùng để định nghĩa một lớp. Một lớp bao gồm các biến và thủ tục.
`del`	dùng để giải phóng tham chiếu tới một đối tượng.
`try, except, finally`	được dùng trong xử lý ngoại lệ. Xem mục Ngoại lệ và xử lý ngoại lệ
`raise`	tạo ngoại lệ theo định nghĩa của người lập trình. Xem mục Ngoại lệ và xử lý ngoại lệ
`and, or, not`	được sử dụng trong mệnh đề logic. Xem mục Toán tử logic.
`for, while`	được sử dụng để tạo vòng lặp. Xem mục Vòng lặp for và Vòng lặp while.
`in`	kiểm tra phần tử có nằm trong một tập hợp hay không. Xem mục Toán tử khai thác
`is`	xác thực phần tử. Xem mục Toán tử xác thực
`if, else, elif`	được sử dụng để tạo cấu trúc rẽ nhánh. Xem mục Câu lệnh rẽ nhánh
`continue, break, pass`	xem mục Từ khoá continue, break, pass
`import`	dùng để thêm thư viện cho file.
`from`	dùng để thêm một hàm cụ thể trong thư viện nào đó cho file.
`as`	dùng để tạo một bí danh cho tên thư viện.
`return`	dùng để trả về một giá trị hoặc `None` trong hàm.
`gobal`	dùng để tạo một biến toàn cục.
`nonlocal`	tương tự như gobal, dùng trong các hàm lồng nhau.
`lambda`	dùng để tạo hàm vô danh.
`print`	dùng để in ra màn hình.
`input`	dùng để nhập liệu từ bàn phím.
`int`	kiểu dữ liệu integer.
`float`	kiểu dữ liệu float.

Bảng 1: Một số từ khoá thông dụng

4 Chú thích

4.1 Chú thích

Chú thích trong Python là một công cụ thiết yếu cho lập trình viên, giúp giải thích, mô tả chức năng của một đoạn code nào đó.

Trong các ngôn ngữ lập trình khác như Java hoặc C++ cung cấp kí hiệu // cho chú thích trên một dòng và cặp kí tự /* */ cho nhiều dòng. Tuy nhiên, Python cung cấp kí hiệu # cho chú thích trên một dòng.

Ví dụ: Hàm kiểm tra một số nguyên có phải là số nguyên tố hay không:

```
1  import math
2  def is_prime_number(n):
3      # input: a integer number
4      # output: if n is a prime number, return True. If not, return False.
5      # if n = 1 or n < 0, n isn't a prime number
6      if n < 1:
7          return False
8      # if n = 2 or n = 3, n is a prime number
9      if n < 4:
10         return True
11     # if n >= 4, use this function to check
12     for i in range(2, int(math.sqrt(n)) + 1):
13         if n % i == 0:
14             return False
15     # if n isn't multiples of i in [2, sqrt(n)], n is a prime number
16     return True
```

4.2 Docstrings

Ngoài cách chú thích sử dụng kí hiệu #, Python hỗ trợ thêm một kiểu chú thích khác là docstrings. Docstring được dùng nhiều trong các module, hàm, lớp hoặc phương thức. Trong docstring, lập trình viên có thể mô tả chức năng của hàm, lớp, phương thức bằng cách để phần mô tả trong cặp dấu """.

Sử dụng hàm `__doc__` để in ra docstrings của một hàm. Docstrings phải được định nghĩa ở vị trí đầu tiên sau tên hàm / lớp. Nếu nằm ở vị trí khác, nó sẽ không được lưu vào hàm `__doc__`.

Ví dụ: Hàm in ra dòng chữ "Hello world!":

```
1  def hello_world():
2      """
3      This function prints "Hello world!"
4      """
5      print("Hello world!")
6  hello_world()
7  print(hello_world.__doc__)
```

Kết quả cho ra ở Console:

```
1  Hello world!
2
3      This function prints "Hello world!"
4
```

5 Toán tử

Toán tử có thể được định nghĩa là ký hiệu chịu trách nhiệm cho một hoạt động, phép tính cụ thể giữa hai toán hạng. Python cung cấp một loạt các toán tử như sau:

- Toán tử số học
- Toán tử so sánh
- Toán tử gán
- Toán tử logic
- Toán tử biwter
- Toán tử khai thác
- Toán tử xác thực

5.1 Toán tử số học

Tương tự như Java và C, trong Python cung cấp các toán tử số học để tính toán trên các toán hạng, ngoài ra bổ sung thêm các toán tử mới.

Toán tử	Mô tả	Ví dụ
+	Phép cộng giữa hai toán hạng	a = 10, b = 2, c = a + b (c = 12)
-	Phép trừ giữa hai toán hạng	a = 10, b = 2, c = a - b (c = 8)
*	Phép nhân giữa hai toán hạng	a = 10, b = 2, c = a * b (c = 20)
/	Phép chia giữa hai toán hạng	a = 10, b = 2, c = a / b (c = 5)
//	Phép chia làm tròn xuống	a = 10, b = 3, c = a // b (c = 3)
**	Phép luỹ thừa	a = 10, b = 2, c = a ** b (c = 100)
%	Phép chia lấy dư	a = 10, b = 3, c = a % b (c = 1)

Bảng 2: Mô tả các toán tử số học

5.2 Toán tử so sánh

Toán tử so sánh dùng để so sánh giá trị của các toán hạng, kết quả trả về `True` nếu đúng và `False` nếu sai.

Bảng dưới đây mô tả các toán tử so sánh với a = 5, b = 2.

