Functional JS – Udemy

Software\_Dev\00303\_WebFront\_JS\_jQuery\_Bootstrap\Functional\_Programming\_For\_JS\_Developers\

# Aspects of imperative prograaming to fix by Functional Programming

1. We were defining global vars, outside the functions where they are assignbed their values.
2. We use independent funcitons
3. We pass values from one to other function, and they are redefined.
4. Native JS and DOM methods mmixed together.
5. Unclear what is happening outside of the script that may affect at the values we are worling with, ror itf other functions may be relaying on those values.
6. May be the function names are repeated in other loaded scripts.
7. Brilttle code, that won’t worl in a different context (for example, if we don’t define those global vars first).

# Main concepts

## Object-oriented JS

It’s good to switch to OO, specially in some env where different people are sending scripts to be loadad in the browser.

### Advantages:

1. Allows data to be treated as object to be passed around the code.
2. Objects have methods for manipulating them, and also their instances.
3. Messages can be passed artound to one object to another.
4. Props of objects can either be hidden or shared allowing encapsulation > security. One function doesn’t interfere with another.
5. Allows to build up functionality of objects by adding to the abilities of an object to Composition.
6. Inheritance: enhancing ..
7. Work with objects takes advantage of encapsulation, inheritance:
8. Use, for a defined object, its property .prototpype. …(), in order that when I have several instances of that object I am just using 1 instance of the prototype in memory.
   1. Example: Object SomeText,

// COnstructor

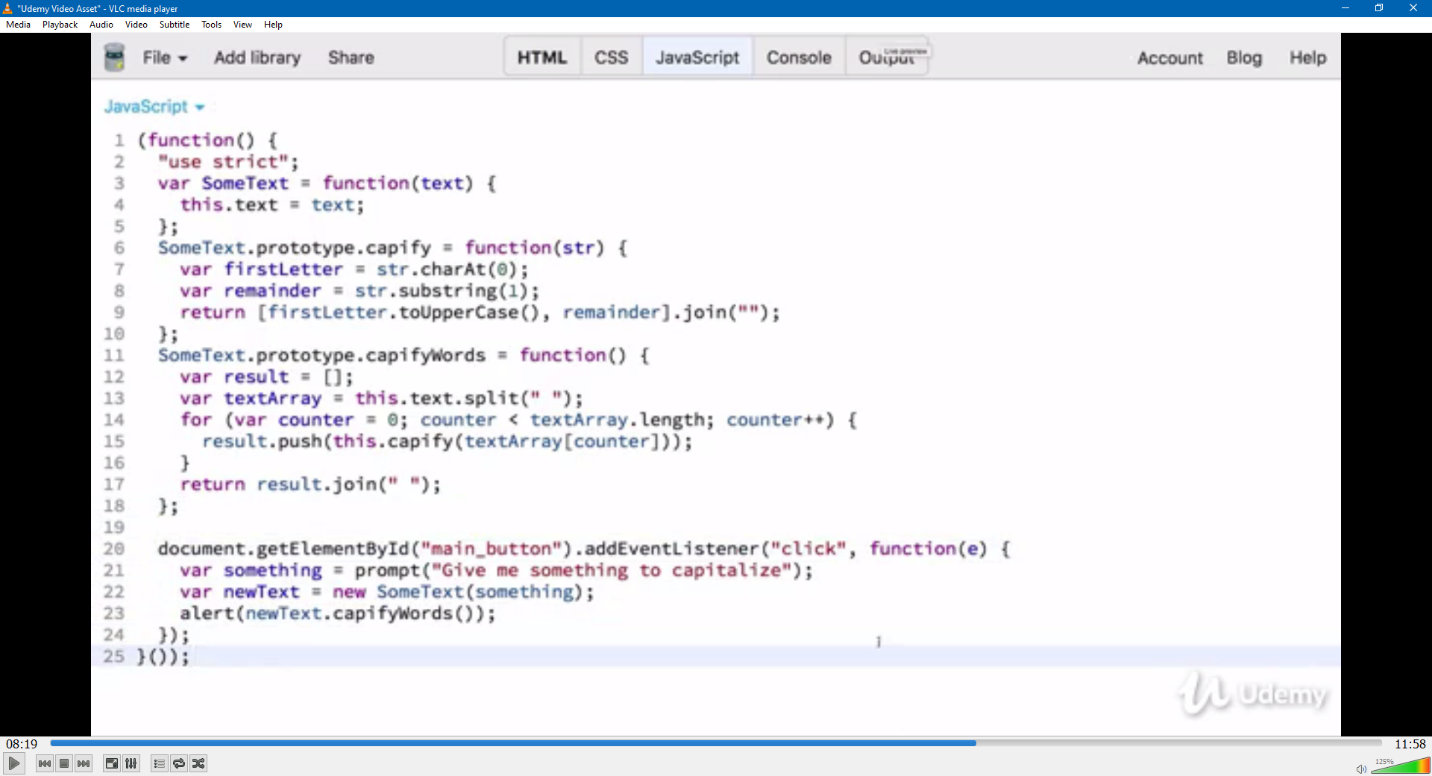
var SomeText = function(textToHandle) {

this.text = textToHandle;

}

with two methods:

* + 1. .prototpype.capitalize(param)
    2. .prototpype.capitalizeWods



The first defines the code just for 1 word, The second is in charge of iteration of calls to the first one.

TIPS:

* Use “use strict” in a limited scope, for example, in a function or inside the definition of an object, since there can be other code not written for this convention, and it can break it.
* Use addEventListener is more versatile than onclick because it can be use for different types of events.

