Wednesday, February 26, 2025

LEC15 - Recursion Continued

List Comprehension

- List comprehension allows us to create new lists based on another list
- Saves lines and is more efficient

Example:

Without List Comprehension:

```
L = [1, 2, 3]
new_list = []
for x in L:
      new_list.append(x * x)
```

With List Comprehension:

```
L = [1, 2, 3]
new_list = [x * x for x in L]
```

- In general, the pattern for a list comprehension statement is `[expression for name in iterable if condition]`
  - `expression` evaluates to a value (ex. a + b)
  - `name` refers to each element in the iterable (ex. item)
  - `condition` (optional) evaluates to `True` or `False`

Filtering using List Comprehension

- Can make sure new list contains only specific elements from the old list
- Can span multiple lines for readability

Example:

```
L = [1, 2, 3, 4, 5, 6]
new_list = [num * 3
            for num in L
            if num <= 4]
```

Wednesday, February 26, 2025

LEC15 - Recursion Continued

Example

Recall the code for summing the elements in a nested list without using list comprehension:

```
def sum_list(L):
      if isinstance(L, list):
            s = 0
            for elem in L:
                  # Calculate the sum recursively
                  s += sum_list(elem)
            return s
      else:
            return L
```

This can be simplified using list comprehension:

```
def sum_list(L):
      if isinstance(L, list):
            return sum([sum_list(elem) for elem in L])
      else:
            return L
```