Lecture 3:

Review of JavaScript

05-431/631 Software Structures for User Interfaces (SSUI)

Fall, 2022

© 2022 - Brad Myers

Logistics

- Switch back to 4x3 format to leave room for the video
- Homework 1 due 1 week from today

- Slides and recordings of labs are on Canvas

© 2022 - Brad Myers

JavaScript Syntax

- Syntax similar to C, C++, Java:
  - Blocks use `{ }`
  - Separate statements with ;
  - Arithmetic and precedence the same: `+, ++, *, %, >=` etc.
  - Loops: `for (let step = 0; step < 5; step++) { … }`
    - Also `do { } while (), while() { }`
  - Conditionals: `if (i<0) {…}`
    - Also `switch() {…}`
    - "ternary": `cond ? exprIfTrue : exprIfFalse;` `(let x = flag ? 3 : 5;)`
      - *Used a lot in JS!*
  - Assignment with `=`
  - Equality test with `==` or `===` (equal value *and* equal type)
    - Almost always use ===
  - Arrays: `[ ]` – zero based
  - Comments are `/* xxx */` or `//xxx`
  - Identifiers with letters, numbers, _ or $ (not -)
  - Case **sensitive**
  - Strings with " " or ' '
    - Can next the other kind inside: `'Brad said "hi".'`

© 2022 - Brad Myers

Dynamically Typed

- Never declare the type of variables, parameters, functions, etc.
  - `let i = 3; i="str"; i = null;`
- Special `undefined` value: `let x;` → `undefined`
- Arrays can contain multiple types: `[3, "foo"]`
- Numbers are `23` or `45.3` (no distinction int <-> float)
- Automatic conversion: `"5"+2+3` → `"523"`
  - Vs. `2+3+"5"` → `"55"`
  - `"11" - 1` → `10`
- `str.length` → note *NOT* a method ~~`str.length()`~~
- But lots of other string methods, e.g., `str.trim()`
- Like Java, strings are immutable (cannot change):
  - ~~`str[2] = 'p';`~~ ← doesn't work
  - All string methods return new strings
- Empty string `""`, `undefined`, `null`, `0` are all `false`: `if(b){}`

© 2022 - Brad Myers

Declaring variables

- `let x` – block scope – inside { }
- `var x` – function scope – anywhere in the function
- Either at top-level of file – global scope (all code running on this web page)
  - Resets if page is reloaded
- `const x` – block scope, and cannot be reassigned, so assign on declaration
  `const x = 123;` ~~`x = 4;`~~ ← error
  - But if x is an object or array, it can be modified
    `const x = [2,3]; x[0]=5;` ← OK

© 2022 - Brad Myers

Debugging

- Chrome debugger has “console” where can type any JavaScript code
  - Can see the values of global variables
    - And locals if at a breakpoint inside a function
  - Can assign values, define functions, evaluate code
  - “Sources” tab allows breakpoints, editing code
    - But not saved, so just for experiments
  - At breakpoints, can see stack (“Scope” tab)
    - Run code in the context of that function
- `console.log(anything);` ← output anything to the console without stopping
- `alert("I am an alert box!");` ← pause
- `debugger;` ← break into debugger when run (ref) - ~~not ()~~

Functions

- ```
  function myFunction(p1, p2) {
      return p1 * p2;
  }
  ```
- Empty parameters: `function myFunc() {}`
  - If no `return` or if it has `return;` then returns `undefined`
- Functions can be values: `const f = myFunction;`
  - `()` signals to invoke it: `f(1,2);`

© 2022 - Brad Myers
  ```

Arrow functions

- Shorter way to write function definitions
  - Especially useful when shorter
  - Very popular, but harder to read
  - Many people use them exclusively
  - Emphasizes that the function is a value connected to the name

```
function h() {
  return "Hello World!";
}
```

- ```
  const h = function() {
    return "Hello World!";
  }
  ```
- ```
  hello = () => {return "hello"};
  ```
  `hello = () => "Hello";` → omit `{}` and `return` if one line
  `hello = (val) => "Hello" + val;` → parameter
  `hello = val => "Hello" + val;` → parameter

Connecting to DOM

- Built-in JavaScript functions to access and set the DOM for the web pages
- **Getting** DOM elements
  - The current page is available as the global variable `document`
  - `document.getElementById(id)` – remember that ID is always unique per page
  - `document.getElementsByTagName(name)` – tags like “div”, “p”, etc.
  - `document.getElementsByClassName(name)` – based on the CSS class name
  - The last 2 return `HTMLCollection`
    - Supports some array functions, like `[0]`, `.length`
    - Or turn into an array with: `Array.from(htmlCollection)`

Change, Create & Add DOM elements

Human-Computer Interaction Institute

- Add content to an element, like for a paragraph
  - Change or add content as a string

```
element.innerHTML = "new html content";
```

- Change attribute, like `href` for `a`, or `src` for `img`, or `classname`

```
element.setAttribute(attribute, value);
element.attribute = new value;
```

  e.g., `element.classname = "product"; //`CSS class becomes product

- Change style property

```
element.style.property = new style;
```

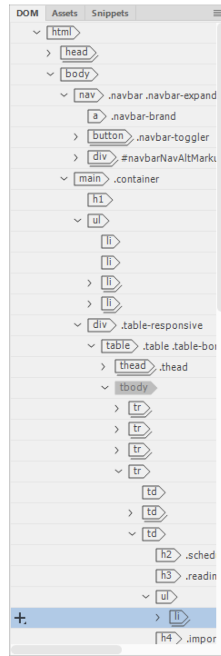
- Create a new element of any kind of tag
  - Note: always created in the document

    ```
    var newdiv = document.createElement("div");
    ```

    (or other tag)
  - Then add to the correct element as a child:

    ```
    element.appendChild(newdiv);
    ```

  - Remember the DOM tree



© 2022 - Brad Myers

Triggering functions

- Will cover event handling in detail in Lecture 4
- For hw1, only need simple event handling
- E.g., to call `initDetails` function when page is loaded, put this in the html file:

```
<body onload="initDetails()">
```

- Call function when button is pressed:

```
myButton.addEventListener("click", myfunction)
```
or
```
element.onclick = myfunction;
```

© 2022 - Brad Myers

Storing values between pages

- Global variables reinitialized on each html page load
- Many options to store information across pages
  - localStorage or sessionStorage APIs – easiest
    - local is permanent, session is reset on browser restart
    - Recommend localstorage for HW1
      `localStorage.setItem('myCat', 'cupcake');`
    - *Hint*: per URL address, so be careful if run same application twice in different tabs!
  - Passing values in the URL and parsing them at the receiving page (`decodeURIComponent`)
  - Cookies (used to be the only way)
  - Store in the browser (browser specific)
  - Store on a remote server (various APIs – hw6)

© 2022 - Brad Myers

Objects

- 2 ways to think of objects:
- 1) Just a collection of name-value pairs:
  `var car = {brand:"Fiat", model:"500", color:"white"};`
- Note: defined inside of `{}` (vs. `[]` for arrays)
  - Both separated by ,
- Access fields the usual way: `car.brand;` → `"Fiat"`
  - Or by array indexed by field name: `car["model"];` → `"500"`
  - Same for assignment: `car.brand = "Honda";`
- New fields can be added dynamically, just by assigning it:
  `car.size = 232;`
- Any value can be a function → method:
  `car.f = function(x) {return this.size+x;}`
  `car.f(12);` → `244;` `car.f;` → returns the function definition
  - Usually use arrow functions

Classes

- 2) Second way is as Classes, with subclasses
  - Think Java classes, not CSS classes
  - Must have a constructor
    - Assign class variables in constructor using `this` – don't declare them:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() { //define a method
    return "I have a " + this.carname;
  }
}
```

  - Create instances with `new` *`classname`*

```
let mycar = new Car("Ford");
```

© 2022 - Brad Myers

Inheritance (subclasses)

- Use `extends`
  - Like Java, subclass has everything of super-class plus whatever is added
  - Constructor *must* call super:

```
class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
```

  - Can *override* methods like in Java, etc.
    - Often call `super` in those as well, to call the super-class's method
- Can add new fields to any instance *dynamically*

```
mycar.price = 5000;
```

  - Can add new methods, since they are just values

Arrow Function and `this`

- Treatment of `this` is different
  - With function, is the object that the function is in dynamically
  - With arrow, is object that was defined in

```
> Obj = {val:4};
{val: 4}
> Obj.f = function(i){return this.val+i;};
f (i){return this.val+i;}
> Obj.f(12); //since function, gets this from Obj
16
> this
Window {parent: Window, …}
> this.val = "window";
"window"
> Obj.v = i => this.val+i; //since arrow, gets 'this'
                          //from scope v is defined in
i => this.val+i
> Obj.v(12);
"window12" // note the meaning of + determined dynamically
```

Shortcut syntaxes

- Lots of shortcut syntaxes
  - Sometimes clear, other times less readable
- "Object Destructuring"

```
const { top, left } = originalRect;
```

  - Uses the names "`top`" and "`left`" *both* as names of the variables *and* names of fields of the object
- Spreading (expand) the values, use `...` operator
  - Takes values of following item, and puts them into the new container; like "flatten"

```
let arry = [...htmlCollection]
myFunction(...obj)
```