Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a `for` statement with a conditional test inside:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

With list comprehension you can do all that with only one line of code:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to `True`.

Example

Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

The condition `if x != "apple"` will return `True` for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

Example

With no `if` statement:

```
newlist = [x for x in fruits]
```

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the `range()` function to create an iterable:

```
newlist = [x for x in range(10)]
```

Same example, but with a condition:

Example

Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]
```

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

```
newlist = [x.upper() for x in fruits]
```

You can set the outcome to whatever you like:

Example

Set all values in the new list to 'hello':

```
newlist = ['hello' for x in fruits]
```

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "orange" instead of "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

The *expression* in the example above says:

"Return the item if it is not banana, if it is banana return orange".

Basic Questions

1. **Squares of Numbers**
   Write a list comprehension to generate the squares of numbers from 1 to 10.

   ```
   python
   Copy code
   # Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
   ```

2. **Even Numbers**
   Write a list comprehension to create a list of all even numbers between 1 and 20.

   ```
   python
   Copy code
   # Output: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
   ```

3. **Words with 'a'**
   From the list `["apple", "banana", "cherry", "date", "grape"]`, create a new list containing only words that contain the letter `'a'`.

   ```
   python
   Copy code
   # Output: ['apple', 'banana', 'grape']
   ```

4. **Uppercase Conversion**
   Convert all words in the list `["hello", "world", "python", "list"]` to uppercase using list comprehension.

   ```
   python
   Copy code
   # Output: ['HELLO', 'WORLD', 'PYTHON', 'LIST']
   ```

Intermediate Questions

6. **Multiples of 3**
   Write a list comprehension that generates the multiples of 3 up to 30.

   ```
   python
   Copy code
   # Output: [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
   ```

7. **Filter and Modify**
   Given a list `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, create a new list where numbers divisible by 2 are doubled, and the rest remain the same.

   ```
   python
   Copy code
   # Output: [1, 4, 3, 8, 5, 12, 7, 16, 9, 20]
   ```