# Strings

- Strings
- Escape sequences
- String Methods
- Chaining Methods



## Strings

- The String object is used to represent and manipulate a sequence of characters.
- Strings are useful for holding data that can be represented in text form.
- Strings can be created as primitives, from string literals, or as objects, using the String() constructor. We will talk about objects later in the course.

```
let str1 = "This is a string"
let str2 = 'This is a string'
let str3 = `This is a string`
let str4 = new String("This is a string")

console.log(typeof str1) // string
console.log(typeof str2) // string
console.log(typeof str3) // string
console.log(typeof str4) // object
```



## Strings

- You can see the length of a string with **length** property.
- length property returns the count of the total number of characters.
- The length of an empty string is 0.

```
let str1 = "Clarusway"
let str2 = 'Full Stack'
let str3 = ""
let str4 = str1 + " " + str2

console.log(str1.length) // 9
console.log(str2.length) // 10
console.log(str3.length) // 0
console.log(str4.length) // 20
```

# Strings

- With indexing you can access elements of strings one by one.
- Indexes always start from 0.
- Valid indexes are 0 to string.length-1


```
let str1 = "Orange"

console.log(str1[0]) // O
console.log(str1[3]) // n

// invalid indexing, out of range
console.log(str1[-1]) // undefined
console.log(str1[10]) // undefined
```



# Strings

- Strings are **immutable**, which means once you assign a value, the value cannot be changed. However, you can assign a new value later.

```
let str1 = "Clarusway"
str1[2] = "A"
console.log(str1) // No change. Output is still Clarusway

// assign a new value
str1 = "CLARUSWAY"
console.log(str1) // CLARUSWAY
```

# Escape sequences

- Special characters can be encoded using escape sequences. Most used ones:

| Escape sequence | Unicode code point |
|---|---|
| \0 | null character |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | newline |
| \t | tab |
| \b | backspace |

```
let str1 = "This is John's Car"
let str2 = 'This is John's Car'
// error: SyntaxError: Unexpected identifier
let str3 = 'This is John\'s Car'
// escape character

let str4 = "Claruswww\bay"
// second w removed with backspace
console.log(str4) // Clarusway
```



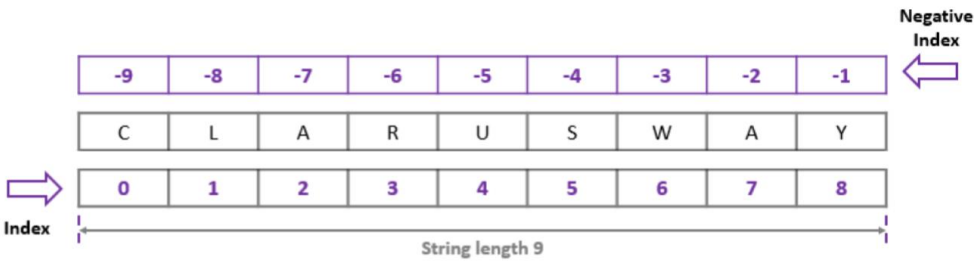
# String Methods

- Remember strings are immutable.
- We can't change a string, but we can generate new strings from a string according to our requirements.
- To generate new strings we use string methods.
- There are many strings methods. You can't memorize all of them, and you don't need to memorize them. You can always google them.
- A good developer should know or recall there is way to do but may not remember how. For example, you may need to capitalize the letters of a string. You need to know there is a method for this. You can google and find the method.
- To google something first you may be able to address the question or problem.



# slice()

- The **slice()** method extracts a section of a string and returns it as a new string
- We can use a negative number to select from the end of the string
- If you don't use the second parameter, the method will slice out the rest of the string.



# slice()

- The character in the second number index is not included to the sliced part.
- Even you use negative numbers, direction is always from left to right.


```
const str= "Orange";

console.log (str.slice(0, 4)); // Oran
console.log (str.slice(3, 5)); // ng
console.log (str.slice(2)); // ange
console.log (str.slice(-4)); // ange
console.log (str.slice(1, -3)); // ra
console.log (str.slice(-6, -1)); // Orang
```



# substring()

- The **substring()** method returns the parts of a string between "start" and "end", not including "end" itself.
- The difference between slice and substring:
  - When startIndex is bigger than endIndex, substring() reverses the roles: startIndex becomes endIndex and vice versa.

```
const str= "Orange";

console.log (str.substring(0, 4)); // Oran
console.log (str.substring(4, 0)); // Oran

console.log (str.substring(2)); // ange
console.log (str.substring(-4)); // ange
```



# concat()

- The **concat()** method is used to combine two or more strings.
- Very similar to the addition/string concatenation operators (+, +=). No difference according to functionality but '+' operator is faster than concat() for performance.

```
let str1 = "Clarusway";
let str2 = "Full";
let str3 = "Stack";
const result = str1.concat(" ", str2, " ", str3)
console.log(result)
```

# includes()

- The **includes()** method specifies whether a string includes the characters of a specified string.
- This method returns true if the characters are in the string, and if not false.
- It is case sensitive.

```
let str = "Hello John, welcome to Clarusway.";

console.log(str.includes("welcome")); // true

console.log(str.includes("Welcome")); // false
```

CLARUSWAY
WAY TO REINVENT YOURSELF

# indexOf()

- The **indexOf()** method returns the index of (the position of) the first occurrence of a specified text in a string:
- This method returns -1 if the value is not found.

```
let str = "Hello John, welcome to Clarusway.";

console.log(str.indexOf("welcome")); // 12

console.log(str.includes("Welcome")); // -1
```

CLARUSWAY
WAY TO REINVENT YOURSELF

# String Methods

- Interactive question :
  - Validate an input of a user whether it contains @ sign

Develop Code!

