Python data type: string, number, list, tuple, set and dictionary

In this section, you'll learn how to use Python as a computer, grasp Python's data types and take the first step towards Python programming.

In the previous section, we became familiar with the first **Python** program, adding two numbers and printing their sum out of the screen. In this section, you will learn how to use Python as a computer, learn numbers, strings, lists and take the first step towards Python programming.

Because the article will list the most important content of all data types in Python, it will be quite long. Each data type comes with a specific example so you can easily imagine.

Make sure you save this **Python Document** page to update the latest articles. Don't forget to do Python exercises to reinforce your knowledge.

Data type in Python

1. Turn in Python

Variables are a location in memory used to store data (values). The variable is uniquely named to distinguish between different memory locations. The rules for writing a variable name are the same as the rules for writing identifiers in Python.

In Python, you don't need to declare the variable before using it, just assign a variable a value and it will exist. There is also no need to declare variable type, the variable type will be automatically received based on the value you assigned to the variable.

Assign values ??to variables:

To assign a value to the variable, use the = operator. Any kind of value can be assigned to a valid variable.

For example:

```
hoa = "H?ng" la = 3 canh = 5.5
```

Above are three assignment statements, "Pink" is a character string, assigned to the flower variable, 3 is an integer and is assigned to la, 5.5 is a decimal number and assigned to the soup.

Assign multiple values:

In Python you can perform multiple assignments in a command like this:

```
hoa, la, canh = "H?ng", 3, 5.5
```

If you want to assign the same value to multiple variables, you can write the following command:

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

The above command will assign a value of 3 to all three variables, flowers, la and soup.

2. Digital data type in Python

Digital data types in Python

Python supports integers, decimals, and complex numbers, which in turn are defined as int, float, and complex classes in Python. Integers and decimals are distinguished by the presence or absence of the decimal point. For example: 5 is an integer, 5.0 is a decimal number. Python also supports complex numbers and uses the suffix `j` or `J` to indicate the virtual part. For example: `3+5j` . In addition to `int` and `float` , Python supports two more types of numbers, `Decimal` and `Fraction` .

We will use the type () function to check which variable or value belongs to the class and isinstance () to check if they belong to any particular class.

```
a = 9 # Output: print(type(a)) # Output: print(type(5.0)) # Output: (10+2j) b = 8
?m tra xem b có ph?i là s? ph?
c không # Output: True print(isinstance(b, complex))
a = 9 # Output: print(type(a)) # Output: print(type(5.0)) # Output: (10+2j) b = 8
?m tra xem b có ph?i là s? ph?
c không # Output: True print(isinstance(b, complex))
a = 9 # Output: print(type(a)) # Output: print(type(5.0)) # Output: (10+2j) b = 8
?m tra xem b có ph?i là s? ph?
c không # Output: True print(isinstance(b, complex))
```

Python integers are not limited to lengths, decimals are limited to 16 numbers after the decimal point.

The numbers they work on are usually a factor of 10, but computer programmers (usually embedded programmers) need to work with binary, hexadecimal and octal systems. To represent these coefficients in

Python, we put an appropriate prefix before that number.

Coefficient prefix for Python numbers:

Number system Binary Prefix '0b' or '0B' Octal system '0o' or '0O' Hexadecimal '0x' or '0X'
(You put the prefix but no sign '').

Here is an example of using coefficient prefixes in Python, and when using the print () function to print their values ??to the screen, we get the corresponding number in the factor of 10.

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

Switch between numeric types in Python

We can convert this number type to another number. This is also called coercion. Operations such as addition and subtraction will implicitly convert integers to decimal numbers (automatically) if there is an operator in the operation that is a decimal.

For example: If you perform the addition of integer 2 and decimal number 3.0, then 2 will be forced to convert into decimal 2.0 and the result will be a 5.0 decimal.

```
>>> 2 + 3.0 5.0
```

We can use built-in Python functions like int (), float () and complex () to switch between numeric types explicitly. These functions can even be converted from strings.

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

When converting from decimal to integer, the number will be dropped, only the integer part is taken.

Python Decimal Module

The built-in float class in Python can surprise us a bit. Normally, if calculating 1.1 and 2.2, we think the result will be 3.3, but it doesn't seem to be so. If you check the correctness and error of this operation in Python, the result will be False.

```
>>> (1.1 + 2.2) == 3.3 False
```

What happened?

This is because decimal numbers are implemented in computer hardware in the form of binary fractions, because computers only understand binary numbers (0 and 1), so most of the decimal fraction we know , cannot be stored correctly in the computer.

For example, we cannot represent 1/3 fraction as a decimal number, since it is an infinite decimal number, with numbers after the decimal point infinitely long, we can only estimate it. .

When converting decimal 0.1, it will lead to infinitely long binary part of 0.0001100110011001100110011 . and the computer only stores part of the finite number after the sign. Its just. Therefore, the stored number is only approximately 0.1 but never equal to 0.1. That is why, the addition we mentioned above does not produce the results we expect. That is the limit of computer hardware, not Python error.

