**Download Your Daily Notes**

**My own views on lists and methods**

 This shows how to access the data within the list (syntax)

**Daily Notes - More on Lists**

 This shows how to access the data within the list (syntax)

**Daily Notes - Using Lists as Stacks and Queues**

 This shows the syntax for getting data from a list, from the end of the list using negative indexing

**Daily Notes - List Comprehensions**

 This shows us how to fetch data from a list, when more than one item is required. It uses slicing in order to fetch a range of numbers, this can be from the beginning to the end.

**Daily Notes - Nested List Comprehensions**

 This topic was about adding and changing elements in lists. Unlike strings or tuples, lists are mutable, meaning their elements can be modified.  
  
To change the value of an item in a list, we can use the assignment operator (=). By assigning a new value to the desired index, we can change individual elements or a range of elements within the list.  
  
To add elements to a list, we have a few options. We can use the append() method to add a single item at the end of the list, or we can use the extend() method to add multiple items by extending the list. Alternatively, we can use the + operator for concatenation, which combines two lists together.  
  
In addition, the \* operator allows us to repeat a list a certain number of times.  
  
Furthermore, we learned that we can insert items at specific positions in a list using the insert() method. This method allows us to insert both single and multiple items by utilizing an empty slice.  
  
Overall, we explored various ways to add and modify elements within lists, giving us flexibility in working with mutable data structures in Python.

**Daily Notes - The del statement / Tuples and Sequences**

 This topic focused on deleting and removing elements from lists in Python. We explored different methods and techniques for this purpose.  
  
The keyword del allows us to delete one or more items from a list. By specifying the index or a range of indices, we can remove specific elements. It's even possible to delete the entire list using del, which results in an error if any reference to the list is made afterward.  
  
The remove() method is used to remove a specific item from a list by specifying its value. On the other hand, the pop() method removes and returns the item at a given index. If no index is provided, pop() removes and returns the last item, making it suitable for implementing stack-like behavior.  
  
Additionally, the clear() method allows us to empty a list, removing all its elements.  
  
Lastly, we learned that we can delete items in a list by assigning an empty list to a slice of elements. By replacing a slice with an empty list, the corresponding items are effectively removed from the original list.  
  
These techniques provide us with the ability to modify and remove elements from lists, enhancing our flexibility in working with mutable data structures in Python.

**Daily Notes - Sets**

 The daily note provides an overview of Python list methods and list comprehension. The list methods include functions such as append(), extend(), insert(), remove(), pop(), clear(), index(), count(), sort(), reverse(), and copy(), which enable various operations on lists. Examples demonstrate the usage of these methods. List comprehension is a concise way to create new lists based on existing lists. It involves an expression followed by a for statement within square brackets. Other list operations covered include membership testing and iterating through a list using for loops.

**Daily Notes - Looping Techniques**

 When looping through dictionaries, the items() method can be used to retrieve both the key and corresponding value simultaneously. Similarly, when looping through a sequence, the enumerate() function can be used to retrieve the position index and corresponding value together. To loop over multiple sequences at the same time, the zip() function can pair the entries from each sequence.  
  
To loop over a sequence in reverse, the reversed() function can be used by specifying the sequence in a forward direction. If you want to loop over a sequence in sorted order, the sorted() function can be used to return a new sorted list without altering the original sequence.  
  
While looping over a list, it's recommended to avoid modifying the list itself. Instead, it's simpler and safer to create a new list if any changes are needed.  
  
In summary, Python provides convenient methods and functions like items(), enumerate(), zip(), reversed(), and sorted() to facilitate looping through dictionaries and sequences in different ways. It's important to consider best practices and avoid modifying a list while iterating over it.

**Daily Notes - More on Conditions**

 summarize the following for a daily note: In Python, the conditions used in while and if statements can involve any operators, not just comparisons.  
  
Comparison operators such as in, not in, is, and is not can be used to check for value membership or object identity. Comparison operators have the same priority, which is lower than numerical operators. Chained comparisons are also possible.  
  
Boolean operators and, or, and not can be used to combine comparisons and Boolean expressions. They have lower priorities than comparison operators, with not having the highest priority and or the lowest. Parentheses can be used for explicit grouping.  
  
The Boolean operators and and or are short-circuit operators, meaning that evaluation stops as soon as the outcome is determined. The return value of a short-circuit operator is the last evaluated argument when used as a general value.  
  
Assigning the result of a comparison or Boolean expression to a variable is possible. This can be useful for conditional assignments.  
  
Note that Python doesn't allow assignment inside expressions, unlike C, which helps avoid certain types of errors.

**Daily Notes - Comparing Sequences and Other Types**

 Sequence objects in Python, such as tuples, lists, and strings, can be compared to other objects of the same sequence type. The comparison follows lexicographical ordering, where the first two items are compared, and the outcome is determined based on their relationship. If they are equal, the comparison moves to the next pair of items until one sequence is exhausted.  
  
If the items being compared are themselves sequences of the same type, the lexicographical comparison is carried out recursively. If all items in two sequences compare equal, the sequences are considered equal. If one sequence is an initial sub-sequence of the other, the shorter sequence is considered smaller.  
  
For strings, lexicographical ordering is based on the Unicode code point numbers of individual characters. Examples of comparisons between sequences of the same type are provided.  
  
It is worth noting that comparing objects of different types using < or > is allowed if the objects have appropriate comparison methods. Mixed numeric types are compared based on their numeric values, while other cases may raise a TypeError exception instead of providing an arbitrary ordering.

**Daily Notes - Activity 1 - Applying new Concepts**

 Completed and uploaded

**My Views on the Day**

 1. Learning about Lists and its various functions.  
  
2. Activity 1, 2 & 3  
  
3. None  
  
4. None

**Daily Notes - Day 1 Reflections**

 1. Learning about Lists and its various functions.  
  
2. Activity 1, 2 & 3  
  
3. None  
  
4. None