Functional Programming

Functional programming is a declarative paradigm of building software by composing pure functions.

Pure Functions

A pure function is a function which:
1. Given the same input, returns the same output.
2. Has no side-effects

Benefits of Functional Programming

It helps us to solve problems effectively in a simpler way

It improves modularity

It reduces complex problems into simple pieces

It helps us to debug the code quickly

Arrow functions

Increment	const incr = x => x + 1
Add 2 numbers	const add = (x, y) => x + y
Square root	const squareRoot = x => Math.sqrt(x)
Square	const square = x => x * x

Currying

Add 2 numbers	const add = x => y => x + y
Multiply with a number	const multiply = m => n => n * m
Find volume	const volume = l => w => h => l *w* h
Find volume of cylinder	const aCylinder = volume(10)(10)(10)

Currying is a transformation of functions that translates a function from callable as f(a, b, c) into callable as f(a)(b)(c).

Array methods

Double each array element	const map1 = array1.map(x => x *2)
Print each array element	array1.forEach(e => console.log(e))
The sum of array elements	array1.reduce((acc, val) => acc + val)
Filter array	[1, 2, 3].filter(n => n > 2)

You can shorten and simplify your code with the Array methods!

Higher-Order functions

```
function greaterThan(n) {
    return x => x > n;
}let greaterThanTwo = greaterTh-
an(2);console.log(greaterThanT-
wo(5));
```

In FP, an important concept is Higher-order functions.

It's a function that accepts (what's typically called) a callback function.
In JS you can benefit by using libraries like Ramda or lodash/f.

Function Composition

var compose = (f, g) => (x) => f(g(x));

Function Composition is an act of composing/creating functions that allow you to further simplify and compress your functions by taking functions as an argument and return an output.

It may also return another function as its output other than numerical/string values.