Now try typing the addition above into Python to see what the result is:

```
>>> 1.1+2.2 3.3000000000000003
```

To fix this problem, we can use the Python Decimal module. While the float only takes 16 digits after the decimal point, the Decimal module allows to customize the length of the number.

```
import decimal # Output: 0.1 print(0.1) # Output: 0.1000000000000000055511151231
```

This module is used when we want to perform operations with decimal numbers to get the results we have learned at school.

This is also quite important, for example 25.50kg will be more accurate than 25.5kg, because 2 decimal places are still more accurate than 1 digit.

```
from decimal import Decimal # Output: 3.3 print(Decimal('1.1') + Decimal('2.2'))
```

If you want a more concise code, you can enter the Decimal module and edit the module name to D.

```
from decimal import Decimal as D # Output: 3.3 print(D('1.1') + D('2.2')) # Outpu
```

In this code, enter the Decimal module and edit its name to D, the result is unchanged from the above code.

You can ask in the multiplication section, why not use Decimal number to multiply it by adding 0 after 4 and 2.5. The answer is efficiency, operations with floats are faster than Decimal operations.

When should I use Decimal instead of floats?

We often use Decimal in the following cases:

1. When creating a financial application, it is necessary to represent the decimal part exactly.
2. When you want to control the accuracy of numbers.
3. When you want to do the math just like you did at school.

Fractions in Python

Python provides fractions related to fractions through fractions modules. A fraction with a numerator and denominator, both are integers. We can create Fraction objects in many different ways:

```
 import fractions # T?o phân s? t? s? th?
p phân print(fractions.Fraction(4.5)) # Output: 9/2 # T?o phân s? t? s?
 nguyên # Code by TipsMake.com print(fractions.Fraction(9)) # Output: 9 # T
?o phân s? b?ng cách khai báo t?, m?u s?
 print(fractions.Fraction(2,5)) # Output: 2/5
```

When creating fractions from floats, we can get unusual results, which is due to computer hardware limitations as discussed in the decimal module.

In particular, you can initialize a fraction from the string. This is the preferred initialization method when using decimal numbers.

```
 import fractions # Kh?i t?o phân s? t?
 float print(fractions.Fraction(0.1)) # Output: 3602879701896397/360287970189639
?i t?o phân s? t?
 string # Code by TipsMake.com print(fractions.Fraction('0.1')) # Output: 1/10
```

Fraction data types fully support basic operations such as addition, subtraction, multiplication, division and logic:

```
# Output: 1 print(F(2,5) + F(3,5)) # Output: 3/5 print(F(2,5) + F(1,5)) # Output
```

Math in Python

Python provides math and random modules to solve other math problems such as trigonometry, logarithm, probability and statistics, etc. Since the math module has many functions and attributes, I calculated will do a separate article to list them. Below is an example of math in Python.

```
from fractions import Fraction as F import math # Output: 3.141592653589793 print
```

Use interpreter like pocket calculator

The interpreter acts as a simple calculator: You can enter a calculation and it will write the value. Expression syntax is quite simple: operators like +, -, * and / work like most other programming languages ??(Pascal, C), parentheses () can be used to group . For example:

```
>>> 2 + 2 4 >>> 50 - 5*6 20 >>> (50 - 5*6) / 4 5.0 >>> 8 / 5 # phép chia luôn tr
? v? m?t s? d?ng th?p phân v?i d?u ch?m 1.6
```

Integers (eg `2` , `4` , `20`) are of type `int` , decimal numbers (such as `5.0` , `1.6`) are of type `float` .

Divide (`/`) always returns type float. To perform the partial division (remove the numbers after the decimal point) you can use the operator `//` ; To calculate the remainder, use `%` as the example below:

```
 >>> 17 / 3 # phép chia th??ng tr? v? s? th?
p phân 5.666666666666667 >>> >>> 17 // 3 # phép chia l?y s? nguyên, lo?i b?
```

```
 ph?n sau d?u th?p phân 5 >>> 17 % 3 # toán t? % tr? v? s? d? c?
a phép chia 2 >>> 5 * 3 + 2 # th??ng * s? chia + s? d? 17
```

With Python, you can use the ** operator to calculate the exponent:

```
 >>> 5 ** 2 # 5 bình ph??ng 25 >>> 2 ** 7 # 2 m? 7 128
```

The equal sign = is used to assign a value to a variable. After that, no results will be displayed before the next command prompt:

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

If a variable is not defined (assign value), try to use that variable, you will get the following error:

```
 >>> n # b?n ?ang c? truy c?p vào bi?n n ch?a ???c gán giá tr?
 Traceback (most recent call last): File "", line 1, in NameError: name 'n' is n
```

Python fully supports floating point, the calculation has both integers and decimals, the result will return the number in decimal (verb: operator with mixed type operand converts integer operand to decimal):

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

In interactive mode, the last printed expression will be assigned to variable _ , making it easier to perform further calculations. For example:

```
>>> tax = 12.5 / 100 >>> price = 100.50 >>> price * tax 12.5625 >>> price + _ 11
```

You should treat this variable as read-only, do not assign a value to it - because creating a variable of the same name will take up this default variable and no longer do good things like that.

3. String (string)

String in Python is a sequence of characters. Computers do not handle characters, they only work with binary numbers. Although you can see the characters on the screen, they are stored and processed internally as a combination of numbers 0 and 1. Converting the numeric character is called encoding and the reverse process called decoding. ASCII and Unicode are 2 of the commonly used common encodings.

