## Kiểu dữ liệu number trong Python

Trong bài này mình sẽ giới thiệu kiểu dữ liệu Number trong Python, tìm hiểu về các loại number khác nhau được sử dụng trong Python, cách chuyển đổi từ loại dữ liệu này sang loại dữ liệu khác và các hoạt động toán học được hỗ trợ trong Python.

Nói đến Number thì chắc ai cũng biết đây là kiểu số. Tuy nhiên, chúng ta có rất nhiều loại số khác nhau như số nguyên (int), số thực (float), số phức (complex number). Vì vậy bạn phải nắm vững cách chuyển đổi giữa chúng.

**Mục lục**

* [1. Number trong Python là gì?](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-0)
* [2. Số thập phân trong Python](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-1)
  + [Chúng ta đang sai ở đâu?](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h3-0)
* [3. Khi nào sử dụng Decimal thay vì float](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-2)
* [4. Phân số](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-3)
* [5. Các hàm toán học](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-4)
* [6. Lời kết](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-5)

### Number trong Python là gì?

Number là một nhóm dữ liệu thường dùng trong Python, nó dùng để lưu trữ hầu hết các kiểu về số như số thực, số phức, số nguyên.

Python hỗ trợ số nguyên, số thực và số phức. Chúng được định nghĩa trong các lớp int, float và lớp complex của Python.

Integer và float được phân biệt bằng dấu chấm động (.). Nếu một số có dấu chấm động là kiểu float, ngược lại là kiểu int.

**Ví dụ**: Số 5 thuộc kiểu int, trong khi 5.0 thuộc kiểu float.

Số phức được viết dưới dạng x + yj, trong đó **x** là phần thực và y là phần ảo.

Chúng ta có thể sử dụng hàm type() để biết một biến đang thuộc lớp nào, và hàm isinstance() để kiểm tra xem nó có thuộc về một lớp cụ thể nào đó không.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | # Output: <class 'int'>  print(type(a))    # Output: <class 'float'>  print(type(5.0))    # Output: (8+3j)  c = 5 + 3j  print(c + 3)    # Output: True  print(isinstance(c, complex)) |

Trong khi số nguyên có thể có độ dài bất kỳ, số dấu phẩy động chỉ chính xác tối đa 15 vị trí thập phân (vị trí thứ 16 không chính xác).

Các số chúng ta xử dụng hàng ngày là hệ thống số thập phân (cơ sở 10). Nhưng các lập trình viên máy tính (nói chung là lập trình viên nhúng) cần phải làm việc với các hệ thống số nhị phân (cơ sở 2), hệ thập lục phân (cơ sở 16) và số bát phân (cơ sở 8).

Trong Python, chúng ta có thể biểu diễn các số này bằng cách đặt một tiền tố thích hợp trước số đó.

Bảng sau liệt kê các tiền tố này.

|  |  |
| --- | --- |
| **Number System** | **Prefix** |
| Binary | '0b' or '0B' |
| Octal | '0o' or '0O' |
| Hexadecimal | '0x' or '0X' |

Dưới đây là một vài ví dụ cách sử dụng các tiền tố prefix:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # Output: 107  print(0b1101011)    # Output: 253 (251 + 2)  print(0xFB + 0b10)    # Output: 13  print(0o15) |

Khi bạn chạy chương trình này thì sẽ cho kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3 | 107  253  13 |

Chúng ta có thể chuyển đổi một từ cơ số này sang cơ số khác. Điều này còn được gọi là chuyển đổi cơ số.

**Ví dụ**: Các phép toán như phép cộng, phép trừ số nguyên sẽ tự động chuyển đổi cơ số sang kiểu float nếu một trong các toán hạng là float.

|  |  |
| --- | --- |
| 1  2 | >>> 1 + 2.0  3.0 |

Như ví dụ trên bạn thấy 1 là kiểu số nguyên, 2.0 là kiểu số động và khi thực hiện phép toán thì kết quả sẽ mang kiểu số động.

Bạn cũng có thể sử dụng các hàm dựng sẵn như int (), float () và comlex () để chuyển đổi các kiểu dữ liệu một cách rõ ràng. Các hàm này thậm chí có thể chuyển đổi từ các chuỗi sang số và số sang chuỗi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> int(2.3)  2  >>> int(-2.8)  -2  >>> float(5)  5.0  >>> complex('3+5j')  (3+5j) |

### Số thập phân trong Python

Lớp built-in trong Python có thể thực hiện một vài phép toán khiến chúng ta ngạc nhiên. Chúng ta đều biết rằng tổng của 1.1 và 2.2 là 3.3, tuy nhiên khi sử dụng toán tử so sánh bằng (==) thì điều đó là không đúng trong Python.

|  |  |
| --- | --- |
| 1  2 | >>> (1.1 + 2.2) == 3.3  False |

#### Chúng ta đang sai ở đâu?

Nó chỉ ra rằng các số dấu phẩy động được triển khai trong phần cứng máy tính dưới dạng phân số nhị phân, vì máy tính chỉ hiểu nhị phân (0 và 1). Vì lý do này, hầu hết các phân số thập phân mà chúng ta biết không thể được lưu trữ chính xác trong máy tính.

**Hãy lấy một ví dụ.** Chúng ta không thể biểu thị phân số 1/3 dưới dạng số thập phân. Điều này sẽ cung cấp cho 0.33333333 ... dài vô hạn và chúng ta chỉ có thể ước chừng nó.

Hóa ra phân số thập phân 0,1 sẽ dẫn đến phân số nhị phân dài vô hạn 0,000110011001100110011 ... và máy tính của chúng ta chỉ lưu trữ một số hữu hạn của nó.

Điều này sẽ chỉ xấp xỉ 0,1 nhưng không bao giờ bằng nhau. Do đó, đó là giới hạn của phần cứng máy tính của chúng tôi và không phải là lỗi trong Python.

|  |  |
| --- | --- |
| 1  2 | >>> 1.1 + 2.2  3.3000000000000003 |

Để khắc phục vấn đề này, chúng ta có thể sử dụng mô-đun **decimal**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import decimal    # Output: 0.1  print(0.1)    # Output: Decimal('0.1000000000000000055511151231257827021181583404541015625')  print(decimal.Decimal(0.1)) |

Mô-đun này được sử dụng khi chúng ta muốn thực hiện các phép tính thập phân như đã học ở trường. Chúng ta biết 25,50 kg chính xác hơn 25,5 kg vì nó có hai chữ số thập phân đáng kể so với một.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from decimal import Decimal as D  # Output: Decimal('3.3')  print(D('1.1') + D('2.2'))    # Output: Decimal('3.000')  print(D('1.2') \* D('2.50')) |

Chúng ta có thể hỏi tại sao không thực hiện Decimal mỗi lần, thay vì float? Lý do chính là hiệu quả, float luôn luôn nhanh hơn.

### Khi nào sử dụng Decimal thay vì float

Chúng ta thường sử dụng Decimal trong các trường hợp sau.

* Khi đang làm cho các ứng dụng tài chính cần đại diện thập phân chính xác.
* Khi muốn kiểm soát mức độ chính xác cần thiết.
* Khi muốn thực hiện khái niệm về số thập phân có ý nghĩa.
* Khi muốn các hoạt động được thực hiện như chúng ta đã làm ở trường

### Phân số

Python cung cấp các hoạt động liên quan đến số phân số thông qua mô-đun **Fraction** của nó. Chúng ta có thể tạo các đối tượng Fraction theo nhiều cách khác nhau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import fractions    # Output: 3/2  print(fractions.Fraction(1.5))    # Output: 5  print(fractions.Fraction(5))    # Output: 1/3  print(fractions.Fraction(1,3)) |

Trong khi tạo phân số từ kiểu float chúng ta có thể nhận được một số kết quả bất thường. Điều này là do biểu diễn số dấu phẩy động nhị phân không hoàn hảo như đã thảo luận trong phần trước.

May mắn thay, phân số cho phép chúng ta khởi tạo bằng chuỗi, đây là các tùy chọn ưa thích khi sử dụng số thập phân.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import fractions    # As float  # Output: 2476979795053773/2251799813685248  print(fractions.Fraction(1.1))    # As string  # Output: 11/10  print(fractions.Fraction('1.1')) |

Kiểu dữ liệu này hỗ trợ tất cả các toán tử cơ bản. Dưới đây là một vài ví dụ.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from fractions import Fraction as F    # Output: 2/3  print(F(1,3) + F(1,3))    # Output: 6/5  print(1 / F(5,6))    # Output: False  print(F(-3,10) > 0)    # Output: True  print(F(-3,10) < 0) |

### Các hàm toán học

Python cung cấp các mô-đun như **math**và **random** để thực hiện các phép toán khác nhau như lượng giác, logarit, xác suất và thống kê, v.v.

