### Programming Guide for Trie Data Structure

#### Overview of Trie

A Trie, also known as a prefix tree, is a tree-like data structure that stores a dynamic set of strings. Tries are used to efficiently store and search for keys in a dataset of strings. Each node in a Trie represents a character of a string.

#### Key Operations

* **Insert**: Add a word to the Trie.
* **Search**: Check if a word is present in the Trie.
* **StartsWith**: Determine if any word in the Trie starts with a given prefix.

#### Pseudo Code for Trie

CLASS TrieNode:  
 FUNCTION \_\_init\_\_():  
 children = Dictionary to store children nodes  
 is\_end\_of\_word = Boolean to mark the end of a word  
  
CLASS Trie:  
 FUNCTION \_\_init\_\_():  
 root = Initialize a new TrieNode  
  
 FUNCTION insert(word):  
 START at the root node  
 FOR each character in the word:  
 IF character is not present in the current node's children:  
 CREATE a new TrieNode as a child  
 MOVE to the child node  
 MARK the last node as the end of a word  
  
 FUNCTION search(word):  
 START at the root node  
 FOR each character in the word:  
 IF character is not in the current node's children:  
 RETURN False (word not found)  
 MOVE to the child node  
 RETURN True if current node is marked as end of a word, else False  
  
 FUNCTION startsWith(prefix):  
 START at the root node  
 FOR each character in the prefix:  
 IF character is not in the current node's children:  
 RETURN False (no word starts with the prefix)  
 MOVE to the child node  
 RETURN True (prefix found in Trie)

#### Implementation Tips

* The Trie should initialize with a root node, which is an instance of TrieNode.
* Each node in the Trie stores a dictionary of child nodes, where keys are characters and values are TrieNode objects.
* For insertion, navigate through the Trie based on the characters of the word, creating new nodes as needed.
* For searching, navigate through the Trie using the characters of the word. If at any point a character is not found in the current node’s children or the end of the word does not correspond to an is\_end\_of\_word node, return False.

#### Applications

* **Autocomplete**: Tries can be used to predict words when a user types in a part of a word.
* **Spell Checker**: Quickly find words in a dictionary to check for misspellings.
* **Word Games**: Tries are useful in games like Scrabble or Boggle where word validation is frequently required.