Alex Cordero
Zak Collins
Matt Cirbo

PEP 202 List Comprehension

Python list comprehension (PEP 202) allows list literals to be conditionally constructed using for and if statements. This was previously implemented was by using the map and filter functions. These functions can be confusing for the user to understand how to implement and would take more time to learn. By using for and if statements, lists can be created much quicker by users without needing to understand all the semantics of the map and filter functions. The map function takes a function as a parameter and a set of values called iterables. It then applies the function to all the iterables and returns a list of the results. The filter function works the same way, but uses a boolean and the true items are added to the list. This can become confusing when more than one iterable parameter is passed. The return is then a list of tuples which is not always ideal to the user. Also, if the user does not want a function applied, but still wants a list, then the identity function can be applied, but making a list in this way takes overhead of the map function and nested loops. With this change to python, the users can now implement list without the need for overhead from functions and the confusion of how these functions actually create the list. This also allows for easy construction of list and allows for the user to more quickly create list. Another advantage is the ability for it to be used in convenient situations such as creating lists from existing lists and has compatibility with the zipped library. The proposed change also comes with some disadvantages. This includes that the code can become less readable and more prone to mistakes to the user since one function is no longer doing all the work. However, the advantages of this change far outweigh the disadvantages that come with the change. List comprehension was proposed as a change since this would be a more versatile system for the users with greater clarity and the users could manipulate their list with greater ease.

One advantage of Python list comprehension is the quickness in which the list can be written and simplification of how the list are initialized. Instead of initializing an empty list before populating it Python list comprehension allows for both operations to occur within a single statement. This allows the user of the language to implement a list object in the way they intended it to be created. If it was made with the map or filter functions, the user is unaware of how the list was created and there could be problems when trying to use the list since the user may not know exactly what the list contains. Skip Montanaro mentions “Part of the motivation for list comprehensions as I recall is that we'd like to get away from such a strong reliance on map, reduce and filter.” which shows that the implementation for list depends on the map and filter functions. The user could now instead write the list in a way that they understand. Also, if the user wanted to create a list and did not want to apply a function to it then they would put a None in for the function applying the identity function and running through it just as if it was a real function. This would take time and extra overhead that is not necessary for the user. The user will have the ability to not apply a function to their list when one is created saving some time and extra code.

The following code shows the difference in two of the ways to implement a list.

With list comprehension

```
list = [i for i in range (2, 100)]
```

With a for loop

```
list = []
for i in range(2,100):
    list.append(i)
```

With a map function

```
list = map(someFunction, ValuesToList)
```

Another advantage of Python list comprehension is that it makes initializing lists from an already pre-existing list much easier than the previous way. Previously If you wanted to implement a list using an already generated list you would have to create a for loop iterating through the list and an if conditional statement inside to test for the specific condition on each element before applying whatever filter the user put within the if statement before adding each element to the new list. This process takes up many more lines of code than the newly proposed example while also forcing the list to be initialized before adding each element individually. The new list comprehension functionality allows for less code to be written and makes it easy to put all of the required expressions on one line without having to specify that you are appending your new list. Take a look at the following code which shows the difference between initializing a list of the square every odd number up to 100.

With List Comprehension

```
squarelist = [x*x for x in range(100) if (x%2 != 0)]
```

vs. With a for loop

```
squarelist = []
for x in range(100):
    if(x%2 != 0):
        squarelist.append(x*x)
```

The disadvantage of this implementation is that the code can become more difficult for others to read and understand and it is more prone to mistakes. The map function is a clear implementation of a list. Someone else can look at this and determine the function that is applied to the values of the passed in iterables, which is then returned as a list, even if they do not understand how it works. The other way is much more confusing to read with variables i and j which could be anything. These are not clear what they are at any time and what the list may be after it is initialized. As an example Python developer

Moshe Kadka while discussing implementing PEP202 stated “List comprehensions' problems is that we cannot find a syntax that we all agree is readable”. This shows that many people think that the new syntax is very unclear on how it works and confuses readers. This becomes much less clear to any reader trying to understand the code. This could further lead to more errors in the code. If someone is unsure how the syntax works when reading it, then they most likely would create many errors when making list, leading to more time debugging. This must be a consideration for everyone as this proposed change is implemented.

The following code which creates a list of prime numbers is much harder to understand without proper naming conventions and prior knowledge of list comprehensions.

```
noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]
```

A change that we believe would be a good addition to Python list comprehension is the addition of case match. It would be very convenient to add multiple filters to a previous list based on specifications within the list comprehension. This would help to make code more efficient and reduce the amount of lines needed to produce such list. Currently to produce a list with multiple cases a user would need to produce multiple list comprehensions and append them to each other. A problem that can be associated with the current format is that creating multiple lists and appending them would not produce lists in the same order. The only way to produce these lists currently would be in the old format which is using a for loop with multiple if statements or a switch statement inside of it. The case method works better than having multiple ifs because multiple ifs is already defined within the list comprehension definition. Currently multiple if statements is treated as an logical and between the two contained expressions.

The following expression would produce a list the square of all numbers divisible by six that are less than 100. This syntax looks similar to having a case match but actually acts as a logical and. While a case match would have produced a list of all numbers divisible by 3 squared and all numbers divisible by 2 squared.

```
list = [i*i for i in range(100) if i % 3 if i % 2]
```

The following proposed syntax would make it easy to have each case separated by a comma at the beginning of the list comprehension declaration. The following code would produce a list of all number divisible by three squared as well as all numbers divisible by two cubed.

```
list = [i*i, i*i*i for i in range(100) case i%3 == 0 case i%2 ==0]
```

List comprehension is a proposed change to python that would allow the users of the language to implement the list themselves instead of using the map and filter functions. These functions are not fully understood by many of the people that use them. Furthermore, these functions take a function as a parameter and apply it to a set of values which get turned into a list. If the user does not want a function

applied to the set of values, they can pass in the identity function, but this still will take overhead when running the program and cause for unnecessary extraneous code to run. The new implementation of list objects would be much quicker and more condensed code. The problem is that some of this code becomes difficult for other users to read and therefore could be more error prone when implementing it. This would lead to more time debugging. However, there are still many advantages that this new implementation is capable of such as creating a list from another list using conditions for what is desired. This makes list much more versatile in python and makes using them much easier. For this reason we believe that the advantages outweigh the disadvantages and this proposed change of list comprehension should be implemented.

https://docs.python.org/2/library/functions.html#filter
http://markmail.org/message/qguevwxeprbg75mn#query:+page:1+mid:ctfezaiyl3iy3b47+state:results
http://www.secnetix.de/olli/Python/list_comprehensions.hawk
http://www.pythonforbeginners.com/basics/list-comprehensions-in-python
http://www.secnetix.de/olli/Python/list_comprehensions.hawk