Study Guide: Lists Operations and Methods

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 segment, 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

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]**

Coding skills

Skill Group 1

- Use a **for** loop to modify elements of a list.
- Use the **list.append(old,new)** method.
- Use the **string.endswith()** and **string.replace()** methods to modify the elements within a list.

```
# This block of code changes the year on a list of dates.
# The "years" list is given with existing elements.
years = ["January 2023", "May 2025", "April 2023", "August 2024", "September 2025", "December 2023"]


# The variable "updated_years" is initialized as a list data type
# using empty square brackets []. This list will hold the new list
# with the updated years.
updated_years = []

# The for loop checks each "year" element in the list "years".
for year in years:

    # The if-statement checks if the "year" element ends with the
    # substring "2023".
    if year.endswith("2023"):

        # If True, then a temporary variable "new" will hold the
        # modified "year" element where the "2023" substring is
        # replaced with the substring "2024".
        new = year.replace("2023","2024")

        # Then, the list "updated_years" is appended with the changed
        # element held in the temporary variable "new".
        updated_years.append(new)

    # If False, the original "year" element will be appended to the
    # the "updated_years" list unchanged.
    else:
        updated_years.append(year)


print(updated_years)
# Should print ["January 2024", "May 2025", "April 2024", "August 2024", "September 2025", "December 2024"]
```

Run
Reset

Skill Group 2

- Use a list comprehension to return values

```
# This list comprehension creates a list of squared numbers (n*n). It
# accepts two integer variables through the function's parameters.
def squares(start, end):

# The list comprehension calculates the square of a variable integer
# "n", where "n" ranges from the "start" to "end" variables inclusively.
# To be inclusive in a range(), add +1 to the end of range variable.
    return [n*n for n in range(start,end+1)]


print(squares(2, 3))  # Should print [4, 9]
print(squares(1, 5))  # Should print [1, 4, 9, 16, 25]
print(squares(0, 10)) # Should print [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Run
Reset

Skill Group 3

- Use the **string[index]** method within a list comprehension.
- Use a list comprehension to modify elements of a list.
- Use the **string.replace()** method within a list comprehension.

```
# This block of code also changes the year on a list of dates using a
# different approach than demonstrated in Skill Group 1. By using a
# list comprehension, you can see how it is possible to refactor the
# code to a shorter, more efficient code block.

# The "years" list is given with existing elements.
years = ["January 2023", "May 2025", "April 2023", "August 2024", "September 2025", "December 2023"]

# The list comprehension below creates a new list "updated_years" to
# hold the command to replace the "2023" substring of the "year"
# element with the substring "2024". This action will be executed if
# the last 4 indices of the "year" string is equal to the substring
# "2023". If false (else), the "year" element will be included in the
# new list "updated_years" unchanged.
updated_years = [year.replace("2023","2024") if year[-4:] == "2023" else year for year in years]


print(updated_years)
# Should print ["January 2024", "May 2025", "April 2024", "August 2024", "September 2025", "December 2024"]
```

Run
Reset

Skill Group 4

- Use the **string.split()** method to separate a string into a list of individual words.
- Iterate over the new list using a **for** loop.
- Modify each element in the list by slicing the element's string at a given [1:] index position and appending the substring to the end of the element.
- Convert a list back into a string.

```
# This function splits a given string into a list of elements. Then, it
# modifies each element by moving the first character to the end of the
# element and adds a dash between the element and the moved character.
# For example, the element "2two" will be changed to "two-2". Finally,
# the function converts the list back to a string, and returns the
# new string.
def change_string(given_string):

# Initialize "new_string" as a string data type by using empty quotes.
    new_string = ""
    # Split the "given_string" into a "new_list", with each "element"
    # holding an individual word from the string.
    new_list = given_string.split()

    # The for loop iterates over each "element" in the "new_list".
    for element in new_list:

        # Convert the list into a "new_string" by using the assignment
        # operator += to concatenate the following items:
```