**Module math**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import math    # Output: 3.141592653589793  print(math.pi)    # Output: -1.0  print(math.cos(math.pi))    # Output: 22026.465794806718  print(math.exp(10))    # Output: 3.0  print(math.log10(1000))    # Output: 1.1752011936438014  print(math.sinh(1))    # Output: 720  print(math.factorial(6)) |

**Module random**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | import random    # Output: 16  print(random.randrange(10,20))    x = ['a', 'b', 'c', 'd', 'e']    # Get random choice  print(random.choice(x))    # Shuffle x  random.shuffle(x)    # Print the shuffled x  print(x)    # Print random element  print(random.random()) |

### Lời kết

Như vậy là mình đã giới thiệu xong toàn bộ kiến thức cơ bản về xử lý number trong Python, có rất nhiều hàm bổ trợ cho number và mình sẽ trình bày nó ở một bài khác. Chúc bạn học tốt.

## Kiểu dữ liệu String trong Python

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu String trong Python, đây là kiểu chuỗi, được dùng để lưu trữ các chuỗi kí tự trong Python.

Mình sẽ hướng dẫn các bạn cách tạo, định dạng, sửa đổi và xóa các chuỗi trong Python. Ngoài ra, bạn sẽ được giới thiệu các thao tác và chức năng liên quan đến chuỗi.

**Mục lục**

* [1. Kiểu dữ liệu string trong Python là gì?](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-0)
  + [Cú pháp tạo string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-0)
  + [Truy cập các ký tự của chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-1)
  + [Thay đổi và xóa chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-2)
* [2. Các phép toán với chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-1)
  + [Nối hai hoặc nhiều chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-3)
  + [Lặp qua từng ký tự trong chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-4)
  + [Kiểm tra chuỗi con trong chuỗi cha](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-5)
* [3. Các định dạng chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-2)
  + [Ký tự nháy đơn và nháy kép](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-6)
  + [Ký tự thoát](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-7)
* [4. Lời kết](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-3)

### Kiểu dữ liệu string trong Python là gì?

Một chuỗi là một dãy các ký tự được sắp xếp với nhau, một ký tự đơn giản là một biểu tượng và máy tính có thể đọc hiểu được ký tự đó thông qua mã máy ASCII.

Máy tính không hiểu ý nghĩa của các ký tự như con người mà chúng sẽ hiểu sẽ chuyển sang kiểu nhị phân (binary). Mặc dù bạn có thể thấy các ký tự trên màn hình của mình, nhưng bên trong nó được lưu trữ và thao tác dưới dạng kết hợp của 0 và 1.

Việc chuyển đổi từ ký tự thành số được gọi là mã hóa và quá trình ngược lại gọi là giải mã. ASCII và Unicode là một số dạng mã hóa được sử dụng phổ biến.

#### Cú pháp tạo string trong Python

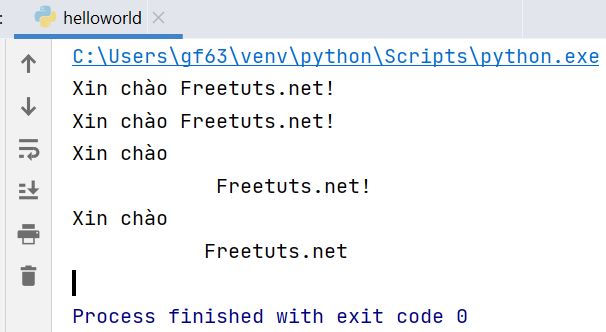
Chuỗi có thể được tạo bằng cách đặt các ký tự bên trong cặp nháy đơn hoặc nháy kép. Nếu bạn muốn viết chuỗi trên nhiều dòng thì phải sử dụng ba dấu nháy, cách này thường được dùng để tạo **docstring**.

**Ví dụ tạo chuỗi**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # Tất cả những ví dụ dưới đây là tương đương với nhau.    # Dùng nháy đơn  my\_string = 'Xin chào Freetuts.net!'  print(my\_string)    # Dùng nháy kép  my\_string = "Xin chào Freetuts.net!"  print(my\_string)    # Dùng ba nháy đơn trên nhiều dòng  my\_string = '''Xin chào             Freetuts.net!'''  print(my\_string)    # Dùng ba nháy kép trên nhiều dòng  my\_string = """Xin chào             Freetuts.net"""  print(my\_string) |

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Xin chào Freetuts.net!  Xin chào Freetuts.net!  Xin chào              Freetuts.net!  Xin chào             Freetuts.net |



Ví dụ trên là một vài cách tạo chuỗi thông thường với nháy đơn và nháy kép,

#### Truy cập các ký tự của chuỗi string trong Python

String được xem như là một **list** các ký tự nên bạn có thể truy cập đến các ký tự thông qua chỉ mục index của nó.

Chỉ mục bắt đầu từ 0, và nếu bạn cố gắng truy cập một ký tự nằm ngoài phạm vi chỉ mục thì sẽ xuất hiện lỗi **IndexError**.

Chỉ mục phải là một số nguyên, không thể sử dụng float hoặc các loại khác vì điều này sẽ dẫn đến **TypeError**.

Chỉ số -1 đề cập đến phần tử cuối cùng, -2 cho phần tử cuối cùng thứ hai, .. (đây là cách tính ngược của chuỗi).

Chúng ta có thể truy cập một loạt các phần tử trong một chuỗi bằng cách sử dụng toán tử slicing (dấu hai chấm).

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | str = 'freetuts.net'  print('str = ', str)    # Ký tự đầu tiên  print('str[0] = ', str[0])    # Ký tự cuối cùng  print('str[-1] = ', str[-1])    #Lấy các kí tự thứ 2 đến thứ 6  print('str[1:5] = ', str[1:5])    # Lấy các kí tự từ thứ 6 đến ky tự thứ 3 tính từ cuối chuỗi  print('str[5:-2] = ', str[5:-2]) |

Chạy chương trình này sẽ cho kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | str =  freetuts.net  str[0] =  f  str[-1] =  t  str[1:5] =  reet  str[5:-2] =  uts.n |

Nhưng nếu bạn cố gắng truy cập đến một phần tử không tồn tại, thì sẽ báo lỗi như ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # index must be in range  >>> my\_string[15]  ...  IndexError: string index out of range    # index must be an integer  >>> my\_string[1.5]  ...  TypeError: string indices must be integers |

### Thay đổi và xóa chuỗi string trong Python

Để thay đổi chuỗi thì bạn có thể sử dụng chỉ mục index ... điều này là **sai**nhé các bạn. Chuỗi là bất biến nên bạn không thể thay đổi chuỗi, chỉ được phép gán một chuỗi khác vào biến.

Ví dụ mình thay đổi ký tự thứ 2 trong chuỗi.

**Lỗi đổi giá trị của chuỗi**

|  |  |
| --- | --- |
| 1  2  3 | # Thử thay đổi ký tự thứ 2 thành ký tự khác  str = 'freetuts.net'  str[1] = "A" |

Thì nhận được kết quả lỗi như sau:



Chúng ta không thể xóa một ký tự trong chuỗi nhưng có thể xóa hoàn toàn chuỗi bằng cách sử dụng từ khóa del, đây là từ khóa được sử dụng rất nhiều từ các bài học trước tới giờ.

**Lỗi xóa một ký tự trong chuỗi**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | str = 'freetuts.net'    # Lệnh này sai, sẽ bị lỗi  del(str[1])    # Lệnh này đúng, nó xóa luôn biến str  del(str)    # Vì vậy khi in giá trị của str thì không tồn tại biến này  print(str) |

### Các phép toán với chuỗi string trong Python

Có nhiều phép toán được thực hiện với chuỗi string làm cho nó trở thành một trong những kiểu dữ liệu được sử dụng nhiều nhất trong Python.

#### Nối hai hoặc nhiều chuỗi

Phép toán nối hai hoặc nhiều chuỗi thành một chuỗi thì ta gọi là phép nối chuỗi.

* Để nối chuỗi thì ta sử dụng toán tử +.
* Nếu muốn lặp lại chuỗi nhiều lần thì dùng toán tử \*.

