# JavaScript For Absolute Beginners

(Daniyal Nagori)

# JavaScript

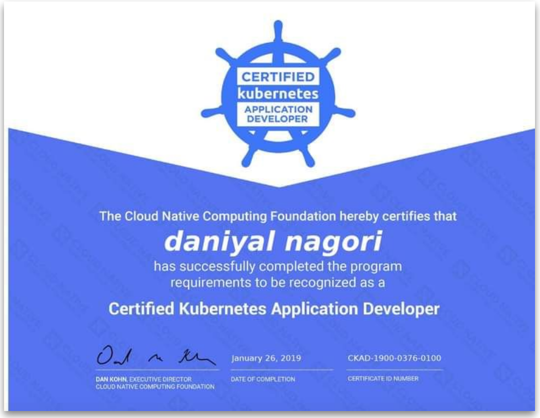
fb.com/daniyalnagori1237

linkedin.com/in/daniyalnagori

github.com/daniyalnagori

twitter.com/daniyalnagori1

# About Instructor

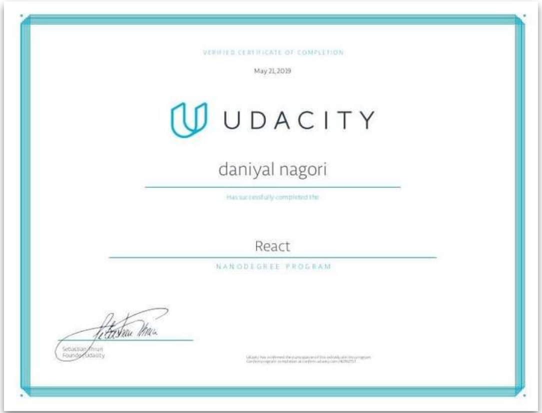

CERTIFIED
kubernetes
APPLICATION DEVELOPER

The Cloud Native Computing Foundation hereby certifies that
***daniyal nagori***
has successfully completed the program requirements to be recognized as a
**Certified Kubernetes Application Developer**

[illegible]
DAN KOHN, EXECUTIVE DIRECTOR
CLOUD NATIVE COMPUTING FOUNDATION

January 26, 2019
DATE OF COMPLETION

CKAD-1900-0376-0100
CERTIFICATE ID NUMBER


VERIFIED CERTIFICATE OF COMPLETION

May 21, 2019

UDACITY

daniyal nagori

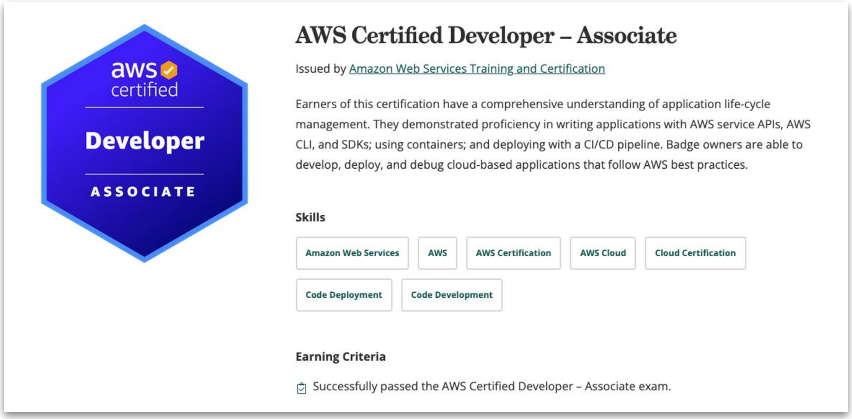
Has successfully completed the

React

NANODEGREE PROGRAM

[illegible]
Sebastian Thrun
Founder, Udacity

[illegible]



**AWS Certified Developer – Associate**

Issued by Amazon Web Services Training and Certification

Earners of this certification have a comprehensive understanding of application life-cycle management. They demonstrated proficiency in writing applications with AWS service APIs, AWS CLI, and SDKs; using containers; and deploying with a CI/CD pipeline. Badge owners are able to develop, deploy, and debug cloud-based applications that follow AWS best practices.

**Skills**

Amazon Web Services | AWS | AWS Certification | AWS Cloud | Cloud Certification | Code Deployment | Code Development

**Earning Criteria**

Successfully passed the AWS Certified Developer – Associate exam.

## Integrated Development Environment

## Setting up your environment

- There are many ways in which you can set up a JavaScript coding environment. Such as:
    - Integrated Development Environment (IDE). Example: VS Code, Sublime Text, Atom, etc.
    - Web browser. Example: Chrome, Firefox, etc.
    - Online editor (optional). Example: StackBlitz, Replit, etc.

## Adding Javascript to a Web Page

## Adding JavaScript to a web page

- There are two ways to link JavaScript to a web page.
  - The first way is to type the JavaScript directly in the HTML between two <script> tags.

```
<html>
    <script type="text/javascript">
        alert("Hello World!");
    </script>
</html>
```

  - The second way is to create a file with extension of .js and link it to our web page.

```
<html>
    <script type="text/javascript" src="hello_world.js"></script>
</html>
```

# ALERT

## ALERT

- The alert() method displays an alert box with a message and an OK button.
- The alert() method is used when you want information to come through to the user.
- The alert box takes the focus away from the current window, and forces the user to read the message.
- Do not overuse this method. It prevents the user from accessing other parts of the page until the box is closed.