Toán tử	Mô tả	Ví dụ
==	So sánh bằng	a == b (False)
!=	So sánh khác	a != b (True)
>	So sánh lớn hơn	a > b (True)
>=	So sánh lớn hơn hoặc bằng	a >= b (True)
<	So sánh bé hơn	a < b (False)
<=	So sánh bé hơn hoặc bằng	a <= b (False)

Bảng 3: Mô tả các toán tử so sánh

5.3 Toán tử gán

Toán tử gán là toán tử dùng để gán giá trị của một đối tượng cho một đối tượng khác. Trong Python, nó cũng được thể hiện giống như các ngôn ngữ khác.

Bảng dưới đây mô tả các toán tử gán với a = 5, b = 2.

Toán tử	Mô tả	Ví dụ
=	Gán giá trị của toán hạng này cho toán hạng khác	a = b (a = 2)
+=	Cộng toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a += b (a = 7)
-=	Trừ toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a -= b (a = 3)
*=	Nhân toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a *= b (a = 10)
/=	Chia toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a /= b (a = 2.5)
%=	Chia lấy dư toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a %= b (a = 1)
**=	Luỹ thừa toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a **= b (a = 25)
//=	Chia làm tròn xuống toán hạng này cho toán hạng kia rồi gán lại cho chính nó	a //= b (a = 2)

Bảng 4: Mô tả các toán tử gán

5.4 Toán tử logic

Tương tự các ngôn ngữ khác, trong Python có 3 loại toán tử logic: `and`, `or`, `not`.

Bảng dưới đây mô tả các toán tử logic trong Python, với a = `True`, b = `False`.

Toán tử	Mô tả	Ví dụ
and	Nếu cả hai vế của toán tử đều là True thì kết quả là True, ngược lại là False	a and b (False)
or	Nếu một trong hai vế của toán tử là True thì kết quả là True, ngược lại là False	a or b (True)
not	Nếu toán tử là True thì kết quả là False, ngược lại là True	not a (False)

Bảng 5: Mô tả các toán tử logic

5.5 Toán tử biwter

Toán tử biwter thực hiện trên các bit nhị phân của các giá trị.

Bảng dưới đây mô tả các toán tử biwter trong Python, với a = 00001100, b = 00001111.

Toán tử	Ví dụ
|	(a | b) (00001111)
&	(a & b) (00001100)
^	(a ^ b) (00000010)
~	(~a) = (00001101)
«	a«a (49152)
»	a»a (0)

Bảng 6: Mô tả các toán tử biwter

5.6 Toán tử khai thác

Toán tử khai thác thường được dùng để kiểm tra xem 1 đối số có nằm trong 1 tập hợp đối số hay không. Trong Python hỗ trợ hai dạng toán tử khai thác là `in` và `not in`.

Bảng dưới đây mô tả các toán tử khai thác trong Python với a = 6, b = [1, 6, 5].

Toán tử	Mô tả	Ví dụ
in	Kiểm tra đối số có nằm trong tập hợp không	a in b (True)
not in	Kiểm tra đối số có nằm ngoài tập hợp không	a not in b (False)

Bảng 7: Mô tả các toán tử khai thác

5.7 Toán tử xác thực

Toán tử xác thực dùng để kiểm hai giá trị có bằng nhau hay không. Trong Python hỗ trợ 2 dạng toán tử xác thực là `is` và `not is`.

Bảng dưới đây mô tả các toán tử xác thực trong Python với a = 6, b = 7.

Toán tử	Mô tả	Ví dụ
is	Kiểm tra toán hạng này có bằng toán hạng kia hay không	a is b (False)
not is	Kiểm tra toán hạng này có khác toán hạng kia hay không	a not is b (True)

Bảng 8: Mô tả các toán tử xác thực

6 Nhập, xuất dữ liệu

Sử dụng lệnh `print(<string>)` để in chuỗi ra ngoài màn hình.

Ví dụ: Chương trình Hello World!

```
1  print("Hello World!")
```

Kết quả cho ra ở Console:

```
1  Hello World!
```

Sử dụng hàm `input()` để đưa dữ liệu từ bàn phím vào trong Python, cú pháp:

```
<variable_name> = input([<output_string>])
```

Ví dụ: Chương trình nhập vào tên người dùng, in ra kết quả **‘‘Hello + tên người dùng’’**:

```
1  userName = input("Type your name here: ")
2  print("Hello " + userName)
```

Kết quả cho ra ở Console:

```
1  Type your name here: An
2  Hello An
```

Mặc định dữ liệu đưa vào khi sử dụng hàm `input()` là một chuỗi:

```
a = input("a = ")
b = input("b = ")
print('a is',type(a), 'format')
print('b is',type(b), 'format')
c = a + b
print('a + b = ', c)
```

Kết quả cho ra ở Console:

```
a = 15
b = 36
a is <class 'str'> format
b is <class 'str'> format
a + b = 1536
```

Khi nhập vào một số, để thuận tiện cho việc tính toán, ta phải ép kiểu chuỗi về số.

Ví dụ: Chương trình nhập vào năm sinh người dùng, trả về tuổi của người đó tính đến năm 2020:

```
year = int(print("Type your year of birth here: "))
age = 2020 - year
print("Your age is: %d" % age)
```

Kết quả cho ra ở Console:

```
Type your year of birth here: 1999
Your age is: 21
```

7 Câu lệnh rẽ nhánh

Câu lệnh rẽ nhánh là câu lệnh quen thuộc và được sử dụng rất nhiều trong các ngôn ngữ lập trình. Câu lệnh này kiểm tra một điều kiện vào, xử lý dữ liệu với từng trường hợp đúng sai của điều kiện đó.