Hãy xem ví dụ dưới đây:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | str1 = 'Hello'  str2 ='World!'    # Sử dụng +  print('str1 + str2 = ', str1 + str2)    # Sử dụng \*  print('str1 \* 3 =', str1 \* 3) |

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2 | str1 + str2 =  HelloWorld!  str1 \* 3 = HelloHelloHello |

Nếu viết hai chuỗi gần kề nhau thì cũng được xem là phép nối chuỗi. Còn nếu muốn nối hai chuỗi có nhiều dòng thì phải bổ sung thêm cặp ngoặc đơn.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Đặt hai chuỗi gần nhau  print('Xin chào'' Freetuts')    # Hai chuỗi gần nhau nhưng có xuống hàng  # Nên thêm cặp ngoặc đơn ()  print(('Xin chào'         ' Freetuts')) |

Kết quả:

|  |  |
| --- | --- |
| 1  2 | Xin chào Freetuts  Xin chào Freetuts |

#### Lặp qua từng ký tự trong chuỗi

Nếu bạn muốn lặp qua từng ký tự trong chuỗi thì có thể kết hợp với [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html).

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5 | count = 0  for letter in 'Freetuts.net':      if(letter == 't'):          count += 1  print(count,'ký tự t được tìm thấy') |

Kết quả:

|  |  |
| --- | --- |
| 1 | 3 ký tự t được tìm thấy |

#### Kiểm tra chuỗi con trong chuỗi cha

Tương tự, để kiểm tra một chuỗi con có xuất hiện trong chuỗi cha hay không thì ta dùng toán tử in và not in.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Trả về True  print('f' in 'freetuts.net')    # Trả về False  print('f' not in 'freetuts.net') |

### Các định dạng chuỗi

#### Ký tự nháy đơn và nháy kép

Nếu trong chuỗi có ký tự nháy đơn hoặc nháy kép thì bạn phải bổ thêm đằng trước nó dấu \, nếu không Python sẽ không hiểu đó là một dãy chuỗi. Việc sử dụng nháy đơn hay kép phụ thuộc vào chuỗi được bao quanh bởi nháy đơn hay kép:

* Nếu bao quanh bởi nháy đơn thì thêm \ vào nháy đơn
* Nếu bao quanh bởi nháy kép thì thêm \ vào nháy kép

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # Sử dụng ba dấu nháy  print('''He said, "What's there?"''')    # Trường hợp nháy đơn  print('He said, "What's there?"')    # Trường hợp nháy kép  print("He said, \"What's there?\"") |

#### Ký tự thoát

Chúng ta có rất nhiều ký tự thoát trong Python, nó là một dạng chuỗi đặc biệt. Ví dụ bạn thêm \n cuối chuỗi thì python sẽ hiểu đó là ký tự xuống hàng.

Dưới đây là danh sách các ký tự thoát.

|  |  |
| --- | --- |
| **Escape Sequence** | **Description** |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| ' | Single quote |
| \" | Double quote |
| \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 | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

### Lời kết

Như vậy là mình đã giới thiệu sơ lược các kiến thức về xử lý chuỗi string trong Python, bài này giải thích rõ phần lý thuyết giúp bạn hiểu một cách tường tận để sau này không phải bở ngỡ trước những bài toán phức tạp. Chúc ban học tốt.

## Kiểu List (mảng) trong Python

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu List (mảng) trong Python. Đây là kiểu dữ liệu được dùng rất nhiều, nhất là trong những bài tập học kỹ thuật lập trình Python.

Kiểu dữ liệu List hay còn được gọi là array (tức là mảng). Mảng là một kiểu dữ liệu rất quan trọng bất kì ngôn ngữ nào như PHP, C/C++ nên việc nắm vững nó là rất cần thiết.

**Mục lục**

* [1. Mảng trong Python là gì?](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-0)
* [2. Các thao tác trên mảng trong Python](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-1)
  + [Thay đổi giá trị của phần tử](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-0)
  + [Đếm chiều dài của mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-1)
  + [Lặp qua từng phần tử](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-2)
  + [Kiểm tra một giá trị có tồn tại trong mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-3)
  + [Thêm phần tử vào mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-4)
  + [Xóa phần tử ra khỏi mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-5)
  + [Thiết lập List rỗng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-6)
* [3. Lời kết](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-2)

### Mảng trong Python là gì?

Trong Python, mảng là một loại dữ liệu đặc biệt, nó gồm nhiều phần tử và mỗi phần tử là một dữ liệu riêng biệt.

Ví dụ bạn cần lưu trữ danh sách sinh viên thì có thể sử dụng mảng, mỗi phần tử của mảng là một sinh viên.

Bạn cũng có thể tưởng tượng một chiếc tủ có nhiều ngăn xếp, lúc này chiếc tủ được xem là mảng và các ngăn xếp là các phần tử của mảng.

**\* Lưu ý**: Trong Python, ngoài tên gọi là mảng ra thì tên chính của nó là **List**nhé các bạn.

**Cách tạo mảng**

Trước tiên hãy tìm hiểu cú pháp khởi tạo một list, xem các ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3 | list1 = ['freetuts', 'blog', 1997, 2000];  list2 = [1, 2, 3, 4, 5 ];  list3 = ["a", "b", "c", "d"] |

Trong ví dụ này mình đã tạo ra ba biến lưu trữ 3 tập hợp List khác nhau, mỗi phần tư trong List không bắt buộc phải có cùng kiểu dữ liệu. Ví dụ dưới đây sẽ in ra giá trị của các phần tử trong List.

|  |  |
| --- | --- |
| 1  2 | thislist = ["apple", "banana", "cherry"]  print(thislist) |

Kết quả:

|  |  |
| --- | --- |
| 1 | ['apple', 'banana', 'cherry'] |

Ngoài ra, bạn có thể tạo list thông qua đối tượng list của Python.

|  |  |
| --- | --- |
| 1  2 | thislist = list(("apple", "banana", "cherry"))  print(thislist) |

Kết quả:

|  |  |
| --- | --- |
| 1 | ['apple', 'banana', 'cherry'] |

### Các thao tác trên mảng trong Python

Bây giờ ta sẽ tìm hiểu những hành động liên quan đến List nhé.

#### Thay đổi giá trị của phần tử

Như ta biết, list bản chất là array nên mỗi phần tử sẽ có một chỉ số đánh dấu riêng, vị trí đầu tiên là 0, tiếp theo là 1, 2, 3, 4, ... Ví dụ bạn có một List gồm 5 phần tử thì lần lược chỉ số đánh dấu của các phần tử là: 0, 1, 2, 3, 4. Như vậy để thay đổi giá trị của phần tử nào thì ta sẽ dựa vào số chỉ mục đánh dấu này.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  thislist[1] = "blackcurrant"  print(thislist) |

Kết quả:

|  |  |
| --- | --- |
| 1 | ['apple', 'blackcurrant', 'cherry'] |

#### Đếm chiều dài của mảng

Để đếm chiều dài của mảng thì ta sử dụng hàm len.

|  |  |
| --- | --- |
| 1  2 | thislist = ["apple", "banana", "cherry"]  print(len(thislist)) |

Kết quả:

|  |  |
| --- | --- |
| 1 | 3 |

#### Lặp qua từng phần tử

Để lặp qua từng phần tử thì ta sư dụng [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html) hoặc [vòng lặp while](https://freetuts.net/vong-lap-while-trong-python-685.html), kết hợp với hàm len để đếm tổng số phần tử của List.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3 | fruits = ['banana', 'apple',  'mango']  for index in range(len(fruits)):     print 'Current fruit :', fruits[index] |

Có một cách đơn giản hơn là bạn sử dụng cú pháp basic như sau:

|  |  |
| --- | --- |
| 1  2  3 | fruits = ['banana', 'apple',  'mango']  for fruit in fruits:     print 'Fruit :', fruit |

#### Kiểm tra một giá trị có tồn tại trong mảng

Ta phải sử dụng lênh if để kiểm tra. Như ví dụ dưới đây kiểm tra chuỗi "Apple" có tồn tại trong list không.

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  if "apple" in thislist:    print("Apple có trong danh sách này") |

### Thêm phần tử vào mảng

Nếu PHP sử dụng cú pháp $var[] = value để thêm một phần tử vào array thì trong Python phải sử dụng phương thức **append** của List.

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  thislist.append("orange")  print(thislist) |

Kết quả:

|  |  |
| --- | --- |
| 1 | ['apple', 'banana', 'cherry', 'orange'] |

#### Xóa phần tử ra khỏi mảng

**Cách 1**: Để xóa một phần tử ra khỏi List thì ta sử dụng phương thức remove() được tích hợp sẵn trong List.

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  thislist.remove("banana")  print(thislist) |

Chương trình này sẽ xóa phần tử có giá trji là "banana".

**Cách 2**: Nếu bạn muốn xóa phần tử có số thứ tự index nào đó thì sử dụng phương thức pop(). Nếu bạn không truyền index vào thì mặc định nó sẽ xóa phần tử cuối cùng.

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  thislist.pop(1)  print(thislist) |

Chương trình này sẽ xóa phần tử có index = 1, tức là "banana".