In Python, string is a sequence of Unicode characters. Unicode includes all characters in all languages ??and provides uniformity in encoding.

How to create a string in Python

Besides numbers, Python can also manipulate strings, represented in many ways. They can be put in single quotes (`'.'`) or double (`"."`) with the same result. used to "escape" these 2 quotes.

```
 >>> 'spam eggs' # d?u nháy ??n 'spam eggs' >>> 'doesn't' # s? d?ng ' ?? vi
?t d?u nháy ??n. "doesn't" >>> "doesn't" # .ho?c s? d?ng d?
u nháy kép "doesn't" >>> '"Yes," he said.' '"Yes," he said.' >>> ""Yes," he said
```

In the interactive interpreter, the result string consists of a quotation mark and special "escape" characters used. Although the output looks a bit different from the input (the enclosed quotes may change), these two strings are equivalent. The string is written in quotation marks when the string contains single quotes and no double quotes, otherwise it will be written in single quotes. The print () function creates a more readable output string, by skipping the apostrophe enclosed and printing special characters, "hiding" the quotes:

```
>>> '"Isn't," she said.' '"Isn't," she said.' >>> print('"Isn't," she said.') "Is
?a là dòng m?i >>> s # không có print(), n s? ???c vi?t trong k?t qu? ??
u ra 'First line.nSecond line.' >>> print(s) # có print(), n s? t?
o ra dòng m?i First line. Second line.
```

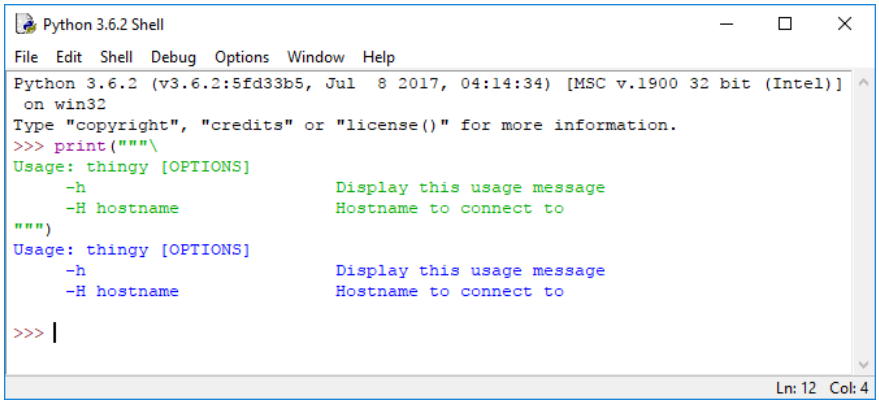
If you do not want the characters added by interpreted by the interpreter as a special character, use the raw string by adding `r` to the first quotation mark:

```
 >>> print('C:somename') # ? ?ây n là dòng m?
i! C:some ame >>> print(r'C:somename') # thêm r tr??c d?u nháy C:somename
```

String strings can be written on multiple lines by using 3 quotes: `"""."""` or `'''.'''` . End lines automatically included in the string, but can prevent this by adding at the end of the line. For example:

```
print(""" Usage: thingy [OPTIONS] -h Display this usage message -H hostname Hostn
```

This is the result (the new original line is not counted):



Here is a list of all escape sequences supported by Python:

Escape SequenceLayewewline Backslash and new line are ignored Backslash 'Single quotation mark' Double quotes 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 The character has an octal value ooo xHH The hexadecimal character is HH
For example: Run each command individually in the compiler to see your results.

```
>>> print("C:Python32TipsMake.com") C:Python32TipsMake.com >>> print("In dòng nà
? x48x45x58") In giá tr? HEX >>>
```

How to access the element of the string

Strings can be indexed with the first character numbered 0. There is no separate character type, each character is simply a number:

```
 >>> word = 'Python' >>> word[0] # ký t? ? v? trí s?
 0 'P' >>> word[5] # ký t? ? v? trí s? 5 'n'
```

Index can also be negative, start counting from the right:

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

Note that since -0 is similar to 0, negative indicators start at -1.

In addition to numbering, slices are also supported. While the index is used to retrieve individual characters, slices will allow you to retrieve substring:

```
 >>> word[0:2] # các ký t? t? v? trí 0 (bao g?m) ??n 2 (lo?i tr?
) 'Py' >>> word[2:5] # các ký t? t? v? trí 2 (bao g?m) ??n 5 (lo?i tr?
) 'tho'
```

Notice how the characters are retained and excluded. It always ensures that `s[:i] + s[i:]` equal to `s` :

```
 >>> word[:2] + word[2:] 'Python' >>> word[:4] + word[4:] 'Python'
```

The index in the chain cut has a pretty default setting, there are 2 indexes that are ignored by default, 0 and the size of the cut string.

```
 >>> word[:2] # các ký t? t? ??u ??n v? trí th? 2 (lo?i b?
) 'Py' >>> word[4:] # các ký t? t? v? trí th? 4(l?y) ??n h?
t 'on' >>> word[-2:] # các ký t? th? hai tính t? cu?i lên (l?y) ??n h?
t 'on'
```

Another way to remember how to cut the work sequence is to visualize the indicators as partitions between characters, with the leftmost character being numbered 0. Then, the last character on the right, in string n characters will have an index n, for example:

```
+---+---+---+---+---+---+ | P | y | t | h | o | n | +---+---+---+---+---+---+ 0 :
```

The first row of numbers brings the position of the index from 0 to 6 in the series. The second row is the corresponding negative indicators. When cutting from i to j will include all characters between i and j, respectively.

