Default Parameters

- In JavaScript, default function parameters allow you to initialize named parameters with default values if no values or undefined are passed into the function.
- Default parameters are parameters which are given by default while declaring a function. But it's value can be changed when calling the function.
- In JavaScript, function parameters default to undefined. However, it's often useful to set a different default value. This is where default parameters can help.

Example

```
function say(message='Hi') {
    console.log(message);
}
say();
```

Example

```
function say(message='Hi') {
    console.log(message);
}
say(); // 'Hi'
say(undefined); // 'Hi'
say('Hello'); // 'Hello'
```

Example

```
let Func = (a, b = 10) => {
 return a + b;
}
Func(20); // 20 + 10 = 30
```

Example

```
let abc = function(a = "Hello World"){
    console.log(a);
}
abc(); // Hello World
```

Example

```
let abc = function(a = "Hello World",b="Hello Students"){
   console.log(a+", "+b);
}
abc(); // Hello World , Hello Students
```

Example

```
let abc = function(a = "Hello World",b="Hello Students"){
   console.log(a+", "+b);
}
abc(undefined,"Rohit"); // Hello World, Rohit
```

Example

```
function hello(a,b=a*5)
{
   console.log(a + ", " + b);
}
hello(4); // 4,20
```

Example

```
abc = () => 25;
function hello(a,b=abc()*5)
{
   console.log(a + " " + b);
}
hello(4); // 4, 125
```

Example

```
abc = () => 25;
function hello(a,b=abc()*5)
{
   console.log(a + " " + b);
}
hello(5,4); // 5,4
```

Example

```
function add(a, b = 8) {
   return a+b;
}
console.log(add(5,6));
console.log(add(7));
console.log(add(10));
```

Example

```
function add(a,b = 2,c = 1) {
  return a + b * c;
}
console.log(add(2,3,4));
console.log(add(5));
```

Arrow Functions in ES6

- ES6 arrow functions(=>) provides us an shorthand way to declare functions.
- Arrow functions also known as Fat arrow function.
- Arrow functions have "Implicit" return statement i.e., if your arrow function is only one line, you can return values without having to use the "return" keyword and the curly brackets {}.

Advantages of Arrow Functions

1) No longer to use .bind( ) method.
2) Code is much cleaner and less verbose.
3) We can skip the explicit return statement.
4) When we use the arrow function, the "this" keyword is inherited from the parent scope.

Arrow Function Vs Regular Function

- First, in the arrow function, the this, arguments, super, new.target are lexical. It means that the arrow function uses these variables (or constructs) from the enclosing lexical scope.
- Second, an arrow function cannot be used as a function constructor. If you use the new keyword to create a new object from an arrow function, you will get an error.

the lexical 'this'

- The this scope with arrow functions is inherited from the context.
- With regular functions, this always refers to the nearest function, while with arrow functions this problem is removed, and you won't need to write var that = this ever again.

When not to use 'arrow functions'

Below there are some condition when we don't want to use arrow functions :-

a) Don't use Arrow functions as 'Click Handlers'.
b) Don't use Arrow functions as 'Object Methods'.
c) Don't use Arrow functions as 'Prototype Methods'.
d) Don't use Arrow functions when you need an arguments object.

Example

```
var abc = () =>{
   console.log("Hello I am Fat Arrow function");
}
abc();
```

Example

```
var abc = () => console.log("Without Braces");
abc();
```

Example

```
let add = (x,y) => x + y;
console.log(add(10,20)); // 30;
```

Example -- With 'return' Statement

If we use the block syntax, we need to specify the 'return' keyword inside arrow functions.

```
let add = (x, y) => { return x + y; };
console.log(add(2,3)) // 5
```

Example -- Checking 'type' & 'instance'

```
let arrFunc = () => console.log("Hello World")
console.log(typeof arrFunc) // function
console.log(arrFunc instanceof Function) // true
```

Example -- With no parameter

```
let logDoc = () => console.log(window.document);
logDoc();
```

Example -- With single parameter

```
let names = ['John', 'Mac', 'Peter'];
let lengths = names.map(name => name.length);
console.log(lengths);
```

Example -- With multiple parameters

```
let numbers = [4, 2, 6];
numbers.sort((a, b) => b - a);
console.log(numbers);
```

Example -- Line break between parameter definition and arrow

```
let multiply = (x, y) =>
x * y;
console.log(multiply(3, 3))
```

Example -- Arrow function body with no braces

```
let square = x => x * x;
console.log(square(3))
```

Example -- Arrow function body with no braces

```
var abc = a => console.log(a * 5);
abc(5);
```

Example -- With Template Literals

```
let name = "Rohit"
let printName = () => {
   console.log(`Hello ${name}`)
}
printName()
```

Example -- Calculating Factorial

```
const factorial = (n) => {
   let product = 1;
   for (let i = 1; i <=n;  i++ ) {
     product *= i;
   }
   return product;
 }
  console.log(factorial(5));
```