**Cách 3**: Sử dụng từ khóa del

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  del thislist[<em>0</em>]  print(thislist) |

Từ khóa del còn có thể xóa toàn bộ phần tử như sau.

|  |  |
| --- | --- |
| 1  2 | thislist = ["apple", "banana", "cherry"]  del thislist |

#### Thiết lập List rỗng

Để thiết lập list rỗng thì bạn sử dụng phương thức clear().

|  |  |
| --- | --- |
| 1  2  3 | thislist = ["apple", "banana", "cherry"]  thislist.clear()  print(thislist) |

### Lời kết

Trên là cách sử dụng mảng trong Python. Nếu bạn muốn tham khảo thêm các function thì hãy xem ở chuyên mục [List Methods](https://freetuts.net/python-function/list-methods) nhé, ở đó có rất nhiều ví dụ về cách sử dụng các phương thức có sẵn trong List như: [clear](https://freetuts.net/list-clear-trong-python-5298.html), [copy](https://freetuts.net/list-copy-trong-python-5297.html), [sort](https://freetuts.net/list-sort-trong-python-5296.html), ...

Như vậy là mình đã giới thiệu xong cách sử dụng List trong Python, và mình cũng có nói sơ lược về các phương thức có sẵn trong List dùng để xử lý những hành động thường gặp. Hẹn gặp bạn ở bài tiếp theo nhé.

## Kiểu dữ liệu Tuple trong Python

Trong bài viết này chúng ta sẽ học cách sử dụng **kiểu dữ liệu tuple trong Python**. Cụ thể hơn sẽ tìm hiểu khái niệm tuple là gì? Cách sử dụng nó như thế nào? Các phương thức được tích hợp sẵn trong tuple.

Ở bài trước chúng ta đã học kiểu dữ liệu mảng và bạn cũng đã biết cách sử dụng rồi phải không nào?

Thực ra tuple cũng không có gì đặc biệt, cách khai báo và sử dụng nó không khác gì mảng. Nhưng tại sao Python lại bổ sung thêm tuple trong khi đã có mảng? Hãy cùng mình tìm hiểu ngay nhé.

**Mục lục**

* [1. Kiểu dữ liệu tuple trong Python là gì?](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-0)
  + [Ưu điểm của tuple so với mảng](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-0)
* [2. Cách tạo Tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-1)
* [3. Truy cập các phần tử của tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-2)
  + [Truy cập thông qua chỉ mục index](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-1)
  + [Số chỉ mục âm](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-2)
  + [Slicing](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-3)
* [4. Thay đổi giá trị cho các phần tử Tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-3)
* [5. Xóa Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-4)
* [6. Các phương thức trong Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-5)
* [7. Một vài thao tác khác với Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-6)
  + [Kiểm tra phần tử có xuất hiện trong Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-4)
  + [Duyệt qua từng phần tử của Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-5)

### Kiểu dữ liệu tuple trong Python là gì?

Trong Python, kiểu dữ liệu Tuple tương tự như [mảng](https://freetuts.net/list-mang-trong-python-1690.html), sự khác biệt giữa chúng là ta **không thể thay đổi** các phần tử của một tuple, trừ khi phần tử đó là một mảng.

#### Ưu điểm của tuple so với mảng

Các phần tử của Tuple rất giống với List nên về cơ bản cả hai đều có thể sử dụng trong một tình huống tương tự nhau. Tuy nhiên, vẫn có một số lợi thế của Tuple so với List:

* Chúng ta thường dùng Tuple cho các kiểu dữ liệu không đồng nhất (các phần tử khác kiểu dữ liệu) và List cho các kiểu dữ liệu đồng nhất (các phần tử cùng kiểu dữ liệu).
* Vì tuple là bất biến, nên việc lặp qua tuple nhanh hơn so với List.
* Tuple có thể được sử dụng làm khóa cho [Dictionary](https://freetuts.net/dictionary-trong-python-1961.html), trong khi list thì không thể.
* Nếu bạn có dữ liệu ít thay đổi, việc triển khai nó dưới dạng Tuple sẽ đảm bảo không bị thay đổi.

### Cách tạo Tuple trong Python

Một tuple được tạo bằng cách đặt tất cả các mục (phần tử) bên trong dấu ngoặc đơn (), được phân tách bằng dấu phẩy. Các dấu ngoặc là tùy chọn, có hoặc không đều được, nhưng tốt nhất là nên sử dụng nó.

**Cú pháp**

|  |  |
| --- | --- |
| 1 | my\_tuple = (element1, element2, ...) |

Một tuple có thể có nhiều phần tử và mỗi phần tử có thể thuộc các kiểu dữ liệu khác nhau (int, float, list, string, v.v.).

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # Các phần tử Tuple có thể có các kiểu dữ liệu khác nhau  my\_tuple = ("freetuts", [8, 4, 6], (1, 2, 3))  print(my\_tuple)  # Kết quả: ("freetuts", [8, 4, 6], (1, 2, 3))    # tuple có thể được tạo bằng cách bỏ đi cặp ngoặc đơn  my\_tuple = 3, 4.6, "blog"  print(my\_tuple)  # Kết quả: 3, 4.6, "blog"    # Bạn có thể tách các phần tử tuple thành nhiều biến nhỏ  a, b, c = my\_tuple  print(a)  print(b)  print(c)  # Kết quả:  # 3  # 4.6  # blog |

Việc tạo một Tuple **chỉ có một phần tử** sẽ phức tạp hơn một xíu, bởi vì trường hợp này thì Python sẽ dễ bị nhầm lẫn sang kiểu [string](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html), vì vậy ta phải bổ sung thêm dấu phẩy ở cuối để trình biên dịch Python nhận biết đó là một Tuple.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # Chỉ có một phần tử nên Python hiểu nhầm qua kiểu string  my\_tuple = ("hello")  print(type(my\_tuple))  # Kết quả: <class 'str'>    # Bạn thêm dấu phẩy để Python biết đây là 1 tuple  my\_tuple = ("hello",)  print(type(my\_tuple))  # Kết quả: <class 'tuple'>    # Bạn cũng có thể bỏ cặp dấu ngoặc đơn nhưng phải có dấu phẩy  my\_tuple = "hello",  print(type(my\_tuple))  # Output: <class 'tuple'> |

### Truy cập các phần tử của tuple trong Python

Có nhiều cách khác nhau để truy cập các phần tử của tuple.

#### Truy cập thông qua chỉ mục index

Chúng ta có thể sử dụng toán tử chỉ mục [] để truy cập đến một phần tử trong một tuple, chỉ mục bắt đầu từ 0.

Ví dụ, một tuple có 6 phần tử thì chỉ mục bắt đầu từ 0 đến 5, nên nếu bạn cố gắng truy cập đến cá phần tử khác như (6, 7, ...) thì sẽ gây ra lỗi **IndexError**.

Chỉ mục phải là một số nguyên (int), vì vậy bạn không thể sử dụng kiểu float hoặc các loại khác, điều này sẽ dẫn đến TypeError.

Tương tự, tuple lồng nhau được truy cập bằng cách sử dụng lập chỉ mục lồng nhau, như trong ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | # Danh sách Tuple  my\_tuple = ('p','e','r','m','i','t')    # Lấy phần tử đầu tiên trong tuple  print(my\_tuple[0])  # Output: 'p'    # Lấy phần tử thứ 6 trong tuple  print(my\_tuple[5])  # Output: 't'    # Mỗi phần tử của tuple là một mảng hoặc 1 tuple khác  n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))    # Lấy phần tử thứ 4 của phần tử thứ nhất  print(n\_tuple[0][3])  # Output: 's'    # Lấy phần tử thứ 2 của phần tử thứ 2  print(n\_tuple[1][1])  # Output: 4 |

#### Số chỉ mục âm

Python cho phép lập chỉ mục số âm, chỉ số -1 đề cập đến phần tử cuối cùng, -2 cho phần tử cuối cùng thứ hai, v.v.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | my\_tuple = ('p','e','r','m','i','t')    # Phần tử cuối cùng  print(my\_tuple[-1])  # Output: 't'    # Phần tử thứ 6 tính từ cuối lên  print(my\_tuple[-6])  # Output: 'p' |

#### Slicing

Chúng ta có thể truy cập vào một loạt các phần tử trong một tuple bằng cách sử dụng **toán tử slicing**, tức là dấu hai chấm ":".

