EDUFY

Project: build a Hangman game

Maximilian Bahrami · 11 November 2026

Build the word-guessing game step by step: a random word, blanks, letter guesses, attempts, winning and losing, and a version you can play on a web page.

Time for a real project. In this lesson you'll build **Hangman**, the word-guessing game, one small step at a time. Every step uses something you've already learned (random numbers, booleans, if, functions, loops, arrays) and adds one new idea. At the end you'll have a game you can play on a web page.

The rules: the computer picks a secret word. The player sees one blank per letter and guesses letters. A right letter shows up in every place it appears; a wrong letter costs one of **5 attempts**. The player can also guess the whole word at once. Find the word before the attempts run out and you win; otherwise you lose.

Each step has a short lesson, an exercise and an answer you can open. Try the exercise yourself first.

Step 1: pick a secret word

Template strings

So far you've joined text with +. A **template string** is often easier: write it between backticks ` ` and put any value inside ${ }`:

```
const name = "Sara";
const age = 16;

console.log("Hi " + name + ", you are " + age); // the old way
console.log(`Hi ${name}, you are ${age}`); // a template string
```

A random item from a list

You met this in the random numbers lesson: Math.floor(Math.random() * list.length) is a random position from 0 to the last one, so list[position] is a random item.

```
const colours = ["red", "green", "blue"];
const index = Math.floor(Math.random() * colours.length);

console.log(`Position ${index}: ${colours[index]}`);
```

Exercise 1: a function that picks the word

> Write a function pickWord(words) that **returns** a random word from the array it's given, in small letters. Use this list:

> ["Camel", "Candy", "Colombia", "Andalusia", "Naruto"]

Answer

```
const possibleWords = ["Camel", "Candy", "Colombia", "Andalusia", "Naruto"];

function pickWord(words) {
const index = Math.floor(Math.random() * words.length);
return words[index].toLowerCase();
}

console.log(pickWord(possibleWords));
console.log(pickWord(possibleWords));
```

> Small letters from the start mean you never have to worry about "C" and "c" being different later.

Step 2: from a word to letters

split

A string can be turned into an **array of its letters** with split(""). Each letter gets its own position, starting from 0, so you can reach any letter by its number:

```
const word = "camel";
const letters = word.split("");

console.log(letters); // ["c", "a", "m", "e", "l"]
console.log(letters.length); // 5
console.log(letters[2]); // "m"
```

join

join does the opposite: it glues an array back into one string, with whatever you put between the brackets in the gaps:

```
console.log(["c", "a", "t"].join("")); // "cat"
console.log(["c", "a", "t"].join(" ")); // "c a t"
```

Filling an array

Array(5).fill("") makes an array of 5 items, all "". It's how you get one blank per letter:

```
const blanks = Array(5).fill("_");
console.log(blanks.join(" ")); // _ _ _ _ _
```

Exercise 2: the blanks

> Write a function makeBlanks(word) that returns an array with one "_" for each letter of the word. Print it with spaces between the blanks.

Answer

```
function makeBlanks(word) {
return Array(word.length).fill("_");
}

const shown = makeBlanks("colombia");
console.log(shown.join(" ")); // _ _ _ _ _ _ _ _
```

> shown is what the player sees. As they guess right letters, you'll swap blanks for letters.

Step 3: guess a letter

The classic for loop

for...of gives you each item, but here you also need its **position**, so you know which blank to fill. The classic for loop counts the positions for you:

```
const letters = ["c", "o", "l", "o", "m", "b", "i", "a"];

for (let i = 0; i < letters.length; i++) {
console.log(`position ${i}: ${letters[i]}`);
}
```

Read for (let i = 0; i < letters.length; i++) as: start i at 0; keep going while i is less than the length; add 1 to i after each round.

Same position, not push

To reveal a letter, put it in the **same position** in shown as in the word. push can't do that: it always adds to the **end** of an array. Writing shown[i] = letter replaces exactly the blank at position i:

```
const shown = ["_", "_", "_", "_"];
shown[1] = "a";
console.log(shown.join(" ")); // _ a _ _

shown.push("x");
console.log(shown.join(" ")); // _ a _ _ x (added at the end!)
```

The word is "colombia" and the player guesses "o". Which positions should be filled?

> • Only position 1
>
> • ✓ Positions 1 and 3
>
> • Positions 2 and 4

c=0, o=1, l=2, o=3. A letter can appear more than once, so the loop must check **every** position and fill each one that matches, not stop at the first.

Exercise 3: reveal a letter

Write a function revealLetter(word, shown, letter) that puts letter into shown at every position where the word has it. It should **return** true if the letter was in the word at least once, and false if not.

