Functions in JS

⇒ fundamental building blocks

⇒ for better code organization, modularity

⇒ for reusability

⇒ functions are executed when they are called.

JS Function Syntax

function name (p1, p2, p3)

{ }

⇒ function keyword used to define function

⇒ name follows naming rule should be camel case.

⇒ Optional paraments are listed inside paranthesses ().

Why functions

⇒ with function we can reuse the code

⇒ write code one and use multiple time with minor customization

⇒ same function used with different arguements to produce different result.

() operator is used call function.

Function Declaration

⇒ A function declaration is standalone statement that defines a named function.

```
function greet() {
    console.log("Hello");
}
```

Hoisting: Function declarations are hoisted meaning they moved to the

top of their scope during compalition. This allow yy to call the function before its actual declaration in the code.

Function Expression

A function expression defines a function as part of another expression, often assigned to a variable. The function can be named or anonymous

```
const greet = function(){
    console.log("Hello");
}

const greet = function myGreet(){
    console.log("Hello");
}
```

No Hoisting: Function expression are not hoisted same way. They are only created when javascript interpreter reaches the line where they are define.

⇒ can't call the a function exp. before its defination in the code.

Arrow Function vs Normal Function

1. Syntax

Normal Function: Defined using function keyword

```
function add (a, b) {
    return a+b;
}
```

Arrow Function

- Uses more consice structure with ⇒ operator
- They are declared in variable

```
const add = (a, b) ⇒ a+b;
```

2. this

Normal Function: Has its own this context, determined by how it's called (method call, function call, constructor call).

Arrow Function:

- Does not have its own this context
- It lexically bind this, means it inherit this from surrounding parent scope where it was defined

This make them ideal for callbacks where this context needs to be preserved.

```
const person = {
  name: "John",
  greetNormal: function() {
    Console.log("Hello, " + this.name);
  },
  greetArrow: () => {
    console.log("Hello, " + this.name);
  }
};

person.greeNormal(); // Hello, John
person.greetArrow(); // Hello, undefined
```

because arrow function always guess this of its nearby parent.

3. Arguments Objects:

Normal Function: Has access to arguments object, which is an array-like object containing all argument passed to the function.

Arrow Function:

Does not its own arguments objed. If needed, rest parameter (...args) can be used to collect arguments

4. Constructor Function

Normal Function:

can be used as constructs with new keyword to create new object instances.

Arrow Function:

Cannot be used as constructors

~~Bold~~

5. Hoisting:

Normal Functions:

Function declarations are hoisted to the top of their scope, meaning they can be called before their definition in code.

Arrow Function:

Are treated like variables and are not hoisted in the same way. They cannot be accessed before their initialization

In short, arrow function are best used when a function is

defined within other function, then in order to access parent context, child function is declared as arraw function.

Parameters:

These are named variables listed in a function's definition. They act as placeholders for values that the function is expected to recieve when it is called.

```
function greet (name, age)
{ }
```

Argument

These are actual value or

expressions passed to a function when it is called.

greet ("Shumaila", 23);

Default Parameters:

(optional arguments) provide fallback values for function parameters when an argument isn't explicity provided during a function call.

This makes functions more flexible and error-proof.

```
function greet(name="Guest")
 {
    console.log (name);
 }
  greet ("Alice");
  greet ();
```

Return Values:

The primary purpose is to allow a function to produce an output that can be used by other part of code.

It returns undefined by default.

```
function sayHi() {
    console.log("Hi");
}
let result = sayHi();
console.log(result);
```

Function Scope

The code section b/w braces {} is the scope of function.

Global Scope:

Declare outside the functions, but can be accessible from everywhere in file

Local Scop:

declared and can be accessed only inside the function

```
let globalVar;
function test(){
    let LocalVar;
    console.log(globalVar);
    console.log(LocalVar);
}  → scope of function
test();
console.log(globalVar);
console.log(localvariable); //error
```

Callback Function

A callback function is a function passed as an argument to another function.

Used when one function needs to wait until another finishes.

Common in asynch programming

```
function greet(name, callback){
    console.log("Hello " +name);
    callback();
}

    function sayBye() {
        console.log("Goodbye");
    }

    greet("Shumaila", sayBye);
```