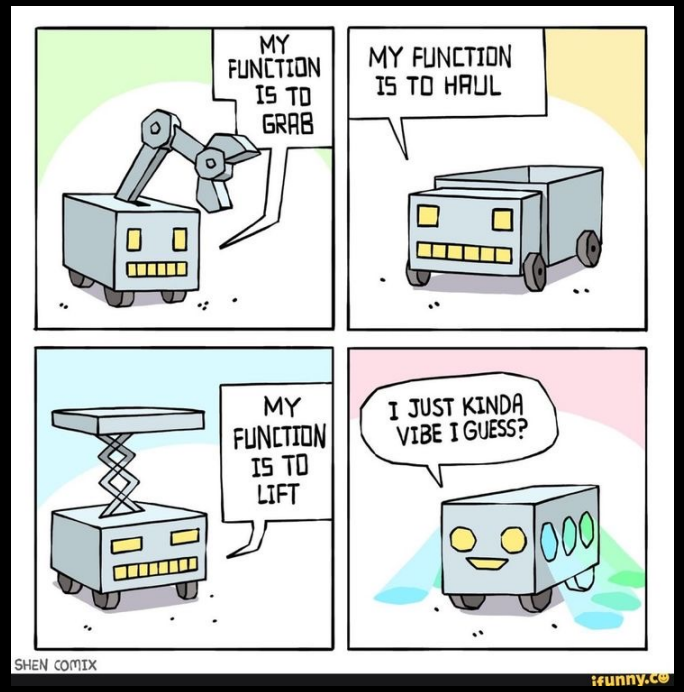
Functions

- What are Functions
- Function Syntax
- Return statement
- Function Parameters
- Call by Value
- Variable Scope

Improve 👊 🖐 ✌ Project

What are Functions?

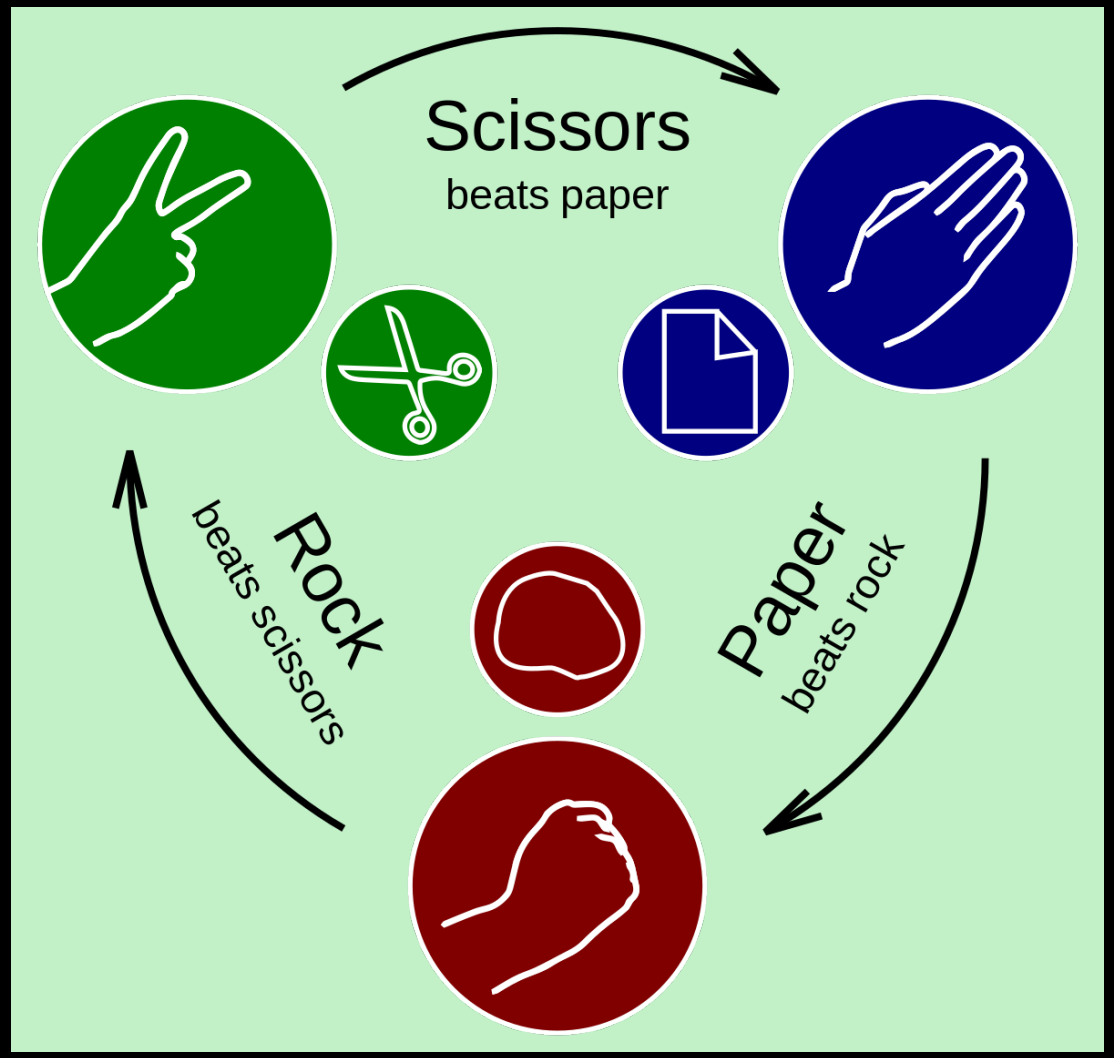



```
function greet(name) {
    // code
}

greet(name);
// code
```