Answer

```
function revealLetter(word, shown, letter) {
let found = false;
for (let i = 0; i < word.length; i++) {
if (word[i] === letter) {
shown[i] = letter;
found = true;
}
}
return found;
}

const word = "colombia";
const shown = Array(word.length).fill("_");

console.log(revealLetter(word, shown, "o")); // true
console.log(shown.join(" ")); // _ o _ o _ _ _ _
console.log(revealLetter(word, shown, "z")); // false
```

A string can be read by position too: word[i] is the letter at i, without split. found starts as false and turns true the moment one letter matches.

Step 4: attempts, winning and losing

A counter that goes down

The number of attempts left is a let, because it changes: each wrong letter takes one away with attempts = attempts - 1 (or the short form attempts--):

```
let attempts = 5;
attempts--;
```

```
attempts--;
console.log(`${attempts} attempts left`); // 3 attempts left
```

Functions that answer yes or no

A function that returns true or false reads like a question in an if. Name it like a question, too:

```
const isEven = (n) => n % 2 === 0;

if (isEven(4)) {
console.log("4 is even");
}
```

Exercise 4: won? lost?

> Write two short functions:
>
> 1. isWon(shown): true when there are no blanks left.
> 2. isLost(attempts): true when there are no attempts left.
>
> Tip: array.includes("_") tells you whether an array still has a blank.

Answer

```
const isWon = (shown) => !shown.includes("_");
const isLost = (attempts) => attempts <= 0;

console.log(isWon(["c", "a", "t"])); // true
console.log(isWon(["c", "_", "t"])); // false
console.log(isLost(0)); // true
console.log(isLost(3)); // false
```

> ! flips the answer: no blank left means won. attempts <= 0 is safer than === 0: if a bug ever made it go below zero, the game would still end.

Step 5: guess the whole word

Comparing two whole strings is one ===. Compare them in small letters, and trim spaces the player typed by accident with trim():

```
const secret = "naruto";
const guess = " Naruto ";

console.log(guess === secret); // false
console.log(guess.trim().toLowerCase() === secret); // true
```

Comparing letter by letter

> **Wrong**

```
for (let i = 0; i < chosen.length; i++) {
if (chosen[i].toLowerCase() === guess[i].toLowerCase()) {
matched[i] = guess[i];
}
}
```

> **Right**

```
const isRight = guess.trim().toLowerCase() === secret;
```

> If the guess is **shorter** than the word, guess[i] runs out and becomes undefined, and undefined.toLowerCase() crashes the game with a TypeError. For "is the whole word right?", compare the whole strings.

Exercise 5: check a word guess

> Write a function guessWord(secret, guess) that returns true when the guess is the secret word. Capitals and spaces around the guess shouldn't matter, and it must not crash on a short guess.

Answer

```
function guessWord(secret, guess) {
return guess.trim().toLowerCase() === secret;
}

console.log(guessWord("naruto", "Naruto")); // true
console.log(guessWord("naruto", "nar")); // false, no crash
console.log(guessWord("naruto", " NARUTO ")); // true
```

Step 6: the game loop

while

A while loop repeats its code **as long as** its condition is true. Something inside the loop must eventually make the condition false, or it never stops:

```
let attempts = 3;

while (attempts > 0) {
console.log(`Attempts left: ${attempts}`);
attempts--;
}
console.log("Out of attempts");
```

In your first version of Hangman, a while (gameIsRunning) loop asks with prompt again and again. This version runs in VS Code with a browser (prompt opens a little box that asks the player to type):

```
const words = ["camel", "candy", "colombia", "andalusia", "naruto"];
const secret = words[Math.floor(Math.random() * words.length)];
const shown = Array(secret.length).fill("_");
let attempts = 5;
let gameIsRunning = true;

while (gameIsRunning) {
const input = prompt(`${shown.join(" ")}\nA letter or the word:`);
if (input === null) break; // the player pressed Cancel

const guess = input.trim().toLowerCase();
if (guess === secret) {
alert("You win!");
gameIsRunning = false;
} else if (guess.length === 1 && secret.includes(guess)) {
for (let i = 0; i < secret.length; i++) {
if (secret[i] === guess) shown[i] = guess;
}
if (!shown.includes("_")) {
alert(`You win! The word was ${secret}`);
gameIsRunning = false;
}
} else {
attempts--;
if (attempts === 0) {
alert(`You lose. The word was ${secret}`);
gameIsRunning = false;
}
}
}
```

It works, but prompt **freezes the whole page** while it waits, and nothing else on the page can happen. Real web games don't loop: they wait for the player to click, run one round, and update the page. That's the version you'll build next.

