COMPSCI 326 - Web Programming
JavaScript Classes

join on the Slack #q-and-a channel as well as Zoom
remember, you can ask questions of your teammates on your group Slack!
please **turn on your webcam** if you can
mute at all times when you aren't asking a question
(https://docs.google.com/document/d/1-5LF53UmHHF8V28y-d8S7LPCBHwg6LcPie_-N5719Fo/edit?usp=sharing)

Background resources:

videos
"crash course " on JS - long, has a ton of stuff (not just for this next class):
https://www.youtube.com/watch?v=hdI2bqOjy3c
specifically higher-order functions (map, forEach, plus filter and sort) - code examples:
https://www.youtube.com/watch?v=rRgD1yVwIvE
this playlist has a whole "object oriented JavaScript" sequence of videos. #4-#6 are especially relevant for this lecture.
https://www.youtube.com/watch?v=4I3bTDlT6ZI&list=PL4cUxeGkcC9i5yvDkJgt60vNVWffpbIB7

web sites
MDN: classes -
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
class tutorial - https://javascript.info/class

Today: JavaScript Classes

Exercise today: create two classes (third for extra credit) based on the decoderRing from last time

Classes - object-oriented programming

- JavaScript has always had object-oriented programming in the form of *prototype-based programming* - Self → Chrome's V8 engine
    - in some sense, this is true "object" orientation - no classes!
    - it is outdated and we are not going to discuss it in lectures
- instead, as of 2015, JavaScript has *classes*, much like in Java and Python
    - classes = templates for stamping out objects in a certain "shape"
    - Note! not supported in Internet Exploder
    - MDN has this for all major browsers - good to know...