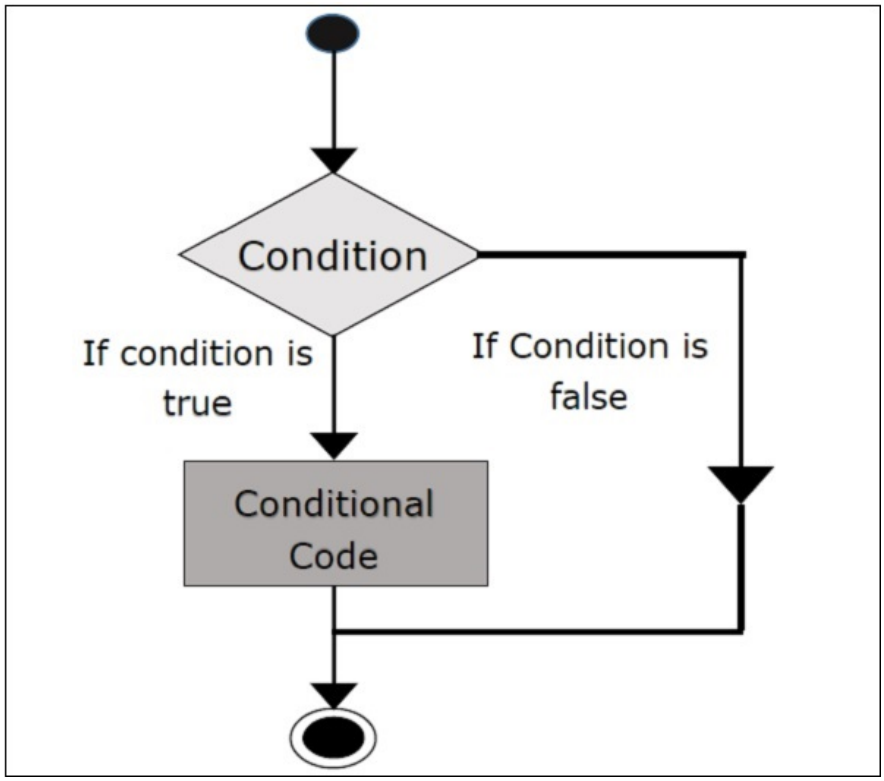
Trong Python, mọi giá trị `<condition>` khác 0 và None được hiểu là `true`.

7.1 Câu lệnh if

Câu lệnh if dùng để kiểm tra một điều kiện với đầu vào là một mệnh đề logic. Nếu điều kiện đúng sẽ thực hiện câu lệnh, nếu sai thì câu lệnh sẽ không được thực hiện.

Cấu trúc của câu lệnh if:

```
if <condition>:
    //conditional code
```



Hình 7.1: Mô tả cách thức hoạt động của câu lệnh if

Ví dụ: Chương trình nhập vào một số nguyên, nếu số đó là số chẵn thì in ra màn hình:

```
number = int(input("Type a number: "))
if number % 2 == 0:
    print("%d is an even number." % number)
```

Kết quả cho ra ở Console:

```
Type a number: 12
12 is an even number.
```

Ví dụ: Kiểm tra tuổi của sinh viên, nếu trên 17 là được chấp nhận:

```
age = int(input("Type your age: "))
if age > 17:
    print("Accepted!")
```

Kết quả cho ra ở Console:

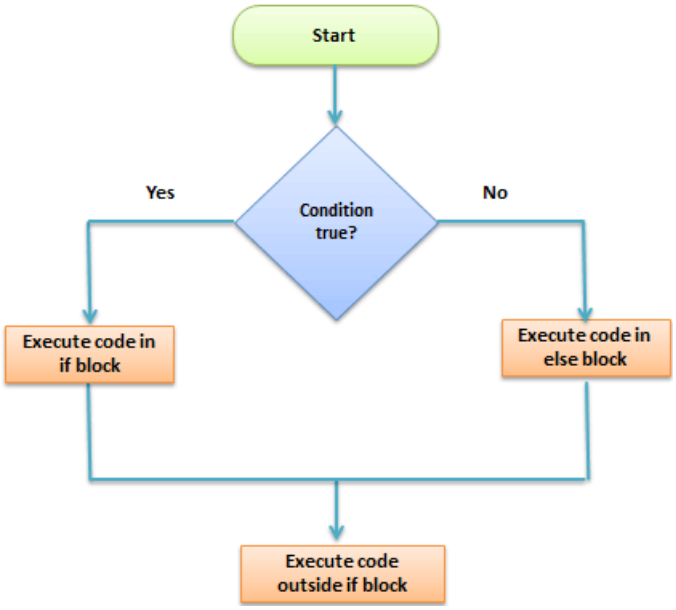
```
Type your age: 16
```

7.2 Câu lệnh if - else

Ở câu lệnh if, chương trình chỉ thực thi khối lệnh nếu điều kiện vào là đúng. Trong nhiều trường hợp, ta cũng cần phải xử lý chương trình khi điều kiện vào là sai. Việc sử dụng từ khoá `else` sẽ giải quyết được trường hợp trên.

Cấu trúc của câu lệnh if - else:

```
if <condition>:
    //if block's code
else:
    //else block's code
```



Hình 7.2: Mô tả cách thức hoạt động của câu lệnh if - else

Ví dụ: Chương trình nhập vào một số nguyên, kiểm tra đó là số chẵn hay lẻ:

```
number = int(input("Type a number: "))
if number % 2 == 0:
    print("%d is an even number." % number)
else:
    print("%d is an odd number." % number)
```

Kết quả cho ra ở Console:

```
Type a number: 11
11 is an odd number.
```

Ví dụ: Chương trình xếp loại, với điểm >= 8 xếp loại giỏi, >= 7 xếp loại khá và còn lại là trung bình:

```
score = float(input("Type your score: "))
if score >= 8:
    print("Excellent")
else:
    if score >= 7:
        print("Good")
    else:
        print("Medium")
```

Kết quả cho ra ở Console:

