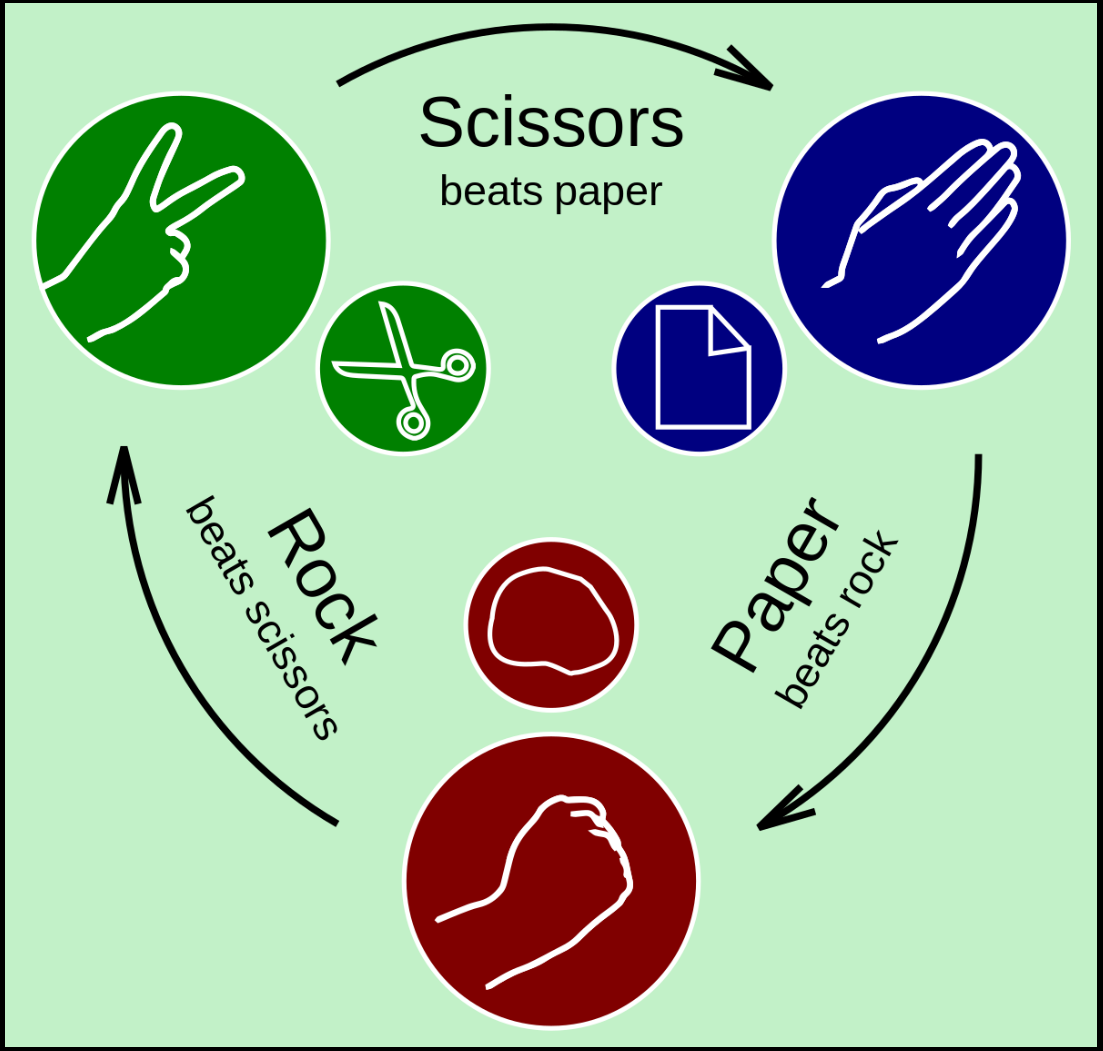
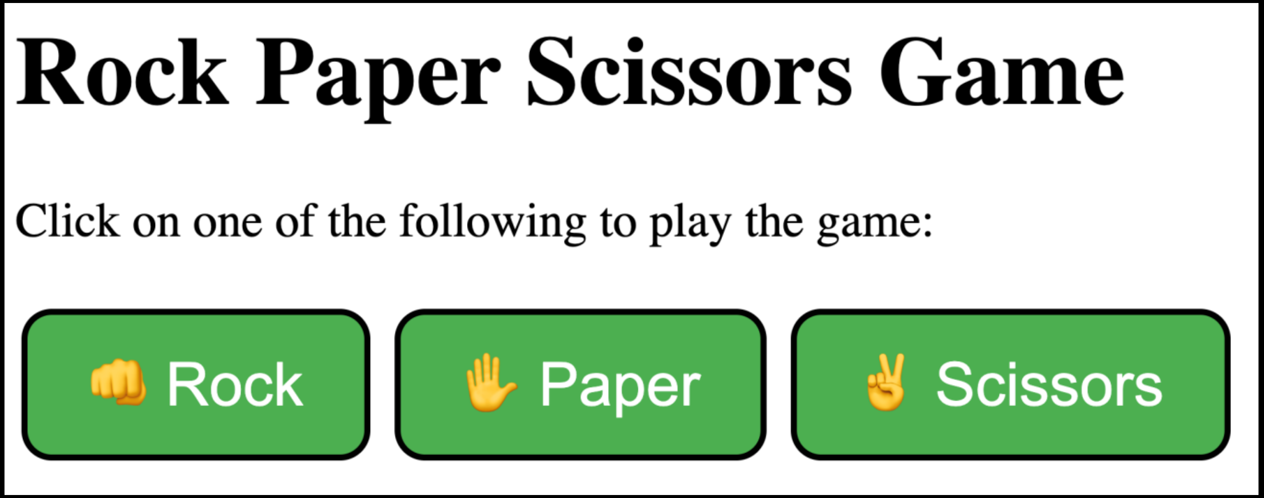
Project Rock-Paper-Scissor Game