* [begin:end] : Trong đó end là biên, tức sẽ ko lấy phần tử **end**mà lấy từ begin đến **end - 1**.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Lấy phần tử thứ 2 đến thứ 4  print(my\_tuple[1:4])  # Output: ('r', 'o', 'g')    # Phần tử đầu tiên đến thứ hai (tức thứ 7 tính từ sau tới)  print(my\_tuple[:-7])  # Output: ('p', 'r')    # Phần tử thứ 8 đến cuối  print(my\_tuple[7:])  # Output: ('i', 'z')    # Lấy toàn bộ phần tử  print(my\_tuple[:])  # Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z') |

Slicing có thể được xem là tốt nhất nếu bạn dùng nó để lấy một dãy các phần tử trong Tuple.

### Thay đổi giá trị cho các phần tử Tuple trong Python

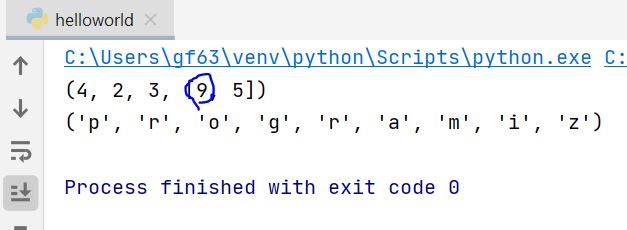
Không giống như mảng, bạn không thể thay đổi các phần tử của Tuple, điều này có nghĩa là các phần tử của một tuple không thể thay đổi một khi nó đã được gán giá trị.

Nhưng nếu phần tử tự nó là một kiểu dữ liệu giống như mảng thì ta có thể thay đổi được.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | my\_tuple = (4, 2, 3, [6, 5])    # Thay đổi giá trị cho phần tử thứ nhất của phần tử thứ 4 trong my\_tuple.  my\_tuple[3][0] = 9  print(my\_tuple)    # Chương trình này sẽ bị lỗi  my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Bỏ dòng này sẽ không có lỗi như hình  # Bạn hãy thử mở lại dòng này sẽ thấy bị lỗi do ta cố tình thay đổi ..  # giá trị cho phần tử tuple  # my\_tuple[1] = 'a'    print(my\_tuple) |

Kết quả:



### Xóa Tuple

Như đã thảo luận ở trên, chúng ta không thể thay đổi các thành phần trong một tuple, điều đó cũng có nghĩa là chúng ta không thể xóa các phần tử của nó, nhưng hoàn toàn có thể xóa một tuple bằng cách sử dụng từ khóa del.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Xóa biến my\_tuple  del my\_tuple    # Lệnh này lỗi vì biến my\_tuple đã bị xóa  my\_tuple |

### Các phương thức trong Tuple

Trong lớp đối tượng Tuple hỗ trợ hai phương thức, đó là:

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count(x)](https://freetuts.net/tuple-count-trong-python-5316.html) | Đếm số lần xuất hiện của một phần tử |
| [index(x)](https://freetuts.net/tuple-index-trong-python-5317.html) | Trả về chỉ mục của phần tử cần tìm |

### Một vài thao tác khác với Tuple

Sau đây là một vài thao tác khác với Tuple thường gặp nhất.

#### Kiểm tra phần tử có xuất hiện trong Tuple

Để kiểm tra một phần tử có xuất hiện trong tuple không thì ta dùng toán tử in.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | my\_tuple = ('a','p','p','l','e',)    # Toán tử in  print('a' in my\_tuple)  # Output: True    print('b' in my\_tuple)  # Output: False    # Toán tử not in  print('g' not in my\_tuple)  # Output: True |

#### Duyệt qua từng phần tử của Tuple

Để lặp qua từng phần tử của Tuple thì ta có thể sử dụng vòng lặp for.

**Vi dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5 | for name in ('John','Kate'):       print("Hello",name)  # Output:  # Hello John  # Hello Kate |

Ngoài ra vẫn còn một vài function khác thuộc lớp [Build-in function](https://freetuts.net/python-function/built-in-functions).

## Kiểu dữ liệu Set trong Python

Trong bài này chúng ta sẽ tìm hiểu **kiểu dữ liệu Set trong Python**với các thao tác như: Cách khởi tạo Set, thêm hoặc xóa các thành phần tử ra khỏi Set, và các thao tác thường dùng trên Set.

Các kiểu dữ liệu trong Python quả thật rườm rà, tại sao có mảng, tuple rồi mà lại thêm kiểu tập hợp Set làm gì nữa? Cái gì cũng có lý do của nó các bạn ạ. Sau đây chúng ta sẽ tìm hiểu về Set nhé.

**Mục lục**

* [1. Set trong Python là gì?](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-0)
  + [Cú pháp](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-0)
  + [Khởi tạo Set rỗng](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-1)
* [2. Làm thế nào để thay đổi Set trong Python?](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-1)
* [3. Làm sao để xóa phần tử ra khỏi Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-2)
* [4. Các hành phép tập hợp của Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-3)
  + [Set union](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-2)
  + [Set Intersection](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-3)
  + [Set Difference](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-4)
  + [Set Symmetric Difference](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-5)
* [5. Các thao tác khác với Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-4)
  + [Kiểm tra phần tử tồn tại trong Set](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-6)
  + [Lặp qua các phần tử của Set](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-7)
* [6. Lời kết](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-5)

### Set trong Python là gì?

Set trong Python là tập các phần tử dữ liệu không có thứ tự, mỗi phần tử là duy nhất (không trùng lặp) và phải bất biến (không thể thay đổi). Tuy nhiên chúng ta có thể thêm hoặc xóa các phần tử ra khỏi Set một cách dễ dàng.

#### Cú pháp

Sau đây là cú pháp tạo Set trong Python.

|  |  |
| --- | --- |
| 1 | my\_set = {element\_1, element\_2, element\_3, ...} |

Trong đó mỗi element là một phần tử và có thể có thuộc nhiều kiểu dữ liệu khác nhau, không nhất thiết phải đồng nhất.

Vì các phần tử không thể thay đổi giá trị nên **Set không cho phép lưu trữ kiểu List hoặc Dictionary**.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Tập hợp set kiểu int  my\_set = {1, 2, 3}  print(my\_set)    # Tập hợp set đa dạng kiểu khác  my\_set = {1.0, "Hello", (1, 2, 3)}  print(my\_set) |

Như bạn thấy, ở ví dụ đầu tiên thì mình truyền dữ liệu đơn giản, còn ở ví dụ thứ hai thì có thểm kiểu Tuple.

**Ví dụ khác**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # Set không chứa dữ liệu trùng nhau  # Như trong tập hợp này mình cố tình cho trùng giá trị 3  # Nhưng khi in ra thì nó chỉ lấy 1 số 3 mà thôi  my\_set = {1,2,3,4,3,2}  print(my\_set)  # Output: {1, 2, 3, 4}    # Set không thể chứa tập hợp có thể thay đổi như Dictionary và List  # Nên đoạn code này sẽ bị lỗi TypeError  # Do nó chứa mảng [3,4]  my\_set = {1, 2, [3, 4]}  # TypeError: unhashable type: 'list'    # Ta có thể khởi tạo giá trị của set qua ...  # đối tượng Set() trong Python  # bằng cách truyền vào một list (array)  my\_set = set([1,2,3,2])  print(my\_set)  # Output: {1, 2, 3} |

Ở ví dụ này mình đã cố tình truyền dữ liệu bị trùng và kết quả là nó lọc hết chỉ giữ lại 1 giá trị bị trùng mà thôi. Ở ví dụ cuối cùng là mình đã khởi tạo một Set từ List chứ không phải tạo phần tử Set chứa kiểu dữ liệu List đâu nhé, đừng nhầm lẫn đoạn đó.

#### Khởi tạo Set rỗng

Việc khởi tạo Set rỗng sẽ có chút nhầm lẫn, việc sử dụng cặp {} sẽ gây hiểu lầm với kiểu dữ liệu [Dictionary](https://freetuts.net/dictionary-trong-python-1961.html), vì vậy để khởi tạo một Set rỗng thì nên sử dụng đối tượng **Set()** và không truyền bất kì đối số nào vào.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Khởi tạo bằng cặp {}  a = {}    # Kiểm tra kiểu dữ liệu thì nó là một dictionary  print(type(a))  # Output: <class 'dict'>    # Khởi tạo bằng set()  a = set()    # Kiểm tra kiểu dữ liệu thì là một Set  print(type(a))  # Output: <class 'set'> |

### Làm thế nào để thay đổi Set trong Python?

Set không phải là kiểu dữ liệu **có chỉ mục** nên việc truy xuất thông qua chỉ mục index là điều không thể. Tuy nhiên nó hỗ trợ một số phương thức giúp ban làm điều này.