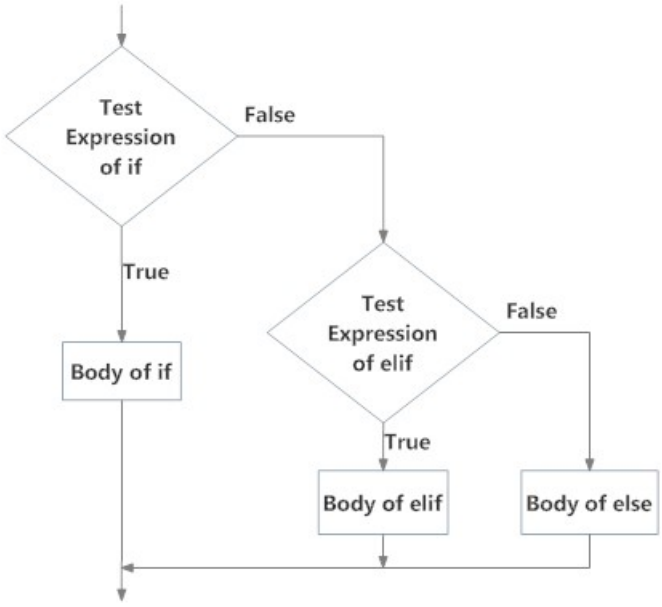
```
Type your score: 6.2
Medium
```

7.3 Câu lệnh if - elif (else if) - else

Trong trường hợp cần xét nhiều hơn 2 điều kiện, thay vì sử dụng `else: if <condition>:`, ta có thể viết gọn thành `elif`.

Cấu trúc của câu lệnh if - elif - else:

```
if <condition>:
    //if block's code
elif <condition>:
    //elif block's code
[elif <condition>:
    //elif block's code]
[...]
else:
    //else block's code
```



Hình 7.3: Mô tả cách thức hoạt động của câu lệnh if - elif - else

Ví dụ: Chương trình nhập vào một số, kiểm tra số đó là số âm, số dương hay số 0:

```
number = float(input("Type a number: "))
if number > 0:
    print("%.3f is a positive number." % number)
elif number < 0:
    print("%.3f is a negative number." % number)
else:
    print("The number is 0.")
```

Kết quả cho ra ở Console:

```
Type a number: -18.25
-18.250 is a negative number.
```

Ví dụ: Chương trình giải phương trình bậc 2:

```
import math

a = int(input("Type your first coefficient: "))
b = int(input("Type your second coefficient: "))
c = int(input("Type your third coefficient: "))
delta = b * b - 4 * a * c
if delta < 0:
    print("The equation has no solution.")
elif delta == 0:
    x = - b / (2 * a)
    print("x = %.2f" % x)
else:
    x1 = (-b - math.sqrt(delta)) / (2 * a)
    x2 = (-b + math.sqrt(delta)) / (2 * a)
    print("x1 = %.2f, x2 = %.2f" % (x1, x2))
```

Kết quả cho ra ở Console với ví dụ giải phương trình $2x^2 + 3x - 1$:

```
Type your first coefficient: 2
Type your second coefficient: 3
Type your third coefficient: 1
x1 = -1.00, x2 = -0.50
```

Ví dụ: Chương trình giải hệ phương trình bậc nhất hai ẩn:

```
a1 = int(input("a1 = "))
b1 = int(input("b1 = "))
c1 = int(input("c1 = "))

a2 = int(input("a2 = "))
b2 = int(input("b2 = "))
c2 = int(input("c2 = "))

d = a1 * b2 - a2 * b1
dx = c1 * b2 - c2 * b1
dy = a1 * c2 - a2 * c1

if d == 0 and dx == dy and dx == 0:
    print("This equations has an infinite number of solutions.")
elif d == 0 and dx != dy:
    print("This equations has no solution.")
else:
    x = dx / d
    y = dy / d
    print("x = %.2f, y = %.2f" % (x, y))
```

Kết quả cho ra ở Console với ví dụ giải hệ phương trình $\begin{cases} x - y = 1 \\ x + y = 3 \end{cases}$:

```
a1 = 1
b1 = -1
c1 = 1
a2 = 1
b2 = 1
c2 = 3
x = 2.00, y = 1.00
```

7.4 Switch - case

Python không có cấu trúc switch case đơn giản. Thay vào đó, chúng ta có thể sử dụng một dictionary để ánh xạ đến các case.

Cấu trúc một dictionary:

```
<name of dictionary> = {
    <key 1>: <value 1>,
    [<key 2>: <value 2>,]
    [<key 3>: <value 3>,]
    [...]
}
```

Ta sử dụng hàm `get(<key>, <value if key not found>)` để lấy value từ một key đã biết trước.

Ví dụ: Chương trình nhập vào tháng dạng số nguyên, xuất ra tên tháng:

```
def get_month_by_number(month):
    switch = {
        1: 'January',
        2: 'February',
        3: 'March',
        4: 'April',
        5: 'May',
        6: 'June',
        7: 'July',
        8: 'August',
        9: 'September',
        10: 'October',
        11: 'November',
        12: 'December'
    }
    return switch.get(month, 'Month must be less than 13 and greater than 0.')

month = int(input("Type your month: "))
print(get_month_by_number(month))
```

Kết quả cho ra ở Console:

```
Type your month: 8
August
```

Ví dụ: Chương trình nhập vào tháng dạng số nguyên, xuất số ngày trong tháng:

```
def get_month_by_number(month):
    switch = {
        1: 'January', 2: 'February', 3: 'March', 4: 'April',
        5: 'May', 6: 'June', 7: 'July', 8: 'August',
        9: 'September', 10: 'October', 11: 'November', 12: 'December'
    }
def get_day_of_february():
    year = int(input("Type a year: "))
    if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
        return 29
    return 28
def get_day_of_month(month):
    switch = {
        1: 31, 2: get_day_of_february(),
        3: 31, 4: 30,
        5: 31, 6: 30,
        7: 31, 8: 31,
        9: 30, 10: 31,
        11: 30, 12: 31
    }
    return switch.get(month, 'Month must be less than 13 and greater than 0.')
month = int(input("Type a month: "))
print(get_month_by_number(month), 'has', get_day_of_month(month), 'days.')
```