# CONSOLE LOG

## CONSOLE LOG

- The **console.log()** method writes (**logs**) a message to the console.
- The **console.log()** method is useful for testing purposes.

## Document Write

## Document Write

- The **document.write()** method writes directly to an open (**HTML**) document stream.
- The **document.write()** method deletes all existing HTML when used on a loaded document.

## VARIABLES

## VARIABLES

- Variable means anything that can vary.
- A JavaScript variable is simply **a name of storage location**.
- A variable must have a unique name.

## Variables

- Variables are values in your code that can represent different values each time the code runs.
- The first time you create a variable, you declare it. And you need a special word for that: `let` , `var` , or `const` .
  Example: `let firstname = "Ali";`
- The commonly used naming conventions used for **variables** are camel-case.
  Example: `let firstName = "Ali";`

## Variables Scope

- **LOCAL**
  - Variables declared within a JavaScript function, become LOCAL to the function.

- **GLOBAL**
  - A variable declared outside a function, becomes GLOBAL.

## VARIABLE Names Legal & Illegal

## VARIABLE Names

- A variable name can't contain any spaces
- A variable name can contain only letters, numbers, dollar signs, and underscores.
- The first character must be a letter, or an underscore (-), or a dollar sign ($).
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are not allowed as the first character of variable.

## Comments

## Comments

- Single line Javascript comments **start with two forward slashes (//).**
- All text after the two forward slashes until the end of a line makes up a comment
- Even when there are forward slashes in the commented text.
- Multi-line Comments
- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored by JavaScript.

## Statements

## Statements

- A computer program is a list of "instructions" to be "executed" by a computer.
- In a programming language, these programming instructions are called statements.
- A JavaScript program is a list of programming statements.
- JavaScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon.

## Data types

## Primitive data types

- String
  - A string is used to store a text value.
    Example: `let firstName = "Ali";`
- Number
  - A number is used to store a numeric value.
    Example: `let score = 25;`
- Boolean
  - A boolean is used to store a value that is either `true` or `false`.
    Example: `let isMarried = false;`
- Undefined
  - An undefined type is either when it has not been defined or it has not been assigned a value.
    Example: `let unassigned;`
- Null
  - null is a special value for saying that a variable is empty or has an unknown value.
    Example: `let empty = null;`

## Template Literals

## Template Literals

A new and fast way to deal with strings is **Template Literals or Template String.**

**How we were dealing with strings before ?**

```
var myName = "daniyal" ;
var hello = "Hello "+ myName ;
console.log(hello); //Hello daniyal
```

## Template Literals

**What is Template literals ?**

As we mentioned before , it’s a way to deal with strings and specially dynamic strings ; so you don’t need to think more about what’s the next quote to use  single or double.

**How to use Template literals**

It uses a `backticks` to write string within it.

## typeof Operator

## Analyzing and modifying data types

- You can check the type of a variable by entering **`typeof`**.
  Example:

  ```
  let testVariable = 1;
  console.log(typeof testVariable);
  ```

- The variables in JavaScript can change types. Sometimes JavaScript does this automatically.
  Example:

  ```
  let v1 = 2;
  let v2 = “2”;
  console.log(v1 * v2); // 4 ← Type Number
  console.log(v1 + v2); // “22” ← Type String
  ```

# Analyzing and modifying data types

- There are three conversion methods:
  - `String()` ← `converts to string type`
  - `Number()` ← `converts to number type`
  - `Boolean()` ← `converts to boolean type`

## Operators

## Operators

- Arithmetic operators:
  - Addition

    Example:

    -
      ```
      let n1 = 1;
      let n2 = 2;
      console.log(n1 + n2); // 3
      ```

    -
      ```
      let str1 = “1”;
      let str2 = “2”;
      console.log(str1 + str2); // “12”
      ```

## Operators

- Arithmetic operators:
    - Subtraction
      Example:

        -
          ```
          let n1 = 5;
          let n2 = 2;
          console.log(n1 - n2); // 3
          ```
    - Multiplication
      Example:

        -
          ```
          let n1 = 5;
          let n2 = 2;
          console.log(n1 * n2); // 10
          ```

## Operators

- Arithmetic operators:
  - Division

    Example:

    -
      ```
      let n1 = 4;
      let n2 = 2;
      console.log(n1 / n2); // 2
      ```
  - Exponentiation

    Example:

    -
      ```
      let n1 = 2;
      let n2 = 2;
      console.log(n1 ** n2); // 4
      ```

## Operators

- Arithmetic operators:
  - Modulus

    Example:

    -
      ```
      let n1 = 10;
      let n2 = 3;
      console.log(n1 % n2); // 1
      ```

