Functional Programming

Quiz

You can find the answers on the next page.

1. What is functional programming and what are its benefits?
2. Name a few functional programming languages.
3. What is a higher order function? Give an example.
4. What is function composition?
5. What is currying? Show an example.
6. What is a pure function?
7. What are the benefits of pure functions?
8. Why does immutability matter?

Answers

1- What is functional programming and what are its benefits?

Functional programming is a programming paradigm or a style of programming. Functional code tends to be more concise, easier to debug, easier to test and more scalable.

2- Name a few functional programming languages.

Clojure and Haskell are pure functional languages. JavaScript is a multi-paradigm language.

3- What is a higher order function? Give an example.

A higher-order function is a function takes a function as an argument or returns a function (or both). Array.map() is an example of a higher-order function.

4- What is function composition?

Function composition is a technique for building complex functions from small, reusable functions. If we have two functions f(x) and g(x), we can compose them as g(f(x)). Here, the result of f(x) gets passed to g(x) as an argument.

5- What is currying? Show an example.

Currying is a technique for reducing the number of parameters of a function to one.

```
const add = a => b => a + b;
```

6- What is a pure function?

A pure function always produces the same result for the same arguments and is free of side effects.

7- What are the benefits of pure functions?

Pure functions are easy to debug, easy to test and more scalable. Everything a pure function needs is specified in its arguments. We don't have to set some global state before testing them. Pure functions don't mutate their arguments or global state. So, we can call them and know the objects we pass to them don't get mutated. There are no surprises.

8- Why does immutability matter?

Immutability allows us to write programs that are more predictable. We know that if call a function and give it an object, the object doesn't get changed by the function. There are no surprises!

It also improves change detection in React and Redux apps because Redux can do a shallow comparison between two object references to tell if an object is changed. It doesn't know to do a full comparison that involves checking every property.

Immutability also allows us to write more scalable code. We can run multiple function calls in parallel. We know that our functions don't mutate data so they don't accidentally modify the state of the program.