function call

1. Definition: Blocks of reusable code.
2. DRY Principle: "Don't Repeat Yourself" it Encourages code reusability.
3. Usage: Organizes code and performs specific tasks.
4. Naming Rules: Same as variable names: camelCase
5. Example: ”Beta Gas band kar de"

Project Rock-Paper-Scissor Game



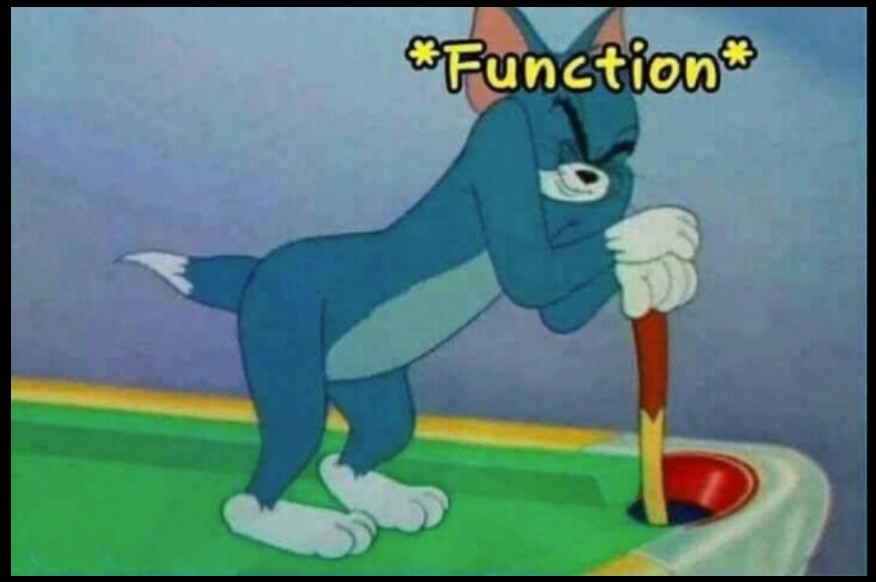
Rock Paper Scissors Game

Click on one of the following to play the game:

👊 Rock | ✋ Paper | ✌ Scissors

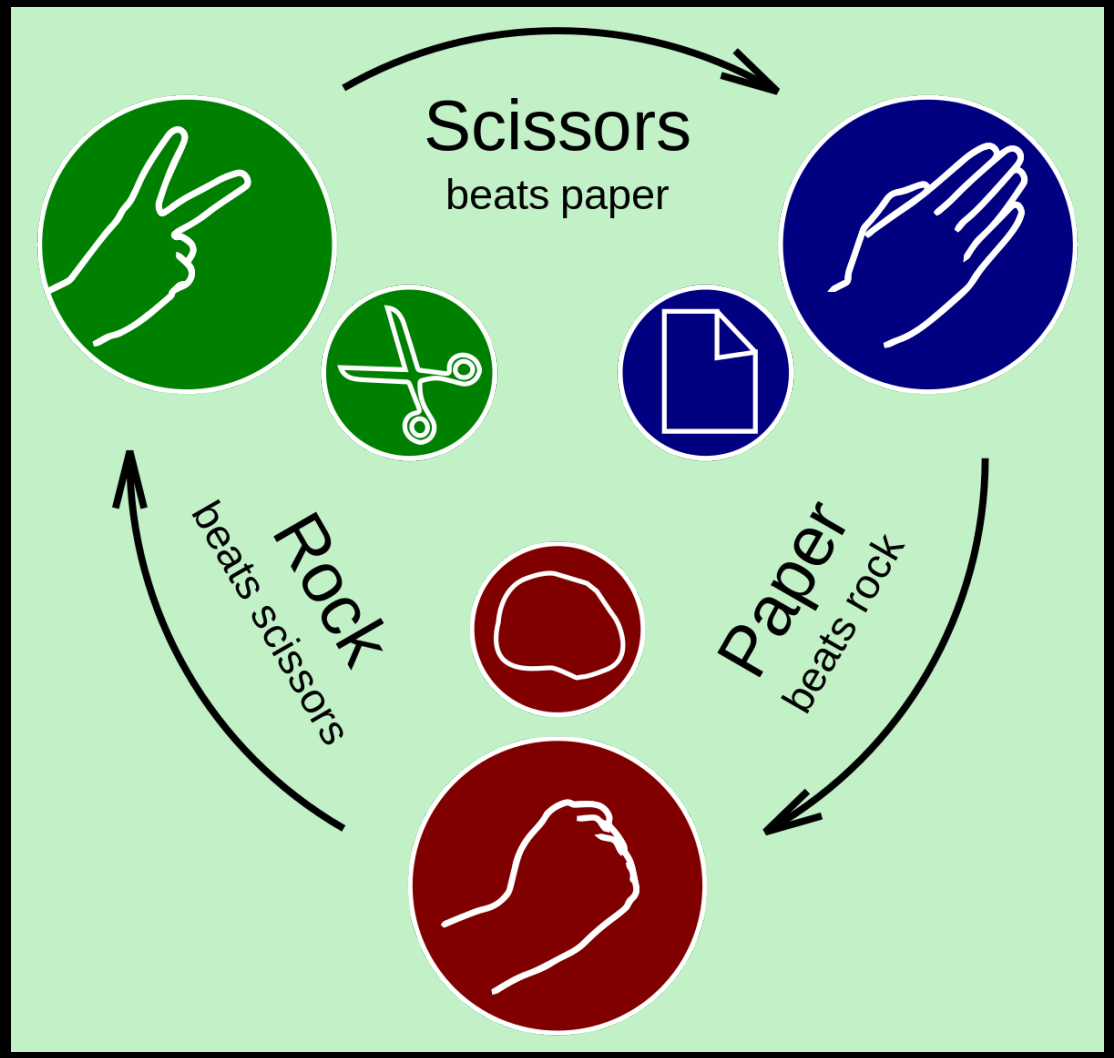
Create functions for onclick, for Random Number & Computer Choice

Return statement



1. Sends a value back from a function.
2. Example: "Ek glass paani laao"
3. What Can Be Returned: Value, variable, calculation, etc.
4. Return ends the function immediately.
5. Function calls make code jump around.
6. Prefer returning values over using global variables.

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:

👊 Rock | ✋ Paper | ✌ Scissors

User return instead of Global Variable

Parameters

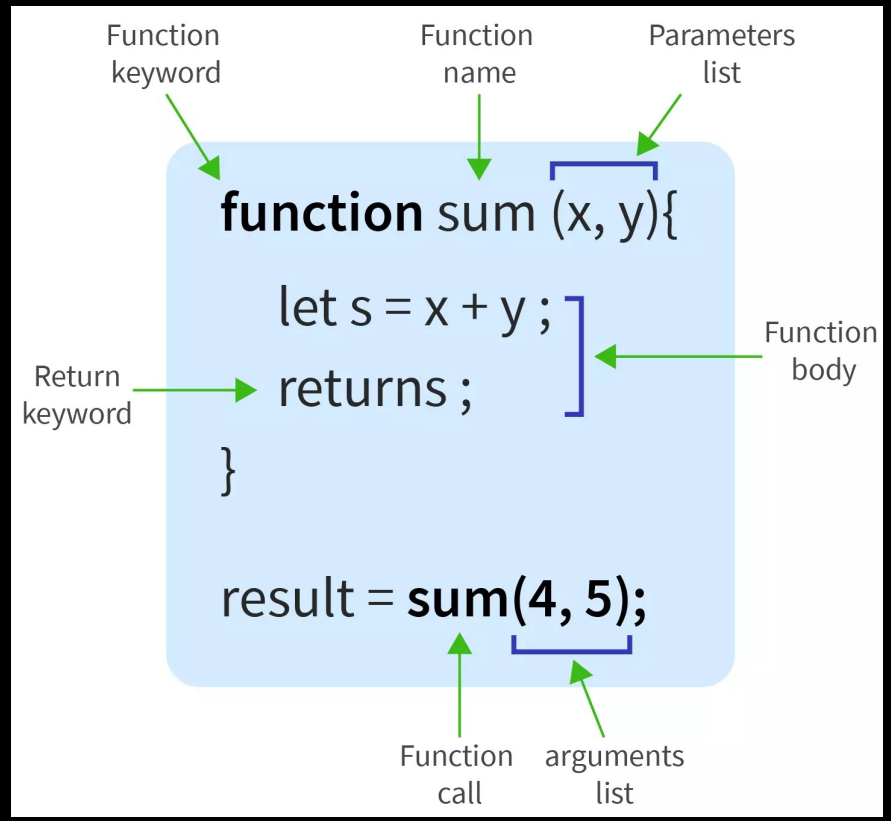
Parameter

Function

Return

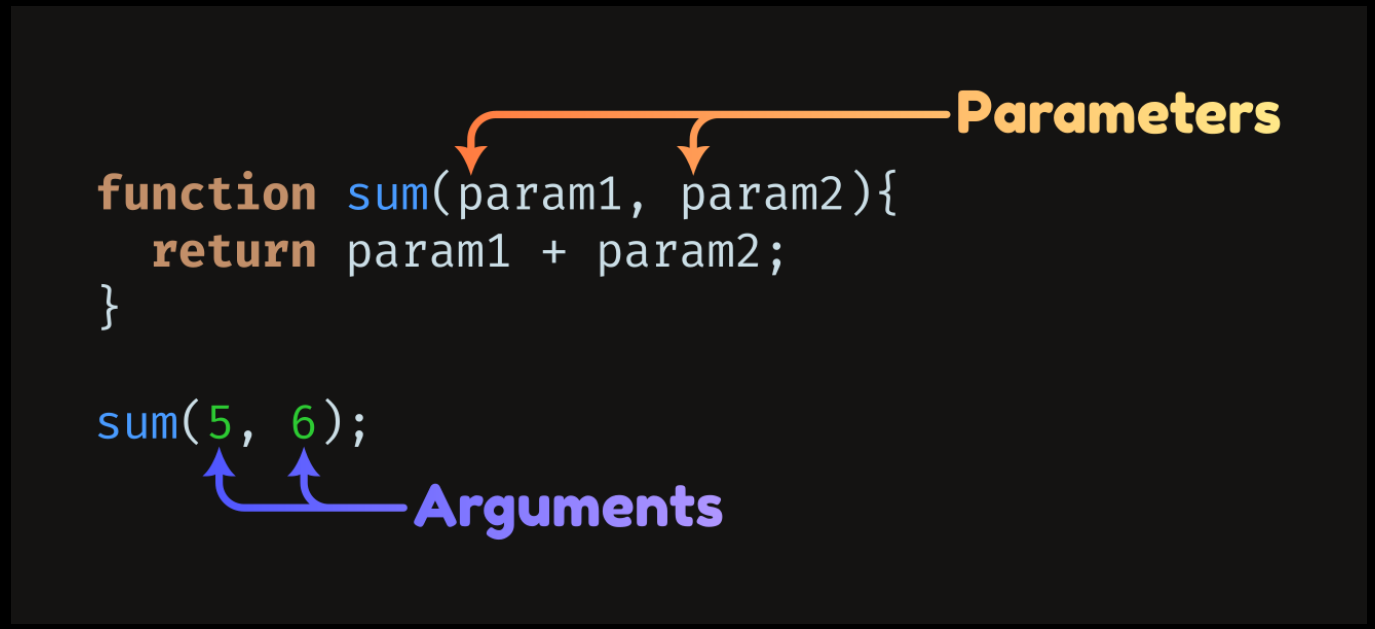
1. Input values that a function takes.
2. Parameters put value into function, while return gets value out.
3. Example: "Ek packet dahi laao”
4. Naming Convention: Same as variable names.
5. Parameter vs Argument
6. Examples: alert, Math.round, console.log are functions we have already used
7. Multiple Parameters: Functions can take more than one.
8. Default Value: Can set a default value for a parameter.

