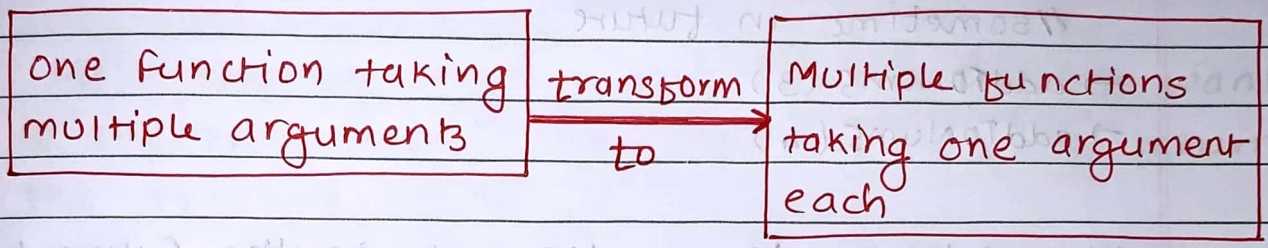
Javascript - Day 11

CURRYING :-

⇒ Currying is a mechanism where we can translate the evaluation of a function that takes multiple arguments into evaluating a sequence of functions that take a single argument.




```
function add (a,b) {
    return a+b;
}
```

```
function addNum1 (a){
return function addNum2(b){
        return a+b
    }
}
```

*** → The reason why we are able to achive this is becoz function addNum2 has access to variable a due to closure.

→ So now, how do we use this functions?

→ addNum2 (5) (3) and not addNum1(5,3); because addNum1 only takes one argument and it returns a function so result of addNum1 (5) is a function (addNum2) and we want to pass the second argument to addNum2.

→ Alright, why would ~~somebody~~ someone even do this in the first place??

Scanned by CamScanner

Let's say we just call
→ addNum1(5)

this will return us a function, so let's store it for future use.

→ const addToNum5 = addNum(5)
// sometime in future
→ addToNum5(3)
→ addToNum5(4)

→ We are trying to run less code in the future by storing addToNum5 for future and not running it everytime.

* Compose :-
→ The data processing that we do should be obvious.

Example:
Let's say we want to make a paste from 5 ingredients mixed together.

item1 —add→ item2 ⟹blend⟹ newItem —add→ item3 —blend→ Find Product

→ so find product is made from item1, item2, item3 and the order doesn't matter.

→ what compose say's is we should be able to do the following.

item2 —add→ item3 → newItem —add→ item1 → final product
blend blend

Scanned by CamScanner

→ Composability is a system design principle.

→ A highly composable enviroment components can be assembled in various combinations and still get the right output.

● Code Example :-

Let's say we have a negative number we want to multiply it with another number and return the absolute value.

$-2 * 3 \Rightarrow -6 \xrightarrow{\text{absolute}} 6$

Let's compose these together.

1. const compossedResult = compose (multiplyBy3, return absolute)

⇓

we need to define our own compose function.

2. const compose = function (funcA, funcB) {
return function (data) {
return funcB (funcA (data))
}
}

↳ We can also do funcA (funcB(data))

>> composeResult (-2)
⇒ 6

→ Let's assume multiplyBy3 and return Absolute are defined.

→ On line one we're calling compose function that

Scanned by CamScanner

takes 2 functions as arguments and returns a functions as arguments and returns a function (line 3), this function is stored in composed result. We can now call composed Result with any data (-2) and it applies funcB on result of funcA.

Scanned by CamScanner