"A closer look at functions"

* Default parameters:→

* Before ES6 we can make the default params to a function like this:→

* As seen here this is the way we make default values earlier.

```
const bookings = [];
const createBooking = function (flightNum, numOfPass = 1, price = 199) {
    numOfPass = numOfPass || 1;
    flightNum = flightNum || 1;

  const booking = {
    flightNum,
    numOfPass,
    price,
  };
};
```

* After ES6:→

The default params could be passed here.

```
const bookings = [];
const createBooking = function (flightNum, numOfPass = 1, price = 199) {
    numOfPass = numOfPass || 1;
    flightNum = flightNum || 1;

  const booking = {
    flightNum,
    numOfPass,
    price,
  };
};
```

This mean if I don't pass any value to this func it will take this value as a default.

* Passing arguments Values vs Reference :→

* Siple Primitive values here is flight will be passed by value.

```
const flight = "LH234";
const zeyad = {
  name: "Zeyad Albadawy",
  passport: 123456789,
};

const checkIn = function (flightNum, passenger) {
  flightNum = "L2H235";
  passenger.name = "MR. " + passenger.name;

  if (passenger.passport === 123456789) alert("Checked IN");
  else alert("Wrong Passport");
};

checkIn(flight, zeyad);
console.log(flight);
console.log(zeyad);
```

* This mean when I change its value in the checkIn and try to prints its value will be 'LH234', There is no change.

* otherwise when I pass zeyad object it will be passed by Reference of its heap, then when I try to log the zeyad object after this function Call it will be changed into MR. Zeyad AlBadawy.

* Note that:— JavaScript doesn't have Call by Reference such as cpp and the thing which happens in the case of zeyad object is that it passed the address of this object in heap which is a value.

* First class and higher order Functions :→

* First Class Functions :→ it means that functions are simply a values.

* higher order Functions :→ means that function which can Return a Function or Function which can accepts a Call back Function or Both of them.

Function accepts a Call back Functions :→

* The Fn function is called the Call back function.

* notice here i don't Call it, i just pass it to another Func

```
const oneWord = function (str) {
  return str.replace(/ /g, "").toLowerCase();
};

const upperFirstWord = function (str) {
  const [first, ...others] = str.split(" ");
  return [first.toUpperCase(), ...others].join(" ");
};

// Hight Order Functions
const transformer = function (str, fn) {
  console.log(`The Original String : ${str}`);
  console.log(`The Transformed String is ${fn(str)}`);
  console.log(`It Is Transformed By ${fn.name}`);
};

transformer("Java Script Is The Best!", upperFirstWord);
transformer("Java Script Is The Best!", oneWord);
```

* note that the Fn accepts a property name which indicates that is treated like objects.

* Function returns other Function :→

```
const greet = function (greeting) {
  return function (name) {
    console.log(`${greeting} ${name}`);
  };
};

const greeterHey = greet('Hey');
greeterHey('Jonas');
greeterHey('Steven');
```

* greetHey is a function which got from return.

* So that we can pass the name to greetHey Func

* The call and apply methods :→

Consider we try to build a booking flight for different air line companies like this.

```
const lufthensa = {
  airline: "lufthensa",
  iateCode: "LH",
  bookings: [],
  book(flightNum, name) {
    console.log(
      `${name} booked a seat on ${this.airline} flight ${this.iateCode}${flightNum}`
    );

    // Array Of Objects ->>
    this.bookings.push({
      flight: `${this.airline} flight ${this.iateCode}${flightNum}`,
      name,
    });
  },
};

lufthensa.book(239, "Zeyad_01");
lufthensa.book(259, "Zeyad_02");
```

* after that I decided to build the same thing for another company called euroings like this :→

but I dont create a book method instead of that I will take it from the First Company like this.

```
const eurowings = {
  airline: "eurowings",
  iateCode: "EW",
  booking: [],
};
```

but when I try that it seems to be an error.

```
const eurowingsBook = lufthensa.book;
eurowingsBook(123, "zeyad_03");
```

```
TypeError: Cannot read properties of undefined (reading 'airline')
    at book (script.js:80:40)
    at script.js:105:1
```

You know why this happens?

