**Lec 34**

**Function Declaration and Function Expression**

Declaration = Function abc(){}

Expression = Const abc= Function(){}

Arrow function = is the simplest form of the Function ExpressionAbc=>3-3;

Arrow function does not get a this keyword so it should not use always.

**We can call the declared function before declaration but not able to call the function before expression**

**\*\*\*\*\*\*\*\*\*\*\***

Functions are just like values that’s why they have name,properties like values

**Default Parameters L128**

We can pass and receive parameters in function call and declaration

**Const abc=function(a=1,b=2,c=3)**

During call if we use null or undefined parameter func will use these default parameters

**Parameters passing by value OR refence L129**

In JS we can pass only parameters pass by value not pass by reference even it looks like pass by reference it was only done in the older languages

In obj we pass reference but it is also a reference of value or value it-self **but not pass by value**

**Function accepting call back functionL131**

Why call back function?

1:>They allow us to make more and more function means in this way we split our code into parts

2:> By this we make abstraction (means on function does’t know about the function of function that is calling inside of it

**THE CALL AND APPLY METHODS**

**Call method**

Call method :> by using we call use function of an object for the others objects that are not having such function

Obj a={

Function report{};

}

Obj b={};>>it does not have report func but we can still use

**Apply method**

It work as call but receive array rather than parameters

But in modern JS its not used

Call method also work means takes array rather than element/parameters JS upgraded call method

**BIND METHOD**

It is also works as same as call method but it return function that is bind to this keyword and by using this we can manipulate function of the object.

{ call method call the function but bind method return the method}

**IIFE { IMMEDIATELY INVOKED FUNCTION EXPRESSION}**

1. Sometimes we need to create a function which call only once is called IIFE.

2. This is used to create a new privacy or scope

**Closure**

1. Closure is not a JS feature we don’t create closure like arrays etc closure is happen only certain conditions
2. Closure remember function birth and variables and everything when it created\\\

**Function:**

**Function Declarations**

Earlier in this tutorial, you learned that functions are **declared** with the following syntax:

function *functionName*(*parameters*) {  // *code to be executed*}

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Semicolons are used to separate executable JavaScript statements.  
Since a function **declaration** is not an executable statement, it is not common to end it with a semicolon.

## Function Expressions

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

const x = function (a, b) {return a \* b};

After a function expression has been stored in a variable, the variable can be used as a function:

const x = function (a, b) {return a \* b};  
let z = x(4, 3);

The function above is actually an **anonymous function** (a function without a name).

Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

The function above ends with a semicolon because it is a part of an executable statement.

The typeof operator in JavaScript returns "function" for functions.