106)sectional overview

It originates from mathematics and some times it can get complicated. Functional programming is gaining popularity nowadays bcoz of its use in languages developed with functional programming in mind work well with distributed computing.

109)Pure fucntions

1)Same input gives same output. 2)functions cannot modify anything outside of itself(no side effects)

If a functions modifies anything outside of it, it is called side effect. We dnt knw what can happen to state. anyone can call this function anywhere and change the state. if we have many functions like this, then we have to go one by one in all functions to figure out, how these functions are modifying the data.and that is one of the problem with having side effects is that reusing the shared state (global variable) that can interact with anything and order of functions call matters. It can causes various bugs.

See code snippet – this is impure function.

So how can we make this code not having any side effects?

110)Pure Function -2

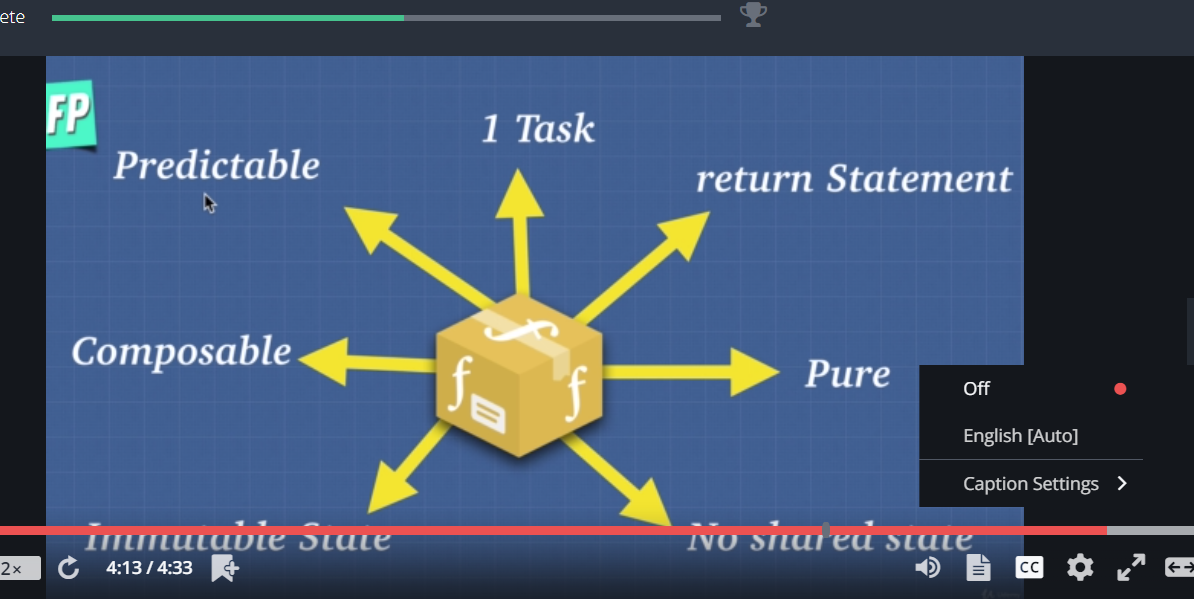
We have a program without any side effect. See code snippet2.js.

Console.log(1); this is also side affect

See refrencial transparency.- if I change function to 7 , it will not have any affect.

Pure functions are very to test, very to compose and it avoids a lot of bugs because we have no mutations, no shared state.

111)Can everything be pure

We cannot have only pure functions. The goal of functional programming is not to make everything pure functions, its goal is to minimize the side effect. The idea is to organize your code such that there is a specific part with sideeffect. So that when we have some bug, we know exactly we need go to that spot, because that is spot having side effect. Rest of ypur code well those are just pure fucntions and because they are pure we dnt have to worry about them as much.

112)Idempotent

For same input we always get same output.

113)Imperative vs declarative

Imperative- what to do and how to do

Declarative- what to do

114)Immutablility

Structural sharing and read article on structural sharing.

115)Higher order functions an closures

Closures only make a fucntions impure if we modify the closed over variable. If we just return a closed over variable without modifying it, we are following the principles of functional programming.

Using clousres we can have private variables. In functional programming, closures are used a lot for this thing in functional programming. We just have to be careful not to modify the state with in closure i.e we can return variables but we cannot modify the

116)Currying

It s technique of translating the evaluation of function that takes multiple arguments into evaluating the sequence of functions each with single argument.

See cod snippet- 3.js

Now what is use of it? I can now create utility functions out of it. See code snippet 3.js.

Currying reminds of methods on prototypes that are shared, they are trying to save some memory. Lets say our multiply by 5 fucntion is called many times, now we are only calling part of original function again and again.

117)Partial Application

It is very similar yet slightly different to currying.

119)MCI: Memorization 1

Memorization is caching the return value of function based on its parameter and if parameter of this function does’nt change like it is here then its memorized. See code snippet 5.js

120)MCI: Memorizaton 2

We dnt want to pollute our global space . so we will use closures.

See code snippet 6.js.

What we learnt here is very powerful because it allows us to be very efficient. Dynamic programming allows us to use what we know about memorization to optimize our code.

121)Compose and pipe