* add() dùng để thêm phần tử
* update() dùng để cập nhật phần tử

Hãy đọc kỹ phần ví dụ dưới đây sẽ giúp bạn hiểu rõ hơn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | # Khởi tạo set my\_set  my\_set = {1,3}  print(my\_set)    # Nếu bạn bỏ comment ở dòng 9,  # bạn set nhận một lỗi là  # TypeError: 'set' object does not support indexing    #my\_set[0]    # Thêm một phần tử vào set  my\_set.add(2)  print(my\_set)  # Output: {1, 2, 3}    # Thêm nhiều phần tử vào Set, ta dùng mảng để chứa các phần tử đó  my\_set.update([2,3,4])  print(my\_set)  # Output: {1, 2, 3, 4}    # Cập nhật set bằng một list và set mới  # Nó sẽ duyệt và loại bỏ bớt giá trị trùng nhau  my\_set.update([4,5], {1,6,8})  print(my\_set)  # Output: {1, 2, 3, 4, 5, 6, 8} |

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2  3  4 | {1, 3}  {1, 2, 3}  {1, 2, 3, 4}  {1, 2, 3, 4, 5, 6, 8} |

### Làm sao để xóa phần tử ra khỏi Set trong Python

Để loại bỏ phần tử ra khỏi Set thì ta sử dụng hai phương thức discard() và remove(). Cả hai đều có chung một mục đích, tuy nhiên điểm khác biệt là với discard() thì sẽ không sinh lỗi nếu giá trị cần xóa không tồn tại, còn remove() thì sẽ sinh lỗi.

Ví dụ dưới đây sẽ minh họa cho điều này:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | # Khởi tạo my\_set  my\_set = {1, 3, 4, 5, 6}  print(my\_set)    # Xóa phần tử có giá trị 4  my\_set.discard(4)  print(my\_set)  # Output: {1, 3, 5, 6}    # Xóa phần tử có giá trị 6  my\_set.remove(6)  print(my\_set)  # Output: {1, 3, 5}    # Xóa phần tử  # không tồn tại trong my\_set  my\_set.discard(2)  print(my\_set)  # Output: {1, 3, 5} => Không bị lỗi    # Xóa một phần tử  # không tồn tại trong my\_set  # thì sẽ bị lỗi, do dùng phương thức remove  my\_set.remove(2)  # Output: KeyError: 2 |

Tương tự, chúng ta có thể xóa và trả về phần tử đã xóa bằng phươn thức pop(), hoặc xóa toàn bộ phần tử bằng phương thức clear().

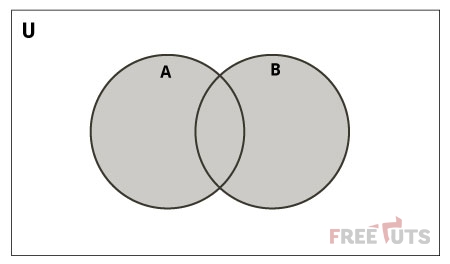
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # Khởi tạo my\_set  my\_set = set("HelloWorld")  print(my\_set)  # Output: {'r', 'd', 'o', 'e', 'W', 'H', 'l'}    # pop một phần tử ngẫu nhiên  print(my\_set.pop())  # Output: Tùy thuộc vào giá trị mà nó xóa mà in kết quả khác nhau    # pop thêm một phần tử ngẫu nhiên khác  my\_set.pop()  print(my\_set)  # Output: Cũng là random    # clear my\_set, kết quả trả về Set rỗng  my\_set.clear()  print(my\_set)  #Output: set() |

### Các hành phép tập hợp của Set trong Python

Set có thể sử dụng với nhiều phép toán như liên kết, hợp, ... bằng cách sử dụng các toán tử và phương thức mà nó hỗ trợ sẵn.

#### Set union

Toán tử union kí hiệu là |, nó sẽ trả về hợp của hai Set

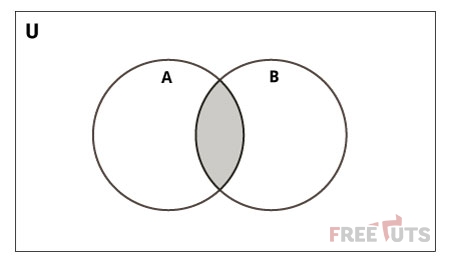


**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Khởi tạo set A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử union |  print(A | B)  # Output: {1, 2, 3, 4, 5, 6, 7, 8} |

#### Set Intersection

Phép giao intersection ký hiệu là &, nó sẽ trả về phần giao của hai set

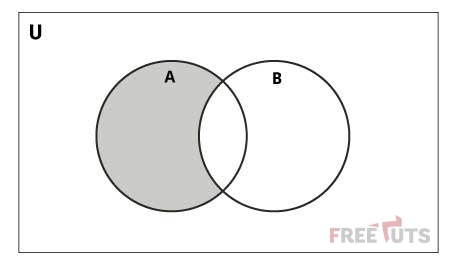


**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Khởi tạo set A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử &  print(A & B)  # Output: {4, 5} |

#### Set Difference

Phép Difference kí hiệu là -, nó sẽ trả về các phần tử của A không có trong B

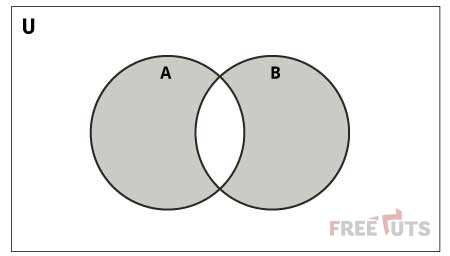


**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Khởi tạo set 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} |

#### Set Symmetric Difference

Toán tử Set Symmetric Difference kí hiệu là ^, Nó sẽ trả về phần tử A không có trong B và phần tử B không có trong A.



**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # 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} |

### Các thao tác khác với Set trong Python

Bây giờ mình sẽ làm một vài ví dụ về các thao tác khác thường gặp trên Set, đây là những thao tác được sử dụng rất nhiều khi bạn làm việc với Python.

#### Kiểm tra phần tử tồn tại trong Set

Để kiểm tra một phần tử có tồn tại không trong Set thì ta sử dụng toán tử in, toán tử này được sử dụng khá nhiều, hầu như tất cả các loại dữ liệu đều có.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # Khởi tạo my\_set  my\_set = set("apple")    # Kiểm tra xem 'a' có tồn tại trong set không  print('a' in my\_set)  # Output: True    # Kiểm tra xem 'p' có tồn tại trong set không  print('p' not in my\_set)  # Output: False |

#### Lặp qua các phần tử của Set

Để lặp qua các phần tử của Set thì ta có thể sử dụng [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html), đây là vòng lặp được sử dụng nhiều nhất trong Python.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for letter in set("apple"):      print(letter)    # Result:  # p  # l  # a  # e |

### Lời kết

Như vậy là mình đã giới thiệu xong kiến thức cơ bản về tập hợp Set trong Python.

Bạn cần lưu ý rằng với Set thì không thể thay đổi giá trị của các phần tử, bản chất khi sử dụng phương thức update là bạn đã bổ sung các phần tử. Tuy nhiên bạn có thể thực hiện các thao tác xóa phần tử ra khỏi set

## Dictionary trong Python: Kiểu dữ liệu từ điển

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu từ điển **Dictionary trong Python**, đây là kiểu dữ liệu rất hay, thường được dùng để kết hợp với JSON để xử lý dữ liệu.

Nếu bạn đã từng học qua PHP thì có thể xem dictionary python là một mảng kết hợp gôm các cặp key : value. Đây cũng là cấu trúc phù hợp với chuỗi JSON.

**Mục lục**

* [1. Tạo dictionary Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-0)
* [2. Truy xuất phần tử của Dictionary trong Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-1)
* [3. Thay đối giá trị phần tử của Dictionary Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-2)
* [4. Xóa phần tử ra khỏi Dictionary](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-3)
* [5. Thay đổi các phần tử dictionary qua vòng lặp](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-4)

### Tạo dictionary Python

Kiểu từ điển dictionary là loại dữ liệu tổng hợp được sắp xếp không có thứ tự, mỗi phần tử sẽ có hai phần đó là key và value.

Nếu bạn đã từng học qua PHP thì nó giống như một mảng Array, còn với Javascript thì nó giống như là một Object.

**Ví dụ**

|  |  |
| --- | --- |
| 1 | my\_dict = {1: 'apple', 2: 'ball'} |

Như trong ví dụ này thì chúng ta có hai phần tử và mỗi phần tử có hai thông số key : value, đó là 1 : 'apple' và 2 : 'ball'.

Khi bạn truy xuất một phần tử trong dictionary sẽ rất nhanh nếu biết trước tên khóa đó là gì.

Giả sử mình cần lấy phần tử thứ có key là 1 thì sẽ lấy như sau:

|  |  |
| --- | --- |
| 1 | my\_dict[1] |

Các key có thể là một số tự nhiên hoặc một chuỗi được đặt trong ngoặc đơn hoặc ngoặc kép.