For non-negative indicators, the length of a slice is the difference of the indicators, if both are within the limit. For example, the length of word [1: 3] is 2.

Trying to use an index that is too large will return an error:

```
 >>> word[42] # t? ch? có 6 ký t?
 Traceback (most recent call last): File "", line 1, in IndexError: string index
```

However, indexes outside the slice range are still handled neatly when used to cut:

```
 >>> word[4:42] # c?t ký t? t? v? trí th? 4 ??n 42 'on' >>> word[42:] # c?
t ký t? sau v? trí 42 ''
```

Change or delete the string

Python strings cannot be changed - they are fixed.Therefore, if you intentionally assign a character to an indexed location, you will receive an error message:

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

If another string is needed, the best way is to create a new one:

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

You cannot delete or remove characters from the string, but like tuple, you can delete the entire string, using the del keyword:

```
qtm_string = 'quantrimang.com' del qtm_string # Output: NameError: name 'qtm_stri
```

Stringing

Strings can be joined together by the + operator and replaced by * :

```
 >>> # thêm 3 'un' vào sau 'ium' >>> 3 * 'un' + 'ium' 'unununium'
```

Two or more characters in the form of strings (ie, characters in the quotes) are joined automatically.

```
 >>> 'Py' 'thon' 'Python'
```

This feature only works with string literal, not with variables or expressions:

```
 >>> prefix = 'Py' >>> prefix 'thon' # không th? n?i m?t bi?n v?i m?t chu?
i . SyntaxError: invalid syntax >>> ('un' * 3) 'ium' . SyntaxError: invalid synta
```

If you want to join variables together or variables with strings, use the + sign:

```
>>> prefix + 'thon' 'Python'
```

This feature is especially useful when you want to break long strings into shorter strings:

```
>>> text = ('Put several strings within parentheses ' . 'to have them joined togo
```

If you want to join strings in many different lines, use parentheses:

```
>>> # s? d?ng () >>> s = ('Xin ' . 'chào!') >>> s 'Xin chào!'
```

Repeat and check the element of the string

Like list and tuple, you also use the for loop when you need to loop through a string, like for example counting the number of "i" characters in the following string:

```
count = 0 for letter in 'TipsMake.com': if(letter == 'i'): count += 1 # Output: C
? i ???c tìm th?y print('Có', count,'ch? i ???c tìm th?y')
```

To check if a substring is in the string, use the keyword print, as follows:

```
>>> 'quantrimang' in 'quantrimang.com' True >>> 'python' in 'quantrimang.com' Fal
```

Python functions are built-in to work with strings

There are two most common functions when working with strings in Python: enumerate () and len ().

The `len()` function is built into Python, which returns the length of the string:

```
>>> s = 'supercalifragilisticexpialidocious' >>> len(s) 34
```

The enumerate () function returns the listed object, containing the value and index pairs of the element in the string, which is quite useful while looping.

```
qtm_str = 'Python' # enumerate() qtm_enum = list(enumerate(qtm_str)) # Output: li
```

Format () method to format the string

The format () method is very flexible and powerful when used to format strings. The string format contains the {} mark as a placeholder or a replacement field to receive the replacement value. You can also use position or keyword arguments to specify the order.

```
 # default(implicit) order thu_tu_mac_dinh = "{}, {} và {}".format('Qu?
n','Tr?','M?ng') print('n--- Th? t? m?c ??
nh ---') print(thu_tu_mac_dinh) # s? d?ng ??i s? v? trí ?? s?p x?p th? t?
 vi_tri_thu_tu= "{1}, {0} và {2}".format('Qu?n','Tr?','M?
```

```
ng') print('n--- Th? t? theo v? trí ---') print(vi_tri_thu_tu) # s? d?ng t?
 khóa ?? s?p x?p th? t? tu_khoa_thu_tu = "{s}, {b} và {j}".format(j='Qu?
n',b='Tr?',s='M?ng') print('n--- Th? t? theo t?
 khóa ---') print(tu_khoa_thu_tu)
```

We have the results when running the above code as follows:

```
 --- Th? t? m?c ??nh --- Qu?n, Tr? và M?ng --- Th? t? theo v? trí --- Tr?
, Qu?n và M?ng --- Th? t? theo t? khóa --- M?ng, Tr? và Qu?n
```