## Operators

- Assignment operators:
  - Assignment operator are used to assigning values to variables. Example:
    - **let n = 5;**
      **console.log(n); // 5**
      **n += 5;**
      **console.log(n); // 10**
      **n -= 5;**
      **console.log(n); // 5**

## Operators

- Comparison operators:
    - Comparison operator are used to compare values of variables. Example:

        -
          ```
          let n = 5;
          console.log(n == 5); // true
          console.log(n === 5); // true
          console.log(n != 5); // false
          console.log(n > 8); // false
          console.log(n < 8); // true
          console.log(n >= 8); // false
          console.log(n <= 8); // true
          ```

## Math Expressions
## Familiar Operators

## Expressions

- An Expression is a combination of values, variables, function call and operators, which computes to a value.
- The computation is called an evaluation.
- “Daniyal” + “Nagori”

## Math Expressions Familiar Operators

- Wherever you can use a number, you can use a math expression.
- **“+”, “-”, “*”, “/”** and **“%”** are commonly used operators.
- **“%” (Modulus)** operator works similar to “/” but instead of the result, It gives you the remainder when the division is executed.
- Examples:
  - let add = 2 + 3;  // 5
  - let subtraction = 8 - 4;  // 4
  - let multiplication = 2 * 2;  // 4
  - let division = 4 / 2;  // 2
  - let modulus = 9 % 3;  // 0

## Math Expressions
## UnFamiliar Operators

## Math Expressions UnFamiliar Operators

- There are several specialized math expressions such as “++”, “--” and “**”.
  - **“++” :** It increments the variable by 1.
  - **“--” :** It decrements the variable by 1.
  - **“**” :** Exponentiation is one of the newer operators in JavaScript, and it allows us to calculate the power of a number by its exponent.

## Math Expressions UnFamiliar Operators

### Post Increment vs Pre Increment

- Post Increment
  - The operator increases the variable var1 by 1 but returns the value before incrementing.
  - Example:

    -
      ```
      let i = 1;
      let num = i++  //  1
      ```
- Pre Increment
  - The operator increases the variable var1 by 1 but returns the value after incrementing.
  - Example:

    -
      ```
      let i = 1;
      let num = ++i  //  2
      ```
- Same rule for the **Decrement**

## Math Expressions
## Eliminating Ambiguity

## Math Expressions Eliminating Ambiguity

Complex arithmetic expressions can pose a problem, one that you may remember from high school algebra.

- Examples:
  - var totalVal = (5 + 2)  *  3 + 6;  // 27
  - var totalVal = (2 * 4) * 4 + 2; // 34

## Concatenating Text String

## Concatenating Text Strings

- The **concat()** method joins two or more strings.
- The **concat()** method does not change the existing strings.
- The **concat()** method returns a new string.
- You can also use “**+**” operator to concatenate multiple strings.
- Examples:
  - let userName = Daniyal
    console.log("Thanks, " + userName + "!")

## Prompts

## Prompts

- The **prompt()** method displays a dialog box that prompts the user for input.
- The **prompt()** method returns the input value **(String)** if the user clicks **"OK"**, otherwise it returns **null**.
- When a **prompt box** pops up, the user will have to click either **"OK"** or **"Cancel"** to proceed.
- Do not overuse this method. It prevents the user from accessing other parts of the page until the box is closed.

## If, Else, Else If Statements

## If, Else and Else If Statements

- Use **if** to specify a block of code to be executed, if a specified condition is true.
- Use **else** to specify a block of code to be executed, if the same condition is false.
- Use **else if** to specify a new condition to test, if the first condition is false.

## If, Else and Else If Statements - Examples

- If Example:

    -
      ```
      let x = prompt("Where does the Pope live?");
      let correctAnswer = "Pakistan";
      if (x == correctAnswer ) {
              alert("Correct!");
       }
      ```
- else - Example

    -
      ```
      let x = prompt("Where does the Pope live?");
      let correctAnswer = "Pakistan";
      if (x == correctAnswer ) {
              alert("Correct!");
       } else {
              alert("Wrong!");
      }
      ```