### Truy xuất phần tử của Dictionary trong Python

Để truy xuất đến các phần tử trong dictionary thì ta dựa vào tên key của phần tử đó, vì vậy bạn phải biết tên key nhé.

Chúng ta có hai cách lấy, cách thứ nhất là thông qua cặp dấu ngoặc vuông và cách thứ hai là thông qua phương thức get() được tích hợp sẵn trong dictionary.

Ví dụ dưới đây thể hiện cho hai cách này.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | my\_dict = {'name':'Jack', 'age': 26}    # Output: Jack  print(my\_dict['name'])    # Output: 26  print(my\_dict.get('age')) |

Sự khác nhau giữa chúng là gì? Thực tế kết quả không có gì khác nhau, chỉ là với phương thức get() thì sẽ trả về none nếu key không tồn tại, còn cặp ngoặc vuông sẽ trả về lỗi.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2 | # Error vì key abc không tồn tại  print(my\_dict['abc']) |

### Thay đối giá trị phần tử của Dictionary Python

Chúng ta có hai thao tác chủ yếu đó là thêm hoặc thay đổi giá trị cho một phần tử, và đương nhiên để thay đổi thì ta phải biết tên key.

Dictionary là kiểu dữ liệu cho phép ta thay đổi các phần tử. Để thêm một phần tử mới thì ta sử dụng toán tử gán (=), nếu phần tử cần thêm chưa tồn tai thì dictionary sẽ bổ sung nó vào, ngược lại nếu đã tồn tại thì sẽ được cập nhật giá trị.

Hãy xem ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | my\_dict = {'name':'Jack', 'age': 26}    # Thay đổi giá trị  my\_dict['age'] = 27    #Output: {'age': 27, 'name': 'Jack'}  print(my\_dict)    # Thêm phần tử  my\_dict['address'] = 'Downtown'    # Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}  print(my\_dict) |

Khi chạy lên kết quả sẽ có dạng như sau:

|  |  |
| --- | --- |
| 1  2 | {'name': 'Jack', 'age': 27}  {'name': 'Jack', 'age': 27, 'address': 'Downtown'} |

### Xóa phần tử ra khỏi Dictionary

Chúng ta có thể xóa một phần tử ra khỏi dictionary bằng cách sử dụng phương thức pop(), tham số truyền vào là tên key cần xóa, kết quả trả về là giá trị của phần tử đã xóa.

Hoặc sử dụng phương thức popitem() để xóa phần tử ra khỏi dictionary và kết quả trả về là cặp giá trị (key, value) đã xóa, và sử dụng phương thức clear() để xóa toàn bộ phần tử.

Ngoài ra nếu bạn muốn xóa luôn dictionary đó thì có thể sử dụng từ khóa del, chi tiết sẽ xem ở ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | # Xóa sử dụng phương thức pop()  # Output: 16  print(squares.pop(4))    # Output: {1: 1, 2: 4, 3: 9, 5: 25}  print(squares)    # Xóa sử dụng phương thức popitem()  # Output: (1, 1)  print(squares.popitem())    # Output: {2: 4, 3: 9, 5: 25}  print(squares)    # Xóa sử dụng từ khóa del  del squares[5]    # Output: {2: 4, 3: 9}  print(squares)    # Xóa tất cả phần tử sử dụng phương thức clear()  squares.clear()    # Output: {}  print(squares)    # Xóa biến squares ra khỏi bộ nhớ  del squares    # Throws Error bởi vì biến squares đã bị xóa nên không tồn tại  # print(squares) |

Rõ ràng kết quả khi chạy chương trình này sẽ là:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 16  {1: 1, 2: 4, 3: 9, 5: 25}  (1, 1)  {2: 4, 3: 9, 5: 25}  {2: 4, 3: 9}  {} |

### Thay đổi các phần tử dictionary qua vòng lặp

Trường hợp bạn muốn tất cả các phần tử được thay đổi thông qua một công thức nào đó thì có thể kết hợp sử dụng vòng lặp for.

**Ví dụ**: cho dictionary sau:

|  |  |
| --- | --- |
| 1 | squares = {1:1, 2:4, 3:9, 4:16, 5:25} |

Bây giờ mình muốn tạo thành một squares mới với mỗi phần tử sẽ được nhân double lên. Với cách làm thông thường thì mình sẽ làm như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | squares = {1:1, 2:4, 3:9, 4:16, 5:25}  squares[1] = squares[1]\*2  squares[2] = squares[2]\*2  squares[3] = squares[3]\*2  squares[4] = squares[4]\*2    #Output: {1: 2, 2: 8, 3: 18, 4: 32, 5: 25}  print(squares) |

Trên là một vài chia sẻ về cách sử dụng kiểu từ điển dictionary trong Python, hy vọng qua bài này bạn hiểu được cách khởi tạo, thêm phần tử, xóa phần tử, cũng như thay đổi giá trị của các phần tử trong dictionary.

## Exceptions trong Python, xử lý lỗi đơn giản

Trong bài này chúng ta sẽ tìm hiểu Exception trong Python, đây là mã lệnh giúp xử lý các lỗi từ cơ bản đến nâng cao trong quá trình lập trình Python.

Một exception có thể được xem là một đoạn code bất thường trong một chương trình dẫn đến sự gián đoạn trong dòng chảy của chương trình đó, hay còn gọi là lỗi.

Bất cứ khi nào có exception xảy ra, chương trình sẽ dừng thực thi nên các đoạn mã phía dưới không được biên dịch. Do đó, một exception là lỗi mà trình biên dịch python không thể chạy được.

**Mục lục**

* [1. Các exception trong Python có sẵn](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-0)
* [2. Chuyện gì xảy ra nếu không dùng exception?](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-1)
* [3. Lệnh try ... except trong Python](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-2)
* [4. Except bị thiếu exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-3)
* [5. Khai báo nhiều exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-4)
* [6. Khối finaly trong Exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-5)
* [7. Nhảy exception với từ khóa raise](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-6)
* [8. Tự tạo exception trong Python](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-7)

### Các exception trong Python có sẵn

Python cung cấp cho chúng ta cách xử lý exception để giúp chương trình không bị gián đoạn. Vì vậy với những đoạn code bạn cảm thấy không an toàn thì hãy đưa nó vào một exception.

Dưới đây là danh sách những exception thường được sử dụng trong lập trình Python:

1. **ZeroDivisionError**: Xảy ra khi một số được chia cho số không.
2. **NameError**: Xảy ra khi tên không tồn tại, có thể là cục bộ hoặc toàn cục.
3. **ValueError**: Xảy ra khi bạn gán dữ liệu cho biến không chính xác.
4. **IndentationError**: Xảy ra khi chương trình thụt hàng không đúng.
5. **IOError**: Xảy ra khi xử lý nhập xuất bị lỗi.
6. **EOFError**: Xảy ra khi kết thúc tập tin mà các thao tác vẫn còn thực hiện trên đó.
7. **KeyError**: Xảy ra khi một key không tồn tại trong dictionary
8. **ImportError**: Xảy ra khi bạn import một package / module không thành công
9. **SyntaxError**: Xảy ra khi lỗi liên quan đến cú pháp
10. **ZeroDivisionError**: Xảy ra khi bạn chia một số cho 0
11. **ArithmeticError**: Lỗi liên quan đến toán học

Tuy nhiên, bạn có thể tự tạo ra các exception trong Python bằng cách sử dụng từ khóa except (mình sẽ trình bày nó ở phần cuối cùng của bài viết này).

### Chuyện gì xảy ra nếu không dùng exception?

Như chúng ta đã thảo luận ở trên, exception là một lệnh điều kiện bắt lỗi trong quá trình thực thi chương trình.

Hãy xem xét ví dụ sau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = int(input("Enter a:"))  b = int(input("Enter b:"))  c = a/b;  print("a/b = %d"%c)    #other code:  print("Vị trí khác của chương trình") |

Giả sử mình nhập số a = 10, b = 0, chạy chương trình lên sẽ có kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Enter a:10  Enter b:0  Traceback (most recent call last):    File "exception-test.py", line 3, in <module>      c = a/b;  ZeroDivisionError: division by zero |

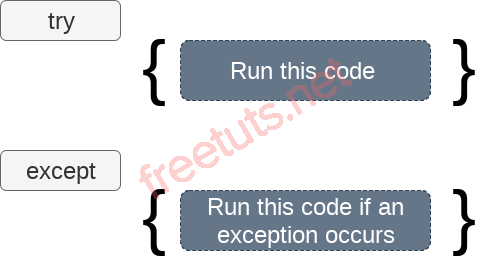
Như vậy bạn sẽ nhận được một thông báo lỗi **ZeroDivisionError**.

