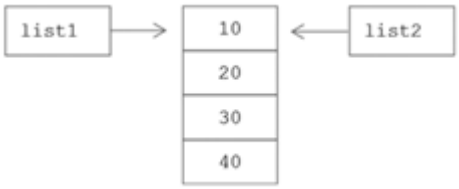
More Python Lists

In this section, you will take a closer look at the assignment of lists. You will also learn a useful and convenient means of generating lists that the `range` function cannot produce, called *list comprehensions*.

Assigning and Copying Lists

Because of the way that lists are represented in Python, when a variable is assigned to another variable holding a list, `list2 = list1`, each variable ends up referring to the *same instance* of the list in memory. This is depicted below:



This has important implications. For example, if an element of `list1` is changed, then the corresponding element of `list2` will change as well,

```
>>> list1 = [10, 20, 30, 40]
>>> list2 = list1
>>> list1[0] = 5
>>> list1
[5, 20, 30, 40]  change made in list1
>>> list2
[5, 20, 30, 40]  change in list1 causes a change in list2
```

Knowing that variables `list1` and `list2` refer to the same list explains this behavior. This issue does not apply to strings and tuples, since they are immutable and therefore cannot be modified.

When needed, a copy of a list can be made as given below,

```
list2 = list(list1)
```

In this case, we get the following results,

```
>>> list1 = [10, 20, 30, 40]
>>> list2 = list(list1)
>>> list1[0] = 5
>>> list1
[5, 20, 30, 40]  change made in list1
>>> list2
[10, 20, 30, 40]     change in list1 does NOT cause any change in list2
```

When copying lists that have sublists, another means of copying, called *deep copy*, may be needed.

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>>list1 = ['red','blue','green']
>>> list2 = list1
>>>list1[2] = 'yellow'
>>>list1
???

>>>list2
???
```

```
>>>list1 = ['red', 'blue', 'green']
>>>list2 = list(list1)
>>>list1[2] = 'yellow'
>>>list1
       ???


>>>list2
       ???
```

Part II - List Comprehensions

The `range` function allows for the generation of sequences of integers in fixed increments. **List comprehensions** in Python can be used to generate more varied sequences. Example list comprehensions are given below:

Example List Comprehensions	Resulting List
(a) `[x**2 for x in [1, 2, 3]]`	`[1, 4, 9]`
(b) `[x**2 for x in range(5)]`	`[0, 1, 4, 9, 16]`
(c) `nums = [-1, 1, -2, 2, -3, 3, -4, 4]` `[x for x in nums if x >= 0]`	`[1, 2, 3, 4]`
(d) `[ord(ch) for ch in 'Hello']`	`[72, 101, 108, 108, 111]`

(e) `vowels = ('a', 'e', 'i', 'o', 'u')`
`w = 'Hello'`
`[ch for ch in w if ch in vowels]`
	`['e', 'o']`

In the table above, (a) generates a list of squares of the integers in list `[1, 2, 3]`. In (b), squares are generated for each value in `range(5)`. In (c), only positive elements of list `nums` are included in the resulting list. In (d), a list containing the character encoding values in the string `'Hello'` is created. Finally, in (e), tuple `vowels` is used for generating a list containing only the vowels in string `w`.

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>>temperatures = [88, 94, 97, 89, 101, 98, 102, 95, 100]
>>> [t for t in temperatures if t >= 100]
???

>>> [(t-32) * 5/9 for t in temperatures]
???
```

Concepts and Procedures

1. For `list1 = [1, 2, 3, 4]` and `list2 = [5, 6, 7, 8]`, give the values of `list1[0]` and `list2[0]` where indicated after the following assignments.

a) `list1[0] = 10 and list 2[0] = 50`
b) `list2 = list1`
c) `list2[0] = 15`
d) `list1[0] = 0`

2. Give an appropriate list comprehension for each of the following.

a) Producing a list of consonants that appear in string variable w.

b) Producing a list of numbers between 1 and 100 that are divisible by 3.

c) Producing a list of numbers, `zero_values`, from a list of floating-point values, `data_values`, that are within some distance, `epsilon`, from 0.

Problem Solving

1. Write a Python program that prompts the user to enter a list of first names and stores them in a list. The program should display how many times the letter 'a' appears within the list.

2. Write a Python program that prompts the user to enter a list of words and stores in a list only those words whose first letter occurs again within the word (for example, 'Baboon'). The program should display the resulting list.

3. Modify the Password Encryption/Decryption program in the chapter so that it allows the user to continue to encrypt and decrypt passwords until they quit.

4. Modify the Password Encryption/Decryption program in the chapter so that the program rejects any en- tered password for encryption that is not considered “secure” enough. A password is considered secure if it contains at least eight characters, with at least one digit and one special character (!, #, etc).

5. Modify the Encryption/Decryption program in the chapter so that a new encryption key is randomly generated each time the program is executed. (See the Python 3 Programmers’ Reference for information on the Random module.)