2024-03-15

Return keyword :

we can use the variable that is inside a function , even though if it is inside a function with this keyword .
The return keyword basically returns the value that we want to return ,

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Rock Paper Scissors</title>

</head>

<body>

<!--

    The algo is as follows :

    1. Computer randomly selects a move

    2. Compare the move to get the results

    3. display the result in popup

-->



    <p>Rock Paper Scissors </p>
```

```
    <!-- ROCK -->

    <button onclick="

           pickcomputermove();



            let result ='';

            if(computermove === 'Rock'){

                result = ' there is a tie';

            }else if (computermove === 'Paper'){

                result ='you lose';

            }else if (computermove === 'Scissors'){

                result= 'you win';

            }

            alert(`You choose Rock . The computer chose ${computermove} . The
result is ${result}.`)



    ">Rock</button>



        <!-- Papers -->

    <button onclick="

   pickcomputermove();



    if (computermove ==='Paper'){
```

```
        result = 'tie';

    }else if (computermove === 'Rock'){

        result = 'you win ';

    }else if (computermove ==='Scissors'){

        result = 'you loose';

    }



    alert(`You choose Paper . The computer chose ${computermove} . The result is
${result}.`)



">Papers</button>



    <!-- Scissors -->

    <button onclick="

    /*let computermove2= '';

    let result2='';

    const randomnum3 = Math.random(); */ //generates a rnadom number between 0
and 1

    const computermove = pickcomputermove();

    /* this saves the returned value from the function and then saves it to
another variable i.e computermove . and then the processing is done on that
variable */
```

```
    if (computermove ==='Paper'){

        result = ' you win';

    }else if (computermove === 'Rock'){

        result = ' you loose ';

    }else if (computermove ==='Scissors'){

        result = 'tie';

    }



    alert(`You choose Scissors . The computer chose ${computermove} . The result
is ${result}.`)



">Scissors</button>



    <!--

        The decision of choice for the computer is as follows :

        1. If the random number is between 0 and 1/3 = rock

        2. If the random number is between  1/3 and 2/3 = paper

        3. If the random number is between 1 and 2/3 = scissor

     -->
```

```
    <script>

        function pickcomputermove() {

            let computermove ='';

            const randomnum = Math.random(); //generates a rnadom number between
0 and 1

            if (randomnum >0 && randomnum<1/3){

                computermove = 'Rock';

            } else if (randomnum >1/3 && randomnum< 2/3){

             computermove = 'Paper';

            }else if(randomnum >2/3 && randomnum< 1) {

                computermove = 'Scissors';

            }



            return computermove ; // this trick is used to access the variable
that is inside the function

        }

    </script>

</body>

</html>
```

Returning a variable like this is usually preferred over using global variables

---

- Parameter : puts a value to a function
- Return : gets the value out of a function

Parameter names follow the same rule as variable name

Objects :

They help to group multiple values together

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Objects</title>

</head>

<body>

    <script>

        const product = {  // it is an object here

            name : ' socks',

            price : 1000

        }

    </script>

</body>

</html>
```

Rules for objects :

- start and end with a curly bracket
- we can put multiple values in the object
- the left side is known as property and the right side is known as value
- there can be many property value pair in an object
- we can use the dot notation to mutate the value

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Objects</title>

</head>

<body>

    <script>

        const product = {

            name : ' socks',

            price : 1000

        };

        console.log(product);

        console.log(product.name); /* shows the name wala property ko value only
*/

        product.name = ' cotton socks '; /* updates the value of the object */

        console.log(product);
```

```
        product.newproperty = true ; /* adds a new property and assigns a value
to the property */

        console.log(product);

    </script>

</body>

</html>
```

#notes objects help us to arrange and group all the related values in a single place and also lets us to use multiple values together

---

Built in objects :

1. Math .
2. console .
3. JSON
    1. this object helps to work with JSON i.e. JavaScript Object Notation
    2. it is a similar syntax to JavaScript object
    3. has less features
    4. the properties and values , both have to use the double quotes
    5. doesn't support functions
    6. JSON syntax can be understood by almost every programming language but the JavaScript object is understood only by JavaScript
    7. it is used when :
        1. we send data between computers
        2. we store data
        3.
4. localstorage

Built in JSON object :

Helps to convert JavaScript Object into JSON

```
JSON.stringify();  // the 'stringify ' is a method of the JSON object .


// it helps to convert the js object into a json object . the function shall be omitted when we convert it . it shall be converted into a string data type
```

```
JSON.parse(); // this shall make convert the json back to object
```

Local storage :

used to store the values more permanently .
it only supports strings

```
localStorage.setItem() // used to store values in the local storage
localStorage.setItem('message' , 'hello');
```

Explanation:

It seems like you're using JavaScript to set an item in the browser's localStorage. You've set the key "message" with the value "hello". This means that the string "hello" will be stored in the browser's localStorage associated with the key "message". You can retrieve this value later using `localStorage.getItem('message')`.

Null vs undefined :

In JavaScript, `null` and `undefined` are both used to represent the absence of a value, but they are used in slightly different contexts:

1. **`null`**: It is explicitly set by developers to indicate that a variable or object intentionally does not have a value. It is a primitive value that represents the intentional absence of any object value. For example:

   javascriptCopy code

   `let myVar = null;`
2. **`undefined`**: It indicates that a variable has been declared but has not been assigned a value, or a property does not exist in an object. It is a type itself (`undefined`) and a

property of the global object. For example:

javascriptCopy code

```
let myVar; console.log(myVar); // Output: undefined let obj = {}; console.log(obj.nonExistentProperty); // Output: undefined
```

In summary, `null` is used to represent an intentional absence of value, while `undefined` represents an uninitialized or non-existent value.

Auto boxing :

```
console.log('hello'.length);
console.log('hello'.toUpperCase());
```

Automatic boxing, also known as auto-boxing, is a feature in JavaScript where primitive data types are automatically converted to their corresponding object wrapper types when a method or property that expects an object is used with a primitive value.

In JavaScript, primitive data types include numbers, strings, booleans, null, and undefined. These primitive types are not objects and do not have properties or methods. However, JavaScript provides object wrapper types (`Number`, `String`, `Boolean`, etc.) that wrap primitive values and provide access to methods and properties.

When you try to access a property or call a method on a primitive value, JavaScript automatically converts the primitive value to an object of the corresponding wrapper type, performs the operation, and then converts it back to a primitive value.

Here's an example of auto-boxing with the `String` object wrapper type:

javascriptCopy code

```
let str = 'hello'; // primitive string let strLength = str.length; // accessing property 'length' console.log(strLength); // Output: 5 // Behind the scenes, auto-boxing occurs: // The primitive string 'hello' is automatically converted to a String object, // the length property is accessed, and then the object is converted back to a primitive string.
```

Similarly, auto-boxing happens for other primitive types when they are used with methods or properties that expect objects.

Auto-boxing simplifies the code by allowing developers to work with primitive values as if they were objects without manually creating wrapper objects. However, it's essential to be

aware of the performance implications, especially in performance-critical code, as auto-boxing involves additional object creation and conversion steps.

#notes objects are just references

```
const object1 = {
name: 'david'
}

const object2 = object1 ; // this doesnt make a copy of the object , it just
makes a
```

```
const message = object.message;
const{ message } = object; // take the message from the object and then stores in
the message variable
```

both the above lines of code do the same thing

Document Object Model :

It is another build in object :

```
document.body.innerHTML = 'hello'; // replaces all the things in the body with
the 'hello' string
// innerhtml controls all the html inside the body property

document.title='good job'; // changes the title of the web page accordingly
```