## If, Else and Else If Statements - Examples

- Else if - Example

  -
    ```
    let x = prompt("Where does the Pope live?");
    let correctAnswer = "Pakistan";
    if (x == correctAnswer ) {
            alert("Correct!");
    } else if (x=="Pakista") {
            alert("Close!");
    } else {
            alert("Wrong!");
    }
    ```

## Comparison Operators

## Comparison Operators

- Comparison and Logical operators are used to test for **true** or **false**.
- Comparison operators are used in logical statements to determine equality or difference between variables or values.
- **“==”**, **“===”**, **“!=”**, **“!==”**, **“>”**, **“<”**, **“>=”** and **“<=”** are some of the comparison operators.

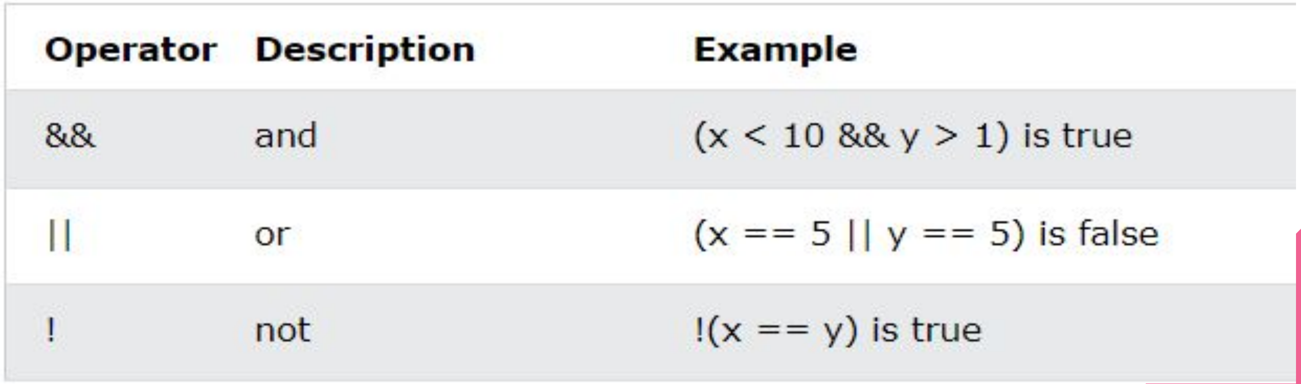
## Comparison Operators - Examples

- let a = 2 + 2 == "4" // true
- let b = 2 + 2 === "4" // false
- let c = 2 + 2 > 4 // false
- let d = 2 + 2 >= 4 // true
- let e = 2 + 3 !== 5 // false

## Testing Sets Of Conditions (Logical Operators)

## Testing Sets Of Conditions (Logical Operators)

- Logical operators are used to determine the logic between variables or values.
- Given that **`x = 6`** and **`y = 3`**, the table below explains the logical operators:



| Operator | Description | Example |
|---|---|---|
| && | and | (x < 10 && y > 1) is true |
| \|\| | or | (x == 5 \|\| y == 5) is false |
| ! | not | !(x == y) is true |

## Testing Sets Of Conditions (Logical Operators) - Examples

- let x = 6
  let y = 10

  True True = True

  let a1 = x < y && x === 6 // true
  let a2 = x < y && x !== 6 // false
  let a3 = x === y || y === 10 // true   If One Statement Two then True
  let a4 = (x===6 && y===4) || x < y // true

  False

## If Statement Nested

## If Statement Nested

- JavaScript allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

## If Statement Nested - Example

```
// Ticketing system
let country = prompt("Where do you live?")
// Number() function is used to convert the string to number
let age = Number(prompt("What's your age?"))

if (country === "pakistan") {
    if (age >= 18) {
        console.log("Here is your ticket")
    } else {
        console.error("Age restriction")
    }
} else {
    console.log("Invalid country")
}
```

## Array

## Array

- **The Problem:** Suppose you have five fruits and you want to store them in the variable, But you have to create five variables to store the fruits which is not an efficient approach, what if you have thousands of fruits?

  -
    ```
    let fruit1 = "apple"
    let fruit2 = "banana"
    let fruit3 = "grapes"
    let fruit4 = "strawberry"
    let fruit5 = "orange"
    ```

- **The Solution:** Here the array comes into play which helps to store multiple data in a single variable.

  -
    ```
    let fruits = ["apple","banana", "orange", "grapes", "strawberry"]
    ```

## Array - More About Array

- An array is a special variable, which can hold more than one value.
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- In JavaScript, arrays always use numbered indexes.
- Array indexes start with 0.
- Examples:

    -
      ```
      let fruits = ["apple","banana", "orange", "grapes", "strawberry"]
      fruits[0] // apple
      fruits[3] // grapes
      ```

    -
      ```
      let x = [1, 2, "daniyal"] // Arrays can store multiple types of data
      ```

## Arrays: Adding and removing elements

## Arrays: Adding and removing elements

- When you work with arrays, it is easy to **remove elements** and **add new elements**. This is what **popping** and **pushing** is.
- The **pop()** method removes the last element from an array:
- The **pop()** method returns the value that was **"popped out"**
- The **push()** method adds a new element to an array (at the end).
- The **push()** method returns the new array length.

## Arrays: Adding and removing elements - Examples

- var pets = [];
  pets[0] = "dog"; // adds “dog” to an array at 0 index
  pets[1] = "cat"; // adds “cat” to an array at index 1

  pets.pop(); // removes the last element of an array which is cat in our case
  pets.push("parrot"); // adds a new element to an array

## Arrays: Removing, inserting, and extracting elements

## Arrays: Removing, inserting, and extracting elements

- Shifting is equivalent to popping, but working on the **first element** instead of the **last**.
- The **shift()** method removes the first array element and "**shifts**" all other elements to a lower index.
- The **shift()** method returns the value that was "**shifted out**".
- The **unshift()** method adds a new element to an array (at the beginning), and "**unshifts**" older elements:
- The **unshift()** method returns the new array length.

## Arrays: Removing, inserting, and extracting elements - Example

- var pets = [];
  pets[0] = "dog"; // adds “dog” to an array at 0 index
  pets[1] = "cat"; // adds “cat” to an array at index 1

  pets.shift(); // removes the first element of an array which is cat in our case
  pets.unshift("parrot"); // adds a new element to an array (at the beginning)

## Arrays: Removing, inserting, and extracting elements

### Splicing and Slicing Arrays

- The **splice()** method adds new items to an array.
  - Example:
    ```
    const fruits = ["Banana", "Orange", "Apple", "Mango"];
    fruits.splice(2, 0, "Lemon", "Kiwi");
    // adds elements to an array at 2nd index
    // deleted 0 elements
    ```
- The **slice()** method slices out a piece of an array.
  - Example:
    ```
    const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
    const citrus = fruits.slice(1); // [Orange,Lemon,Apple,Mango]
    ```
  - **Notes:**
    The **slice()** method creates a new array.

## JavaScript Objects

# JavaScript Objects

- In real life, a car is an object: A car has properties like weight and color, and methods like start and stop.
- All cars have the same properties, but the property values differ from car to car.
- All cars have the same methods, but the methods are performed at different times.
    - Example:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

| Object | Properties | Methods |
|---|---|---|
| | car.name = Fiat<br>car.model = 500<br>car.weight = 850kg<br>car.color = white | car.start()<br>car.drive()<br>car.brake()<br>car.stop() |

## JavaScript Objects- Example