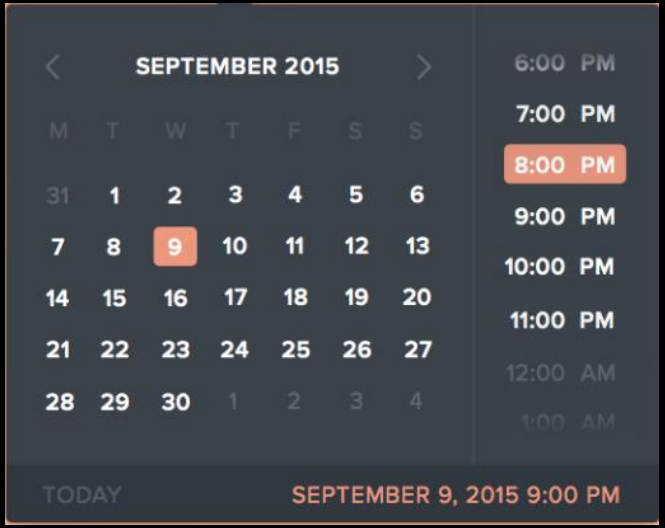


Rock Paper Scissors Game

Click on one of the following to play the game:

1. Score will survive browser refresh.
2. Add Reset Button To clear or reset stored data.

Date



1. new Date() Creates a new Date object with the current date and time.
2. Key Methods:
   - getTime(): Milliseconds since Epoch.
   - getFullYear(): 4-digit year
   - getDay(): Day of the week
   - getMinutes(): Current minute
   - getHours(): Current hour.
3. Crucial for timestamps, scheduling, etc.