The format () method can have optional format specifications. They are separated from the field name with a:. For example, you can align left , right> or center ^ a string in the given space. It is possible to format integers such as binary and hexadecimal numbers; Decimals can be rounded or displayed in exponential form. There are many formats you can use.

```
 >>> # ??nh d?ng s? nguyên >>> "Khi chuy?n {0} sang nh? phân s?
 là {0:b}".format(12) 'Khi chuy?n 12 sang nh? phân s? là 1100' >>> # ??nh d
?ng s? th?p phân >>> "S? th?p phân {0} ? d?ng m? s?
 là {0:e}".format(1566.345) 'S? th?p phân 1566.345 ? d?ng m? s?
 là 1.566345e+03' >>> # Làm tròn s? th?p phân >>> "1 ph?
n 3 là: {0:.3f}".format(1/3) '1 ph?n 3 là: 0.333' >>> # c?n ch?nh chu?
i >>> "|{:10}|{:^10}|{:>10}|".format('Qu?n','Tr?','M?ng') '|Qu?n | Tr? | M?
ng|'
```

Old style string format:

You can format the string in Python about sprintf () style used in programming language C with% operator.

```
 >>> x = 15.1236789 >>> print('Giá tr? c?a x là %3.2f' %x) Giá tr? c?
a x là 15.12 >>> print('Giá tr? c?a x là %3.4f' %x) Giá tr? c?
a x là 15.123712.3457
```

The method is often used in string

There are many methods built into Python to work with strings. In addition to the format () mentioned above, there is lower (), upper (), join (), split (), find (), replace (), etc. .

```
>>> "QuanTriMang.Com".lower() 'quantrimang.com' >>> "QuanTriMang.Com".upper() 'Q
?m Com".split() ['Quan', 'Tri', 'Mang', 'Ch?
m', 'Com'] >>> ' '.join(['Quan', 'Tri', 'Mang', 'Ch?
m', 'Com']) 'Quan Tri Mang Ch?m Com'' >>> 'Quan Tri Mang Ch?
m Com'.find('Qua') 0 >>> 'Quan Tri Mang Ch?m Com'.replace('Ch?
m','.') 'Quan Tri Mang . Com'
```

4. List (list)

Python provides a variety of complex data, often called sequences, used to group different values. The most versatile is the list (list).

How to create lists in Python

In Python, the list is represented by a series of values, separated by commas, within []. Lists may contain multiple items of different types, but usually items with the same type.

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

List unlimited number of items, you can have many different data types in the same list, such as string, integer, decimal, .

```
list1 = [] # list r?ng list2 = [1, 2, 3] # list s?
nguyên list3 = [1, "Hello", 3.4] # list v?i ki?u d? li?u h?n h?p
```

You can also create nested lists (lists contained in the list), for example:

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

Or declare a nested list from the beginning:

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

Access the list element

There are many different ways to access the element of a list:

Index (index) of the list:

Use the index [] operator to access an element of the list. Index starts at 0, so a list with 5 elements will have an index of 0 to 4. Accessing an index element other than the index of the list will give rise to an IndexError error. Index must be an integer, cannot use float, or other data type, will create a TypeError error.

```
qtm_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # TypeE
?a list ph?i là s? nguyên ho?c slice, không ph?i s? th?p phân qtm_list[2.0]
```

Nested lists can be accessed by nested index:

```
qtm_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Outpu
?ng nhau ln_list = ["Happy", [1,3,5,9]] # Index l?
ng nhau # Output: a print(ln_list[0][1]) # Output: 9 print(ln_list[1][3])
```

Negative Index:

Python allows negative indexing for strings. Index -1 is the last element, -2 is the last element from the last. Simply said, index is negative when you count the element of the string backwards from the beginning to the

beginning.

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

Slicing lists in Python

Python allows access to a range of elements of the list by using the slice operator: (colon). Every list cut action returns a new list containing the required elements.

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

To slice the list, you only need to use the sign: between the two indexes to retrieve the elements. [1: 5] will take elements 1 to 5, [: -8] taken from 0 to -8, .

If the following cut action is performed, it will return a new list as a copy of the original list:

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

Change or add elements to the list

List also supports operations like list join:

```
 >>> squares + [36, 49, 64, 81, 100] [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Unlike strings, fixed, the list is a data type that can be changed. For example, you can change items in the list:

```
 >>> cubes = [1, 8, 27, 65, 125] # có v? sai sai >>> 4 ** 3 # l?p ph??ng c?
a 4 là 64, không ph?i 65! 64 >>> cubes[3] = 64 # thay th? giá tr?
 sai >>> cubes [1, 8, 27, 64, 125]
```

You can also add new items to the end of the list using methods, such as append ():

```
 >>> cubes.append(216) # thêm l?p ph??ng c?
a 6 >>> cubes.append(7 ** 3) # và l?p ph??ng c?
a 7 >>> cubes [1, 8, 27, 64, 125, 216, 343]
```

Assigning slices can also be done and can even change the size of the list or delete it completely:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g'] >>> letters ['a', 'b', 'c', 'd
? vài giá tr?
 >>> letters[2:5] = ['C', 'D', 'E'] >>> letters ['a', 'b', 'C', 'D', 'E', 'f', 'g
?
 thì xóa chúng >>> letters[2:5] = [] >>> letters ['a', 'b', 'f', 'g'] >>> # xóa
?ng cách thay t?t c? các ph?n t? b?ng m?t list r?
ng >>> letters[:] = [] >>> letters []
```

Wool functions () can also be applied to lists:

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

Delete or remove an element from the list in Python

You can delete one or more elements from the list using the del keyword, you can delete the list completely.

```
my_list = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # xóa ph
?n t?
 có index là 2 del my_list[2] # Output: ['q', 'u', 'n', 't', 'r', 'i', 'm', 'a',
?n t? có index t? 1 ??
n 7 del my_list[1:7] # Output: ['q', 'a', 'n', 'g', '.', 'c', 'o', 'm'] print(my_
?
 list my_list del my_list # Error: NameError: name 'my_list' is not defined print
```

You can also use remove () to remove a given element or pop () to remove an element at a certain index. pop () removes the element and returns the last element if the index is not specified. This helps to deploy the list in the form of a stack (data structure first in last out - first, last).