```
// Person Object - Key Value Pair syntax
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  "eye-Color": "blue",
};

// Access Obj Properties
person.age; // 50
person["eye-Color"]; // blue
```

## The Arrays Object

## The Arrays Object

We Can store multiple Objects in an Array

```
// Persons Array - Key Value Pair syntax
const persons = [
  {
    firstName: "Hamzah",
    lastName: "Syed",
    age: 22,
    "eye-Color": "brown",
  },
  {
    firstName: "John",
    lastName: "Doe",
    age: 50,
    "eye-Color": "blue",
  },
];

// Access Person 1
persons[0].age // 22
persons[1].age // 50
```

## For Loops

## For Loops

- Loops are handy, if you want to run the same code over and over again, each time with a different value.

```
// Syntax:
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

- From the example above, you can read
- Expression 1 sets a variable before the loop starts (let i = 0).
- Expression 2 defines the condition for the loop to run (i must be less than 5).
- Expression 3 increases a value (i++) each time the code block in the loop has been executed.

## For Loops - Examples

- Example 1

```
for (let i = 0; i < 3; i++) {
   console.log("Hello World")
}
```

- Example 2

```
for (let i = 0; i < 3; i++) {
  console.log("Hello World" + i)
}
```

## For Loops - Examples

- Example 3

```
var cleanestCities = ["Karachi", "Lahore", "Islamabad", "Peshawar"];

for (var i = 0; i <= 4; i++) {
  if ("Islamabad" === cleanestCities[i]) {
    console.log("It's one of the cleanest cities");
    break;
  }
}
```

## For Loops - Examples

- Example 4

```
var cleanestCities = ["Karachi", "Lahore", "Islamabad", "Peshawar"];

var matchFound = "no";

for (var i = 0; i <= 4; i++){
if ("Islambad" === cleanestCities[i]) {
  matchFound = "yes";
  alert("It's one of the cleanest cities");
}
}

if (matchFound === "no") {
  alert("It's not on the list");
}
```

## For Loops - Examples

- Example 5

```
var cleanestCities = ["Karachi", "Lahore", "Islamabad", "Peshawar"];

var numElements = cleanestCities.length;
var matchFound = false;

for (var i = 0; i < numElements; i++) {
  if ("Islamabad" === cleanestCities[i]) {
    matchFound = true;
    console.log("It's one of the cleanest cities");
    break;
  }
}
if (matchFound === false) {
  console.log("It's not on the list");
}
```

## Nested For Loops

## Nested For Loops - Example

- A nested loop is a loop within a loop.

```
var firstNames = ["BlueRay ", "Upchuck ", "Lojack ", "Gizmo ", "Do-Rag "];
var lastNames = ["Zzz", "Burp", "Dogbone", "Droop"];
var fullNames = [];

for (var i = 0; i < firstNames.length; i++) {
  for (var j = 0; j < lastNames.length; j++) {
    fullNames.push(firstNames[i] + lastNames[j]);
  }
}
```

## Changing case

## Changing case

- The toLowerCase() method converts a string to lowercase letters.
- The toLowerCase() method does not change the original string.
- The toUpperCase() method converts a string to uppercase letters.
- The toUpperCase() method does not change the original string.