Kết quả cho ra ở Console:

```
Type a month: 2
Type a year: 2003
February has 28 days.
```

8 Vòng lặp

Sử dụng vòng lặp khi ta muốn lặp đi lặp lại một đoạn code nhiều lần. Có hai loại vòng lặp:

- Vòng lặp xác định (definite loop): việc lặp sẽ được thực hiện với số lần biết trước.
- Vòng lặp không xác định (indefinite loop): việc lặp sẽ được thực hiện với số lần không biết trước.

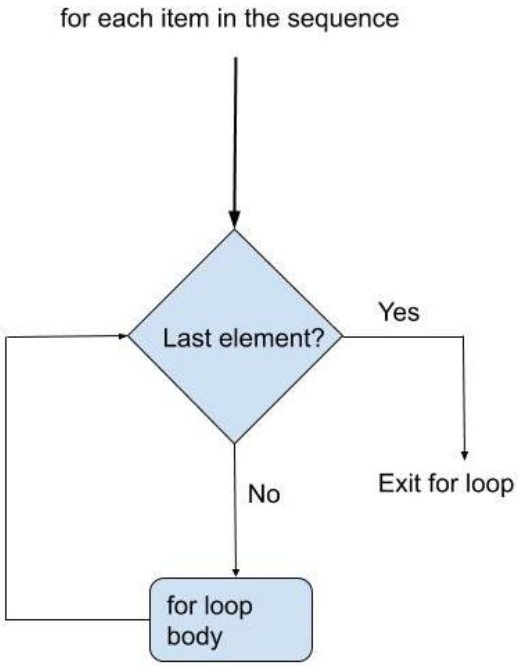
8.1 Vòng lặp for

Sử dụng vòng lặp for để lặp đi lặp loại một khối lệnh khi biết số lần lặp cụ thể. Cấu trúc vòng lặp for:

```
for <iterator_var> in <sequence>:
    // for block's code
```

Trong đó:

- `sequence` là chuỗi hoặc collection ... cần lặp.
- `iterator_var` là biến lặp, nhận giá trị của từng phần tử bên trong `sequence` ở mỗi lần lặp.



Hình 8.1: Mô tả cách thức hoạt động của vòng lặp for

Do vòng lặp for của Python chỉ duyệt qua các phần tử có trong `sequence` nên đối với các bài toán không thao tác trên chuỗi hoặc các collections, ta phải dùng thêm hàm `range()` để trả về một danh sách các số hạng theo thứ tự cần lặp.

Cấu trúc	Giá trị	Ví dụ	Kết quả
range(n)	[0, 1, 2,... n - 1]	range(3)	[0, 1, 2]
range(x, y)	[x, x + 1, x + 2,... y - 1]	range(2, 5)	[2, 3, 4]
range(x, y, n)	[x, x + n, x + 2n,.... y'] (y' <= y - 1)	range(3, 8, 2)	[3, 5, 7]

Bảng 9: Mô tả kết quả hàm range()

Ví dụ: Chương trình nhập vào một số, tính giai thừa của số đó:

```
n = int(input("Type a number: "))
factorial = 1
for i in range(2, n + 1):
    factorial *= i
print("Factorial of %d: %d" % (n, factorial))
```

Kết quả cho ra ở Console:

```
Type a number: 6
Factorial of 6: 720
```

Ví dụ: Chương trình nhập vào một số, tính tổng số chẵn trong đoạn [2, n]:

```
n = int(input("Type a number: "))
sum = 0
for i in range(2, n + 1, 2):
    sum += i
print("Sum of even numbers in [2, %d]: %d" % (n, sum))
```

Kết quả cho ra ở Console:

```
Type a number: 6
Sum of even numbers in [2, 6]: 12
```

Ví dụ: Chương trình nhập vào một chuỗi, in ra từng ký tự trong chuỗi:

```
string = input("Type a sentence: ")
for char in string:
    print(char)
```

Kết quả cho ra ở Console:

```
Type a sentence: Python
P
y
t
t
o
n
```

Ví dụ: Chương trình tính tổng, tích một dãy số sử dụng vòng lặp for duyệt qua các phần tử trong mảng:

```
array = [1, 3, 5, 8, 9]
sum = 0
product = 1
for element in array:
    sum += element
    product *= element
print("Sum of array: ", sum)
print("Product of array: ", product)
```

Kết quả cho ra ở Console:

```
Sum of array: 26
Produce of array: 1080
```

Chúng ta cũng có thể duyệt mảng bằng các sử dụng chỉ số của từng phần tử. Sử dụng hàm `len(obj)` để trả về độ dài của mảng.