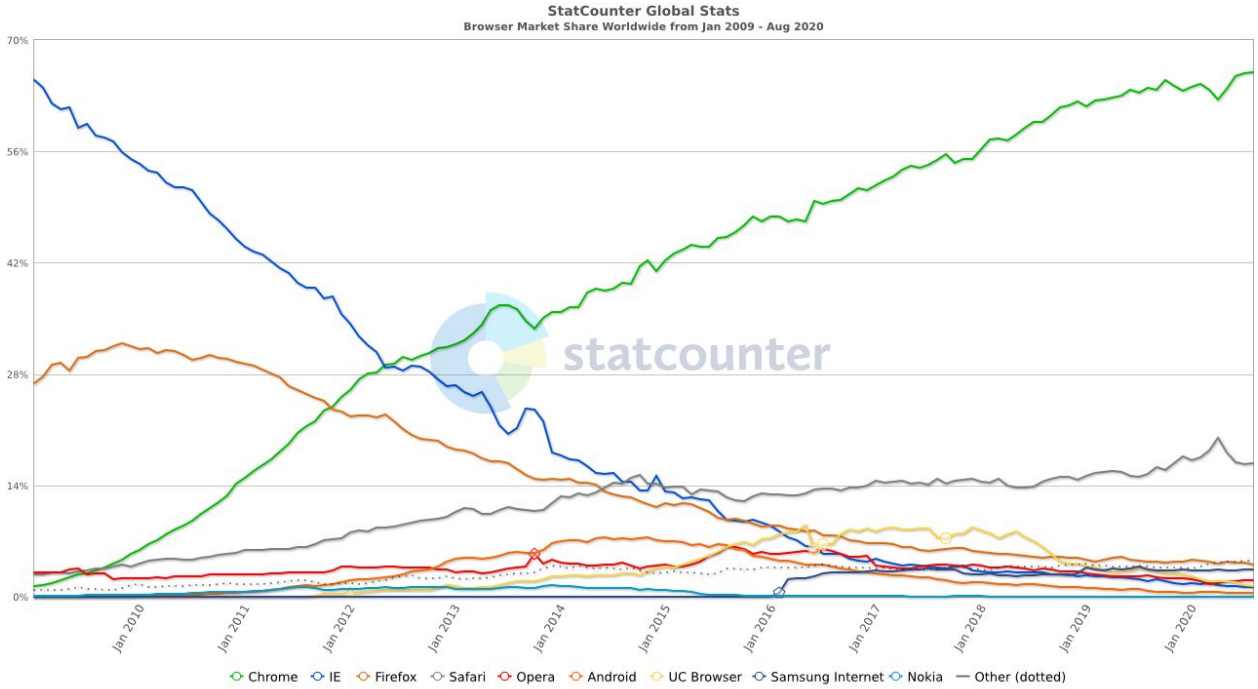
Browser compatibility

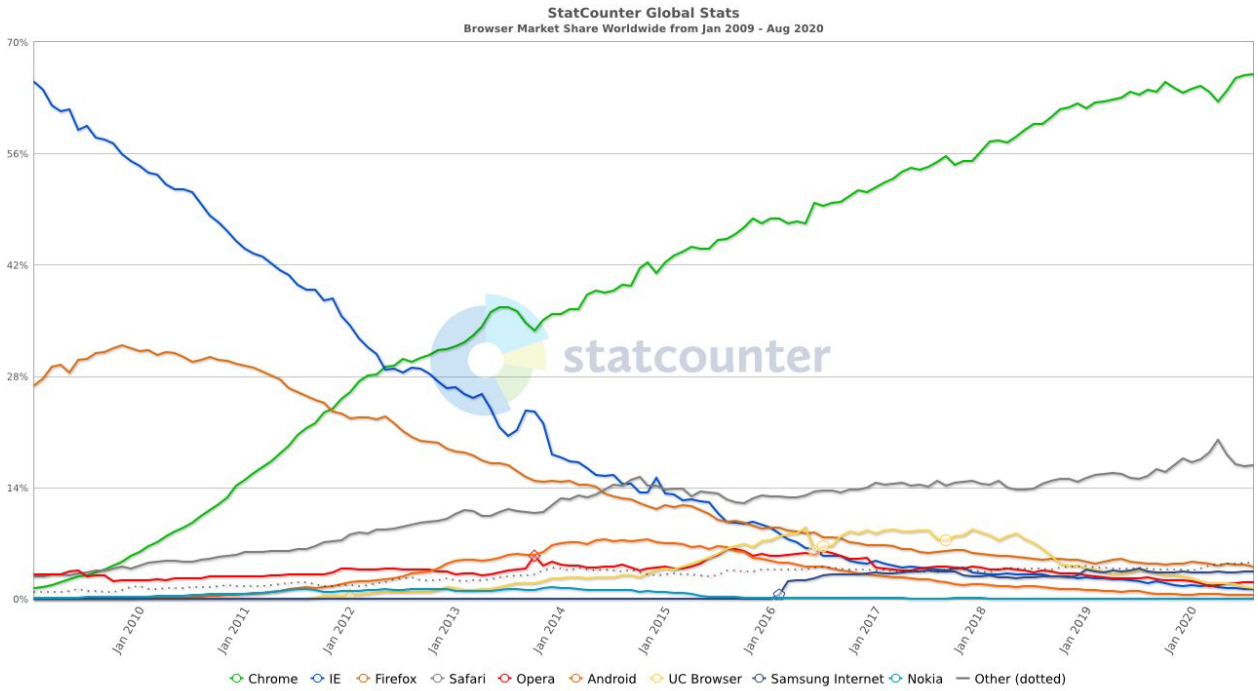
Update compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet	Node.js
`classes`	49	13	45	No	36	9	49	49	45	36	9	5.0	6.0.0
`constructor`	49	13	45	No	36	9	49	49	45	36	9	5.0	6.0.0
`extends`	49	13	45	No	36	9	49	49	45	36	9	5.0	6.0.0
Private class fields	74	79	No	No	62	14	74	74	No	53	14	No	12.0.0
Public class fields	72	79	69	No	60	14	72	72	No	51	14	No	12.0.0
`static`	49	13	45	No	36	9	49	49	45	36	9	5.0	6.0.0
Static class fields	72	79	75	No	60	No	72	72	No	51	No	No	12.0.0

- does this matter?

browser share over time...

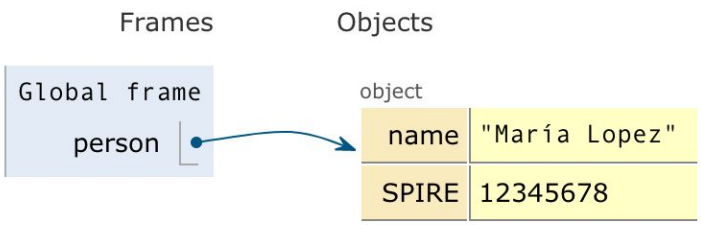




- three different approaches:
  - (only if you have a small about of code you need) Add the Polyfills: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill
  - Babel: https://babeljs.io/
  - TypeScript: https://www.typescriptlang.org/ (we will use this later)

Back to our person example:

```
let person = { 'name' : 'María Lopez', 'SPIRE' : 12345678 };
```



Instead of doing this one at a time, let's make a Person *class* and then we can easily make Person *objects*.

```
class Person {
  // … more to come
}
```

Then we will be able to do things like this:

```
// pretend we read this from a database
let people = [new Person('María Lopez', 12345678), new Person('José
Martínez', 24681357), new Person('Sara Silverman', 90908080)];

console.log(people[0].name); → 'María Lopez'
```

Woo! While yes, you can do this, it is boring and not that interesting. This is not why we actually use classes!

We *actually* use classes to:

- organize code conceptually instead of having a bunch of functions lying around
- encapsulate related code *and* data
- simplify reuse of code (you will see this in the exercise)
- take advantage of types (especially with TypeScript, coming soon)
- "information hiding" (especially with TypeScript, coming soon)

Things to know:

- **constructors** (making an object)

- **new** to make an object
- **getters** and **setters** to access and modify fields

```
class Person {
  constructor(thename) {
      // initialize the fields
      this._name = thename;
  }
  // a function
  print() {
    return this._name;
  }
  // a getter
  get name() {
    return this._name;
  }
  // a setter
  set name(newname) {
    this._name = newname;
    // a setter can do anything, and is useful for updating state
    console.log('name changed to ' + newname);
  }
}

let p = new Person('Janet Yellow');
p.print() → 'Janet Yellow';
p.name → 'Janet Yellow';
p.name = 'Janet Yellin';
→ name changed to Janet Yellin
```

Inheritance

- **extends** to create a subclass
- **super** to call the superclass's constructor with its arguments
- static functions

```
class Student extends Person {
```

```
  constructor(name, spire) {
    // have to initialize the super-class first
    super(name);
    // now do anything extra
    this._spire = spire;
  }
  print() {
    return this._name + ' ' + this._spire;
  }
  get spire() {
    return this._spire;
  }
  set spire(newSpire) {
    console.log('can't change SPIRE IDs');
    // maybe throw an exception?
  }
  static printType() {
    // this method works on classes, not objects
    // no "this" available - it's for EVERY object of this class
    // you can refer to this method as Student.printType()
    console.log('STUDENT');
  }
}
```

Exercise!

https://docs.google.com/document/d/1HPhZWpQN9cmhLxWkokZknR-cMrCq_VtUVVm3-FUIZL8/edit?usp=sharing