In addition, the clear () method is also used to empty a list (deleting all elements in the list).

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

The last way to delete elements in a list is to assign an empty list to the element slices.

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

List method in Python

The methods available for lists in Python include:

1. **append ()** : Add an element to the end of the list.
2. **extend ()** : Add all elements of the current list to another list.
3. **insert ()** : Insert an element into the given index.
4. **remove ()** : Delete the element from the list.
5. **pop ()** : Delete the element from the list and return the element at the given index.
6. **clear ()** : Delete all elements of the list.
7. **index ()** : Returns the index of the first matching element.
8. **count ()** : Returns the number of counted elements in the list as an argument.
9. **sort ()** : Arrange the elements in the list in ascending order.
10. **reverse ()** : Reverse the order of elements in the list.
11. **copy ()** : Returns the copy of the list.

For example:

```
QTM = [9,8,7,6,8,5,8] # Output: 2 print(QTM.index(7)) # Output: 3 print(QTM.coun
```

Example 2:

```
QTM = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Output: 3 
```

List comprehension: How to create a new list briefly

List comprehension is an expression associated with a for statement that is enclosed in square brackets [].

For example:

```
cub3 = [3 ** x for x in range(9)] # Output: [1, 3, 9, 27, 81, 243, 729, 2187, 65
```

The above code is equivalent to:

```
 cub3 = [] for x in range (9): cub3.append(3**x) print(cub3)
```

In addition to for, if can also be used in a list comprehension of Python. The if command can filter the elements in the current list to create a new list. Here is an example:

```
cub3 = [3 ** x for x in range(9) if x > 4] # Output: [243, 729, 2187, 6561] print
? ','L?p trình '] for y in ['Python','C++']] # Output: ['Ngôn ng?
 Python', 'Ngôn ng? C++', 'L?p trình Python', 'L?
p trình C++'] print(noi_list)
```

Check if the element is in the list

Use the keyword in to check if an element is already in the list. If the element already exists, the result is True, and vice versa will return False.

```
QTM = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Output: Tru
```

For loop in list

Use the for loop to loop through the elements in the list as shown below:

```
 for ngon_ngu in ['Python','Java','C']: print("Tôi thích l?
p trình",ngon_ngu)
```

The result returned will be as follows:

```
 Tôi thích l?p trình Python Tôi thích l?p trình Java Tôi thích l?p trình C
```

Python functions integrated with the list

Built-in Python functions like all (), any (), enumerate (), len (), max (), min (), list (), sorted (), . are often used with lists for real Show different tasks.

1. **all ()** : Returns true if all elements of the list are true or list empty.
2. **any ()** : Returns True when any element in the list is true. If the list is empty the function returns False.
3. **enumerate ()** : Returns the enumerate, containing index object and the value of all elements of the list as tuple.
4. **len ()** : Returns the length (number of elements) of the list.
5. **list ()** : Convert an object that can loop (tuple, string, set, dictionary) into a list.
6. **max ()** : Returns the largest element in the list.
7. **min ()** : Returns the smallest element in the list.
8. **sorted ()** : Returns the new sorted list.
9. **sum ()** : Returns the sum of all elements in the list.

5. Tuple

Tuple in Python is a sequence of ordered elements like list. The difference between list and tuple is that we cannot change the elements in the tuple when assigned, but in the list, the elements can change.

Tuple is often used for data that does not allow modifications and is faster than lists because it cannot be changed automatically. A tuple is defined by parentheses (), elements in tuple are separated by commas (,).

For example:

```
t = (10, "quan tri mang", 2j)
```

You can use the [] cut operator to extract the element in tuple but cannot change its value.

```
t = (10, "quan tri mang", 2j) #t[0:2] = (10, 'quan tri mang') print("t[0:2] = ",
```

Run the above code we get the result:

```
t[0:2] = (10, 'quan tri mang')
```

Where is Tuple more than the list?

Because tuples and lists are quite similar, they are often used in similar situations. However, tuples still have certain advantages over the list, such as:

1. Tuple is often used for heterogeneous (different) data types and lists often use for identical (identical) data types.
2. Since tuple cannot be changed, iterating through tuple elements is faster than list. So in this case, tuple is a bit more dominant than the list.

3. Tuple contains unchanged elements, which can be used as keys for dictionary. With the list, this cannot be done.
4. If there is data that does not change its deployment as a tuple, it will ensure that the data is write-protected.

Create a tuple

Tuple is created by placing all of its elements in parentheses (), separated by commas. You can remove parentheses if you want, but add it to make the code more clear.

