Agenda

- Functions
- Objects
- ~~Array~~

Anonymous function

- Function without a name is called as anonymous function
- Syntax: var = function() { // body }
- E.g.

```
var multiply = function(p1, p2) {
console.log(“p1 * p2 = “ + (p1 * p2));
}
```

Arrow Functions

- Arrow functions are a modern, concise way to write function expressions in JavaScript, introduced in ECMAScript 2015 (ES6).
- They provide a shorter syntax than traditional function expressions, making the code more readable, especially for simple, one-line functions.
- An arrow function has three main parts:
    1. Parameters: Enclosed in parentheses ().
    2. The "fat arrow": The => syntax.
    3. Function body: The code to be executed.

```
const add = (a, b) => {
    const res = a + b
    console.log(res)
}

// Omitting parentheses for a single parameter:
const square = n => {
    const res = n * n
    console.log(res)
}

//Omitting curly braces and return for a single expression: For one-line arrow functions, the return is implicit.
const square = x => x * x;

//Returning an object literal: To implicitly return an object, you must wrap the curly braces in parentheses to avoid a syntax error.
const createObject = () => ({ message: "Hello" });
```

- If the function body has multiple lines, you must use curly braces and an explicit return statement.

Higher Order Functions

- A higher-order function (HOF) is a function that either takes one or more functions as arguments, returns a function, or both.
- This is possible in JavaScript because functions are "first-class citizens," meaning they can be treated like any other value (such as a string or number).
- A function that is passed as an argument to a higher-order function is known as a callback function.
- This is a common pattern used to handle asynchronous operations and create reusable code.

```
//executer function
// Higher Order functions
function executer(n1, n2, fn) {
    const res = fn(n1, n2)
    console.log('res - '+res)
}

const add = (n1,n2)=>{
    const res = n1+n2;
    return res;
}
// Here add is called as callback function
executer(10,20,add)

// we can pass the callback function by directly declaring it in the argument
executer(10,20,(n1,n2)=>n1-n2)
executer(10,20,(n1,n2)=>n1*n2)
```

Function Hidden Parameters

- Every function in JS accepts 2 hidden parameters

1. this:
    - refers the current object on which the function is called
    - if function is called on an instance, the instance becomes this
    - E.g.

```
var p = new Object
p.myCanVote = canVote;

// the variable p becomes this inside canVote()
p.myCanVote()
```

    - if function is called without an instance then Window becomes this

```
```

```
function canVote() { /* body */}

// inside canVote Window becomes this
canVote();
```

2. arguments:

- is of type array
- array of parameter values passed to a function
- E.g.

```
function add() {
var answer = 0;
for (var index = 0; index < arguments.length; index++) {
answer += arguments[index];
}
console.log(answer)
}
```

Object

- collection of properties (data members or fields) and methods
- Everything in JS is an Object, even functions are also objects
- To create an object (instance)
    - Use Object Literal ({})
    - Use new keyword
    - Use constructor function

```
// using Object
var c1 = new Object();
c1.model = "i10";
c1.company = "Hyundai";

// using construction function
function Car(model, company) {
    this.model = model;
    this.company = company;
}
var c2 = new Car("Fabia", "Skoda");

// using Object Literal
// also called as JSON
var c3 = {
    model: "X5",
    company: "BMW"
};
```

```
var cars = [c1, c2, c3];
for (var index = 0; index < cars.length; index++) {
    var car = cars[index];
    console.log("Model: " + car.model);
    console.log("Company: " + car.company);
}
```