Ví dụ: Chương trình nhập vào một dãy số, tính tổng các số lẻ trong dãy:

```
length = int(input("Type length of array: "))
array = []
sum = 0
for i in range(length):
    array.append(int(input("Type your number %d: " % (i + 1))))
print("Iterating over array by index:")
for i in range(len(array)):
    print("Index %d: %d" %(i, array[i]))
    if array[i] % 2 == 1:
        sum += array[i]
print("Sum of odd numbers in array: ", sum)
```

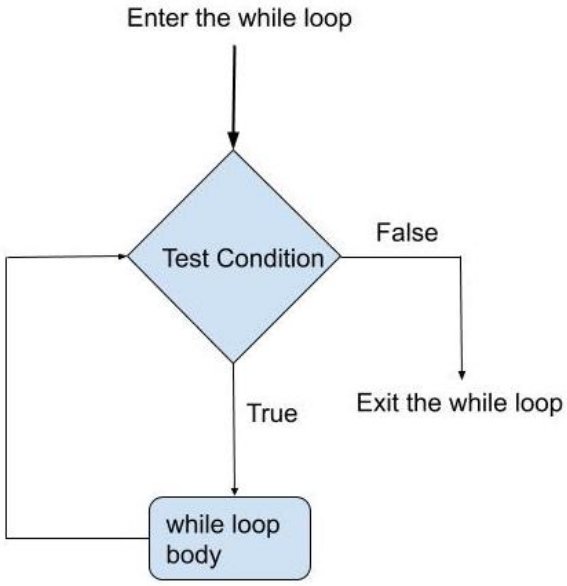
Kết quả cho ra ở Console:

```
Type length of array: 3
Type your number 1: 1
Type your number 2: 4
Type your number 3: 9
Iterating over array by index:
Index 0: 1
Index 1: 4
Index 2: 9
Sum of odd numbers in array: 10
```

8.2 Vòng lặp while

Sử dụng vòng lặp while để lặp đi lặp loại một khối lệnh khi số lần lặp phụ thuộc vào điều kiện lặp. Cấu trúc của vòng lặp while:

```
while <condition>:
    //while block's code
[else:
    //else block's code]
```



Hình 8.2: Mô tả cách thức hoạt động của vòng lặp while

Ví dụ: Chương trình nhập vào một số, in ra các bội nhỏ hơn 100 của số đó :

```
n = int(input("Type a number: "))
multiple = n
i = 2
while multiple < 100:
    print(multiple)
    multiple = n
    multiple *= i
    i += 1
```

Kết quả cho ra ở Console:

```
Type a number: 28
28
56
84
```

Ví dụ: Chương trình nhập vào hai số tự nhiên, tính ước chung lớn nhất của hai số đó:

```
print("GCD(a, b)")
print("a must be greater than b")
a = int(input("a = "))
b = int(input("b = "))
while a % b != 0:
    temp = a
    a = b
    b = temp % b
else:
    print("GCD:", b)
```

Kết quả cho ra ở Console:

```
GCD(a, b)
a must be greater than b
a = 126
b = 12
GCD: 6
```

Ví dụ: Chương trình nhập vào số thập phân, in ra màn hình số nhị phân của số đó:

```
n = int(input("Type your decimal number: "))
binary = ""
while n > 0:
    binary += "%s" % (n % 2) # convert surplus of n / 2 to string and add to binary
    n = int(n / 2)
print("Binary:", binary[::-1]) # reverse binary string
```

Kết quả cho ra ở Console:

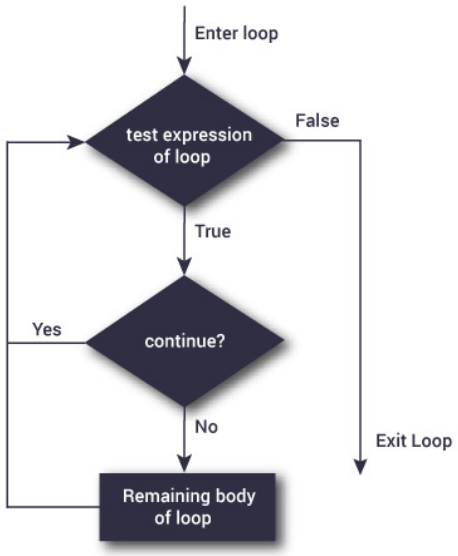
```
Type your decimal number: 13
Binary: 1101
```

8.3 Từ khoá continue, break, pass

Trong vòng lặp, ta cần chú ý ba từ khoá quan trọng là `continue`, `break` và `pass`.

8.3.1 Continue

Sử dụng từ khoá `continue` khi người lập trình muốn bỏ qua giá trị hiện tại, tiếp tục thực hiện vòng lặp với giá trị tiếp theo.



Hình 8.3: Mô tả cách thức hoạt động của từ khoá continue

Ví dụ: Tính căn bậc 2 của các phần tử trong dãy cho trước, bỏ qua các phần tử âm:

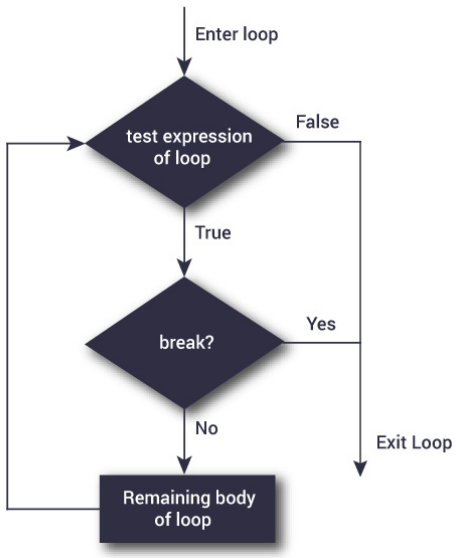
```
1  import math
2  array = [1, 4, -5, 25, -8, 16]
3  for i in array:
4      #skip element which is less than 0
5      if i < 0:
6          continue
7      print(math.sqrt(i))
```

Kết quả cho ra ở Console:

```
1  1.0
2  2.0
3  5.0
4  4.0
```

8.3.2 Break

Sử dụng từ khoá `break` khi người lập trình muốn dừng và thoát khỏi vòng lặp ngay lập tức.



