PythonForBeginners

(https://www.pythonforbeginners.com/
)

List Comprehension in Python

Author: Josh Petty (https://www.pythonforbeginners.com/author/josh)
Last Updated: June 22, 2023

Python provides us with different tools for data manipulation. One such tool is list comprehension. In this article, we will discuss the basics of list comprehension with syntax and examples.

Table of Contents

Chances are you'll use a lot of lists as a Python programmer. While we're all big fans of for loops (and nested for loops), Python provides a more concise method for handling lists such as list comprehension. In order to keep your code elegant and readable, it's recommended that you use Python's comprehension features.

Syn

List comprehension is a powerful and concise method for creating lists in Python that becomes essential the more you work with lists, and lists of lists. It has the following syntax:

(https://www.pythonforbeginners.com/courses/python-3-for-beginners)

```
my_new_list = [ expression for item in iterable_object ]
```

In the above syntax,

- The literal iterable_object represents an iterable object such as a list, tuple, set, etc. We use the elements of the iterable_object to create a new list.
- The term item represents an element of the iterable_object.
- The expression represents the operation that we perform on the item. You can perform mathematical operations on item or pass it to a function.
- my_new_list is the list created by executing expression on elements of iterable_object.

List comprehensions can also contain conditional statements. For instance, consider the following syntax.

```
my_new_list = [ expression for item in iterable_object if condition]
```

In the above syntax, if condition is True, only then the output of expression is stored in my_new_list. Otherwise, the item and expression for which condition is False are discarded from the output.

To understand the list comprehension, imagine that you're going to perform an expression on each item in the list. The expression will determine what item is eventually stored in the output list. For instance, consider the following example.

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[item**2 for item in myList]
print("The output list is:",newList)
```

Output:

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [1, 4, 9, 16, 25, 36]
```

In the above example, we have created newList using the elements of myList. You can observe that we have squared all the elements of myList and stored them in newList in a single Python statement.

If you compare the statement `newList=[item**2 for item in myList]` with the syntax for list comprehension, you can easily understand how it works.

Now, suppose that we want to find the squares of only the even numbers in the original list. In this case, we will use the list comprehension containing the conditional statement as shown below.

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[item**2 for item in myList if item%2==0]
print("The output list is:",newList)
```

Output:

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [4, 16, 36]
```

In the above code, we introduced the conditional statement item%2==0 in place of condition. DUe to this, the odd elements for which condition is False are discarded from the output.

Not only can you perform expressions on an entire list in a single line of code, but, as we'll see later, it's possible to add conditional statements in the form of filters, which allows for more precision in the way lists are handled.

- Comprehension methods are an elegant way to create and manage lists.
- In Python, list comprehensions are a more compact way of creating lists.
- More flexible than for loops, list comprehension is usually faster than other methods.

List Comprehension Examples in Python

To understand list comprehension in a better manner, let us discuss some examples.

Creating a List With List Comprehension

Let's start by creating a list of numbers using Python list comprehensions. We'll take advantage of Python's **range()** method as a way to create a list of digits.

```
# construct a basic list using range() and list comprehensions
# syntax [ expression for item in list ]
digits = [x for x in range(10)]
print(digits)
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Let's break down this Python example by starting with the first 'x'. This is our expression. It doesn't do anything because we're simply recording the number. The second 'x' represents each item in the list created by the **range()** method.

In the Python example above, we're using the **range()** method to generate a list of numbers. Python iterates(or loops (https://www.pythonforbeginners.com/loops/for-while-and-nested-loops-in-python)) through each item in that range, and saves a copy of the item in a new list called digits.

Perhaps that seems redundant? That's only because you've yet to see the real potential of list comprehension.

Comparing List Creation Methods in Python

To better illustrate how list comprehension can be used to write more efficient Python code, we'll take a look at a side-by-side comparison. In the following example, you'll see two different techniques for creating a Python list.

First, we will use a for loop to create a new list with squares of elements of an existing list as shown below.

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[]
for item in myList:
    square=item**2
    newList.append(square)
print("The output list is:",newList)
```

Output:

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [1, 4, 9, 16, 25, 36]
```

Here, we have used a for loop to traverse through the elements of the existing list. Then, we used the append() method to append the square of each element into the new list.

We can perform the same task using a list comprehension in a single statement as shown below

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[item**2 for item in myList]
print("The output list is:",newList)
```

Output:

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [1, 4, 9, 16, 25, 36]
```

Even in this basic example, it's obvious that list comprehensions reduce the code necessary to complete a rather complicated task when working with a list.

Multiplication With List Comprehensions

What if we wanted to multiply every number in a list by a number in Python? We could write a for loop and store the results in a new list. Or, we could use list comprehensions to multiply all the elements of a list with a number as shown below.

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[item*3 for item in myList]
print("The output list is:",newList)
```

Output:

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [3, 6, 9, 12, 15, 18]
```

In the above example, we have multiplied all the elements of the existing list by three. What if we want to multiply only a part of the existing list by a number? For this, we can use the condition parameter discussed in the syntax above.

For instance, if you only wanted to multiply only the even numbers in a given list, you can use the conditional statement with list comprehension as shown below.

