EDUFY

Random numbers in JavaScript: dice, lists and fair shuffles

Maximilian Bahrami · 10 November 2026

Make a random whole number in any range with Math.random() and Math.floor(), pick a random item, shuffle a list fairly, avoid the Math.round trap, and know when to use crypto instead.

Dice, shuffled cards, a random tip of the day, a colour that changes on every click: they all start with one function, Math.random(). By the end of this lesson you'll be able to make a random whole number in any range, pick a random item from a list, shuffle a list fairly, and know when Math.random() is the wrong tool.

Math.random()

Math.random() gives you a decimal number that is **0 or more, and always less than 1**. It never gives exactly 1. Press **Run** a few times: you get new numbers every time.

```
console.log(Math.random());
console.log(Math.random());
console.log(Math.random());
```

On its own, a number like 0.7351... isn't much use. The trick is to **stretch** it and then **cut off** the decimals.

Stretch it: multiply

Multiply by 10 and the range becomes 0 up to (but not including) 10:

```
console.log(Math.random() * 10); // 0 to 9.999...
console.log(Math.random() * 100); // 0 to 99.999...
```

Cut it: Math.floor()

Math.floor() rounds a number **down** to the whole number below it. It never rounds up:

```
console.log(Math.floor(4.2)); // 4
console.log(Math.floor(4.99)); // 4
console.log(Math.floor(0.5)); // 0
```

Put the two together and you get a random **whole** number from 0 to 9. Ten possible numbers, each as likely as the others:

```
const n = Math.floor(Math.random() * 10);
console.log(n); // 0, 1, 2 ... 9
```

Good to know

> Math.random() * 10 can be 9.999..., but never 10. So Math.floor turns it into **9 at most**. Multiplying by 10 gives you the 10 numbers 0 to 9.

Any range: min to max

Real code rarely wants "0 to 9". It wants "1 to 6" for a dice, or "18 to 65" for an age. This is the formula developers use, wrapped in a function so you only write it once:

```
// A random whole number from min to max, both included.
function randomInt(min, max) {
return Math.floor(Math.random() * (max - min + 1)) + min;
}

console.log(randomInt(1, 6)); // a dice: 1 to 6
console.log(randomInt(18, 65)); // an age
console.log(randomInt(-5, 5)); // works below 0 too
```

How it works for a dice, randomInt(1, 6):

1. max - min + 1 is 6 - 1 + 1 = **6**: how many numbers there are.
2. Math.floor(Math.random() * 6) gives **0 to 5**.
3. + min moves it up by 1, so you get **1 to 6**.

What can Math.floor(Math.random() * 6) + 1 give?

> • 0 to 6
>
> • 1 to 7
>
> • ✓ 1 to 6
>
> • 0 to 5

> Math.floor(Math.random() * 6) gives 0 to 5. Adding 1 moves every result up by one, to **1 to 6**: exactly a dice.

A dice you can click

Here is the same function in a page. Press **Run**, then click the button in the result.

```
<!DOCTYPE html>
<html lang="en">
<body>
<button id="roll">Roll the dice</button>
<p id="result" style="font-size: 48px; margin: 12px 0">?</p>
<script>
```

```
function randomInt(min, max) {
return Math.floor(Math.random() * (max - min + 1)) + min;
}

const button = document.getElementById("roll");
const result = document.getElementById("result");

button.addEventListener("click", () => {
const roll = randomInt(1, 6);
result.textContent = roll;
console.log("You rolled", roll);
});
</script>
</body>
</html>
```

A random item from a list

Lists (arrays) start counting at 0, and list.length is how many items there are. So a random position is exactly "0 up to length, not included". That's the first formula, with length in place of 10:

```
const tips = ["Drink water", "Take a walk", "Stretch", "Call a friend"];

const tip = tips[Math.floor(Math.random() * tips.length)];
console.log(tip);
```

It works for a list of any size, and you never go past the last item.

