Differentiating between data structures like lists, tuples, sets, arrays, and linked lists involves understanding their characteristics and use cases. Here are the major differences between them:

1. **List, Tuple, Set (Commonly referred to as Abstract Data Types or ADTs):**
   * **List:** An ordered collection of elements that can contain duplicate values. Lists are mutable, meaning you can modify their content without creating a new list.
   * **Tuple:** Similar to lists, tuples are ordered collections of elements. However, tuples are immutable, meaning once a tuple is created, its content cannot be changed.
   * **Set:** An unordered collection of unique elements. Sets are used when the existence of an element in a collection is more important than the order or frequency of elements.
2. **Array, Linked List (Commonly referred to as Concrete Data Structures):**
   * **Array:** An ordered collection of elements of the same type, stored in contiguous or neigbouring memory locations. Arrays have a fixed size, and accessing elements is fast because elements can be accessed directly using their index.
   * **Linked List:** A sequence of elements, where each element points to the next element in the list. Unlike arrays, linked lists do not require contiguous or neigbouring memory space. They allow for efficient insertion and deletion of elements but have slower access times since elements need to be accessed sequentially.

**Major Differences:**

* **Memory Allocation:** Arrays require contiguous memory allocation, whereas linked lists do not. Lists, tuples, and sets in high-level languages like Python handle memory allocation automatically, often using arrays or linked list structures internally.
* **Mutability:** Tuples and arrays (when size and type are fixed) are immutable, while lists, sets, and linked lists are mutable.
* **Access Efficiency:** Arrays offer fast access to elements via indexing, while linked lists require sequential access, which is slower. Lists, tuples, and sets in languages like Python provide varying efficiencies for access, insertion, and deletion based on their implementation.
* **Storage of Elements:** Lists, tuples, and sets can store elements of different types (in languages like Python), whereas arrays and linked lists typically store elements of the same type.

In summary, the differentiation can be in terms of their mutability, memory allocation, access efficiency, and the homogeneity of stored elements. The broad categorization as ADTs and concrete data structures is also a way to differentiate them based on their level of abstraction and implementation details.



so basically both ADTs and Concrete data structure are used to store data in computers right?

Yes, that's correct. Both Abstract Data Types (ADTs) and Concrete Data Structures are used for storing and organizing data in computers, but they serve different roles in this process:

1. **Abstract Data Types (ADTs):**
   * ADTs are more about the behavior and properties of a data structure. They define the type of data they can hold and the operations that can be performed on them.
   * Examples include Lists, Stacks, Queues, Sets, and Maps. These data types do not specify how the data will be organized in memory or how the operations will be implemented. They are more about what operations are possible and what properties these operations have.
   * ADTs provide a theoretical framework or a contract, which concrete data structures then implement.
2. **Concrete Data Structures:**
   * Concrete data structures are actual implementations of ADTs. They define how data is organized and stored in memory and how various operations are performed.
   * Examples include Arrays, Linked Lists, Trees, Hash Tables, etc. These structures have specific memory layouts and algorithms for performing operations like addition, deletion, and access.
   * Concrete data structures are what actually get used in programming and system design. They are the tangible representations of the abstract concepts defined by ADTs.

In essence, while ADTs describe what a data structure should do, concrete data structures define how those operations are carried out. Both are integral in computer science for efficiently storing, accessing, and manipulating data.



So Adts is like an inbuilt datatypes that python provide us where Concrete data structure are used in the backend that the python programmers already wrote.

Your understanding is partially correct, but it's important to clarify the relationship between Abstract Data Types (ADTs) and Concrete Data Structures, especially in the context of a programming language like Python.

1. **Abstract Data Types (ADTs):**
   * ADTs are theoretical concepts in computer science. They define a set of operations and the behavior of those operations, but not their implementation.
   * In Python, when we talk about Lists, Tuples, Sets, and Dictionaries, we are often referring to Python's specific implementations of these ADTs. These data types in Python provide the user with a certain behavior and set of operations, like adding or removing elements, but the user doesn't need to worry about how these operations are implemented under the hood.
2. **Concrete Data Structures:**
   * Concrete data structures are the actual implementations of these abstract concepts. They define how data is stored in memory and how various operations are executed.
   * In Python, the List, Tuple, Set, and Dictionary types are actually concrete implementations of more abstract concepts. For example, Python's List is a dynamic array, and its Dictionary is implemented as a hash table. These are concrete data structures.
   * The distinction here is that, while Python provides these as built-in types, they are still concrete implementations. The user doesn't usually see the underlying code, but these structures have specific ways they handle data in memory.

So, in summary, in Python and other high-level programming languages, when you use built-in data types like Lists, Tuples, Sets, and Dictionaries, you are actually using concrete data structures that the language has provided as implementations of various ADTs. These implementations are designed to handle the complexities of data storage and manipulation internally, allowing programmers to use them without needing to understand the details of how they are implemented.