```
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[item*3 for item in myList if item%2==0]
print("The output list is:",newList)
```

Output

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [6, 12, 18]
```

In this output, you can observe that only the even numbers of the original list are multiplied by 3 and stored in the output list. The rest of the elements are discarded from the output.

Suggested Reading: Set Comprehension in Python (https://www.pythonforbeginners.com/basics/set-comprehension-in-python)

Show The First Letter of Each word Using Python

So far we've seen examples of building a list of numbers in Python. Next, let's try working with strings and the various ways list comprehensions can be used to elegantly handle a list of strings.

We can select the first letter of the words from a list of words. For this, we can use the string indexing (https://www.pythonforbeginners.com/strings/string-indexing-in-python) operator to select the first character for each word in any given list as shown in the following example.

```
# a list of the names of popular authors
authors = ["Ernest Hemingway","Langston Hughes","Frank
Herbert","Toni Morrison",
    "Emily Dickson","Stephen King"]

# create an acronym from the first letter of the author's names
letters = [ name[0] for name in authors ]
print(letters)
```

Output

```
['E', 'L', 'F', 'T', 'E', 'S']
```

List comprehensions can make solving issues involving strings much easier using simplified expressions. These methods can save time and precious lines of code.

Separating The Characters in a String Using List Comprehension

We can also separate the characters in a string using list comprehension. As strings are iterable objects, we can iterate over all the characters of the string using list comprehension and store them in an output list as shown below.

```
myStr="PythonForBeginners"
print("The string is:",myStr)
newList=[character for character in myStr]
print("The characters are:",newList)
```

Output:

```
The string is: PythonForBeginners
The characters are: ['P', 'y', 't', 'h', 'o', 'n', 'F', 'o', 'r', 'B',
'e', 'g', 'i', 'n', 'n', 'e', 'r', 's']
```

Lower/Upper Case Converter Using List Comprehension in Python

To convert all the words in a list of strings to lowercase or uppercase, we can use the lower() method or upper() method. To convert elements of a list into lowercase, we can invoke the lower() method on the string in list comprehension. Similarly, we can use the upper() method to convert elements of a list to uppercase as shown below.

```
lower_case = [ letter.lower() for letter in ['A','B','C'] ]
upper_case = [ letter.upper() for letter in ['a','b','c'] ]
print(lower_case, upper_case)
```

Output

```
['a', 'b', 'c'] ['A', 'B', 'C']
```

Suggested Reading: Dictionary Comprehension in Python (https://www.pythonforbeginners.com/dictionary/dictionary-comprehension-in-python)

Print Numbers Only From a Given String in Python

Another interesting exercise is to extract numbers from a string using list comprehension. For instance, we may have a database of names and phone numbers. It would be useful if we could separate the phone numbers from the names. Using list comprehensions, we can do just that.

Taking advantage of Python's **isdigit()** method, we can extract the phone number from the user data. The isdigit() method, when invoked on a string, returns True if it only contains digits.

To extract numbers from a given string, we will use list comprehension and invoke the isdigit() method on the characters in the if condition. After this, we will only get digits in the output list. You can observe this in the following example.

```
# user data entered as name and phone number
user_data = "Elvis Presley 987-654-3210"
phone_number = [ x for x in user_data if x.isdigit()]
print(phone_number)
```

Output

```
['9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
```

Parsing a File Using List Comprehension in Python

You can also read files in Python using list comprehension. Files are iterable objects. Hence, you can iterate through the lines of the file using list comprehension and create a list.

To demonstrate, I've created a file called *dreams.txt* and pasted the following text, a short poem by Langston Hughes.

```
Hold fast to dreams
For if dreams die
Life is a broken-winged bird
That cannot fly.
-Langston Hughes
```

Using list comprehension, we can iterate through the lines of text in the file and store their contents in a new list as shown below.

```
# open the file in read-only mode
file = open("dreams.txt", 'r')
poem = [ line for line in file ]

for line in poem:
    print(line)
```

Output

```
Hold fast to dreams
For if dreams die
Life is a broken-winged bird
That cannot fly.
-Langston Hughs
```

Suggested Reading: Tuple Comprehension in Python (https://www.pythonforbeginners.com/basics/tuple-comprehension-in-python)

Using Functions in List Comprehension

So far we've seen how to use list comprehension to generate lists using some basic Python methods like **lower()** and **upper()**. But what if we wanted to use our own Python functions?