Functions Syntax



1. Use function keyword to declare.
2. Follows same rules as variable names.
3. Use () to contain parameters.
4. Invoke by using the function name followed by ().
5. Fundamental for code organization and reusability.

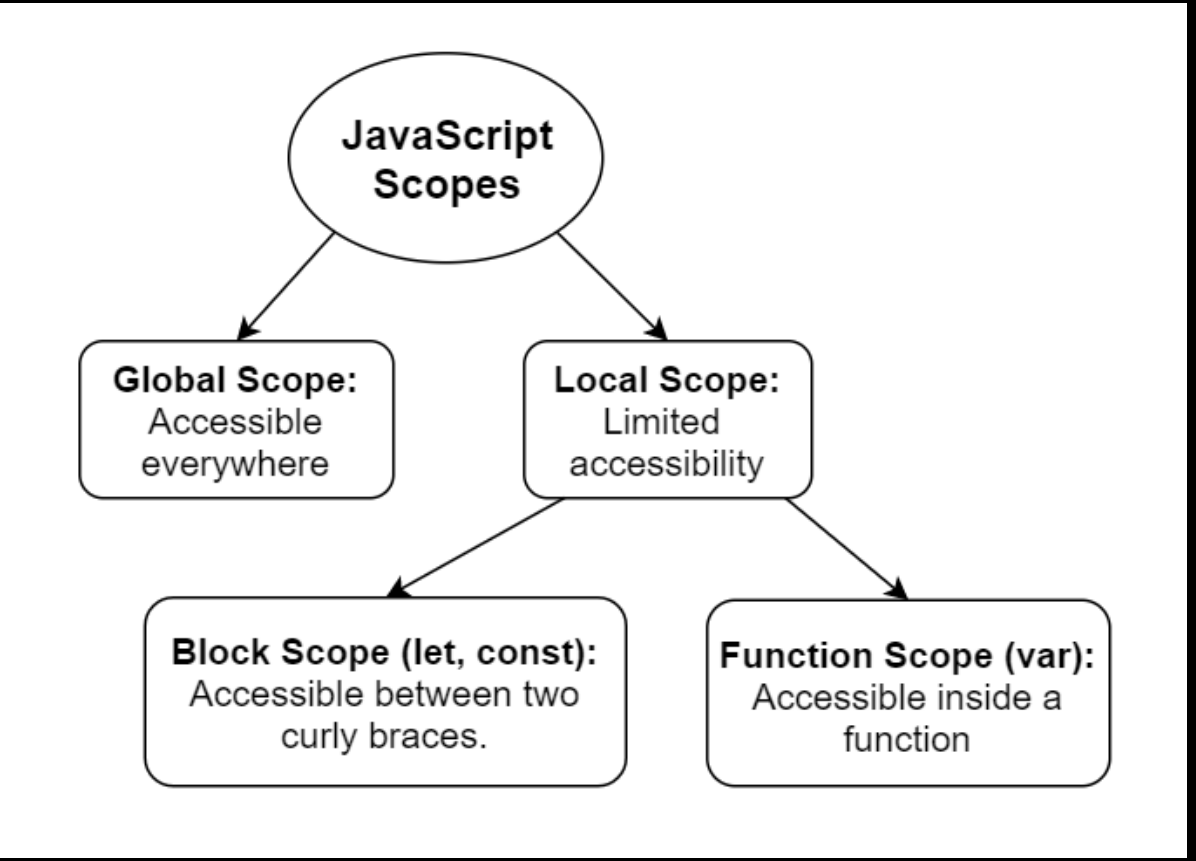
Argument vs Parameter



1. Parameters: Variables in a function definition, acting as placeholders.
2. Arguments: Actual values passed to a function at call time.

