**JavaScript Functions: A Detailed Guide**

Functions are the building blocks of structured, reusable code in JavaScript. They encapsulate a set of instructions that perform a specific task and can be invoked or called when needed. Here's a breakdown of the key concepts and examples you provided:

**1. Function Types:**

* **Normal Functions (Function Declarations):**
  + Defined using the function keyword followed by a name, parentheses for arguments, and curly braces for the function body.
  + Can be called before their declaration (hoisting).

JavaScript  
function sayHi() {  
 console.log("hi");  
}  
  
sayHi(); // Output: hi (Hoisted)

* **Function Expressions:**
  + Defined by assigning a function to a variable.
  + Cannot be called before their assignment.

JavaScript  
let sayHello = function() {  
 console.log("hello");  
};  
  
sayHello(); // Output: hello

* **Arrow Functions:**
  + Introduced in ES6 (ECMAScript 2015) for a concise way to write functions.
  + Defined using arrow (=>) syntax.
  + Implicitly return a value if the body is a single expression.

JavaScript  
let greet = () => {  
 console.log("greetings!!");  
};  
  
greet(); // Output: greetings!!  
  
let sum3 = (a, b) => a + b; // Implicit return

* **Need of function expression:**

let welcome;

function welcomeUser(age) {

if (age > 18) {

welcome = function () {

console.log("hello adult");

}

} else {

welcome = function (){

console.log("hi minor");

}

}

}

welcomeUser(12);

welcome(); // this works only with function expression

**2. Function Parameters and Arguments:**

* Functions can take zero or more parameters, which act as placeholders for values passed during the function call (arguments).

JavaScript

let sum = function(a, b) { // a and b are parameters  
 return a + b;  
};  
  
let result = sum(5, 3); // 5 and 3 are arguments passed to a and b

**3. Function Expressions as First-Class Citizens:**

* In JavaScript, functions are first-class citizens, meaning they can be:
  + Assigned to variables
  + Passed to other functions as arguments
  + Returned from other functions

**4. Function Calls and Return Values:**

* Functions are invoked or called using their name followed by parentheses.
* They can optionally return a value using the return statement.

**5. Higher-Order Functions (HOFs):**

* Functions that take other functions as arguments or return functions are called higher-order functions.

**Example: Calculator Function**

JavaScript

let add = (a, b) => a + b;  
let sub = (a, b) => a - b;  
let pro = (a, b) => a \* b;  
  
function calculator(calc, a, b) {  
 console.log(calc(a, b));  
}  
  
calculator(pro, 10, 40); // Output: 400 (10 \* 40)

**6. Function Closures:**

* When a function is defined within another function, it has access to the outer function's variables and scope, even after the outer function has returned. This is called closure.

JavaScript

// Function returning a function with closure  
function createMultiplier(n) {  
 return num => n \* num;  
}  
  
let doubler = createMultiplier(2);  
let tripler = createMultiplier(3);  
let quadrupler = createMultiplier(4);  
  
console.log(tripler(10)); // Output: 30  
console.log(doubler(10)); // Output: 20  
console.log(quadrupler(10)); // Output: 40

**7. Currying:**

* A technique of transforming a function that takes multiple arguments into a sequence of functions that take one argument at a time.

JavaScript

function addFourNumber(a) {  
 return b => c => d => a + b + c + d;  
}  
  
console.log(addFourNumber(1)(2)(3)(4)); // Output: 10