Tuple is not limited to the number of elements and can have many different data types such as integers, decimals, lists, strings, .

```
 # Tuple r?ng # Output: () my_tuple = () print(my_tuple) # tuple s?
 nguyên # Output: (2, 4, 16, 256) my_tuple = (2, 4, 16, 256) print(my_tuple) # tu
?u ki?u d? li?
u # Output: (10, "TipsMake.com", 3.5) my_tuple = (10, "TipsMake.com", 3.5) print
?
ng nhau # Output: ("QTM", [2, 4, 6], (3, 5, 7)) my_tuple = ("QTM", [2, 4, 6], (3,
? ???c t?o mà không c?n d?u () # còn g?i là ?
óng gói tuple # Output: (10, "TipsMake.com", 3.5) my_tuple = 10, "TipsMake.com",
? gói (unpacking) tuple c?ng có th? làm ???
c # Output: # 10 # TipsMake.com # 3.5 a, b, c = my_tuple print(a) print(b) print
```

Creating a tuple has only a little bit of a complicated element, if you create the usual way of putting that element in the sign () is not enough, you need to add a comma to indicate that this is tuple.

```
 # t?o tuple ch? v?
i () # Output: my_tuple = ("TipsMake.com") print(type(my_tuple)) # khi thêm d
?u ph?y vào cu?
i # Output: my_tuple = ("TipsMake.com",) print(type(my_tuple)) # d?
u () là tùy ch?n, b?n có th? b? n?
u thích # Output: my_tuple = "TipsMake.com", print(type(my_tuple))  # t?
o tuple ch? v?
i () # Output: my_tuple = ("TipsMake.com") print(type(my_tuple)) # khi thêm d
?u ph?y vào cu?
i # Output: my_tuple = ("TipsMake.com",) print(type(my_tuple)) # d?
u () là tùy ch?n, b?n có th? b? n?
u thích # Output: my_tuple = "TipsMake.com", print(type(my_tuple))  # t?
o tuple ch? v?
i () # Output: my_tuple = ("TipsMake.com") print(type(my_tuple)) # khi thêm d
?u ph?y vào cu?
i # Output: my_tuple = ("TipsMake.com",) print(type(my_tuple)) # d?
u () là tùy ch?n, b?n có th? b? n?
u thích # Output: my_tuple = "TipsMake.com", print(type(my_tuple))  # t?
o tuple ch? v?
i () # Output: my_tuple = ("TipsMake.com") print(type(my_tuple)) # khi thêm d
?u ph?y vào cu?
i # Output: my_tuple = ("TipsMake.com",) print(type(my_tuple)) # d?
u () là tùy ch?n, b?n có th? b? n?
u thích # Output: my_tuple = "TipsMake.com", print(type(my_tuple))
```

Access the elements of tuple

There are many different ways to access the elements of a tuple, quite similar to the list, so I don't take specific examples, you can review the list of the list.

Index: S? d?ng toán t? index [] ?? truy c?p vào ph?n t? trong tuple v?i index b?t ??u b?ng 0. Ngh?a là n?u tuple có 6 ph?n t? thì index c?a nó s? b?t ??u t? 0 ??n 5. N?u c? g?n truy c?p ??n index 6, 7 thì s? t?o l?i IndexError. Index b?t bu?c ph?i là s? nguyên, mà không th? là s? th?p phân hay b?t k? ki?u d? li?u nào khác, n?u không s? t? o l?i TypeError. Nh?ng tuple l?ng nhau ???c truy c?p b?ng cách s? d?ng index l?ng nhau:

```
 # tuple l?
ng nhau n_tuple = ("TipsMake.com", [2, 6, 8], (1, 2, 3)) # index l?
ng nhau # Output: 'r' print(n_tuple[0][5]) # index l?
ng nhau # Output: 8 print(n_tuple[1][2])
```

Index âm: Python cho phép l?p ch? m?c âm cho các ??i t??ng d?ng chu?i. Index -1 tham chi?u ??n ph?n t? cu?i cùng, -2 là th? 2 tính t? cu?i tính lên.

C?t lát: Có th? truy c?p ??n m?t lo?t ph?n t? trong tuple b?ng cách s? d?ng toán t? c?t lát : (d?u 2 ch?m).

Thay ??i m?t tuple

Không gi?ng nh? list, tuple không th? thay ??i. ?i?u này có ngh?a là các ph?n t? c?a m?t tuple không th? thay ??i m?t khi ?ã ???c gán. Nh?ng, n?u b?n thân ph?n t? ?ó là m?t ki?u d? li?u có th? thay ??i (nh? list ch?ng h?n) thì các ph?n t? l?ng nhau có th? ???c thay ??i. Chúng ta c?ng có th? gán giá tr? khác cho tuple (g?i là gán l?i - reassignment).

```
 my_tuple = (1, 3, 5, [7, 9]) # không th? thay ??i ph?n t? c?a tuple # N?
u b?n b? d?u # ? dòng 8 # B?n s? nh?n ???c l?
i: # TypeError: 'tuple' object does not support item assignment #my_tuple[1] = 9
?ng ph?n t? có index 3 trong tuple là list # list có th? thay ??i, nên ph?
n t? ?ó có th? thay ??
i # Output: (1, 3, 5, [8, 9]) my_tuple[3][0] = 8 print(my_tuple) # N?u c?
n thay ??i tuple hãy gán l?i giá tr?
 cho nó # Output: ('q', 'u', 'a', 'n', 't', 'r', 'i', 'm', 'a', 'n', 'g') my_tup
```

