Thanks for downloading it. I randomly chose 6 topics for this demo, from the latest draft. The final version may vary in format and content.

What are closures in JavaScript?

Closures are mechanisms used for enabling data privacy by combining a function enclosed with reference to its surrounding environment. A closure function has access to the scope in which it is created and not the scope in which it is executed.

```
/*
 * Example of simple closure
 */
function parentFunction() {
  let outerVar = 'I am in the parent scope!';
  function innerFunction() {
    // Variable outerVar is store in the scope
    console.log(outerVar);
  }
  return innerFunction;
}

const myClosure = parentFunction();
myClosure();
// Output: I am in the parent scope!

/*
 * Example of useful closure
 */
function multiply(a) {
  return function innerMultiply(b) {
    // Variable a is store in the scope
    return a * b;
  }
}

const double = multiply(2);
double(3);
// Output: 6
double(8);
// Output: 16
```

What is the difference between slice and splice?

Use slice to pick elements from an array that you do not want to mutate, whereas splice primary use is to modify the original array inserting and removing elements.

```
const food = ['burrito', 'pasta', 'noodles'];

// Using slice to pick elements form an array based on elements index
const myFood = food.slice(1, 2);
// myFood is ['pasta']
// food is ['burrito', 'pasta', 'noodles']

// Using splice to update the original array based on elements index
food.splice(1, 1, 'burger');
// Output: ['pasta']
// food is ['burrito', 'burger', 'noodles']
```

What is scope in JavaScript?

The scope of a variable refers to the lifetime and accessibility of variables within the context of code and is generally of two types, global and local. A global variable is accessible outside the block where it is declared, whereas local variables will only be accessible within the block where it is declared.

```
// Initialize a global variable
var globalVariable = "I'm a global variable";

const testingScopes = () => {
  // Initialize a local variable only visible in this block
  var localVariable = "I'm a local variable";
  // Log global variable as functions has access to the outer scope
  console.log(globalVariable);
};

// Log global and local variables
console.log(globalVariable);
// Output: "I'm a global variable"
console.log(localVariable);
// Output: ReferenceError: localVariable is not defined
testingScopes();
// Output: "I'm a global variable"
```

Why is JavaScript a dynamic language? Is it strongly or weakly type?

JavaScript is dynamic because variables can be created at the runtime, and the type of them will be determined at runtime. Moreover, JavaScript is a weakly typed dynamic language since it does not need to specify the type of variable that will be stored. The language automatically assigns a type to the variable during code execution.

```
var thisIsAString = 'I am a string';
> typeof thisIsAString;
// Output: string

// Assign an integer
thisIsAString = 1;
> typeof thisIsAString;
// Output: number
```

What are generics in TypeScript?

Generics is a tool in TypeScript that allows the creation of reusable code components that can work with many types and empower classes, interfaces, and types to act the parameters want them to.

```
function getArray<T>(items : T[] ) : T[] {
    return new Array<T>().concat(items);
}

let myNumArr = getArray<number>([100, 200, 300]);
let myStrArr = getArray<string>(["Hello", "World"]);

myNumArr.push(400); // OK
myStrArr.push("Hello TypeScript"); // OK

myNumArr.push("Hi"); // Compiler Error
myStrArr.push(500); // Compiler Error
```

What are error boundaries in React

Generics is a tool in TypeScript that allows the creation of reusable code components that can work with many types and empower classes, interfaces, and types to act the parameters want them to.

```
class ErrorBoundary extends Component {
  state = {
    error: null
  };

  static getDerivedStateFromError(error) {
    // Update state so next render shows fallback UI.
    return { error: error };
  }

  componentDidCatch(error, info) {
    // Log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  render() {
    if (this.state.error) {
      // You can render any custom fallback UI
      return <p>Something went wrong</p>;
    }
    return this.props.children;
  }
};
```