Plan for today

JavaScript data types
  Arrays, objects, iteration

Modules
  Import and export

Classes
Array syntax

```javascript
let arr = [10, 20, 30];
/* Usual indexed for loop */
for (let i = 0; i < arr.length; i++)
    console.log(arr[i]);
/* Loop over elements */
for (let elem of arr)
    console.log(elem);
```

Caution: for ... of is very different than for ... in
Useful **Array** operations

```
arr.push(elem1, elem2, ...)
    Add element(s) to an array
arr.indexOf(value)
    Get index of value in arr (-1 if not found)
arr.slice(start, end)
    Return a subarray (also works for strings)
arr.splice(index, delCount, newElem1, ...)
    Insert and/or remove elements at index
```

Warning: delete arr[i] doesn't work!
JS Objects

(Plain) **Object** is a key-value store

Keys must be strings, values can be anything

**Syntax**

```javascript
let obj = {
    binky: 42,
    winky: "Hello",
    "key w/ $pecial_chars": []
};
console.log(obj["binky"]);```

JS Objects

Shorthand syntax

If key is a valid identifier, can use dot

console.log(obj.binky);

obj.dinky = 193;

Best practice: Use dot when possible
JS Object operators

Operators

"key" in obj

Check membership

Note: obj.nonexistentKey -> undefined

if (!obj.nonexistentKey) is common/useful, but be careful of falsy values

delete obj.key

Remove key/value pair
JS Object functions

Functions

("static" on Object, not methods on individual objects)
Key/value pairs iterated in insertion order

`Object.keys(obj)`
Array of object keys

`Object.values(obj)`
Array of object values

`Object.entries(obj)`
Array of pairs (arrays with length 2) of [key, value]
for (let key of Object.keys(obj))
console.log(key + ": " + obj.key);

for (let [key, value] of Object.entries(obj))
console.log(key + ": " + value);

for ... in can also iterate Object keys
  Recommendation: Avoid for ... in because it's confusing
Arrays and Objects are mutable

Variables and arguments store references

```javascript
const addElem = (arr) => {
  arr.push(42);
};
let arr = [1, 2, 3];
addElem(arr);
console.log(arr); // [1, 2, 3, 42]
```
Aside: some useful language features

**Destructuring: assign to multiple vars**

/* Get first and second elems of arr */
let [first, second] = arr;

/* Variable name matters here! */
let { binky, winky } = obj;

/* Fancier technique, "rest" value */
let [first, ...rest] = arr;

**Template strings**

for (let [key, value] of Object.entries(obj))
    console.log(`The key ${key} has value ${value}`);
Can contain any JS expression
Module exports

MDN reference

Module's variables not global
- Not automatically accessible from other module
- Need to be exported

export
- export let exportedVar = ...;
- export const exportedFn = () => { ... }
These are "named exports" (see next slide)

export default
- export default /* function, class, etc. */;
This is the "default export"
import Binky from "./Binky.js";
   Gets the default export from Binky.js, names it Binky
import { exportedFn } from "./Binky.js";
   Gets a named export (name must match exactly)
import Binky, { exportedVar, exportedFn} from "./Binky.js";
   Combined syntax

Paths must start with "./" (or "../" for parent)
Module strategies

Debugging strategies

Use the debugger to step/inspect variables
Use `console.log` + right-click "Store as global variable"
Assign to `window` object
(Of course, don't leave these in your final submissions)

Third-party libraries

Some libraries don't support modules (yet)
Include with `<script>` tag (without `type="module"`)
Access via global variable (window)
class Counter {
  constructor(start = 0) {
    this._count = start;
  }
  value() { return this._count; }
  add(n) { this._count += n; }
}

let c = new Counter(10);
c.add(5);
c.add(5);
console.log(c.value());
JS classes

MDN class syntax

constructor
  Special method name, called by `new`

Methods
  Define in class body

Fields (instance variables)
  Accessed through `this`
  Initialize in constructor (or method)
  Can add/delete dynamically
JS classes

Visibility

Mostly, everything is "public" (like Python)

One convention: prefix with _ for "private" members
   Don't access fields/methods starting with _ from outside

Newer: can make truly private fields with #
   Recommendation: I haven't seen this widely used yet, and it has some caveats and quirks, so I'd avoid for now.

this keyword

Not implicit (like Python, not like C++)

Determined at call time
   Huh? We'll figure out what this means next time...
Exceptions

try/catch blocks

```javascript
try {
    ...
    throw new Error("Boom");
    ...
} catch (e) {
    console.log(e.stack);
}
```
Exceptions

`throw <expression>`
Can technically throw anything
But probably should throw Errors

`new Error(message)`
Automatically builds a stack trace
Displays nicely in the console
Can have subclasses of errors
Summary

So far
  JavaScript language and syntax

Before next time
  assign0 due tonight
  assign1 out tomorrow

Next time
  Using JS with web pages
    Events, interactors