Variable Scope

1. No Keyword: Declares a variable globally, regardless of block scope, often leading to unintended consequences if not managed carefully.
2. var: Declares a variable with function scope, subject to hoisting, allowing it to be accessed (as undefined) before its actual declaration line.
3. let: Declares a block-scoped variable, accessible only after its declaration and not subject to hoisting, preventing use before declaration.
4. const: Declares a block-scoped variable that must be initialized at the time of declaration and cannot be reassigned, ensuring immutable bindings (though not immutable values).
5. Declare variables in the narrowest scope possible.



Variable Scope

Feature	No Keyword	var	let	const
Scope	Global unless in a module	Function scope	Block scope	Block scope
Hoisting	Not Hoisted	Hoisted within function	Temporal Dead Zone (TDZ)	Temporal Dead Zone (TDZ)
Re-declaration	Allowed globally	Allowed within scope	Not allowed	Not allowed
Re-assignment	Allowed	Allowed	Allowed	Not allowed
Use in Strict Mode	Error if used	Allowed	Allowed	Allowed
Initialization Requirement	Not required	Not required	Not required	Required
Typical Use Case	Avoid in modern JS	Legacy JS code	General variables	Constants

Variable Scope

```
// Example of 'no keyword' declaration and redefining
function globalTestNoKeyword() {
  // Global variable are not hoisted
  //console.log(globalVar); // ReferenceError: globalVar is not defined
  globalVar = "I am global initially"; // Declared without any keyword, defaults to global
  scope
  console.log(globalVar); // Output: I am global initially
  globalVar = "I am globally redefined"; // Redefining the variable
  console.log(globalVar); // Output: I am globally redefined
}
globalTestNoKeyword();
console.log(globalVar); // Output: I am globally redefined

// Example of 'var' declaration, hoisting, and redefining
function varTest() {
  console.log(hoistedVar); // Output: undefined due to hoisting
  var hoistedVar = "I am hoisted";
  console.log(hoistedVar); // Output: I am hoisted
  var hoistedVar = "I am redefined"; // Redefining the variable
  console.log(hoistedVar); // Output: I am redefined
}
varTest();
```

Variable Scope

```
// Example of 'let' declaration and redefining within the same scope
function letTest() {
  //console.log(blockVar); // ReferenceError: Cannot access 'blockVar' before initialization
  let blockVar = "I am block-scoped";
  console.log(blockVar); // Output: I am block-scoped
  //let blockVar = "redefined"; // SyntaxError: Identifier 'blockVar' has already been declared
  console.log(blockVar); // Output: I am redefined
}
letTest();


// Example of 'const' declaration and attempted redefining
function constTest() {
  const constantVar = "I am constant";
  console.log(constantVar); // Output: I am constant
  // constantVar = "Attempt to redefine"; // TypeError: Assignment to constant variable
}
constTest();
```

