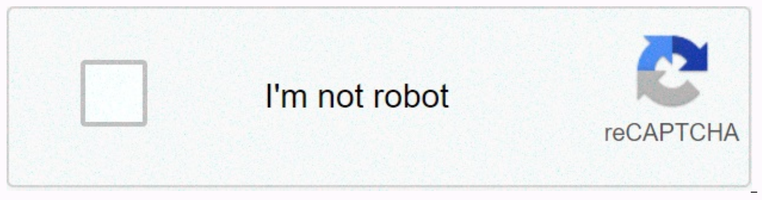

Continue

Python string format list comprehension

A list is exactly what it sounds like, a container containing different Python objects, which could be integers, words, values, etc. It is the equivalent of a table in other programming languages. It is represented by brackets (and this is one of the characteristics that differentiates it from tuples, which are separated by parentheses). It is also variable, i.e. it can be modified or updated. unlike birds, which are unchanged. In this python tutorial, you will learn: Examples of python lists: Python lists can be homogeneous, which means they can contain the same type of objects; or heterogeneous, including different types of objects. Examples of homogeneous lists include: list of integers = [1, 2, 3, 8, 33] list of animals = ['dog', 'cat', 'goat'] list of names = ['John', 'Travis', 'Sheila'] list of variable numbers = [2.2, 4.5, 9.8, 10.4] Examples of heterogeneous directories include: [2, 'cat', 34.33, 'Travis'] [2.22, 33, 'pen'] Access to values within lists To access in-directory values, the index of objects within the directories can be used. An index in Python lists refers to the location of an item within a sorted list. For example: list = [3, 22, 30, 5.3, 20] The first value in the above list, 3, has an index of 0Th second value, 22, has index 1Th third value, 30, has index 2and so on. To access each of the values in the list, you would use: the list[0] to access list 3[1] to access list 22[2] to access list 30[3] to access list 5.3[4] to access 20 The last member of a list can also be accessed using index -1. For example, list[-1] = 20 Slicing list is the method of separating a subset of a list, and the markers of list objects are also used for it. For example, using the same list example above. directory[:] = [3, 22, 30, 5.3, 20] (all members of the list]; list[1:3] = [22, 30] (list members from index 1 to index 3, without member in index 3); list[:4] = [3, 22, 30, 5.3] (members of the list from index 0 to index 4, without the member in index 4) list[2:-1] = [30, 5.3] (members of the list from index 2, which is the third item, in the second to last item in the list, which is 5.3). Python lists are exclusive up, and this means that the last pointer when slicing the list is usually ignored. That is why list[2:-1] = [30, 5.3] , and not [30, 5.3, 20]. The same applies to all other examples of shredding list listed above. Update listsS say you have a list = chemistry, mathematics], and you want to change the list to [biology, chemistry, mathematics], effectively changing the member to index 0. This can easily be done by assigning this index to the new member you want. That is, list = [physics, chemistry, mathematics] list[0] = biology printing(list) Production: [biology, chemistry, mathematics] This replaces the member in index 0 (physics) with the new value you want (chemistry). This can be done for any member or subset of the list you want to change. To give another Suppose you have a list called integers that contains the numbers [2, 5, 9, 20, 27]. To replace 5 in this list with 10, you can do so with: integers = [2, 5, 9, 20, 27] integers[1] = 10 print(integers) >>> [2, 10, 9, 20, 27] To replace the last member of the integer list, which is 27, with a free number such as 30.5, you would use: integers = [26] , 5, 9, 20, 27] integers[-1] = 30.5 print(integers) >>> [2 , 5, 9, 20, 30.5] Delete list itemsThere are 3 Python methods for deleting list items: list.remove(), list.pop(), and del operator. For example: directory = [3, 5, 7, 8, 9, 20] To delete 3 (the 1st item) from the list, you could use: list.remove(3) or list.pop[0] or del list[0]To delete 8, item in index 3, from the list, you could use: list.remove(8), or list.pop[3]ing list Append items To append items to a list , the method is used, and this adds the item to the end of the list. For example: list_1 = [3, 5, 7, 8, 9, 20] list_1.append(3.33) printing(list_1) >>> list_1 = [3, 5, 7, 8, 9, 20, 3.33] list_1.append(cats) print(list_1) >>> list_1 = [3, 5, 7, 8, 9, 20, 3.33, cats] List of built-in functions (methods)The following is a list of built-in functions and methods with their descriptions : len(list): this gives the length of the directory as output. For example: numbers = [2, 5, 7, 9] print(len(numbers)) >>> 4 max(list): returns the item to the list with the maximum value. For example: numbers = [2, 5, 7, 9] print(max(numbers)) >>> 9 minutes(list): returns the item to the list with the minimum value. For example: numbers = [2, 5, 7, 9] print(min(numbers)) >>> 2 list(number): converts a multitude object to a list. For example, for example. animals = (cat, dog, fish, cow) printing (list(animals)) >>> [cat, dog, fish, cow] list.append(item): appends the item to the list. For example, for example. numbers = [2, 5, 7, 9] numbers.append(15) print(numbers) >>> [2, 5, 7, 9, 15] list.pop(index): removes the item in the specified index from the list. For example, for example. numbers = [2, 5, 7, 9, 15] numbers.pop(2) print(numbers) >>> [2, 5, 9, 15] list.remove(element):d displays the item from the list. For example, for example. values = [2, 5, 7, 9] values.remove(2) print(values) >>> [5, 7, 9] list.reverse(): reverses the objects in the list. For example, for example. values = [2, 5, 7, 10] values.reverse() print(values) [10, 7, 5, 2] list.index(element): to get the index value of an item within the list. For example, for example. animals = ['cat', 'dog', 'fish', 'cow', 'goat'] fish_index = animals.index('fish') print(fish_index) >>> 2 sum(list): to obtain the sum of all the values in the list, if the values are all numbers (integers or decimals). For example, for example. values = [2, 5, 10] sum_of_values = print(sum_of_values) >>> 17 If the list contains an item that is not a number, such as a string, the sum method would not work. You will receive a mistake TypeError: unsupported operand type for +: 'int' and 'str'' list.sort(): to arrange a list of integers, floating-point numbers, or strings in ascending or descending order. For example: values = [1, 7, 9, 3, 5] # To sort values in ascending order: values.sort() print(values) >>> [1, 3, 5, 7, 9] Another example: values = [2, 10, 7, 14, 50] # To sort values in descending order: values.sort (reverse = True) print(values) >>> [50, 14, 10, 7, 2] A list of strings can also be sorted alphabetically or along strings. For example, for example. # to sort the list along the data strings = ['cat', 'mammal', 'goat', 'is'] sort_by_alphabet = strings.sort() sort_by_length = strings.sort(key = len) print(sort_by_alphabet) print(sort_by_length) >>> ['cat', 'goat', 'is', 'mammal'] ['is', 'cat', 'goat', 'mammal'] We can sort the same directory alphabetically using 'strings'. Looping through listsLooping through directories can be done in exactly the same way as any other looping operation in Python. This way, a method can run on multiple items in a list at once. For example: list = [10, 20, 30, 40, 50, 60, 70]. To loop through all the items in this list, and let's say, add 10 to each item: for elem in the list: elem = elem + 5 print (elem) >>>>15 25 35 45 55 65 75 To loop through the first three items in the list, and delete all of them; for elem in directory[:3]: list.remove(elem) >>> list = [40, 50, 60, 70] To loop through the 3rd (index 2) to the last item in the list and append them to a new list called new_list: new_list = [] for elem in the list[2:]: print new_list.append(elem) (New list: {}.format(new_list)) Output: New list: [30, 40, 50, 60, 70] In this way, any method or function can be applied to members of a list to perform a specific operation. You can either repeat all members of the list or

