Lists and sets in different programming languages

Sets and lists are built-in types in many languages with a general overarching theme common to most. In the actual implementation, there are some subtle differences.

In this reading, you will explore some of the differences between lists and sets in different programming languages.

Mutability

The concept of immutability has already been discussed. Concretely, it is how a data structure is created. Immutable objects cannot be changed when created, while mutable ones can be changed after creation. Sets are an example of an immutable structure in most programming languages. However, objects that are added to sets can be treated differently depending on the language. In JavaScript, you can add mutable objects to a set, which is not permitted in Python. Understanding this fundamental difference will give you insights into how the language uses sets.

Why are sets as quick as they are?

The reason for the speed at which sets can find an item is down to their underlying architecture. Sets store values using a hashing approach. To hash something is to take it and generate a unique output from it. This is done by applying an algorithm to it that converts the input into a simple alpha-numeric (words and letters) output. Every time the algorithm sees the input, it will always generate the same alpha-numeric output. This is then used in a hashtable, which uses the unique hash to inform where to store an item in memory. So, it is possible to know with one computation whether the value is in memory or not. Instead of searching for every element in the list, just apply the hashing function and check if it is being used.

Therefore, sets are fast but the reason they're so fast is also why you can't change the value being stored. If you create a hash for an element and store it according to that hash, then you change the item, and the hash will no longer be able to find it. This is one reason why Python only allows you to store immutable objects that can't be changed in sets. Not all languages offer this protection. For example, JavaScript does allow the storing of mutable objects. Thus the object's protection is left to the user when coding with JavaScript sets. Suppose you wish to alter a mutable value in JavaScript. In that case, extracting and deleting the mutable object is advisable before reinserting it into the set as a new element.

Like Python, Kotlin will not allow elements to be altered once they have been added to a set. This means that there is read-only access to the set. If you want sets that can be altered, then Kotlin has another type of set called a `MutableSet`. Sets in Kotlin are very versatile and offer a range of built-in methods like `max`, `min`, `sum` and `average`. This makes them very handy when looking for specific items within a group of numbers.

Lists

Another important collection type that is good to know about is lists. Lists store elements in a given order which can be accessed using an index. An index is just a number passed to the list that indicates that the element found at that number location should be returned. The index is how elements are searched in a list. If someone wants to know if something is in a list, they would have to do a search and check each item, so they are slower than a set. Additionally, a list will allow you to store all kinds of elements in them. There is no distinction made between whether an item is mutable or immutable. Lists will also allow you to store duplicates of items. There are different types of lists that are implemented differently.

However, the fundamental point that is worth remembering is that a list allows you to store repeating items and items of different types. A list is ordered, which means that it will retain the ordering of items as they are inserted. This is different from sets, which might store items in different locations from where they were initially entered.

Knowing these subtle differences can be helpful when deciding what type of data structure you want to select. Both

sets and lists are useful, but because they act differently, they will be more useful in some situations than others. Making sure that a list or a set is most suitable in a situation is the trick!

Conclusion

In this reading, a brief overview was given on lists and sets in different languages. The strengths and weaknesses of both have been discussed, as well as some interesting insights into how they work. Hopefully, you will think about these the next time you are planning to store data!