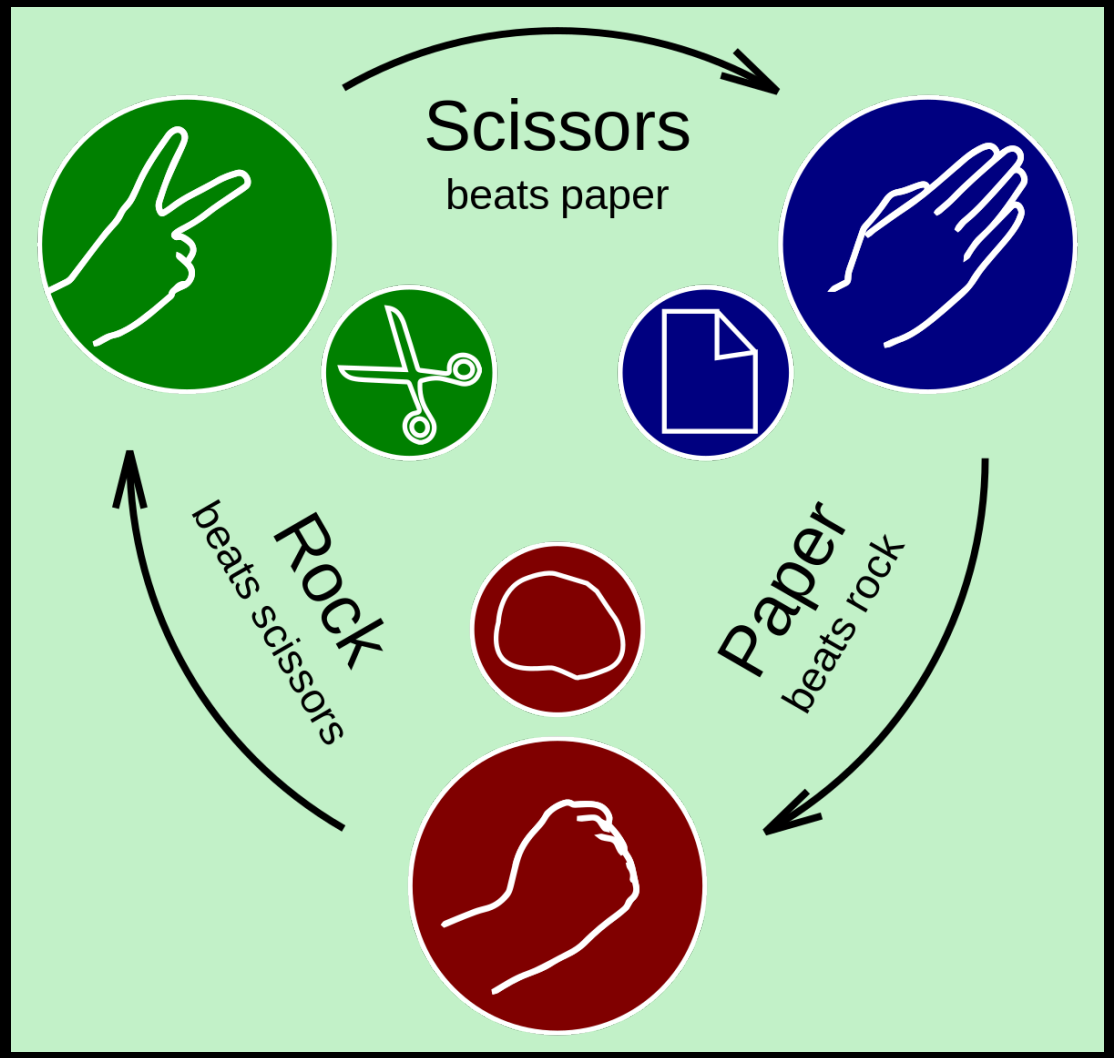
Call by Value

```javascript
// Function to try to swap two numbers
function trySwap(a, b) {
    let temp = a;
    a = b;
    b = temp;
    console.log(`Inside trySwap - a: ${a}, b: ${b}`);
}

function main() {
    let x = 10, y = 20;
    console.log(`Before trySwap - x: ${x}, y: ${y}`);
    trySwap(x, y);  // Attempt to swap x and y
    // The original values are unchanged
    console.log(`After trySwap - x: ${x}, y: ${y}`);
}

main();
```

```
Before trySwap - x: 10, y: 20
Inside trySwap - a: 20, b: 10
After trySwap - x: 10, y: 20
```

1. Value Copy: Passes argument's copy, not the original.
2. Separate Memory: Parameters use distinct memory locations.
3. Data Safety: Original data remains unchanged by the function.
4. Direct Modification: Cannot modify original arguments directly.
5. Efficiency: Good for small data types, less so for large structures.
6. Ease of Use: Straightforward and safe for functions not altering inputs.

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:

👊 Rock | ✋ Paper | ✌️ Scissors

Create function for comparing user choice & Showing Result alert

Practice Exercise

Functions

1. Create a function to check if a number is odd or even.
2. Create a function to return larger of the two numbers.
3. Create function to convert Celsius to Fahrenheit
   F = (9/5) * C + 32
4. Define a function square that takes a number and returns its square.
5. Demonstrate with a function increment that the original number passed to it does not change after incrementing it inside the function.
6. Define a function getAverage that takes five numbers and returns the average.
7. Create a function concatStrings that takes two strings and returns their concatenation.