Hình 8.4: Mô tả cách thức hoạt động của từ khoá break

Ví dụ: Duyệt một chuỗi cho trước, lưu từng ký tự của chuỗi đó vào chuỗi mới, gặp kí tự 'E' thì ngừng lại:

```
string = "contentE0xp"
result = ""
for i in string:
    if i == 'E':
        break
    result += i
print(result)
```

Kết quả cho ra ở Console:

```
content
```

8.3.3 Pass

Khi người lập trình viết một vòng lặp hoặc một hàm, nhưng chưa có ý tưởng gì cho nó, thì có thể sử dụng từ khoá `pass` để thêm vào như là một câu lệnh giữ chỗ.

Nói một cách khác, `pass` là một lệnh trống, không làm gì cả. Python quy định các hàm và vòng lặp không được để trống. Vì thế, sử dụng từ khoá `pass` như là một câu lệnh để hợp thức hoá quy định trên.

Ví dụ về từ khoá `pass`:

```
1  array = ['p', 'a', 's', 's']
2  for i in array:
3      pass
4  print("Pass successfully!")
```

Kết quả cho ra ở Console:

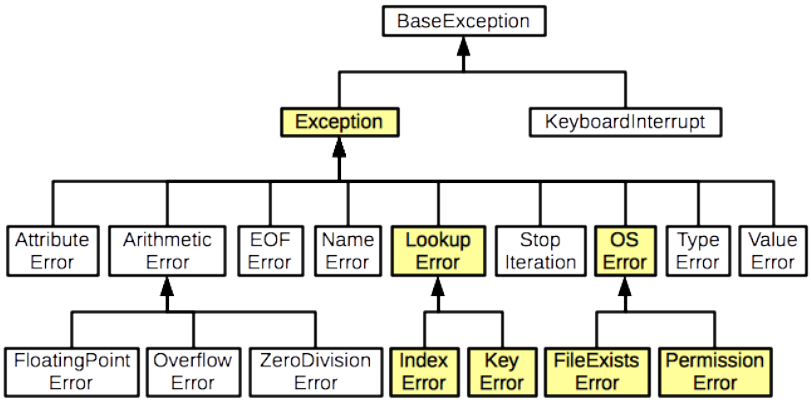
```
1  Pass successfully!
```

9 Ngoại lệ và xử lý ngoại lệ

9.1 Ngoại lệ là gì? Các loại ngoại lệ trong Python

Ngoại lệ (exception) là lỗi xuất hiện làm dừng chương trình đang thực thi. Trong Python hỗ trợ các lập trình viên xử lý ngoại lệ để duy trì hoạt động cho chương trình.

Các Exception sẵn có của Python thông thường được bắt nguồn từ BaseException. Trong khi đó các exception của lập trình viên nên thừa kế từ lớp Exception hoặc từ các lớp con của nó.



Hình 9.1: Sơ đồ mô tả các loại Exception trong Python

Tên ngoại lệ	Chú thích
Exception	Đây là lớp cơ sở cho tất cả các exception, nó sẽ xuất hiện khi có bất cứ một lỗi nào xảy ra.
StopIteration	Xuất hiện khi phương thức next() của interator không trỏ đến một đối tượng nào.
SystemExit	Xuất hiện khi dùng phương thức sys.exit().
ArithmeticError	Xuất hiện khi có lỗi tính toán giữa các số với nhau.
OverflowError	Xuất hiện khi thực hiện tính toán và giá trị của nó vượt quá ngưỡng giới hạn cho phép của kiểu dữ liệu.
FloatingPointError	Xuất hiện khi tính toán float thất bại.
ZeroDivisonError	Xuất hiện khi thực hiện phép chia cho 0.
AssertionError	Xuất hiện trong trường hợp lệnh assert thất bại.
AttributeError	Xuất hiện khi không tồn tại thuộc tính này, hoặc thiếu tham số truyền vào nó.
EOFError	Xuất hiện khi không có dữ liệu từ hàm input() hoặc cuối file.
ImportError	Xuất hiện khi lệnh import thất bại.
KeyboardInterrupt	Xuất hiện khi ngắt trình biên dịch.
LookupError	Lớp cơ sở cho tất cả các lỗi về lookup.
IndexError	Xuất hiện khi index không tồn tại trong list, string,...
KeyError	Xuất hiện khi key không tồn tại trong dictionary.
NameError	Xuất hiện khi một biến không tồn tại trong phạm vi bạn gọi nó.
EnvironmentError	Xuất hiện khi có bất kỳ một lỗi nào ngoài phạm vi của Python.
IOError	Xuất hiện khi xử dụng input/ output thất bại, hoặc mở file không thành công.
OSError	Xuất hiện khi có lỗi từ hệ điều hành.
SyntaxError	Xuất hiện khi chương trình có lỗi cú pháp.
IndentationError	Xuất hiện khi bạn thụt dòng không đúng.
SystemError	Xuất hiện khi trình biên dịch có vấn đề nhưng mà nó lại không tự động exit.
SystemExit	Xuất hiện khi trình biên dịch được thoát bởi sys.exit().
TypeError	Xuất hiện khi thực thi toán tử hoặc hàm mà kiểu dữ liệu bị sai so với kiểu dữ liệu đã định nghĩa ban đầu.
ValueError	Xuất hiện khi chúng ta build 1 function mà kiểu dữ liệu đúng nhưng khi chúng ta thiết lập ở tham số là khác so với khi truyền vào.
RuntimeError	Xuất hiện khi lỗi được sinh ra không thuộc một danh mục nào.
NotImplementedError	Xuất hiện khi một phương thức trừu tượng cần được thực hiện trong lớp kế thừa chứ không phải là lớp thực thi.
UnboundLocalError	Xuất hiện khi chúng ta cố tình truy cập vào một biến trong hàm hoặc phương thức, nhưng không thiết lập giá trị cho nó.