Not only can we write our own functions with list comprehensions, but we can also add filters to better control the statements. You can use any function at the position of expression in the list comprehension syntax.

For example, we can create a function double() and pass all the elements of an existing list to double the elements as shown below.

```
def double(x):
    return x*2
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[double(item) for item in myList]
print("The output list is:",newList)
```

Output

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [2, 4, 6, 8, 10, 12]
```

We can also use conditional statements along with the function in list comprehension as shown below.

```
def double(x):
    return x*2
myList=[1,2,3,4,5,6]
print("The original list is:",myList)
newList=[double(item) for item in myList if item%2==0]
print("The output list is:",newList)
```

Output

```
The original list is: [1, 2, 3, 4, 5, 6]
The output list is: [4, 8, 12]
```

Conclusion

Hopefully, you've seen the potential of list comprehensions and how they can be used to write more elegant Python code. Writing compact code is essential for maintaining programs and working with teams. Learning to make use of advanced features like list comprehension will save time and improve productivity.

Not only will your colleagues thank you when your Python code is more concise and easier to read, but you'll thank yourself when you come back to a program you haven't worked on in months and the code is manageable.

Interested in learning more about Python programming? Follow these links for additional resources that will aid you in your Python journey.

- Learn how even a single Python comment (https://www.pythonforbeginners.com/comments/comments-in-python) can improve a program.
- Explore Python string concatenation (https://www.pythonforbeginners.com/concatenation/string-concatenation-and-formatting-in-python)

I hope you enjoyed reading this article. Stay tuned for more informative articles.

Happy Learning!

Recommended Python Training

Course: Python 3 For Beginners

Over 15 hours of video content with guided instruction for beginners. Learn how to create real world applications and master the basics.

Enroll Now (https://www.pythonforbeginners.com/courses/python-3-for-beginners)

More Python Topics

API (https://www.pythonforbeginners.com/api) Argv (https://www.pythonforbeginners.com/argv) Basics (https://www.pythonforbeginners.com/basics) Beautiful Soup (https://www.pythonforbeginners.com/beautifulsoup) Cheatsheet (https://www.pythonforbeginners.com/cheatsheet) Code (https://www.pythonforbeginners.com/code) Code Snippets (https://www.pythonforbeginners.com/code-snippets-source-code) Command Line (https://www.pythonforbeginners.com/commandline) Comments (https://www.pythonforbeginners.com/comments) Concatenation (https://www.pythonforbeginners.com/concatenation) crawler (https://www.pythonforbeginners.com/crawler) Data Structures (https://www.pythonforbeginners.com/data-structures) Data Types (https://www.pythonforbeginners.com/data-types) deque (https://www.pythonforbeginners.com/deque) Development (https://www.pythonforbeginners.com/development) Dictionary (https://www.pythonforbeginners.com/dictionary) Dictionary Data Structure In Python (https://www.pythonforbeginners.com/dictionary-data-structure-in-python) Error Handling (https://www.pythonforbeginners.com/error-handling) Exceptions (https://www.pythonforbeginners.com/exceptions) Filehandling (https://www.pythonforbeginners.com/filehandling) Files (https://www.pythonforbeginners.com/files) Functions (https://www.pythonforbeginners.com/functions) Games (https://www.pythonforbeginners.com/games) GUI (https://www.pythonforbeginners.com/gui) Json

(https://www.pythonforbeginners.com/json) Lists (https://www.pythonforbeginners.com/lists) Loops (https://www.pythonforbeginners.com/loops) Mechanzie (https://www.pythonforbeginners.com/mechanzie) Modules (https://www.pythonforbeginners.com/modules) Modules In Python (https://www.pythonforbeginners.com/modules-in-python) Mysql (https://www.pythonforbeginners.com/mysql) OS (https://www.pythonforbeginners.com/os) pip (https://www.pythonforbeginners.com/pip) Pyspark (https://www.pythonforbeginners.com/pyspark) Python (https://www.pythonforbeginners.com/python) Python On The Web (https://www.pythonforbeginners.com/python-on-the-web) Python Strings (https://www.pythonforbeginners.com/python-strings) Queue (https://www.pythonforbeginners.com/queue) Requests (https://www.pythonforbeginners.com/requests) Scraping (https://www.pythonforbeginners.com/scraping) Scripts (https://www.pythonforbeginners.com/scripts) Split (https://www.pythonforbeginners.com/split) Strings (https://www.pythonforbeginners.com/strings) System & OS (https://www.pythonforbeginners.com/systems-programming) urllib2 (https://www.pythonforbeginners.com/urllib2)