### DISAdvantages:

1. Methods are taken from prototype, and use properties of the object set with a very simple constructor, which hardens to keep track of what is happening in the code.
2. There are a couple of reasons why looping can be messy:
   1. Loop like this, relies on having an extra var, counter. It is good to avoid creating extra state you will need to track: avoid creating extra variables.
   2. The loop is changing vars outside itself, which dificults to know what the code is doing.
3. Using of the ‘new’ keyword can create hidden and silent errors.
4. The code is still brittle, not portable: it relies on prototype in multiple places, and JS is more complex. Also the front and back end requirements … []. Also for testing, it is improvable.

## Functional style

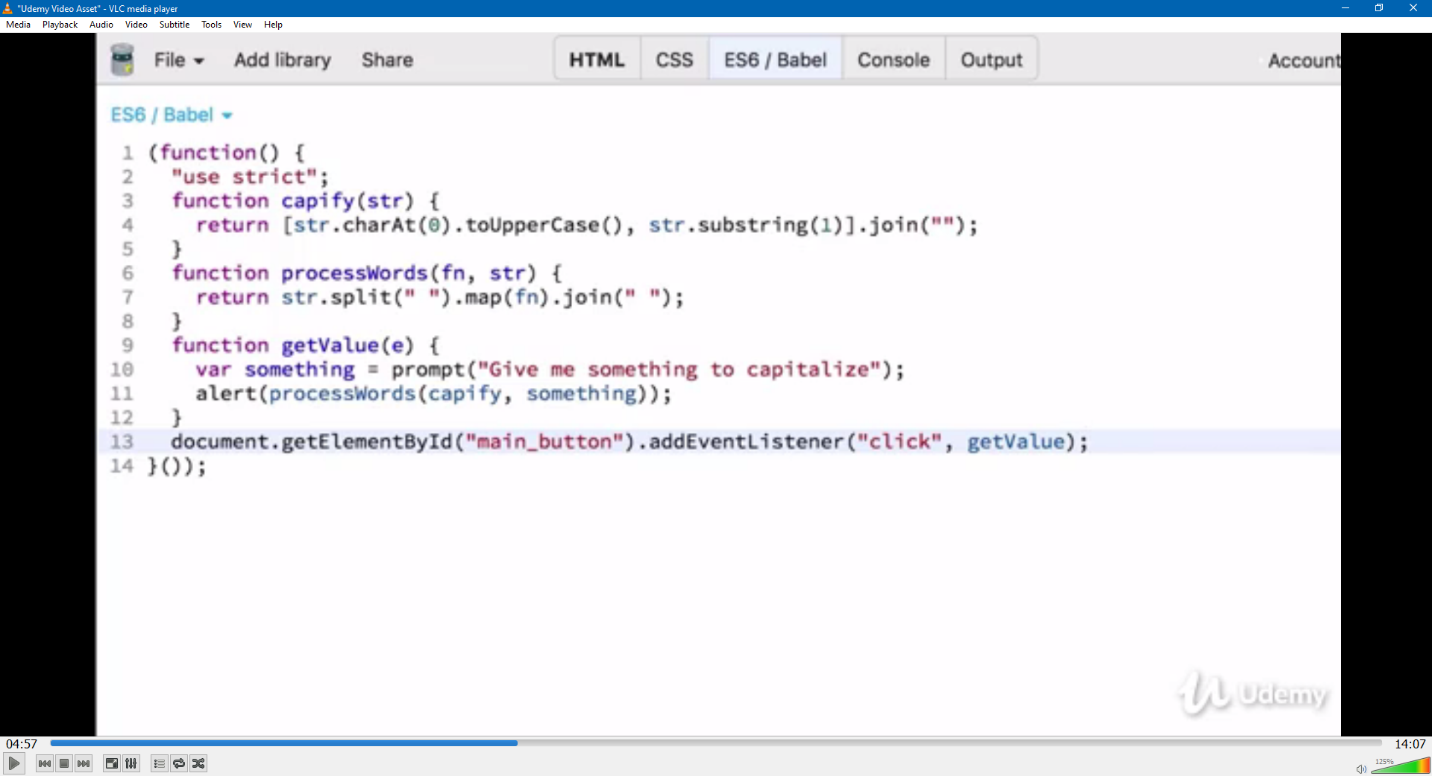
* JS is a very versatile language, can be used in a:
  + Imperative style
  + OO style
  + Functional style

No one is best, it depends on the case.

Functional style:

1. takes advantage of the fact that JS allows you to use functions as 1st-class objects: use them as arguments and pass them between functions.
2. Encourages to not to make changes to values that are outside of the function, or change the state of app while executing the function
3. Ideally, in the function you are going to describe the problem or solution, not the step-by-step; which is a little bit complex.

Example:



COMMENTS:

* Code is much more concise.
* Fuinctions are independent and reusable.
* No reliance on abstract object passed around: we have the vars, passed through. No outside objects that affects to the result inside of the functions.
* Much easier to unit test: we know input and it’s associated output, then. Regardless of the context.
* We pass a fn to processWords(), and inside processWords() the arg fn IS NOT CALLED AS A CALLBACK BUT AS A PREDICATE FN, INSIDE AN ARROW METHOD AS .map(). There are some JS features and methods we saw at the beginning of the course, they are useful for functional programming, and this is one of them perhaps: it doesn’t LOOP, or passes an array as a final argument of the caller function through a “callback” call, but instead makes an iteration from a “group” method as map(); which is much more efficient. Map() also allows not relying on extra-variables as in a loop.