# String Methods

- Sample code :

```
let email = prompt("Enter your email")

if (email.indexOf("@") === -1) {
  console.log("Invalid email")
}
```



# indexOf()

- You can also specify the start index by using the second parameter.

```
let str = 'You do not know what you do not know until you know.';

let result = str.indexOf("know", 12); //start search from 12th position

console.log (result);
```



# indexOf()

- Example: find the count of occurrences of a text in a string

```
function findCount(source, search) {
  let count = 0; // counter will start from 0
  let position = source.indexOf(search) // first search

  while (position !== -1) {// as long as we find search string, loop will go on
    count++ // increase the counter
    position = source.indexOf(search, ++position) // search again, from where we stop +1
  }
  return count
}

let str = 'You do not know what you do not know until you know.';
console.log (findCount(str, "know")); // 3
```

# lastIndexOf()

- The **lastIndexOf()** method returns the index of the last occurrence of a specified text in a string
- This method returns -1 if the value is not found.

```
let str = "Hello John, welcome to Clarusway.";

console.log(str.lastIndexOf("o")); // 21

console.log(str.lastIndexOf("o", 20));// 16
```



# search()

- The **search()** method searches a string for a given value and returns the position of the match.
- This method returns -1 if the value is not found.
- The search() method also accepts a **regular expression** and returns the index of the first match in a string.
- A regular expression is a sequence of characters that forms a search pattern.
- It is very handy especially validating some formatted strings such as emails. However it is quite confusing at first glance. We will not focus on regular expressions on this course.
- If you need to validate a pattern you can easily google and establish that pattern without knowing details.

# search()

```
let str = 'You do not KNOW what you do not know until you know.';

console.log (str.search("know"));  // 32

console.log (str.search(/know/)); // 32 with regular expression

console.log (str.search(/know/i)); // 11 (i means not case sensitive)
```



# search()

- The difference between **search()** method and the **indexOf()** method:
  - The search() method cannot take a second start position argument.
  - The indexOf() method cannot take powerful search values (regular expressions).



# replace()

- The **replace()** method looks for a string for a given value and returns a new string to replace the specified values.
- By default, the replace() method replaces only the first match and it is case sensitive.
- You can also use regular expressions.

```
let str = "Mr Brown has a brown house and a brown car";

let newStr = str.replace("brown", "red");

console.log(newStr);
// Mr Brown has a red house and a brown car
```

# replaceAll()

- The **replaceAll()** method looks for a string for a given value and returns a new string to replace all of the occurrences with the specified values.
- It is a new method introduced in ES2021

```
let str = "Mr Brown has a brown house and a brown car";

let newStr = str.replaceAll("brown", "red");

console.log(newStr);
// Mr Brown has a red house and a red car
```



# split()

- The **split()** method divides a string into an array of substrings, and returns the new array
- We will learn arrays in the following class
- Takes a parameter as separator and divides the string according to this separator.

```
let str = "Welcome to: Full Stack";

console.log(str.split()) // no separator
// output: [ 'Welcome to: Full Stack' ]

console.log(str.split(" ")) // separator is space
// output: [ 'Welcome', 'to:', 'Full', 'Stack' ]

console.log(str.split(":")) // separator is colon
// output: [ 'Welcome to', ' Full Stack' ]
```

# toLowerCase()

- The **toLowerCase()** method transforms a string to lowercase letters

```
let str = "Welcome to Full Stack";

console.log(str.toLowerCase()) // welcome to full stack
```



# toUpperCase()

- The **toUpperCase()** method transforms a string to uppercase letters

```
let str = "Welcome to Full Stack";

console.log(str.toUpperCase()) // WELCOME TO FULL STACK
```



# String Methods

- Interactive question :
  - Convert a text into sentence format
  - Example:
    - "a Good Developer MAY bE aBle to Search Solutions to the Problems."
    - "A good developer may be able to search solutions to the problems."

Develop Code!

# String Methods

- Sample code :

```
let str = "a Good Developer MAY bE aBle to Search Solutions to the Problems.";

const first = str[0].toUpperCase()

let remaining = str.slice(1)
remaining = remaining.toLowerCase()

const result = first + remaining

console.log(result)
```



# trim()

- The **trim()** method eliminates whitespace from both sides of a string

```
let str = "     Welcome to Full Stack    ";

console.log(str.trim()) // Welcome to Full Stack
```



# startsWith()

- The **startsWith()** method determines whether a string begins with the characters of a specified string, returning true or false as appropriate.
- You can also specify the start index with the second parameter.

```
let str = "clarusway@Clarusway.com is our e-mail address";

console.log(str.startsWith("clarusway")) // true
console.log(str.startsWith("Clarusway")) // false

console.log(str.startsWith("Clarusway", 10)) // true
```

# endsWith()

- The **endsWith()** method determines whether a string begins with the characters of a specified string, returning true or false as appropriate.
- You can also specify the start index with the second parameter.

```
                  01234567890123456789
let str = "clarusway@Clarusway.com is our e-mail address";

console.log(str.endsWith("address")) // true

console.log(str.endsWith("Clarusway", 19)) // true
```



# Chaining Methods

- Function chaining is a pattern in JavaScript where multiple functions are called on the same object consecutively.
- Remember string methods return new strings, and we can continue working on the returned value with additional methods.



# Chaining Methods

```
let str = "         Clarusway Full Stack"

let str1 = str.trim() // Clarusway Full Stack

let str2 = str1.slice(0,9) // Clarusway

let str3 = str2.toUpperCase() // CLARUSWAY

console.log(str3) // CLARUSWAY

console.log(str.trim().slice(0,9).toUpperCase()) // CLARUSWAY
```