# Searching Arrays And Lists: Takeaways



## Syntax

- Retrieving an index of a list:

```
list.index("item")
```

- Implementing linear search:

```
def linear_search(array, search):
indexes = []
for i, item in enumerate(array):
    if item == search:
        indexes.append(i)
return indexes
sevens = linear_search(times, 7)
```

- Implementing a more complex linear search:

```
def linear_multi_search(array, search):
indexes = []
for i, item in enumerate(array):
    if item == search:
        indexes.append(i)
return indexes
transactions = [[times[i], amounts[i]] for i in range(len(amounts))]
results = linear_multi_search(transactions, [56, 10.84])
```

- Implementing binary search:

```
def binary_search(array, search):
    counter = 0
    insertion_sort(array)
    m = 0
    i = 0
    z = len(array) - 1
    while i<= z:
        counter += 1
        m = math.floor(i + ((z - i) / 2))
        if array[m] == search:
             return m
        elif array[m] < search:
             i = m + 1
        elif array[m] > search:
             z = m - 1
    return counter
```

## Concepts

- You'll want to implement your own searching logic in some cases. Example cases include:
    - You want to find all occurrences of a term.
    - You have custom search logic across multiple fields in a row.
    - You have a data structure that doesn't have built-in search, like a linked list.
    - You want a higher-performance search algorithm for your use case.
- Time complexity of a linear search if you're only looking for the first element that matches your search:
    - In the best case, when the item you want to find is first in the list, the complexity is $O(1)$.
    - In the average case, when the item you want is in the middle of the list, the complexity is $O(n/2)$, which simplifies to $O(n)$.
    - In the worst case, when the item you want is at the end of the list, the complexity is $O(n)$.
- Space complexity of a linear search:
    - When searching for multiple elements, linear search has $O(n)$ space complexity.
    - When searching for the first matching element, it has space complexity of $O(1)$.
- The binary search algorithm looks for the midpoint of a given range and keeps narrowing the window in its search until the value is found.
- Binary search performs better than a linear search since it doesn't have to search every single element of the array.
- In general, you should use a linear search if:
    - You only need to search once.
    - You don't need to sort the list for another reason (like viewing items in order).
    - You have complex search criteria, or require external lookups.
- You should use binary search if:
    - The data is already sorted, or you need to sort it for another reason.
    - You need to perform multiple searches.
    - You can distribute the sort across multiple machines, so it runs faster.

## Resources

- List of Unicode characters
- Binary Search