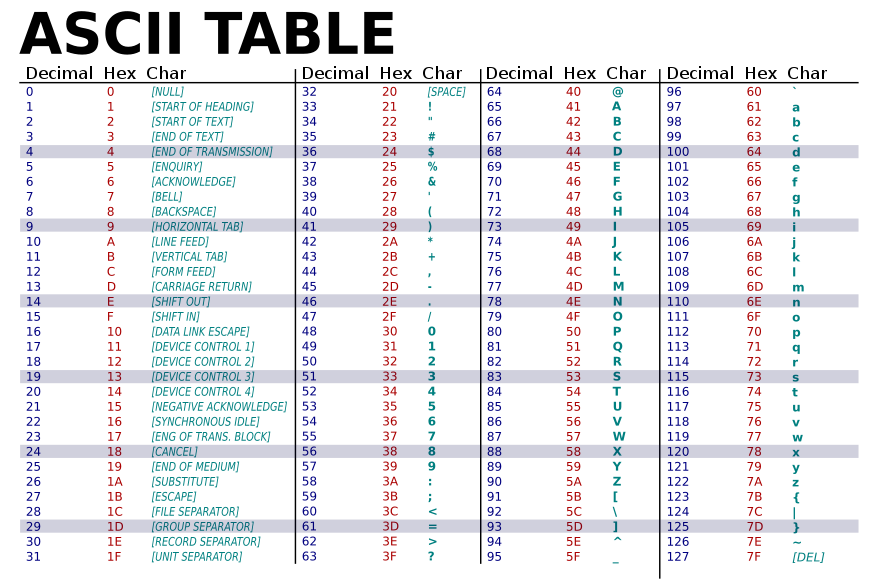
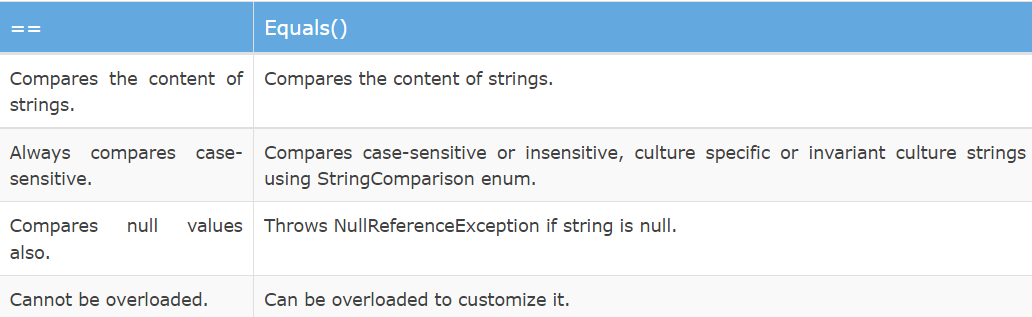
* strings are composed of a **sequence of characters.**
* characters can be ASCII or Unicode, depending on how we choose to do our encoding
* ASCII is only 256 characters,
* it can be represented by a **single byte.**



* Unicode, on the other hand, is larger and more generalized.
* We have UTF-8, UTF-16, and even UTF-32

## **Fast Facts: C#**

* Mutable? No
* Primitive? No
* Comparison: s1.equals(s2) || ==
* Access the ith character: s1[i]



## Useful C# String Methods:

* Length – Returns the length of the string
* strVal(int i) – Returns the character at index i
* substring (int i, int j) – Returns the substring. String is 0 Based
* Contains (String s) – Returns a True if s is contained in the string
* Contains (char s) – Returns a True if s is contained in the string
* indexOf(String s) – Returns the starting index of the first occurrence of s
* ToCharArray () – Converts a string to a character array (useful if we want to repeatedly modify the string)

string strVal = "sujeet";

int a =strVal.Length;

char [] arra= strVal.ToCharArray();

// Retrieves a substring from this instance. The substring starts at a specified

// character position and has a specified length

// output su

string strVal1 = strVal.Substring(0,2);

//output -115 (from sujeet) - ASCII decimal value

int aa = strVal[0];

// output = s (from sujeet)

char ab = strVal[0];

//Loop

foreach (char item in arra){ }

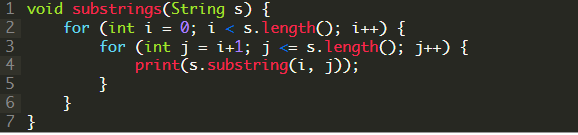
bool s= strVal.Contains('s');

# Using a Length-256 Integer Array

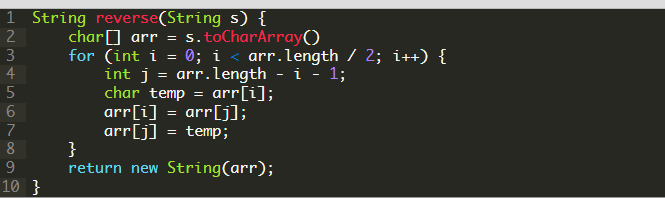
* count the occurrences of each character in a string
* use some sort **of hashtable or dictionary**.
* We could just map the characters to the count of the number of times they occur
* <https://leetcode.com/problems/longest-substring-without-repeating-characters/>
* <https://www.geeksforgeeks.org/sort-string-characters/>
* <https://www.byte-by-byte.com/anagrams/>

# Using 2 Pointers

* Algorithms that use multiple pointers are super common all over the place.
* For starters, we can talk about finding all substrings of a string
* **Array Way**



* **use two pointers-** is to reverse a string in place

****

* [**https://leetcode.com/problems/remove-duplicates-from-sorted-array/**](https://leetcode.com/problems/remove-duplicates-from-sorted-array/)
* [**https://leetcode.com/problems/valid-palindrome/**](https://leetcode.com/problems/valid-palindrome/)
* [**https://www.interviewcake.com/question/cpp/reverse-words/?utm\_source=bbb&utm\_medium=affiliate&utm\_campaign=bbb**](https://www.interviewcake.com/question/cpp/reverse-words/?utm_source=bbb&utm_medium=affiliate&utm_campaign=bbb)

# String Math

* <https://www.geeksforgeeks.org/convert-string-binary-sequence/>
* <https://leetcode.com/problems/string-to-integer-atoi/>
* <https://leetcode.com/problems/compare-version-numbers/>

# String Sliding Windows

* come up a lot when talking about strings and arrays
* dramatically improve our time complexity in certain cases.
* Essentially, sliding windows are a special case of our two pointer pattern
* But with a sliding window, we can use a greedy approach
* <https://leetcode.com/problems/find-all-anagrams-in-a-string/description/>
* <https://leetcode.com/problems/minimum-window-substring/description/>
* <https://leetcode.com/problems/substring-with-concatenation-of-all-words/description/>

# String Comparison, Alignment, and Matching

* <https://www.byte-by-byte.com/longestsubstring/>
* <https://leetcode.com/problems/edit-distance/>
* <https://leetcode.com/problems/regular-expression-matching/>

# Regular Expressions

* <https://leetcode.com/problems/regular-expression-matching/>
* <https://leetcode.com/problems/wildcard-matching/>

# String Algorithms

* KMP (Knuth Morris Pratt)
  + <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>
* Boyer Moore
  + <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/>
* Rabin-Karp
  + <https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm>