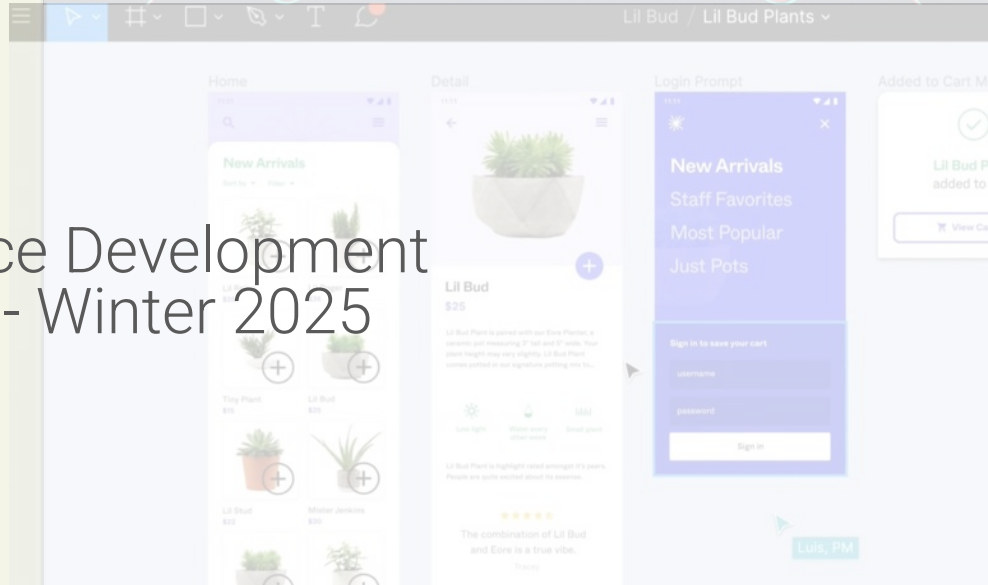
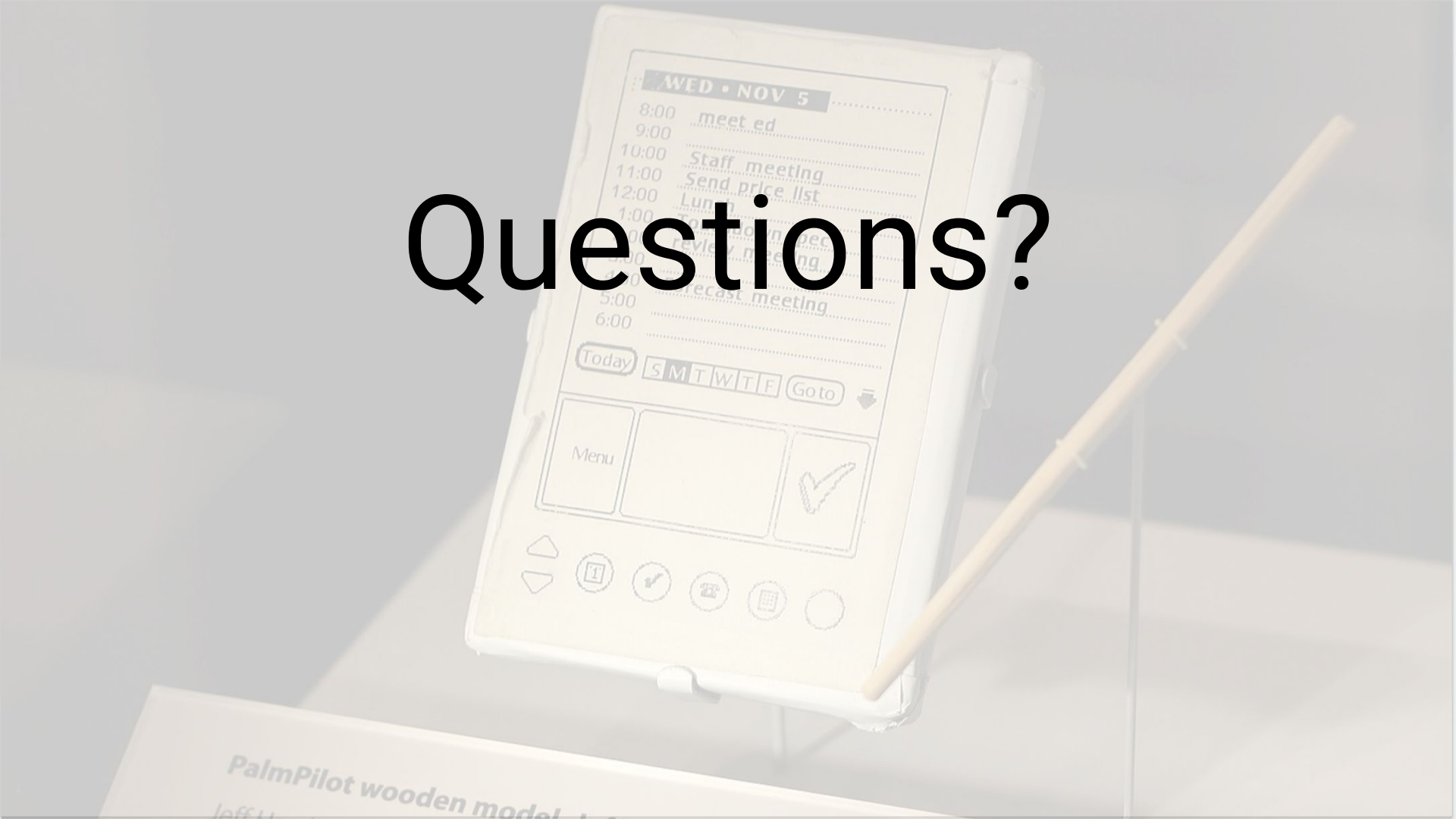
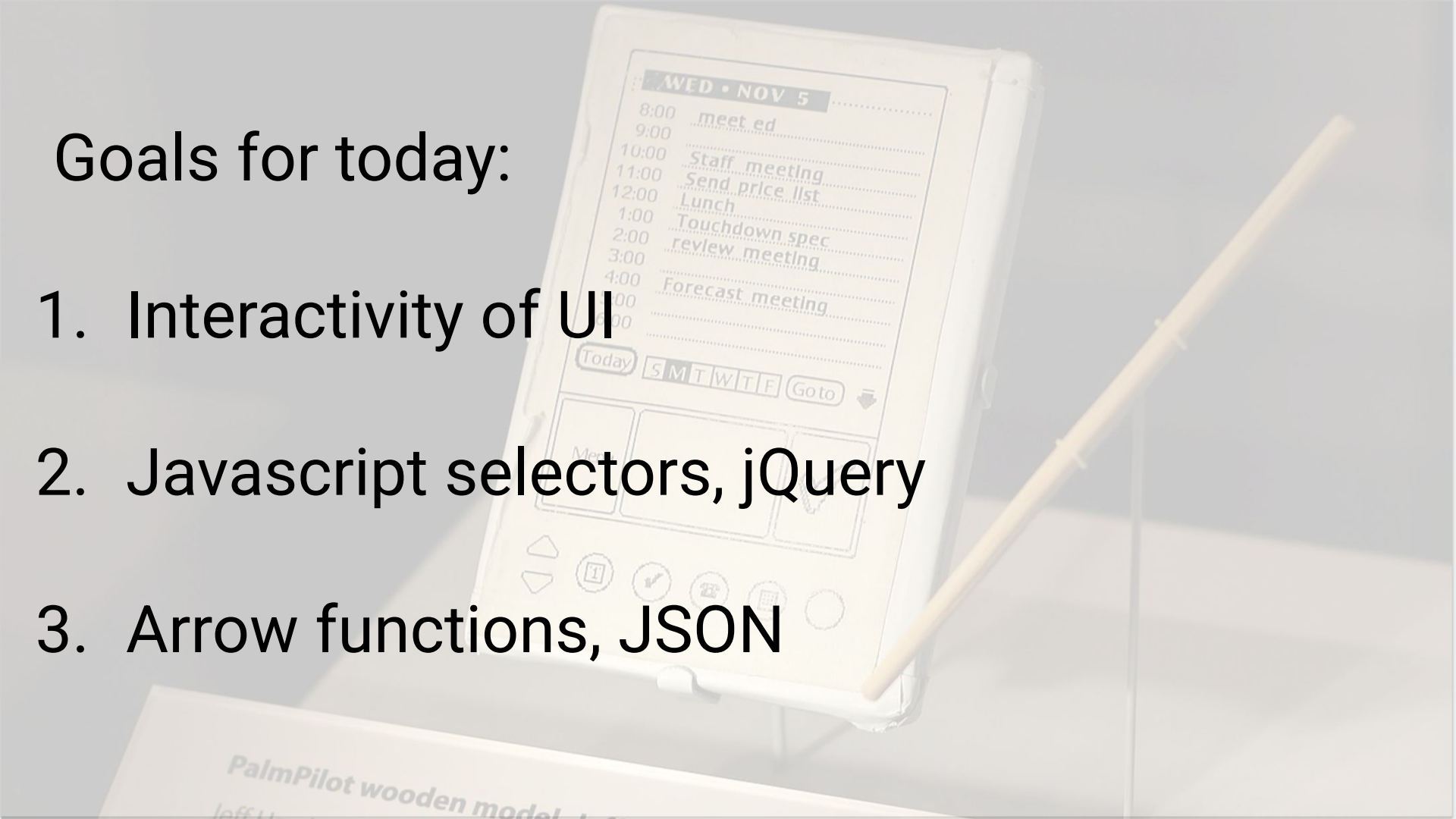
JavaScript Part 2

User Interface Development
EECS 493 - Winter 2025




WED • NOV 5
8:00 meet ed
9:00
10:00 Staff meeting
11:00 Send price list
12:00 Lunch
5:00
6:00
Today
S M T W T F
Goto
Menu
PalmPilot wooden model

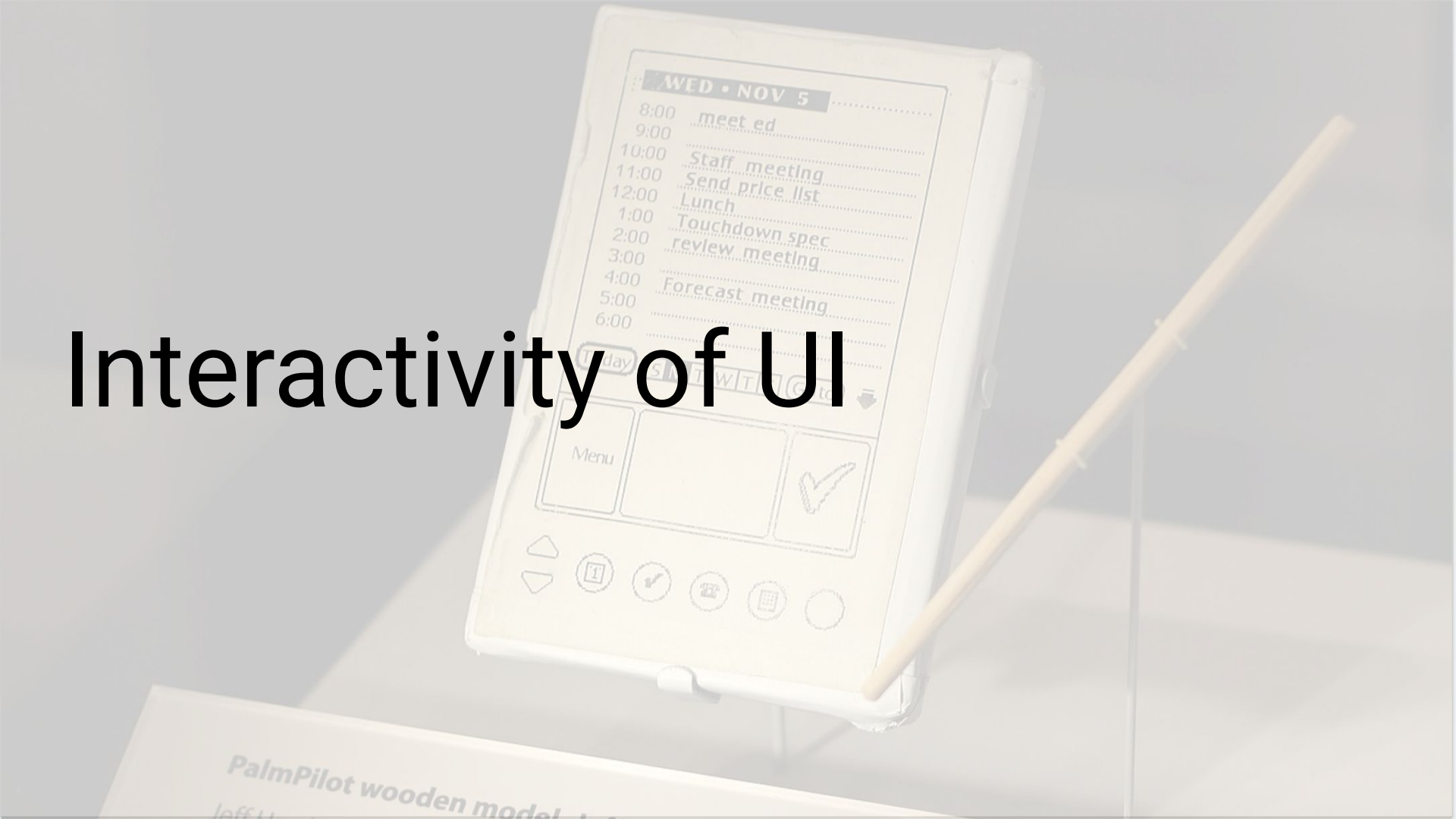
Questions?



Goals for today:

1. Interactivity of UI
2. Javascript selectors, jQuery
3. Arrow functions, JSON

Interactivity of UI



Difference of UI vs. other software

- Many other software focus on procedures:
  - Calculate X, retrieve Y from a database, etc.
  - The system decides when to move to a new state, execute a function, etc.
  - Goal: as fast as possible, as little resource as possible
- UI software:
  - Users need to provide input, e.g., type text, push button, navigate a menu, etc.
  - More critical goal: ***User Agency and Control!***

Why do we need JavaScript?

- HTML defines what is on our page (content)
- CSS defines how our content is laid out (layout)
- JavaScript defines how it all works (interactivity)

Amazon.com. Spend less. Smil

amazon.com/ref=nav_logo

Incognito

amazon Hello Select your address All Hello, Sign in Account & Lists Returns & Orders 0 Cart

All Best Sellers Prime Customer Service New Releases Today's Deals Pharmacy Books Fashion Registry Toys & Games Kindle Books Gift Cards Amazon Home Sell Computers Automotive Coupons

4+ star men's fall fashion

See more

Customers' Most-Loved

Women's

Men's

Beauty

Home

4+ star fashion, beauty & more

Back to School fashion

Clothing

Uniforms

Shoes

Backpacks

Shop smart picks

Sign in for the best experience

Sign in securely

Off to College for her

Off to College fashion

Stock up for college

Get lost in a new release

Basic Interaction Types

- Click/MouseUp/MouseDown
  - Can be broken into: mouse-down, mouse-up
  - Also, can be left click, right click, center click, etc.
- Key press
  - Can be broken into: key-down, key-up
- Mouse move
- Hover/Focus/Enter/Leave
- Drag
- [Lots of other stuff]
  - E.g., pen-based interactions: click, path, …
  - E.g., touch-based interactions: press, long-press, n-finger press, pinch, …

Livecoding example

05-livecoding

mouseevents.html

Simple procedures are not enough

No problem! We know how to handle input!
…right?

```
public static void main () {
    cout << “How old are you?”;
    cin >> age;
    cout << “Wow, you don’t look a day over”;
    cout << age - 1 << “!\n”;
}
```

Simple procedures are not enough

So we get something like…

```
public static void main () {
   hoverIn >> menu;
   openMenu(menu);
}
```

Does this even work in terms of interaction?

Some options

- We could poll
  - “Did you click the mouse? Now? Now? Now?”

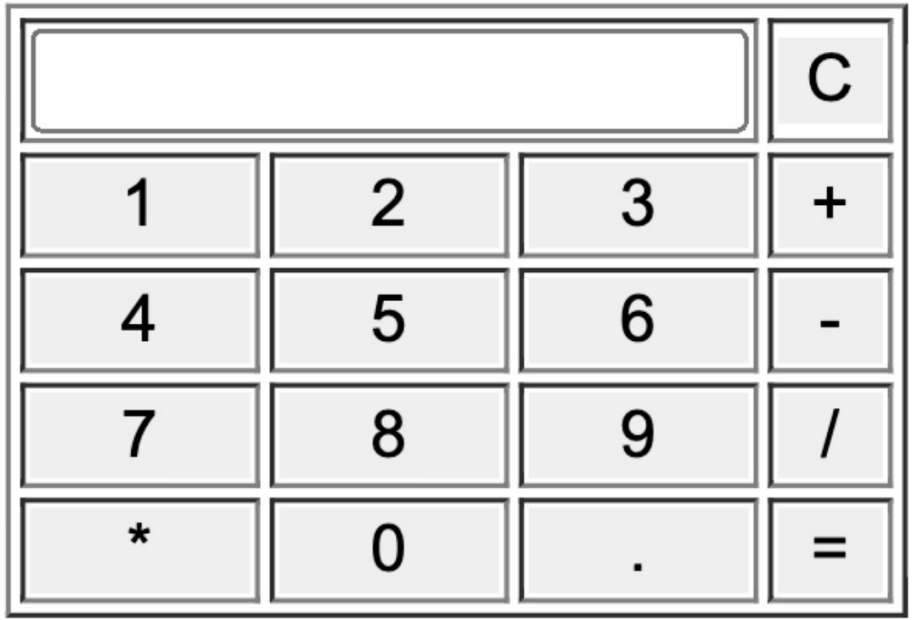
Some options

- We could poll
  - “Did you click the mouse? Now? Now? Now?”
- Alternatively we could have the polling loop running as a parallel thread or process.
  - If the mouse button goes down and up, do this.
  - If “this” is hard-coded, not very flexible

Some options

- We could poll
  - “Did you click the mouse? Now? Now? Now?”
- Alternatively we could have the polling loop running as a parallel thread or process.
  - If the mouse button goes down and up, do this.
  - If “this” is hard-coded, not very flexible
- Or, the code could say, “When the mouse button is up, trigger this function (stored in a variable) that I am specifying (callback).”

Let’s build a calculator (calculator.html)



JavaScript: Handling Keypresses

Each keypress just adds a character to the results field.

```
function addkey(val){
    document.getElementById('result').value+=val;
}
```

```
<body>
    <div id='wrap'>
    <table border="1">
        <tr>
            <td colspan="3"><input type="text" id="result" readonly></td>
            <td><input type="button" value="C"  onClick="cleard()"></td>
        </tr>
        <tr>
            <td><input type="button" value="1" onClick="addkey('1')"></td>
            <td><input type="button" value="2" onClick="addkey('2')"></td>
            <td><input type="button" value="3" onClick="addkey('3')"></td>
            <td><input type="button" value="+" onClick="addkey('+')"></td>
        </tr>
```

Listeners: Event Handlers

- Listener receives event
- One event can be heard by any number of Listener (as in any Observer Pattern), e.g., a button can have multiple listeners.
- Once a Listener hears an event, triggers associated Handler
- Listeners execute Callback Functions
  - Callback functions: i.e., “Call me back when you hear the word”

How this is implemented?

UI:

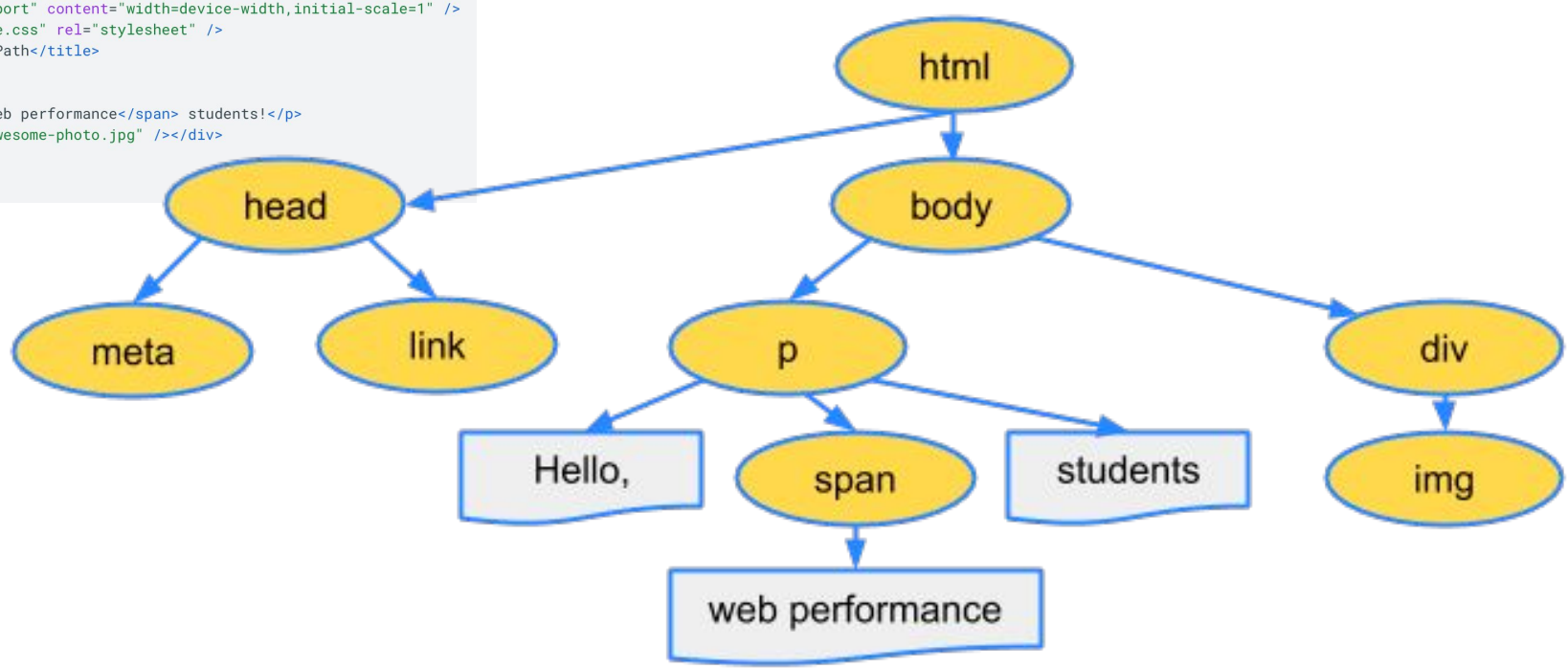
```
<button id=‘myBtn’></button>
```

Code:

```
   <button onclick="handleClick()">,
or
   document.querySelector("myBtn").onclick =
   function(event) { … }
or
   myBtn.addEventListener(“click”, function(event));
```

Example of a DOM Tree

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <link href="style.css" rel="stylesheet" />
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div><img src="awesome-photo.jpg" /></div>
  </body>
</html>
```



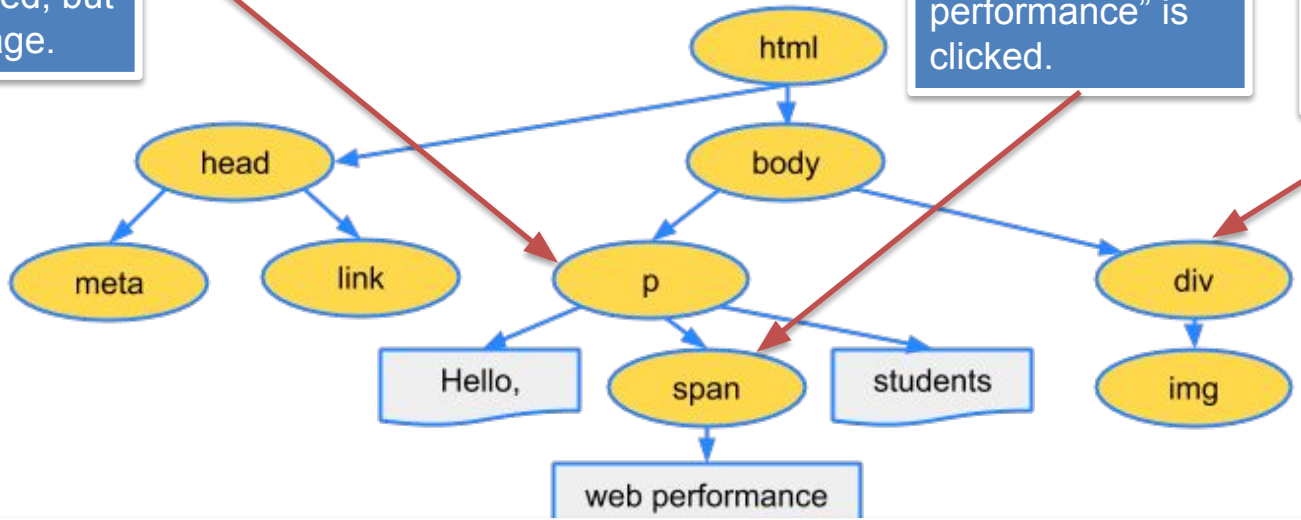
https://web.dev/articles/critical-rendering-path/constructing-the-object-model

Events propagate up!

If you handle clicks here, the handler (“C”) responds if any text is clicked, but not the image.

If you handle clicks here, the handler (“B”) only responds if “web performance” is clicked.

If you handle clicks here, the handler (“A”) only triggers when the image is clicked.



It’s completely okay to have all three click handlers doing different things.
Which handler fires when you click on “web performance”? When you click on “students”?

Javascript Selectors

Vanilla JS Selectors

- Just like in CSS, we need selectors to interact with the DOM (our instance tree)

- JS selectors look something like this:
  - `document.getElementById(’name').addEventListener('click', clickHandler);`
  - `document.getElementsByClassName(‘name’).addEventListener('click', clickHandler);`

Livecoding Example – selectors.html

```
<html>
<head>
<title>Playing with JavaScript</title>

</head>
<body>

<div >
<p id="section1">This is the first paragraph.</p>
<ul>
<li>Apple
<li>Bear
<li>Cat
</ul>

<div >
<p id="section2">This is the second paragraph.</p>
<ul>
<li>Donkey
<li>Epsilon
<li>Pharma
</ul>
```

This is the first paragraph.

- Apple
- Bear
- Cat

This is the second paragraph.

- Donkey
- Epsilon
- Pharma

Example – selectors.html

Defining a click listener

```
document.getElementById("section1").addEventListener("click",clickFunction);

function clickFunction() {
  document.getElementById("section1").style.fontSize="xx-large";
}
```

Defining a double-click listener

```
document.getElementById("section2").addEventListener("dblclick",dblClickFunction);

function dblClickFunction() {
  document.getElementById("section2").innerHTML = "I was double-clicked!";
}
```

Example – selectors.html

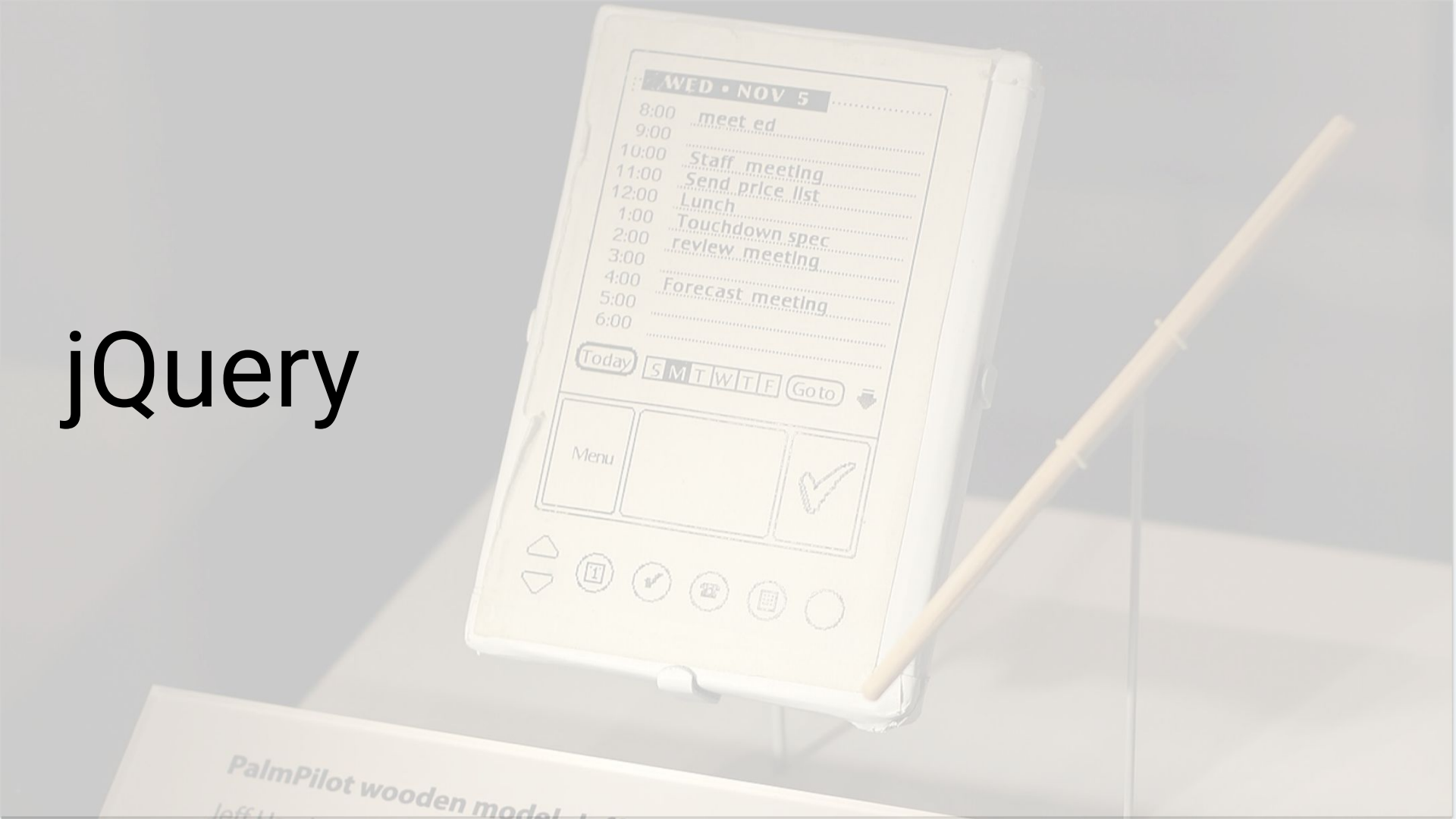
```
document.getElementById("section1").addEventListener("click",clickFunction());

function clickFunction() {
  document.getElementById("section1").style.fontSize="xx-large";
}
```

What does that line do? *

- [ ] Works fine - click on #section1, and it'll call clickFunction(), which increases the size of the text.
- [ ] Gets an error
- [ ] Always make #section1 XX-large because it calls clickFunction()
- [ ] It's broken, but could be fixed if clickFunction() returned a function

jQuery


WED • NOV 5
8:00 meet ed
9:00
10:00 Staff meeting
11:00 Send price list
12:00 Lunch
1:00 Touchdown spec
2:00 review meeting
3:00
4:00 Forecast meeting
5:00
6:00
Today
S M T W T F
Goto
Menu
PalmPilot wooden model

The jQuery Library

- jQuery is a layer above JavaScript that provides simplifications
  - more easily attach event handlers
  - more easily manipulate the DOM

jQuery Example

Suppose in your HTML you have:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

To add a callback function `submitForm` to the `onclick` listener using JavaScript:

```
document.getElementById(’myButton’)
    .addEventListener('click’, submitForm);
```

The same, using jQuery:

```
$(‘#myButton’).click(submitForm);
```

JQuery selector

Listener

Callback function

jQuery Selectors

Suppose in your HTML you have:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

Select a specific element by its ID:

```
$(‘#myButton’)
```

Select elements by class:

```
$(‘.formButton’)
```

Select elements by element name and class:

```
$(‘button.formButton’)
```

Select the whole document:

```
$(document)
```

jQuery Listeners

Suppose in your HTML you have:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

To add a callback function `submitForm` to the `onclick` listener:

```
$(‘#myButton’).click(submitForm);
```

To add a callback function `submitForm` to the `ondoubleclick` listener:

```
$(‘#myButton’).doubleclick(submitForm);
```

To add a callback function `showInstructions` to the `mouseover` listener:

```
$(‘#myButton’).mouseover(showInstructions);
```

jQuery: Modifying DOM

Suppose you want to **add** this button to the `body` of your HTML:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

Using JQuery:

```
$(document.body).append('<button id="myButton"
class="formButton">Submit</button>')
```

jQuery: Modifying DOM

Suppose you have this button in your HTML:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

To *get* the text of this button:

```
$(‘#myButton’).html();
```

To *set* the text of this button to ”Submit Form”::

```
$(‘#myButton’).html('Submit Form');
```

jQuery: Modifying DOM

Suppose you have this button in your HTML:

```
<button id=”myButton” class=“formButton”>Submit</button>
```

Syntax to set any CSS parameter to a particular value:

```
$(‘#myButton’).css(<parameter>, <value>);
```

For example, to set the background color to yellow:

```
$(‘#myButton’).css(‘background-color’, ‘yellow’);
```

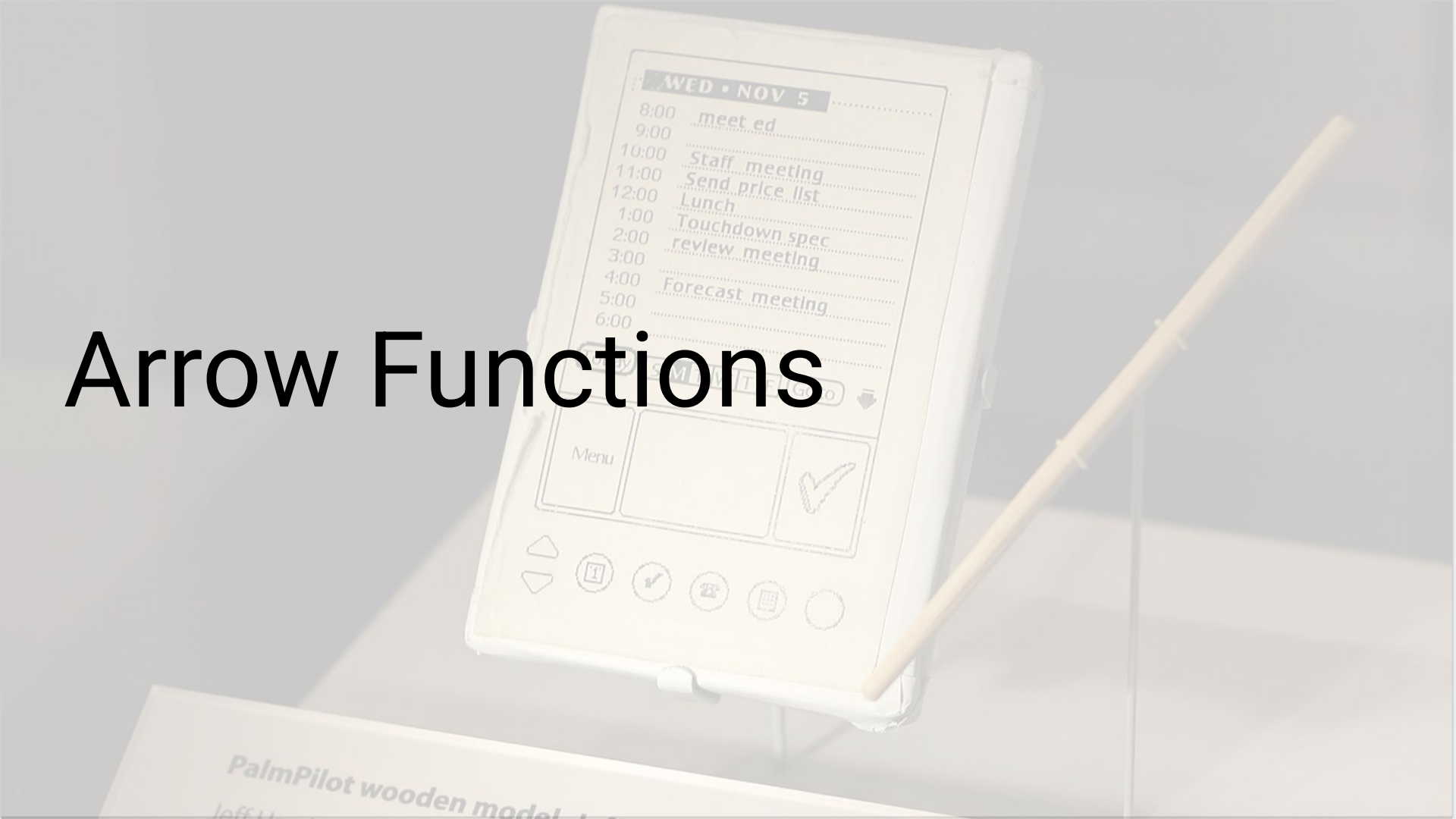
A3 Example of where you'll use jQuery

If myComet is a div of a certain class, and #gameScreen is the div containing the game screen:

```
$('#gameScreen').append(myComet);
```

makes the comet appear

Arrow Functions



Arrow Functions

- Arrow functions allow us to write shorter function syntax.

Array Processing

- Array.forEach(...)
- Processing arrays
  - Array.map(function) – applies function to each element in array.
  - Array.filter(function) – only returns elements where function returns true
  - Array.reduce(function) – combines successive elements of an array

Let’s create an array

Linked via Syllabus in 05-livecoding.zip > Places.html

```
let places = [
{name: "Michigan Stadium",
 city: "Ann Arbor",
 state:"MI",
 maxsize: "111238"},
{name: "Spartan Stadium",
 city: "East Lansing",
 state: "MI",
 maxsize: "75005"},
 {name: "Indiana Memorial Stadium",
 city: "Bloomington",
 state:"IN",
 maxsize: "52626"},
{name: "Ohio Stadium",
 city: "Columbus",
 state: "OH",
 maxsize: "102780"}]
```

Using forEach to walk the array

```
> places.forEach(function(p){console.log(p);})
```

VM1071:1
▶ {name: "Michigan Stadium", city: "Ann Arbor, MI", maxsize: "111238"}

VM1071:1
▶ {name: "Spartan Stadium", city: "East Lansing, MI", maxsize: "75005"}

VM1071:1
▶ {name: "Indiana Memorial Stadium", city: "Bloomington, IN", maxsize: "52626"}

VM1071:1
▶ {name: "Ohio Stadium", city: "Columbus, OH", maxsize: "102780"}

You will need Places.html for in-class activity

Only the ones over 100,000

```
> places.forEach(function(p){if (p.maxsize > 100000)
{console.log(p)};})
```

VM1132:1

```
▶ {name: "Michigan Stadium", city: "Ann Arbor, MI", maxsize: "111238"}
```

VM1132:1

```
▶ {name: "Ohio Stadium", city: "Columbus, OH", maxsize: "102780"}
```

Noticed maxsize was a string. JavaScript isn't particularly worried about types.

Using filter() with an arrow function

```
> places.filter((p) => p.maxsize > 100000)
<- ▼(2) [{…}, {…}] i
     ▶0: {name: "Michigan Stadium", city: "Ann Arbor, MI", maxsize: …
     ▶1: {name: "Ohio Stadium", city: "Columbus, OH", maxsize: "1027…
      length: 2
     ▶__proto__: Array(0)

> (places.filter((p) => p.maxsize >
  100000)).forEach((s)=>console.log(s.name))
  Michigan Stadium
  Ohio Stadium
```

This says: (1) filter places *p* where maxsize > 100000 then (2) foreach stadium *s* put the name in the console.

Use reduce() to sum sizes

```
> (places.map((p)=>p.maxsize)).reduce((a,b) => a+b)
```

```
> (places.map((p)=>p.maxsize)).reduce((a,b) => a+b)
<· "111238750055262610278O"
```

```
> (places.map((p)=>1.0*p.maxsize)).reduce((a,b) => a+b)
<· 341649
```

Write the one line of JavaScript (using arrow functions filter and reduce) that sums up the sizes of the MI stadiums *

Your answer

Write the one line of JavaScript (using arrow functions filter and reduce) that sums up the sizes of the MI stadiums *

Your answer

```
(places.filter((p) => p.state=='MI').map((l)=>
1.0*l.maxsize).reduce((a,b)=>(a+b)))
```

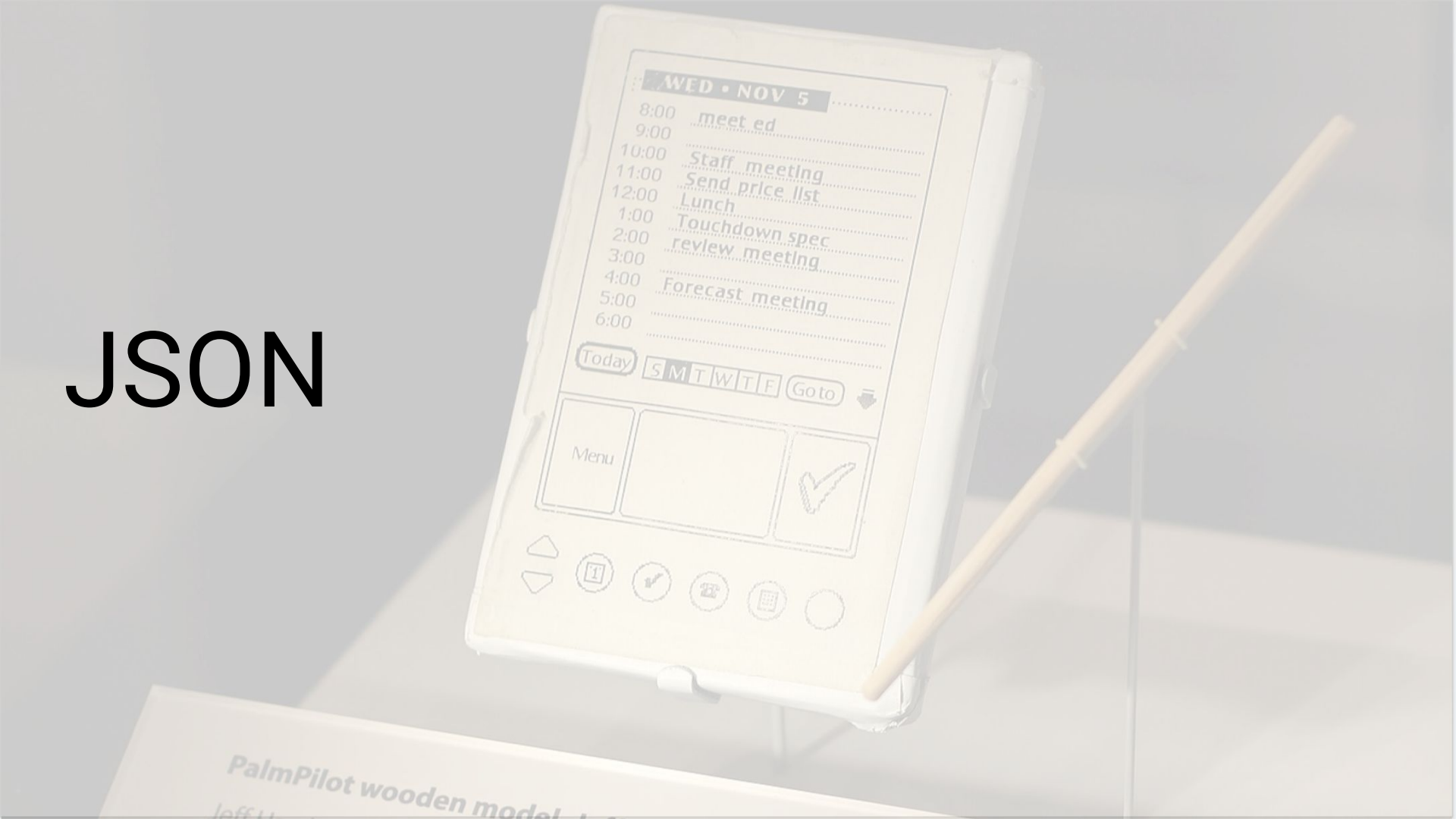
Imagine that you have an array of comets, and you want to find the comets that might be intersecting with your avatar's rectangle. Which one might you use? *

- ○ map
- ○ filter
- ○ reduce
- ○ foreach

Arrow functions are a shorthand for *

- ○ Array.map
- ○ Array.foreach
- ○ Anonymous functions
- ○ Click handlers

JSON


WED • NOV 5
8:00 meet ed
9:00
10:00 Staff meeting
11:00 Send price list
12:00 Lunch
1:00 Touchdown spec
2:00 review meeting
3:00
4:00 Forecast meeting
5:00
6:00
Today
S M T W T F
Goto
Menu
PalmPilot wooden model

JSON: Javascript Object Notation

- It is a string representation of an object that adheres to strict syntax rules.
- It is commonly used for data exchange between systems (e.g. between a server and a web application)

```
// JSON syntax
{
    "name": "John",
    "age": 22,
    "gender": "male",

}
```

JSON object

- Keys must be strings enclosed in double quotes.
- Values can be strings, numbers, arrays, booleans, null or other JSON objects.

```
{
  "name": "Alice",
  "age": 30,
  "isStudent": false,
  "hobbies": ["reading", "cycling", "hiking"],
  "address": {
    "city": "Ann Arbor",
    "state": "MI"
  }
}
```

Accessing JSON data

```
// JSON string
const jsonString = `{
  "name": "Alice",
  "age": 30,
  "isStudent": false,
  "hobbies": ["reading", "cycling", "hiking"],
  "address": {
    "street": "123 Main St",
    "city": "Ann Arbor",
    "state": "MI"
  }
}`;
```

Accessing JSON data

```
28  // Convert JSON string to JavaScript object
29  const jsObject = JSON.parse(jsonString);
30
31  // Accessing properties of the JavaScript object
32  console.log(jsObject.name); // Output: Alice
33  console.log(jsObject.age); // Output: 30
34  console.log(jsObject.isStudent); // Output: false
35  console.log(jsObject.hobbies); // Output: [ 'reading',
    'cycling', 'hiking' ]
36  console.log(jsObject.address.city); // Output: Ann Arbor
37
```

Javascript to JSON

```
39  const jsObject = {
40    name: "Alice",
41    age: 30,
42    isStudent: false,
43    hobbies: ["reading", "cycling", "hiking"],
44    address: {
45      street: "123 Main St",
46      city: "Ann Arbor",
47      state: "MI"
48    }
49  };
```

Javascript to JSON

```
51  // Convert JavaScript object to JSON string
52  const jsonString = JSON.stringify(jsObject);
53
54  // Output the JSON string
55  console.log(jsonString);
56
```

Comparison

JSON	JavaScript Object
The key in key/value pair should be in double quotes.	The key in key/value pair can be without double quotes.
JSON cannot contain functions.	JavaScript objects can contain functions.
JSON can be created and used by other programming languages.	JavaScript objects can only be used in JavaScript.

JSON: Javascript Object Notation

```
> var myObj = {name: "John", age: 31,
  city: "New York"};
< undefined
> myObj
< ▶ {name: "John", age: 31, city: "New Yor
  k"}
> JSON.stringify(myObj)
< "{"name":"John","age":31,"city":"New Yor
  k"}"
```

- In JavaScript, we can directly type in the properties and values of an object.
  - It's a string that we can write and read, in files and databases (MongoDB).

Holes in JSON

- JSON is not a complete representation of JavaScript objects.
- Here are two holes:
  1. Functions
  2. Object references

JSON doesn’t know functions

- Your functions don’t get output in a JSON format.
  - It’s data only.

