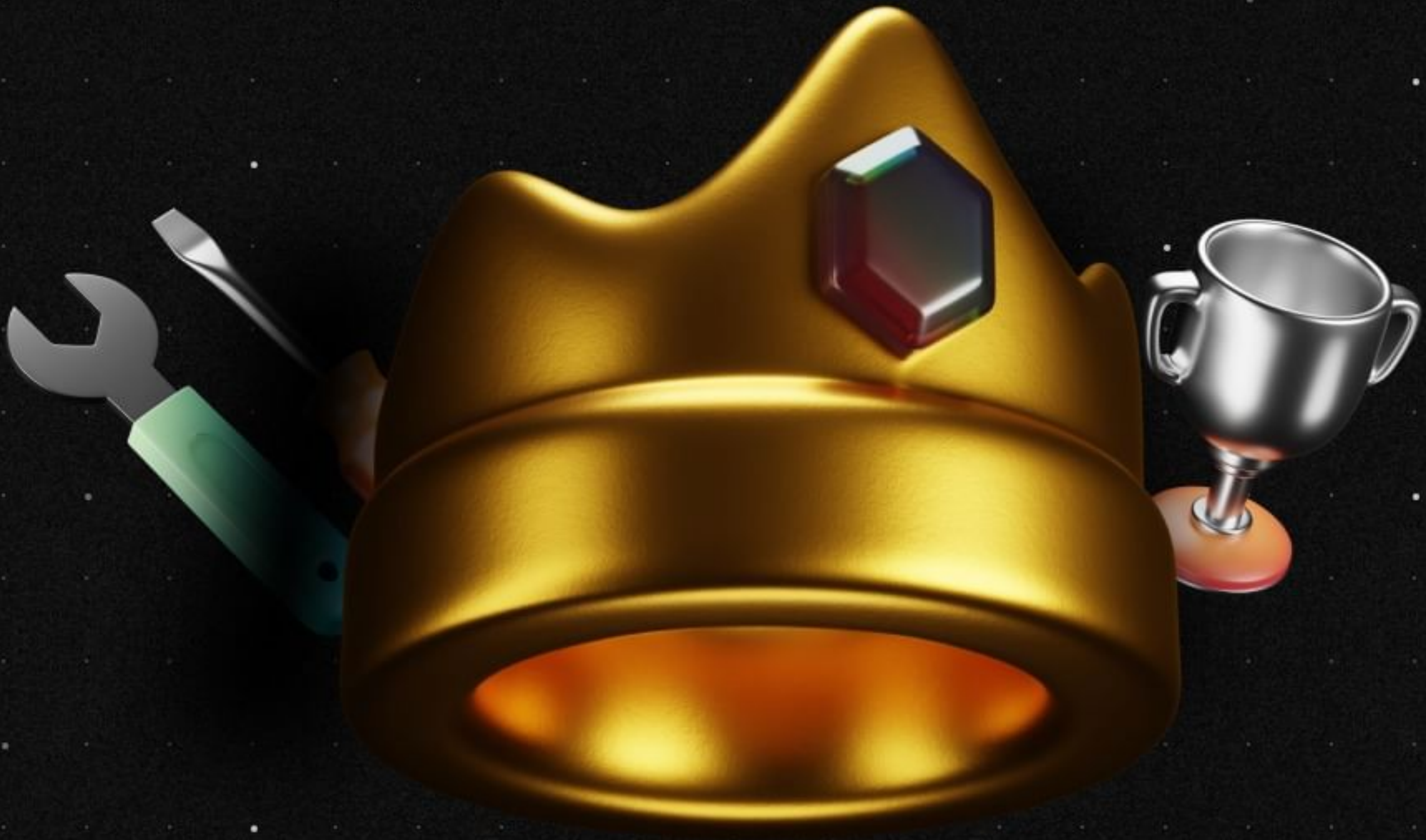
JS

5 JavaScript concepts every developer should know



codewithsloba.com

Swipe ⟶

Asynchronous Programming with Promises

Asynchronous operations in JavaScript allow tasks to be **executed independently** of the main program flow, typically used for tasks like fetching data from a server.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error
fetching data:', error));
```

Closures

Closures allow functions to **retain access to variables** from their parent scope even after the parent function has finished executing.

```
function outerFunction() {
  let outerVar = 'I am from outer';
  function innerFunction() {
    console.log(outerVar);
  }
  return innerFunction;
}
const inner = outerFunction();
inner(); // Output: I am from outer
```

Swipe ⟶

JS

Prototype-based Inheritance

JavaScript objects can **inherit** properties and methods **from other objects** through a prototype chain, enabling hierarchical relationships between objects.

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.speak = function() {
  console.log(this.name + ' makes a noise.');
};
function Dog(name) {
  Animal.call(this, name);
}
Dog.prototype = Object.create(Animal.prototype);
const dog = new Dog('Rover');
dog.speak();
```

Swipe ⟶

JS

Event Loop

A core concept in JavaScript for **managing asynchronous operations**.

It ensures that tasks like network requests and timers can run without blocking the main execution thread.

Asynchronous tasks' callback functions are placed in a queue, waiting to be executed.

The Event Loop continuously checks if the execution stack is empty, dequeuing and executing callbacks when it is.

This mechanism allows JavaScript to **handle multiple tasks concurrently**, ensuring a responsive user experience.

Swipe ⟶

JS

Higher Order Functions

Functions that can **take other functions as arguments** or return functions, allowing for functional programming paradigms such as map, filter, and reduce.

```javascript
function multiplier(factor) {
  return function(x) {
    return x * factor;
  };
}
const double = multiplier(2);
console.log(double(5)); // Output: 10
```