Để bắt lỗi này và giúp chương trình không bị gián đoạn thì ta sử dụng lệnh try ... except.

### Lệnh try ... except trong Python

Nếu chương trình python chứa những đoạn code đáng ngờ thì bạn hãy đặt chúng vào một exception, bằng cách ném chúng vào một khối lệnh **try .. except**.

Thường trong try là đoạn code nghi ngờ có lỗi, trong except là đoạn code thông báo lỗi hoặc trả về lỗi. Bạn có thể định nghĩa nhiều except.

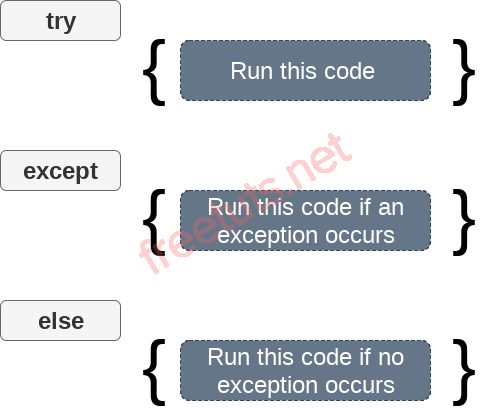


**Cú pháp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | try:      #block of code    except Exception1:      #block of code    except Exception2:      #block of code    #other code |

Chúng ta cũng có thể sử dụng từ khóa ELSE kết hợp trong **try-except** để thực thi những đoạn code trong trường hợp **except-block** không chạy.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try:      #block of code    except Exception1:      #block of code    else:      #Đoạn này sẽ chạy nếu except block không chạy |



**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try:      a = int(input("Enter a:"))      b = int(input("Enter b:"))      c = a/b;      print("a/b = %d"%c)  except Exception:      print("Không thể chia cho 0")  else:      print("Đây là đoạn code ELSE") |

**Kết quả**:

|  |  |
| --- | --- |
| 1  2  3  4 | Enter a:10  Enter b:2  a/b = 5  Đây là đoạn code ELSE |

### Except bị thiếu exception

Ở các ví dụ trên mình đã xác định rõ lỗi cho các except bằng cách đặt tên lỗi phía sau, lúc này nếu đoạn code nào trong **try block** bị lỗi nào thì sẽ chạy đoạn code ở **except** tương ứng với lỗi đó.

|  |  |
| --- | --- |
| 1 | except Exception: |

Tuy nhiên bạn hoàn toàn có thể bỏ từ khóa Exception như ở ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try:      a = int(input("Enter a:"))      b = int(input("Enter b:"))      c = a/b;      print("a/b = %d"%c)  except:      print("Không thể chia cho 0")  else:      print("Đoạn code trong Else") |

Lúc này dù thể loại lỗi là gì đi nữa thì lỗi đó vẫn được ném vào except.

**Tóm lại bạn cần phải ghi nhớ những điều sau:**

* Python cho phép chúng ta không cần phải chỉ định **exception**ở lệnh **except**.
* Có thể khai báo nhiều **except,**vì khối **try**có thể ném ra nhiều loại **exception**khác nhau.
* Chúng ta cũng có thể chỉ định một khối **else** cùng với câu lệnh **try-except** và sẽ được thực thi nếu không có **exception**nào sinh ra trong khối **try**.
* Các câu lệnh không ném exception nên đặt bên trong khối **else**.

**Ví dụ:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try:      #Đoạn này sẽ sinh ra lỗi nếu file không tồn tại.      fileptr = open("file.txt","r")  except IOError:      print("File không tồn tại")  else:      print("File mở thành công")      fileptr.close() |

### Khai báo nhiều exception

Nếu bạn muốn nhiều exception cùng thực hiện chung một đoạn code thì sử dụng cú pháp sau để kai báo.

|  |  |
| --- | --- |
| 1 | except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>) |

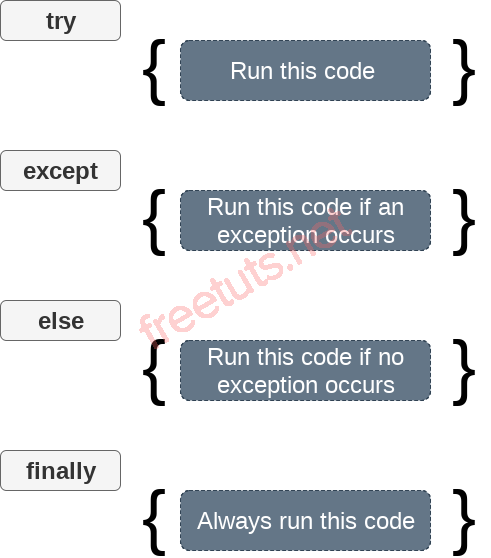
**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | try:      a=10/0;  except ArithmeticError,StandardError:      print "Lỗi tính toán"  else:      print "Thành công" |

### Khối finaly trong Exception

Nếu khối else sẽ được thực hiện nếu không có exception nào được sinh ra thì khối finany sẽ **luôn luôn được thực thi** vì nó là một khối đặc biệt. Cú pháp của nó như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | try:      # Khối chứa đoạn code      # có khả năng sinh lỗi  finally:      # Khối chứa đoạn code      # luôn luôn thực thi |



**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try:      fileptr = open("file.txt","r")      try:          fileptr.write("Nội dung ghi vào file")      finally:          fileptr.close()          print("Đóng file")  except:      print("Mở file lỗi") |

### Nhảy exception với từ khóa raise

Ở những ví dụ trên sẽ sinh ra những exception mà Python có hỗ trợ. Trường hợp bạn muốn tự định nghĩa exception, hoặc nhảy đến exception nào đó thì có thể sử dụng từ khóa **raise**.

**Cú pháp**:

|  |  |
| --- | --- |
| 1 | raise Exception\_class,<value> |

**Ví dụ**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try:      age = int(input("Nhập tuổi của bạn?"))      if age<18:          raise ValueError      else:          print("Tuổi không hợp lệ")  except ValueError:      print("Tuổi quá nhỏ") |

Nếu bạn chạy và nhập tuổi bé hơn 18 thì sẽ xuất hiện lỗi "Tuổi quá nhỏ".

### Tự tạo exception trong Python

Python cho phép chúng ta tạo ra các exception bằng cách sử dụng từ khóa except.

Mình khuyên bạn nên đọc phần này sau khi học xong kiến thức về lớp, bởi để định nghĩa một exception thì bạn phải sử dụng **class**.

**Ví dụ**: Tự tạo ra exception **ErrorInCode**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class ErrorInCode(Exception):      def \_\_init\_\_(self, data):          self.data = data      def \_\_str\_\_(self):          return repr(self.data)    try:      raise ErrorInCode(2000)  except ErrorInCode as ae:      print("Received error:", ae.data) |

Như mình đã nói ở trên, vì trong ví dụ này có sử dụng class để tạo ra một đối tượng exception nên bạn phải hiểu về class-object thì mới hiểu được.

Trên là cách sử dụng exception để xử lý ngoại lệ trong Python. Hy vọng qua bài này bạn sẽ hiểu và vận dụng trong quá trình học tập và làm việc với Python.

## Bài tập Python: Thực hành với các kiểu dữ liệu

Trong bài này mình sẽ tổng hợp các bài tập Python thực hành với các kiểu dữ liệu như kiểu mảng list, tuple, set, dictionary. Đây là những bài tập ở mức cơ bản dành cho newbie.

Để làm được những bài này thì bạn phải có kiến thức nền tảng về Python, vì vậy đừng quên học toàn bộ những nội dung kiến thức ở các bài trước nhé, nhất là kiến thức lệnh rẻ nhánh và vòng lặp.

**Câu hỏi thường gặp liên quan:**

* [In ra các số chẵn trong mảng bằng Python](https://freetuts.net/ref/in-ra-cac-so-chan-trong-mang-bang-python-419.html)
* [Tìm kiếm phần tử trong mảng bằng Python](https://freetuts.net/ref/tim-kiem-phan-tu-trong-mang-bang-python-420.html)
* [Sự khác nhau giữa mảng, tuple, dictionary và set trong Python](https://freetuts.net/ref/su-khac-nhau-giua-mang-tuple-dictionary-va-set-python-422.html)
* [Sắp xếp mảng tăng / giảm dần bằng Python](https://freetuts.net/ref/sap-xep-mang-tang-giam-dan-bang-python-423.html)
* [Chương trình quản lý sinh viên Python lưu mảng và dictionary](https://freetuts.net/ref/chuong-trinh-quan-ly-sinh-vien-python-421.html)