## Strings: Measuring length and extracting parts

# Changing case

- The length property returns the length of a string.
- The length property of an empty string is 0.
- The slice() method extracts a part of a string.
- The slice() method returns the extracted part in a new string.
- The slice() method does not change the original string.
- The start and end parameters specifies the part of the string to extract.
- The first position is 0, the second is 1, ...
- A negative number selects from the end of the string.
  - Example:

    -
      ```
      let cityToCheck = "pakistan"
      var firstChar = cityToCheck.slice(0, 1); // p
      ```

## Changing case - Examples

- Example 1

```
let cityToCheck = "pakistan"

var firstChar = cityToCheck.slice(0, 1); //p
var otherChars = cityToCheck.slice(1); // akistan
firstChar = firstChar.toUpperCase();  // P
otherChars = otherChars.toLowerCase(); // akistan
var cappedCity = firstChar + otherChars; // Pakistan
```

## Changing case - Examples

- Example 2

```
var month = prompt("Enter a month");
var charsInMonth = month.length;
if (charsInMonth > 3) {
    monthAbbrev = month.slice(0, 3);
}
console.log(monthAbbrev)
```

## Strings: Finding segments

## Strings: Finding segments

- The indexOf() method returns the position of the first occurrence of a value in a string.
- The indexOf() method returns -1 if the value is not found.
- The indexOf() method is case sensitive.
  - Example:

    -
      ```
      var text = "To be or not to be."; // 3
      Var segIndex = text.indexOf("be");// 3
      ```
- The lastIndexOf() method returns the index (position) of the last occurrence of a specified value in a string.
- The lastIndexOf() method searches the string from the end to the beginning.
- The lastIndexOf() method returns the index from the beginning (position 0).
- The lastIndexOf() method returns -1 if the value is not found.
- The lastIndexOf() method is case sensitive.
  - Example:

    -
      ```
      var text = "To be or not to be.";
      var segIndex = text.lastIndexOf("be");  // 16
      ```

## Strings: Finding a character at a location

## Strings: Finding a character at a location

- The charAt() method returns the character at a specified index (position) in a string.
- The index of the first character is 0, the second 1, …
    - Example
        - let firstName = "hamzah"
          var firstChar = firstName.charAt(0) // h

## Strings: Replacing characters

## Strings: Replacing characters

- The **replace()** method searches a string for a value or a regular expression.
- The **replace()** method returns a new **string** with the value(s) replaced.
- The **replace()** method does not change the original string.

Example:

```javascript
// Example 1
let text = "Visit Microsoft!";
let result = text.replace("Microsoft", "W3Schools"); // Visit W3Schools!

// Example 2
let text = "Mr Blue has a blue house and a blue car";
let result = text.replace(/blue/g, "red"); // Mr Blue has a red house and a red car
```

## Rounding numbers

## Rounding numbers

- The **Math** object allows you to perform mathematical tasks.
- **Math** is not a constructor. All **properties/methods** of Math can be called by using Math as an object.
- The **Math.round()** method rounds a number to the nearest integer.
- 2.49 will be rounded down (2), and 2.5 will be rounded up (3).

Example:

```
let scoreAvg = 132.23
var numberOfStars = Math.round(scoreAvg); // 132
```

## Rounding numbers

- The **Math.ceil()** method rounds a number rounded UP to the nearest integer.
- The **Math.floor()** method rounds a number DOWN to the nearest integer.

Example

```
let a = Math.ceil(0.60); // 1
let b = Math.ceil(-4.40); // -4

let c = Math.floor(0.60); // 0
let d = Math.floor(-4.40); // -5
```

## Generating random numbers

## Generating random numbers

- The **Math.random()** method returns a random number from 0 (inclusive) up to but not including 1 (exclusive).

Example:

```
let x = Math.random(); // 0.0000000000000000 - 0.9999999999999999
let y = Math.random() * 10; // 0 - 9.999
let z = Math.random() * 100; // 0 - 99.999999
```

## Converting strings to integers and decimals

## Converting strings to integers and decimals

- The **parseInt** method parses a value as a string and returns the first integer.
- The **parseFloat()** method parses a value as a string and returns the first number.

Example:

```
// parseInt
let x = parseInt("10"); // 10
let y = parseInt("10.00"); // 10
let z = parseInt("10.33"); // 10
let p = parseInt("Hello"); // NaN
// parseFloat
let x = parseFloat("10"); // 10
let y = parseFloat("10.00"); // 10
let z = parseFloat("10.33"); // 10.33
let p = parseFloat("Hello"); // NaN
```

## Converting strings to numbers, numbers to strings

## Converting strings to numbers, numbers to strings

- The **Number()** method converts a value to a number.
- The **String()** method converts a value to a string.

Example:

```
// Number()
let x = Number("10"); // 10
let z = Number("10.33"); // 10.33
let y = Number(true); // 1
let p = Number("Hello"); // NaN
// String()& .toString()
let p = String(22); // "22"
let p2 = 44
let p3 = p2.toString() // "44"
```

## Controlling the length of decimals

## Controlling the length of decimals

- The **toFixed()** method converts a number to a string.
- The **toFixed()** method rounds the string to a specified number of decimals.