```
> let hourlyEmployee = {
    name: "Unassigned",
    hoursPerWeek : 40,
    computePay : function()
    {return this.hoursPerWeek *
this.hourlyRate;},
    reportPay : function()
    {console.log(this.name+" is paid
"+this.computePay());}
    }
<· undefined
> JSON.stringify(hourlyEmployee)
<· "{"name":"Unassigned","hoursPerWeek":4
0}"
```

JSON object references

- If your object references another object, JSON includes the referenced object

```
let place = {
    name: "Beyster Building",
    address: "2260 Hayward Street",
    city: "Ann Arbor",
    state: "Michigan"};
let AnhongOffice = {
    name: "Anhong Guo",
    building: place,
    roomNumber: 3741};
```

```
> JSON.stringify(AnhongOffice)
< "{"name":"Anhong Guo","building":{"name":"Beyster Building","addres
s":"2260 Hayward Street","city":"Ann Arbor","state":"Michigan"},"roomN
umber":3741}"
```

Offices in Beyster

```
let place = {
 name: "Beyster Building",
 address: "2260 Hayward Street",
 city: "Ann Arbor",
 state: "Michigan"};
let AnhongOffice = {
 name: "Anhong Guo",
 building: place,
 roomNumber: 3741};
 let XuOffice = {
 name: "Xu Wang",
 building: place,
 roomNumber: 3737};
```

JavaScript knows that:

```
> AnhongOffice
◂· ▶ {name: 'Anhong Guo', building: {…}, roomNumber: 3741}
> XuOffice
◂· ▶ {name: 'Xu Wang', building: {…}, roomNumber: 3737}
> AnhongOffice.place == XuOffice.place
◂· true
>
```

JSON object references

- JSON doesn't know they are the same object

```
> JSON.stringify(AnhongOffice)
'{"name":"Anhong Guo","building":{"name":"Beyster Building","address":"2260 Hayward Street","city":"Ann Arbor","state":"Michigan"},"roomNumber":3741}'
> JSON.stringify(XuOffice)
'{"name":"Xu Wang","building":{"name":"Beyster Building","address":"2260 Hayward Street","city":"Ann Arbor","state":"Michigan"},"roomNumber":3737}'
> JSON.parse(JSON.stringify(AnhongOffice)).building
▶ {name: 'Beyster Building', address: '2260 Hayward Street', city: 'Ann Arbor', state: 'Michigan'}
> JSON.parse(JSON.stringify(XuOffice)).building
▶ {name: 'Beyster Building', address: '2260 Hayward Street', city: 'Ann Arbor', state: 'Michigan'}
> JSON.parse(JSON.stringify(AnhongOffice)).building ==
JSON.parse(JSON.stringify(XuOffice)).building
false
```

Ways around it

- Manage it yourself.
  - Give places unique IDs.
    When you read in the JSON, match the object reference to the place IDs.

How can you get JavaScript object references right when generating JSON for your objects? *

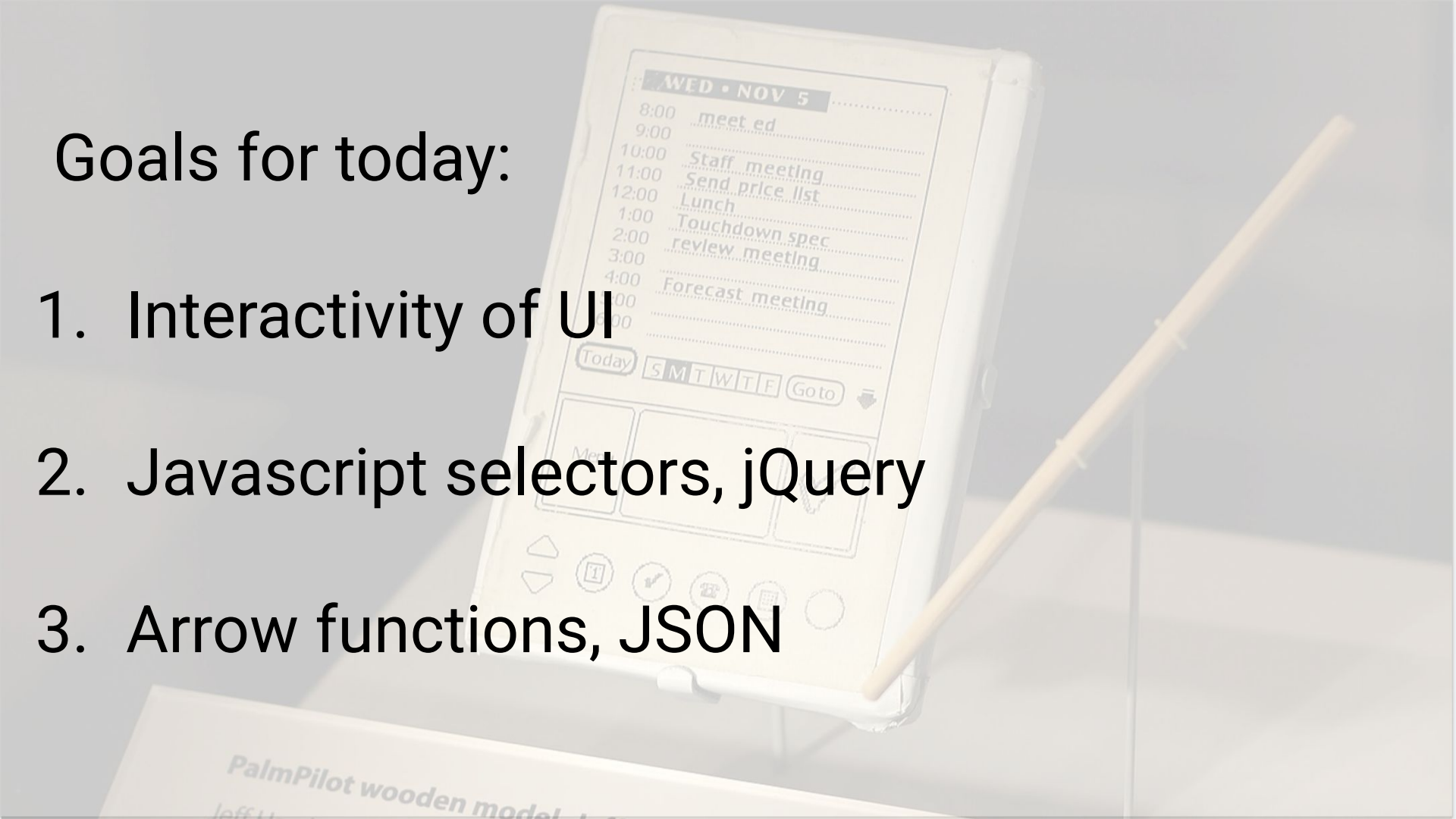
- ◯ It's not a problem -- it just works.
- ◯ Don't use object references - reference unique IDs to look up.
- ◯ It's not a problem because JavaScript objects can't contain other object references.

Which of the following is true? *

- JSON is no big deal -- most object-oriented languages have a way of typing in objects directly.
- JSON could be easily used to output C++ objects.
- JSON has an asynchronous version called JASON.
- JSON is unable to represent functions, dates, or shared object references.

Assume myCount is an object. What will be the result of evaluating: myCount === JSON.parse(JSON.stringify(myCount)) *

- ○ It's true, obviously.
- ○ It's false, because JSON.parse will capitalize the property names
- ○ It's false, because they become two different objects, even if values are the same
- ○ It will generate an error



Goals for today:

1. Interactivity of UI
2. Javascript selectors, jQuery
3. Arrow functions, JSON