loop a subset of the list by using list shredding. List understandings are Python functions that are used to create new sequences (such as lists, dictionaries, etc.) using sequences that have already been created. They help reduce larger loops and make it easier to read and maintain your code. For example, for example. Suppose you wanted to create a list that contains the squares of all numbers from 1 to 9: list_of squares = [] for int in range(1, 10): square = int ** 2 list_of_squares.append(square) print(list_of_squares) List_of_squares using for loop: [1, 4, 9, 16, 25, 36, 49, 64, 81] To do the same thing with directory understandings: list_of_squares_2 = for int in the area(1, 10)] print('List of squares using directory comprehension: {}'.format(list_of_squares_2)) Output using directory comprehension: [1, 4, 9, 16, 25, 36, 49, 64, 81] As shown above, writing the code using the list understandings is much smaller than using traditional loops, and is also faster. This is just one example of using list understandings in place for loops, but this can be replicated and used in many places where loops can also be used. Sometimes going with one for loop is the best option, especially if the code is complicated, but in many cases, the list understandings will make your encoding easier and faster. The following is a table that contains some list functions and methods and their descriptions. Built-in function function description Round() Rounds the number passed as an argument to a specified number of digits and returns the minimum return element of the Min() floating-point value of a given Max() list return item function , therefore we do not need to manually measure Filter() tests if each item in a list is true or not Lambda An expression that can appear in places where a def (to create functions) is not syntax , within a list literally or the call arguments of a Map(function) returns a list of results after applying the given function to each element of a given editable Filter() to apply a specific function passed to its argument in all list items returns a list of list items that contains the intermediate results Sum() Returns the sum of all numbers in the Cmp() list This is used to compare two lists and returns 1 if the first list is longer than the second list. Insert an import item into the list in a specific location List MethodsSYNOGRAPH APPENDIX() Adds a new item to the end of the Clear() list Removes all items from the Copy() list Returns a copy of the original Extend() list Add multiple items at the end of the Count() list Returns the number of occurrences of a specific item in an Index() Returns the index of a specific item in a Pop list() Deletes the item from the list in a specific index (delete by location) Remove() Deletes the specified item from the list (delete by value) On-site reversal method that reverses the order of items in the SummaryA list is exactly what it sounds like, a container containing different Python objects, which could be integers, words, values, etc. Python lists can be homogeneous, meaning they can contain the same type of objects; or heterogeneous, containing different types of objects. To access values within lists, the index of objects within the lists can be used. List slicing is the method of separating a subset of a list, which is why markers of list objects are also used. Three methods for deleting list items are: 1)list.remove(), 2)list.pop() and 3)del operator The append method is used to append items. This adds the item to the end of the list. The Python program looping can run on multiple items in a data list at once. List understandings are Python functions that are used to create new sequences (such as lists, dictionaries, etc.) by using sequences that have already been created. Created. Created.