Bảng 10: Các ngoại lệ trong Python

Ví dụ: Duyệt qua một list số nguyên, in ra giá trị nghịch đảo của từng phần tử trong list:

```
list = [4, 5, 0, 7]
for i in list:
    print(1 / i)
```

Kết quả cho ra ở Console:

```
0.25
0.2
Traceback (most recent call last):
File "Exception.py", line 3, in <module>
    print(1/i)
ZeroDivisionError: division by zero
```

Khi duyệt đến phần tử `array[2] = 0`, việc tính toán giá trị $\frac{1}{i}$ gặp lỗi chia cho 0. Do đó, chương trình dừng lại ngay lập tức và in lỗi ra ngoài Console, không tiếp tục duyệt các phần tử còn lại của list.

Ví dụ: Duyệt qua một list chứa nhiều kiểu dữ liệu, in ra tổng giá trị các số trong list:

```
1  list = [1, 2.3, 7.6, "Ex"]
2  sum = 0.0
3  for i in list:
4      sum += i
5  print(sum)
```

Kết quả cho ra ở Console:

```
1  Traceback (most recent call last):
2  File "Exception.py", line 4, in <module>
3      sum += i
4  TypeError: unsupported operand type(s) for +=: 'float' and 'str'
```

Khi duyệt đến phần tử `"Ex"`, Python không thể chuyển kiểu dữ liệu chuỗi sang kiểu số và cộng vào biến `sum`, từ đó xảy ra lỗi `TypeError`.

9.2 Xử lý ngoại lệ với try - except - finally

Ta có thể xử lý ngoại lệ bằng cách đặt phần code có khả năng xảy ra lỗi vào trong câu lệnh `try - except`. Lập trình viên có thể in các mô tả về ngoại lệ xảy ra trong đoạn code ra màn hình, hoặc có thể bỏ qua chúng.

Cấu trúc câu lệnh try - catch - finally:

```
try:
    # code
except <exception_name> as <variable_name>:
    # handle code
[finally:
    # code]
```

Cách thức hoạt động của try - catch - finally diễn ra như sau: Khi các câu lệnh trong try được phát hiện xảy ra ngoại lệ, chương trình sẽ chuyển xuống phần except và thực hiện các câu lệnh xử lý ngoại lệ trong nó. Sau khi xử lý xong ngoại lệ, tiếp tục thực hiện các câu lệnh trong finally (nếu có). Trường hợp trong try không có ngoại lệ, chương trình vẫn thực hiện các câu lệnh trong finally.

Khác với Java, lập trình viên có thể khai báo nhiều ngoại lệ có thể xảy ra trong chương trình mà không cần phải tuân thủ quy tắc từ cụ thể đến chung nhất. Python sẽ tự tìm kiếm một exception phù hợp trong các exception mà lập trình viên đã khai báo mà không quan tâm đến các ngoại lệ còn lại. Thứ tự xét các exception là từ trên xuống.

Ví dụ: In các giá trị căn bậc 2 của từng phần tử trong một list, bỏ qua các giá trị âm:

```
from math import sqrt
list = [4, 16, -7, 25]
for i in list:
    try:
        print(sqrt(i))
    except Exception as e:
        pass # skip if it's a negative number
    finally:
        print("Done!")
```

Kết quả cho ra ở Console:

```
2.0
Done!
4.0
Done!
Done!
5.0
Done!
```

Khi xét đến phần tử -7, do không thể lấy căn bậc 2 của một số âm nên chương trình xét đoạn lệnh trong phần `except`. Từ khoá `pass` không thực hiện câu lệnh nào cả, chương trình chạy đến phần `finally`, in dòng chữ `"Done!"` ra ngoài màn hình rồi tiếp tục xét các giá trị khác trong list.

Ví dụ: Duyệt và in ra các phần tử trong một list bằng chỉ số:

```
ls = [1, 2, 0.2, 3]
for i in range(len(ls) + 1):
    try:
        print(ls[i])
    except ZeroDivisionError as e:
        print(e)
    except LookupError as ex:
        print(ex.__doc__) # output: Sequence index out of range.
    except IndexError as exc:
        print(ex) # output: list index out of range.
```

Kết quả cho ra ở Console:

```
1
2
0.2
3
Sequence index out of range.
```

Do chỉ có 4 phần tử trong list `ls` nên các phần tử trong list được đánh số từ 0 đến 3. Hàm `range(len(ls) + 1)` trả về một mảng số nguyên `[0, 1, 2, 3, 4]`. Khi duyệt đến phần tử có chỉ số 4 trong list, chương trình phát sinh ra lỗi và tìm đến exception phù hợp đầu tiên mà người lập trình đã khai báo là `LookupError` rồi thực hiện câu lệnh `print(ex.__doc__)`, bỏ qua exception `IndexError` mặc dù ngoại lệ này là cụ thể nhất cho lỗi phát sinh.

9.3 Ngoại lệ do người dùng tự định nghĩa

Ngoài các ngoại lệ đã mô tả ở phần trên, lập trình viên có thể tự định nghĩa một ngoại lệ riêng sao cho phù hợp với từng chương trình cụ thể bằng từ khoá `raise`.

Ví dụ: Chương trình tính tiền thuê phòng khách sạn, mỗi giờ là $5, số giờ không được dưới 1 và trên 12.:

```
hour = [1, 5, 7, 0.5, 11]
for i in hour:
    if i < 1 or i > 12:
        raise Exception("Hour must be greater than 1 and less than 12")
    print("Price: %d$" % (i * 5))
```

Kết quả cho ra ở Console:

```
Traceback (most recent call last):
File "Exception.py", line 4, in <module>
    raise Exception("Hour must be greater than 1 and less than 12")
Exception: Hour must be greater than 1 and less than 12
Price: 5$
Price: 25$
Price: 35$
```