Example:

```
let total = 25.154123
let prettyTotal = total.toFixed(2); // 25.15
```

## Javascript Built-in Constructors

## Javascript Built-in Constructors

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the new keyword.The purpose of a constructor is to create a new object and set values for any existing object properties.

```
new String()     // A new String object
new Number()     // A new Number object
new Boolean()    // A new Boolean object
new Object()     // A new Object object
new Array()      // A new Array object
new Function()   // A new Function object
new Date()       // A new Date object
```

## Getting the current date and time

## Getting the current date and time

- JavaScript **Date** Objects let us work with dates:

Example:

```
const date = new Date()// Mon Oct 31 2022 19:37:34 GMT+0500 (Pakistan Standard Time)
const date = new Date("2000", 11, 2) // Sat Dec 02 2000 00:00:00 GMT+0500 (Pakistan Standard Time)
```

## Extracting parts of the date and time

## Extracting parts of the date and time

- Method in Date Object

```
let currentMonth = d.getMonth();
let dayOfMonth = d.getDate();
let currYr = d.getFullYear();
let currentHrs = d.getHours();
let currMins = d.getMinutes();
let currSecs = d.getSeconds();
let currMills = d.getMilliseconds();
let millsSince = d.getTime(); // getTime gives you the number of milliseconds that have elapsed since
midnight, Jan. 1, 1970.
```

## Specifying a date and time

## Specifying a date and time

- **new Date(date string)** creates a date object from a date string:

```
const d1 = new Date("October 13, 2014 11:13:00"); // Mon Oct 13 2014 11:13:00 GMT+0500 (Pakistan
Standard Time)
const d2 = new Date("2022-03-25"); // Fri Mar 25 2022 05:00:00 GMT+0500 (Pakistan Standard Time)
const d3 = new Date(2018, 11, 24, 10, 33, 30, 0); // Mon Dec 24 2018 10:33:30 GMT+0500 (Pakistan
Standard Time)
```

## Functions

## Functions

- **The Problem**
  Suppose we have a website where we need to show a greeting message 3 times per page and we have 4 pages now we need to write a greeting message 3 * 4 = 12 times manually. What if we need to change something in the greeting message? We need to reflect the changes in all the places this is not an efficient approach.

- **Solution:**
  Here **Functions** come into play. We created a simple greeting message function and we can just call it all over the website this way we just need to write a greeting message once and reuse it all over the website

```
// Creating a Function
function showGreetingMessage() {

}
// Invoking a Function
showGreetingMessage() // "Hamzah, Welcome to our Website"
```

## Functions

- A **JavaScript function** is a block of code designed to perform a **particular task.**
- A JavaScript function is executed when "something" **invokes it** (calls it).
- A JavaScript function is defined with the **function** keyword, followed by a name, followed by **parentheses ()**.
- Function names can contain letters, digits, underscores, and dollar signs **(same rules as variables).**

## Functions - Parameters and arguments

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

```
// Defining a Function
function showGreetingsMessage(name) {
    console.log(name + ", Welcome to our Website")
    // OR
    // console.log(`${name}, Welcome to our Website`)
}
// Invoking a Function
showGreetingsMessage("Hamzah") // "Hamzah, Welcome to our Website"
showGreetingsMessage("Ali") // "Ali, Welcome to our Website"
```

## Functions - return

- When JavaScript reaches a **return** statement, the function will stop executing.
- Functions often compute a return value. The return value is "returned" back to the "caller":

```
// Defining a Function
function showGreetingsMessage(name) {
    return `${name}, Welcome to our Website`
}
// Invoking a Function
let greetingMessage1 = showGreetingsMessage("Hamzah") // "Hamzah, Welcome to our Website"
let greetingMessage2 = showGreetingsMessage("Ali") // "Ali, Welcome to our Website"
```

## Functions - return

```
function addTwoNumbers(x, y) {
    let result = x + y
    return result
}
let result = addTwoNumbers(100, 200)

if (result > 100) {
    console.log("Big Number")
}
```

## Functions - Default or unsuitable parameters

```
// name = "Hello" is the default value
function showGreetingsMessage(name = "Hello") {
    console.log(`${name}, Welcome to our Website`)
}

showGreetingsMessage() // "Hello, Welcome to our Website"
showGreetingsMessage("Ali") // "Ali, Welcome to our Website"
```

## Functions - Arrow Function

- Arrow functions were introduced in ES6.
- Arrow functions allow us to write shorter function syntax:

```
const showGreetingsMessage = (name = "Hello") => `${name}, Welcome to our Website`

let greetingMessage1 = showGreetingsMessage() // "Hello, Welcome to our Website"
let greetingMessage2 = showGreetingsMessage("Ali") // "Ali, Welcome to our Website"
```

## What is **this**?

## What is **this**?

- In JavaScript, the **this** keyword refers to an **object**.
- **Which** object depends on how **this** is being invoked (used or called).
- The **this** keyword refers to different objects depending on how it is used:

## What is **this**? - **this** Alone

- When used alone, this refers to the global object.
- Because this is running in the global scope.
- In a browser window the global object is [object Window]:

```
console.log(this===window) // true

let x = this
console.log(x) // [object Window]
```

## What is **this**? - **this** in Object

- When used in an object method, **this** refers to the **object**.
- In the example on top of this page, **this** refers to the person object.
- Because the **fullName** method is a method of the **person** object.

```
const person = {
    firstName: "John",
    lastName: "Doe",
    id: 5566,
    fullName: function () { // Method
        return this.firstName + " " + this.lastName;
    }
};

console.log(person.fullName()) // Hamzah Syed
```