B?n có th? dùng toán t? + ?? n?i 2 tuple, toán t? * ?? l?p l?i tuple theo s? l?n ?ã cho. C? + và * ??u cho k?t qu? là m?t tuple m?i.

```
 # N?
i 2 tuple # Output: (2, 4, 6, 3, 5, 7) print((2, 4, 6) + (3, 5, 7)) # L?p l
?
i tuple # Output: ('TipsMake.com', 'TipsMake.com', 'TipsMake.com') print(("TipsMa
```

Xóa tuple

Các ph?n t? trong tuple không th? thay ??i nên chúng ta c?ng không th? xóa, lo?i b? ph?n t? kh?i tuple. Nh?ng vi ?c xóa hoàn toàn m?t tuple có th? th?c hi?n ???c v?i t? khóa del nh? d??i ?ây:

```
QTM = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Không th
? xóa ph?n t? c?a tuple # N?u b?n b? d?u # ? dòng 8, # s? t?o ra l?
i: # TypeError: 'tuple' object doesn't support item deletion #del QTM[3] # Có th
? xóa toàn b? tuple # K?t qu? ch?y code s? hi?n ra l?
i: # NameError: name 'my_tuple' is not defined del QTM QTM
```

Ph??ng th?c và hàm dùng v?i tuple trong Python

Ph??ng th?c thêm ph?n t? và xóa ph?n t? không th? s? d?ng v?i tuple, ch? có 2 ph??ng th?c sau là dùng ???c:

1. **count(x)** : ??m s? ph?n t? x trong tuple.
2. **index(x)** : Tr? v? giá tr? index c?a ph?n t? x ??u tiên mà nó g?p trong tuple.

```
QTM = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Count # Ou
```

Các hàm dùng trong tuple khá gi?ng v?i list, g?m có:

1. **all()** : Tr? v? giá tr? True n?u t?t c? các ph?n t? c?a tuple là true ho?c tuple r?ng.
2. **any()** : Tr? v? True n?u b?t k? ph?n t? nào c?a tuple là true, n?u tuple r?ng tr? v? False.
3. **enumerated()** : Tr? v? ??i t??ng enumerate (li?t kê), ch?a c?p index và giá tr? c?a t?t c? ph?n t? c?a tuple.
4. **len()** : Tr? v? ?? dài (s? ph?n t?) c?a tuple.
5. **max()** : Tr? v? ph?n t? l?n nh?t c?a tuple.
6. **min()** : Tr? v? ph?n t? nh? nh?t c?a tuple.
7. **sorted()** : L?y ph?n t? trong tuple và tr? v? list m?i ???c s?p x?p (tuple không s?p x?p ???c).
8. **sum()** : Tr? v? t?ng t?t c? các ph?n t? trong tuple.
9. **tuple()** : Chuy?n ??i nh?ng ??i t??ng có th? l?p (list, string, set, dictionary) thành tuple.

Ki?m tra ph?n t? trong tuple

B?n có th? ki?m tra xem m?t ph?n t? ?ã t?n t?i trong tuple hay ch?a v?i t? khóa in.

```
 QTM = ['q','u','a','n','t','r','i','m','a','n','g','.','c','o','m'] # Ki?
m tra ph?n t?
 # Output: True print('a' in QTM) # Output: False print('b' in QTM) # Not in ope
```

L?p qua các ph?n t? c?a tuple trong Python

S? d?ng vòng l?p for ?? l?p qua các ph?n t? trong tuple.

```
 for ngon_ngu in ('Python','C++','Web'): print("Tôi thích l?
p trình",ngon_ngu)
```

K?t qu? tr? v? s? nh? sau:

```
 Tôi thích l?p trình Python Tôi thích l?p trình C++ Tôi thích l?p trình Web
```

6. Set

Set trong Python là t?p h?p các ph?n t? duy nh?t, không có th? t?. Các ph?n t? trong set phân cách nhau b?ng d?u ph?y và n?m trong d?u ngo?c nh?n {}. Nh? k? r?ng các ph?n t? trong set không có th? t?. Nh?ng các ph?n t? trong set có th? thay ??i, có th? thêm ho?c xóa ph?n t? c?a set. Set có th? ???c s? d?ng ?? th?c hi?n các phép toán nh? t?p h?p, giao,.

Cách t?o set

Set ???c t?o b?ng cách ??t t?t c? các ph?n t? trong d?u ngo?c nh?n, phân tách b?ng d?u ph?y ho?c s? d?ng hàm set(). Set không gi?i h?n s? l??ng ph?n t?, nó có th? ch?a nhi?u ki?u bi?n khác nhau, nh?ng không th? ch?a ph?n t? có th? thay ??i ???c nh? list, set hay dictionary.

Ví d? v? set:

```
a = {5,2,3,1,4}
```

N?u th?c hi?n l?nh in nh? sau:

```
print("a=", a)
```

div="">