Date

```javascript
// Create a new Date object for the current date and time
const currentDate = new Date();

// Get the current timestamp in milliseconds since January 1, 1970
console.log("Current Time (ms since 1970):", currentDate.getTime());

// Get the current day of the week (0 = Sunday, ..., 6 = Saturday)
console.log("Day of the Week:", currentDate.getDay());

// Get the current year
console.log("Current Year:", currentDate.getFullYear());

// Get the current month (0 = January, ..., 11 = December)
console.log("Current Month:", currentDate.getMonth());

// Get the current date of the month
console.log("Date of the Month:", currentDate.getDate());

// Create a specific date (e.g., December 25, 2024)
const specificDate = new Date("2024-12-25");

// Log the specific date
console.log("Specific Date:", specificDate.toDateString());
```

DOM Properties & Methods

DOM and Element Properties

1. location
2. title
3. href
4. domain
5. innerHTML
6. innerText
7. classList

DOM and Element Methods

1. getElementById()
2. querySelector()
3. classList: add(), remove()
4. createElement()
5. appendChild()
6. removeChild()
7. replaceChild()

Practice Exercise

JSON ,Local Storage, Date & DOM

1. Display good morning, afternoon and night based on current hour.
2. Add the name to the output too.
3. Create a Button which shows the number how many times it has been pressed.
   - Also, it has different colors for when it has been pressed odd or even times.
   - The click count should also survive browser refresh.

Objects Equality (== & ===)

```
const obj1 = { a: 1 };
const obj2 = { a: 1 };
const obj3 = obj1;

console.log(obj1 == obj2); // Output: false (different instances)
console.log(obj1 == obj3); // Output: true (same instance)

console.log(obj1 === obj2); // Output: false (different instances)
console.log(obj1 === obj3); // Output: true (same instance)
```

1. ==: When comparing objects, == checks if the two operands refer to the same object in memory. It does not compare the contents of the objects. Therefore, even if two objects have the same properties and values, they will be considered unequal unless they refer to the same instance.
2. ===: Like ==, the === operator checks if the operands refer to the same object in memory. It does not consider the object's properties or values.

Objects Equality (Using JSON)

```javascript
function jsonEqual(obj1, obj2) {
  return JSON.stringify(obj1) === JSON.stringify(obj2);
}

const obj1 = { a: 1, b: 2 };
const obj2 = { a: 1, b: 2 };

console.log(jsonEqual(obj1, obj2)); // Output: true
```

For simple objects without circular references, functions, or undefined values, you can use JSON serialization as a quick comparison

Objects Equality (Shallow Comparison)

```javascript
function shallowEqual(obj1, obj2) {
  // Check if both are objects and not null
  if (typeof obj1 !== 'object' || obj1 === null ||
    typeof obj2 !== 'object' || obj2 === null) {
    return false;
  }

  // Compare the number of properties
  const keys1 = Object.keys(obj1);
  const keys2 = Object.keys(obj2);
  if (keys1.length !== keys2.length) {
    return false;
  }

  // Compare each property value
  for (let key of keys1) {
    if (obj1[key] !== obj2[key]) {
      return false;
    }
  }

  return true;
}
```

For a shallow comparison, you can check if two objects have the same set of properties with identical values. This approach doesn't compare nested objects.

```javascript
const objA = { a: 1, b: 2 };
const objB = { a: 1, b: 2 };
const objC = { a: 1, b: 3 };

console.log(shallowEqual(objA, objB)); // Output: true
console.log(shallowEqual(objA, objC)); // Output: false
```

Objects Equality (Deep Comparison)

```javascript
function deepEqual(obj1, obj2) {
  if (obj1 === obj2) {
    return true; // Same reference or both null
  }

  if (typeof obj1 !== 'object' || obj1 === null ||
     typeof obj2 !== 'object' || obj2 === null) {
    return false;
  }

  const keys1 = Object.keys(obj1);
  const keys2 = Object.keys(obj2);

  if (keys1.length !== keys2.length) {
    return false; // Different number of properties
  }

  for (let key of keys1) {
    if (!keys2.includes(key) ||
     !deepEqual(obj1[key], obj2[key])) {
      return false;
    }
  }
  return true;
}
```

For a deep comparison, you need to recursively compare properties that might themselves be objects.

```javascript
const objD = { a: 1, b: { c: 3 } };
const objE = { a: 1, b: { c: 3 } };
const objF = { a: 1, b: { c: 4 } };

console.log(deepEqual(objD, objE)); // Output: true
console.log(deepEqual(objD, objF)); // Output: false
```