## Spread Operator And Rest Parameters

## Spread And Rest Operator - Spread Operator

- The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo]; // [1, 2, 3, 4, 5, 6]
```

## Spread Operator And Rest Parameters - Rest Parameter

- Similar to the spread operator, we have the rest parameter.
- It has the same symbol as the spread operator, but it is used inside the function parameter list

```
const addUnlimitedNumbers = (...numbers) => {
    let result = 0
    for (let i = 0; i < numbers.length; i++) {
        result += numbers[i]
    }
    return result
}

const result = addUnlimitedNumbers(0,2,3,45)
console.log(result)
```

## Local vs Function Scope

## Local vs Function Scope

- **var** is function scope
- **let** is block scope

```
// var Example
function doingStuff() {
    if (true) {
        var x = "local";
    }
    console.log(x); // local
}
console.log(x); // error - because var is function scoped
doingStuff();

// let Example
function doingStuff() {
    if (true) {
        let x = "local";
    }
    console.log(x); // error - because let is block scoped
}
doingStuff();
```

## Immediately invoked function expression (IIFE)

## Immediately invoked function expression (IIFE)

- An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined

```
// Normal Function
(function () {
    console.log("Hello Hamzah, Welcome to our Website")
})();

// Arrow Function
((name) => {
    console.log(`Hello ${name}, Welcome to our Website`)
})("hamzah");
```

## Recursive functions

## Recursive functions

- A function is recursive if it calls itself and reaches a stop condition.

```
function getRecursive(nr) {
    console.log(nr);
    if (nr > 0) {
        getRecursive(--nr); // Calling Itself
    }
}
getRecursive(3);
```

- `getRecursive(3)`
    - `getRecursive(2)`
        - `getRecursive(1)`
            - `getRecursive(0)`
            - done with `getRecursive(0)` execution
        - done with `getRecursive(1)` execution
    - done with `getRecursive(2)` execution
- done with `getRecursive(3)` execution

## Recursive functions - Example

```
function logRecursive(nr) {
    console.log("Started function:", nr);
    if (nr > 0) {
        logRecursive(n - 1);
    } else {
        console.log("done with recursion");
    }
    console.log("Ended function:", nr);
}
logRecursive(3);
```

## Nested functions

## Nested functions

- Just as with loops, if statements, and actually all other building blocks, we can have functions inside functions.

```
function doOuterFunctionStuff(nr) {
    console.log("Outer function");
    function doInnerFunctionStuff(x) {
        console.log(x + 7);
        console.log("I can access outer variables:", nr);
    }
    doInnerFunctionStuff(nr);
}
doOuterFunctionStuff(2);
```

## Anonymous functions

## Anonymous functions

- We can also create functions without names if we store them inside variables.

```
let functionVariable = function () {
    console.log("Not so secret though.");
};
functionVariable()
```

## Concurrency

## Concurrency

- Concurrency is whenever things are happening "at the same time" or in parallel.

## Promises

## Concurrency - Promises

- A JavaScript Promise object can be:
  - Pending
  - Fulfilled
  - Rejected
- While a Promise object is "**pending**" (working), the result is undefined.
- When a Promise object is "**fulfilled**", the result is a value.
- When a Promise object is "**rejected**", the result is an error object.

## Concurrency - Promises Example

```
let promise = new Promise(function (resolve, reject) {
    // do something that might take a while
    // let's just set x instead for this example
    let x = 20;
    if (x > 10) {
        return resolve(x); // on success
    } else {
        reject("Too low"); // on error
    }
});

promise.then((value) => {
    console.log("Success:", value);
}).catch((err) => console.log(err))
```

## Concurrency - Promises Example

```
let promise = new Promise(function (resolve, reject) {
    let x = 20;
    if (x > 10) {
        // Wait 2 seconds
        setTimeout(() => {
            return resolve(x); // on success
        }, 2000)

    } else {
        reject("Too low"); // on error
    }
});

promise.then((value) => {
    console.log("Success:", value);
}).catch((err) => console.log(err))
```

# Concurrency - Promises Example

```
// Promise Chain
const promise = new Promise((resolve, reject) => {
    resolve("success!");
})
    .then(value => {
        console.log(value);
        return "we";
    })
    .then(value => {
        console.log(value);
        return "can";
    })
    .then(value => {
        console.log(value);
        return "chain";

    })
    .then(value => {
        console.log(value);
        return "promises";
    })
    .then(value => {
        console.log(value);
    })
    .catch(value => {
        console.log(value);
    })
```

## Async/Await

## Async/Await

- **async** and **await** make promises easier to write
- **async** makes a function return a **Promise**
- **await** makes a function wait for a Promise
- The **await** keyword can only be used inside an async function.
- The **await** keyword makes the function pause the execution and wait for a resolved promise before it continues:

## Async/Await

```
function saySomething(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve("something" + x);
        }, 2000);
    });
}
async function talk(x) {
    const words = await saySomething(x);
    console.log(words);
}
talk(2);
talk(4);
talk(8);
```