Shuffle a list fairly

To shuffle (quiz questions, a deck of cards), developers use the **Fisher–Yates shuffle**: go through the list from the end and swap each item with a random one at or before it.

```
function shuffle(list) {
const copy = [...list];
for (let i = copy.length - 1; i > 0; i--) {
const j = Math.floor(Math.random() * (i + 1));
[copy[i], copy[j]] = [copy[j], copy[i]];
}
return copy;
}

console.log(shuffle([1, 2, 3, 4, 5]));
console.log(shuffle(["A", "B", "C", "D"]));
```

[...list] makes a copy first, so the original list stays as it was.

Common mistakes

Using Math.round instead of Math.floor

Wrong

```
const n = Math.round(Math.random() * 10);
```

Right

```
const n = Math.floor(Math.random() * 11);
```

Math.round gives 0 to 10, but **not evenly**: only numbers from 0 to 0.5 become 0, and only 9.5 to 10 become 10. So 0 and 10 come up half as often as the rest. Run this to see it with 110,000 tries:

```
const counts = { round: {}, floor: {} };
for (let i = 0; i < 110000; i++) {
const r = Math.round(Math.random() * 10);
const f = Math.floor(Math.random() * 11);
counts.round[r] = (counts.round[r] || 0) + 1;
counts.floor[f] = (counts.floor[f] || 0) + 1;
}
console.log("round, 0:", counts.round[0], " 5:", counts.round[5], " 10:", counts.round[10]);
console.log("floor, 0:", counts.floor[0], " 5:", counts.floor[5], " 10:", counts.floor[10]);
```

Forgetting the + 1

Wrong

```
Math.floor(Math.random() * (max - min)) + min;
```

Right

```
Math.floor(Math.random() * (max - min + 1)) + min;
```

Without + 1, max can never come up: a "dice" from 1 to 6 would only ever give 1 to 5.

Shuffling with sort

Wrong

```
list.sort(() => Math.random() - 0.5);
```

Right

```
const mixed = shuffle(list); // Fisher–Yates, above
```

It looks clever and you'll see it online, but sort was not made for random answers. Some orders come up far more often than others. Use the shuffle above.

Not for passwords, codes or security

> Math.random() is fast but **predictable** if someone tries hard enough. Never use it for passwords, sign-in codes, tokens or anything secret. The browser has a secure version for that:

```
console.log(crypto.randomUUID());

const bytes = new Uint32Array(1);
crypto.getRandomValues(bytes);
console.log(bytes[0]);
```

Exercise

A random background colour

> Make a page with a button. Each click gives the page a random background colour and prints the colour in the console.

> 1. Write a randomInt(min, max) function.
> 2. On a click, make three random numbers from 0 to 255: red, green and blue.
> 3. Set document.body.style.background to "rgb(" + red + ", " + green + ", " + blue + ")".

> Extra: show the colour's name on the page too, and make the text readable on dark colours.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<button id="change">New colour</button>
<p id="name">Click the button</p>
<script>
function randomInt(min, max) {
return Math.floor(Math.random() * (max - min + 1)) + min;
}

document.getElementById("change").addEventListener("click", () => {
const red = randomInt(0, 255);
const green = randomInt(0, 255);
const blue = randomInt(0, 255);
const colour = "rgb(" + red + ", " + green + ", " + blue + ")";

document.body.style.background = colour;
// Light text on dark colours, dark text on light ones.
const light = red * 0.3 + green * 0.6 + blue * 0.1 > 140;
document.body.style.color = light ? "black" : "white";
document.getElementById("name").textContent = colour;
```

```
console.log(colour);
});
</script>
</body>
</html>
```

> Each colour part is a whole number from 0 to 255, so randomInt(0, 255) is exactly right: 256 possible values, each as likely.

Key points

- Math.random() gives a decimal from 0 up to, but never, 1.
- Math.floor(Math.random() * n) gives a whole number from 0 to n − 1.
- From min to max, both included: Math.floor(Math.random() * (max - min + 1)) + min.
- A random item: list[Math.floor(Math.random() * list.length)]. To shuffle, use Fisher–Yates, not sort.
- Use Math.floor, not Math.round, and never Math.random() for anything secret: use crypto.