Changing the wrong variable

Wrong

```
let gameIsRunning = false;
let startingGame = prompt("Do you want to start?");
if (startingGame.toLowerCase() === "yes") {
startingGame = true; // the loop checks gameIsRunning!
}
while (gameIsRunning) { ... } // never starts
```

Right

```
let gameIsRunning = false;
const answer = prompt("Do you want to start?");
if (answer !== null && answer.toLowerCase() === "yes") {
gameIsRunning = true;
}
```

The loop checks gameIsRunning, so that's the one to set. Keep one variable for one job: answer holds what was typed, gameIsRunning says whether the game runs. And prompt gives null when the player presses Cancel, so check that before calling toLowerCase().

Step 7: the game on a page

The page version uses everything above. One function, play(guess), runs **one round**: it checks the guess, updates shown or attempts, then calls draw() to show the new state. The button calls play each time it's clicked, so there's no loop and the page never freezes. Press **Run** and play:

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<p id="word" style="font: 700 32px monospace; letter-spacing: 6px"></p>
<p id="info"></p>
<input id="guess" maxlength="12" placeholder="A letter or the word" />
<button id="go">Guess</button>
<button id="again">New game</button>
<p id="message" style="font-weight: bold"></p>
<script>
const words = ["camel", "candy", "colombia", "andalusia", "naruto"];
let secret, shown, attempts, used, gameIsRunning;

function newGame() {
secret = words[Math.floor(Math.random() * words.length)];
shown = Array(secret.length).fill("_");
attempts = 5;
used = [];
gameIsRunning = true;
draw("Guess a letter!");
}

function draw(message) {
document.getElementById("word").textContent = shown.join(" ");
document.getElementById("info").textContent =
`Attempts left: ${attempts} · Tried: ${used.join(", ")}`;
document.getElementById("message").textContent = message;
}
```

```
function play(input) {
const guess = input.trim().toLowerCase();
if (!gameIsRunning || guess === "") return;

if (guess.length > 1) {
if (guess === secret) {
shown = secret.split("");
gameIsRunning = false;
return draw("You win!");
}
attempts--;
} else if (used.includes(guess)) {
return draw(`You already tried ${guess}.`);
} else {
used.push(guess);
let found = false;
for (let i = 0; i < secret.length; i++) {
if (secret[i] === guess) {
shown[i] = guess;
found = true;
}
}
if (!found) attempts--;
}

if (!shown.includes("_")) {
gameIsRunning = false;
draw("You win!");
} else if (attempts <= 0) {
gameIsRunning = false;
shown = secret.split("");
draw("You lose. This was the word.");
} else {
draw(guess.length > 1 ? "Not that word." : "Next guess!");
}
}

const box = document.getElementById("guess");
document.getElementById("go").addEventListener("click", () => {
play(box.value);
box.value = "";
box.focus();
});
document.getElementById("again").addEventListener("click", newGame);
newGame();
</script>
</body>
</html>
```

A few things to notice:

- **newGame()** sets everything back to the start, so "New game" is one function call.
- **used** remembers tried letters, so the same wrong letter isn't counted twice. Here push is right: the order of tries doesn't matter.
- **return draw(...)** shows a message and stops the round in one line.

Exercise 6 (extra): a hint button

> Add a **Hint** button to the game. Clicking it reveals one letter that's still hidden, but costs one attempt. It should do nothing when the game is over or only one attempt is left.

> Tip: loop through secret and stop at the first i where shown[i] === "_". Then play(secret[i]) reveals that letter everywhere it appears.

Answer

> Add a button next to the others:

```
<button id="hint">Hint</button>
```

> and this before newGame(); at the end of the script:

```
document.getElementById("hint").addEventListener("click", () => {
if (!gameIsRunning || attempts <= 1) return;
for (let i = 0; i < secret.length; i++) {
if (shown[i] === "_") {
attempts--;
play(secret[i]);
return;
}
}
});
```

> return inside the loop stops it after the first hidden letter. Since that letter is in the word, play reveals it without taking another attempt.

Key points

> - Template strings: ` Hi ${name} put values into text without +`.
> - split("") turns a word into letters, join(" ") turns them back into text, Array(n).fill("_") makes the blanks.
> - The classic for (let i = 0; i < list.length; i++) gives you positions; shown[i] = letter fills the right blank (push only adds at the end).
> - while repeats while something is true, but on a web page, let clicks drive the game: one function per round, then redraw.
> - Small functions with one job (pickWord, revealLetter, isWon, guessWord) make a big program easy to build and to fix.