That's because there is a this keyword in the first Company and when I try to do this

Const eurowingBook = lufthensa.book;

This makes eurowingBook Treated like a regular Function and this in the regular Function will Point to undefined.

When I try to access a property of undefined This Causes the problem.

So how to overCome such a Problem?

note that:— Functions has Properties like objects.

* I need to make this keyword point to the object which I tries to access, and this can be done by the call method.

```
const eurowingsBook = lufthensa.book;

eurowingsBook.call(eurowings, 123, "Zeyad_03");
console.log(lufthensa.bookings);
console.log(eurowings.bookings);
```

The object which I tries to access.

other params of the original method which is book.

* in the parameters of call we need to specify which object this will refer to at each call.

*instead of that we can use the bind which works the same but returns aspesific func

```
const bookEW = book.bind(eurowings);
bookEW(23, 'Steven Williams');
```

Const Book = lunthensa.book;

This will contain the book method.

* BookEW will be attached with eurowings only.

* I want to attach Buy plane function to lufthansa like this

```
lufthensa.planes = 300;
lufthensa.buyPlane = function () {
  console.log(this);
  this.planes++;
  console.log(this.planes);
};
document.querySelector(".buy").addEventListener("click", lufthensa.buyPlane);
```

* The call back function when I click on buy will call Buy plane method but in this case The This keyword will point to the selected element with a class buy not the lufthensa object and This will cause NaN when I try to plane ++;

* to over come such a problem, you should add bind Method to the call back function, so that This keyword will point to the selected obj.

```
document
  .querySelector(".buy")
  .addEventListener("click", lufthensa.buyPlane.bind(lufthensa));
```

By this small modification This will point to the obj

Note that:— It is not acceptable to use call here because at the call back function wait a function and bind returns a func so that you must use bind here.

Bind (This Points, Func Params)

another use of Bind method : →

```
const addTAX = (rate, value) => value + value
* rate;
```

if the rate of tax is Const instead of hard coded it we can use the bind method.

```
const addVAT = addTAX.bind(null, 0.23);
console.log(addVAT(100));
```

addVAT simply will be a function of addTAX but the rate is Const to 0.23 and I will pass only the value to it.

There is no this

* Closure : →

```
const secureBooking = function () {
  let passengerCount = 0;

  return function () {
    passengerCount++;
    console.log(`${passengerCount} passengers`);
  };
};

const booker = secureBooking();

booker();
booker();
booker();
```

. Can you guess the output of Booker() First, Second, third.

* at the first look I think When I Call booker will give me an error.

. That's Because secureBooking function stores the return Function in the Booker Variable and goes away Right !

. When I Call the Booker Function it simply try to update the passenger Count and log it into the Console but where is the initial value of the passenger Count, It is not there.

. The closure gets there, The Booker Function Can access every thing From the Func which Birth place it which in this Case secureBooking even after the call stack of secureBooking gone, The Booker Can access it.

. Then on every Call it Can increment the no of Passengers and log it to the Console.

another example:→

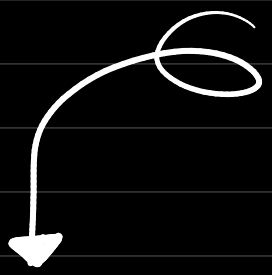
```
let f;
const g = function () {
  const a = 23;
  f = function () {
    console.log(a * 2);
  };
};

const h = function () {
  const b = 77;
  f = function () {
    console.log(b * 2);
  };
};
```

* after the callback function of g() gone away and I try to call f()

* The F function has the ability to access the a var from g Func which gone away.

* another example:→

```
const boardPassengers = function (n, wait) {
  const perGroup = n / 3;
  setTimeout(function () {
    console.log(`We are Boarding all ${n} passengers`);
    console.log(`we are boarding in ${perGroup} passenger`);
  }, wait * 1000);
  console.log(`boarding opens in ${wait} seconds`);
};
boardPassengers(180, 3);
```



When I call board Passengers → perGroup variable get initialized and SetTimeout called and wait For 3 seconds but the Program doesnt wait and Continue executing then log to the Console "Boarding opens in 3 seconds"

* The Function will be gone From Call stack But SetTimeout can access the variables of the Parent Func.