This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the [Lists and Tuples video](https://www.coursera.org/learn/python-crash-course/lecture/jxAkv/lists-and-tuples), you learned about the differences between lists and tuples, how to modify the contents of a list, how to iterate over lists and tuples, how to use the enumerate() function, and how to create list comprehensions.

## ****Knowledge****

### **Common sequence operations**

Lists and tuples are both sequences and they share a number of sequence operations. The following common sequence operations are used by both lists and tuples:

* **len(sequence)** - Returns the length of the sequence.
* **for element in sequence** - Iterates over each element in the sequence.
* **if element in sequence** - Checks whether the element is part of the sequence.
* **sequence[x]** - Accesses the element at index [x] of the sequence, starting at zero
* **sequence[x:y]** - Accesses a slice starting at index [x], ending at index [y-1]. If [x] is omitted, the index will start at 0 by default. If [y] is omitted, the len(sequence) will set the ending index position by default.
* **for index, element in enumerate(sequence)** - Iterates over both the indices and the elements in the sequence at the same time.

### **List-specific operations and methods**

One major difference between lists and tuples is that lists are mutable (changeable) and tuples are immutable (not changeable). There are a few operations and methods that are specific to changing data within lists:

* **list[index] = x** - Replaces the element at index [n] with x.
* **list.append(x)** - Appends x to the end of the list.
* **list.insert(index, x)** - Inserts x at index position [index].
* **list.pop(index)** - Returns the element at [index] and removes it from the list. If [index] position is not in the list, the last element in the list is returned and removed.
* **list.remove(x)** - Removes the first occurrence of x in the list.
* **list.sort()** - Sorts the items in the list.
* **list.reverse()** - Reverses the order of items of the list.
* **list.clear()** - Deletes all items in the list.
* **list.copy()** - Creates a copy of the list.
* **list.extend(other\_list)** - Appends all the elements of other\_list at the end of list
* **map(function, iterable)** - Applies a given function to each item of an iterable (such as a list) and returns a map object with the results
* **zip(\*iterables)** - Takes in iterables as arguments and returns an iterator that generates tuples, where the i-th tuple contains the i-th element from each of the argument iterables.

### **Tuple use cases**

Remember, there are a number of cases where using a tuple might be more suitable than other data types:

* Protecting data: Because tuples are immutable, they can be used in situations where you want to ensure the data you have cannot be changed. This can be particularly helpful when dealing with sensitive or important information that should remain constant throughout the execution of a program.
* Hashable keys: Because they're immutable, tuples can be used as keys on dictionaries, which can be useful for complex keys.
* Efficiency: Tuples are generally more memory-efficient than lists, making them advantageous when dealing with large datasets.

### **List comprehensions**

A list comprehension is an efficient method for creating a new list from a sequence or a range in a single line of code. It is a best practice to add descriptive comments about any list comprehensions used in your code, as their purpose can be difficult to interpret by other coders.

* **[expression for variable in sequence]** - Creates a new list based on the given sequence. Each element in the new list is the result of the given expression.
  + Example: **my\_list = [ x\*2 for x in range(1,11) ]**
* **[expression for variable in sequence if condition]** - Creates a new list based on a specified sequence. Each element is the result of the given expression; elements are added only if the specified condition is true.
  + Example: **my\_list = [ x for x in range(1,101) if x % 10 == 0 ]**

Note that tuples do not have comprehensions but a similar functionality can be achieved with:

**tuple(i for i in (1, 2, 3))**

### **When to use** for **loops or list comprehensions**

In Python, list comprehensions are generally used for creating new lists from existing ones in a concise and readable manner, especially when the task involves simple transformations or filtering of elements.

**for** loops are more versatile and are preferred when the operation is more complex, requires multiple lines of code, involves statements other than expression (like **print**, **pass**, **continue**, **break**), or when you need to iterate over a list without creating a new one.