JavaScript and high-order functions
Why JavaScript?

• Linga franca of the Internet
  ➢ Used in the browsers, used server-side, used for IoT
  ➢ Still evolving to address growing needs (EcmaScript)

• Interesting goals and design trade-offs

• Illustrates many core concepts of CSE 130
Week 1

• A little bit of JavaScript history

• Concepts from JavaScript
  ➤ First-class functions
  ➤ Objects
  ➤ Language flexibility
May 1995
We need a scripting language for the browser!
May 1995

We need a scripting language for the browser!

How about Scheme?
How about Scheme?

Ha? No! Make it look like Java!
One week later...
One week later...

Here is a hacked up prototype!
One week later...

Here is a hacked up prototype!

Great! Let’s ship it!
One week later...

Here is a hacked up prototype!

Great! Let’s ship it!

(It really took another year to embed it in the browser)
JavaScript’s design goals [Eich, ICFP 2005]

• Make it easy to copy/paste snippets of code
  ➤ Tolerate “minor” errors — e.g., missing semicolons

• Simplify event handling (inspired by HyperCard)

• Pick a few hard-working, powerful primitives
  ➤ First-class functions (based off Scheme/Lisp)
  ➤ Objects everywhere (based off Self/Smalltalk)

• Leave all else out!
JavaScript has evolved

• EcmaScript 5 and 6 introduced many new features
  ➤ block scoping
  ➤ new types (Map, Set, Symbols, Uint8Array, etc.)
  ➤ strict mode
  ➤ module system
  ➤ classes

• How could JavaScript have been useful without these?
First-class functions!
First-class functions

• What does it mean for a language to have first class functions? (functions are values)
First-class functions

• What does it mean for a language to have first class functions? (functions are values)
  ➤ can be declared within any scope
First-class functions

• What does it mean for a language to have first class functions? (functions are values)
  ➤ can be declared within any scope
  ➤ can be passed as arguments to a function
First-class functions

• What does it mean for a language to have first class functions? (functions are values)
  ➤ can be declared within any scope
  ➤ can be passed as arguments to a function
  ➤ can be returned as result of function call
Function as scoping primitive

• Today: JavaScript has block scoping

• But, until recently, JavaScript only had function-level scoping

➤ What does this mean?

➤ How did people survive?
goto: scoping examples
Function as scoping primitive

• Whenever you want a new scope:
  ➤ declare a new function
  ➤ immediately call it

• Key requirement from language design:
  ➤ being able to declare function in any scope
Okay! But...

- Why do we want to pass functions as arguments?
- Or return functions as results?
Functions as args

• Original reason: simple way to do event handling
  ➤ E.g., onclick(function() { alert(“button clicked!”); })

• Still true today. But many other reasons, including:
  ➤ performance: asynchronous callbacks
  ➤ expressiveness: filter, map-reduce, etc.
Performance?

• Don’t need to block when reading file

• Can tell runtime system to call your “callback” function once it’s read the file

➤ This allows runtime to schedule other IO concurrently
goto: performance examples
Expressive power of passing functions

• Say more with less!
  ➤ E.g., filter all positive elements from array
  ➤ E.g., add 42 to every element of the array

• In both cases: we are expressing the computation we care about without telling the computer what to do
  ➤ Don’t need to clutter code with low-level mechanisms!
Why return functions?

• With the other 2 properties: let’s you compose functions from other functions
  ➤ Functions that do this are called “high-order”

• E.g., function composition: \((f \circ g)(x) = f(g(x))\)
  ➤ Here \(\circ\) is a function that takes 2 functions: \(f\) and \(g\)
  ➤ How is this useful?
goto: expressive example
Are these just function pointers?

A: yes, B: no

discuss with peers!
No! JavaScript functions are closures!

- Closure = function code + environment
  - Function pointers don’t keep track of environment
  - We’ll see this in more detail in a few lectures
goto: closure examples
What else can functions be used for?

• Modules! EcmaScript now has notion of modules, but most implementations still use functions

• How can we use functions to implement modules?
  ➤ Closures are good for information hiding
  ➤ Locally declared variables are scoped to the function (“module”)
  ➤ Function called with exports object which is used to expose public variables/functions
goto: module examples
Week 1

• A little bit of JavaScript history ✓

• Concepts from JavaScript ✓
  ➤ First-class functions ✓
  ➤ Objects
  ➤ Language flexibility
What are JavaScript Objects?
What are JavaScript Objects?

• Objects are maps of names (strings) to values
What are JavaScript Objects?

• Objects are maps of names (strings) to values

  ➤ E.g., object created with object literal notation:
What are JavaScript Objects?

• Objects are maps of names (strings) to values
  ➤ E.g., object created with object literal notation:
    ➤ e.g., const obj = { x: 3, y: “w00t” }
What are JavaScript Objects?

- Objects are maps of names (strings) to values
  - E.g., object created with object literal notation:
    - e.g., `const obj = { x: 3, y: “w00t” }`
  - Properties are accessed with dot or bracket notation:
What are JavaScript Objects?

- Objects are maps of names (strings) to values
  - E.g., object created with object literal notation:
    - e.g., const obj = { x: 3, y: “w00t” }
  - Properties are accessed with dot or bracket notation:
    - e.g., obj.x or obj[“x”]
What are JavaScript Objects?

- Objects are maps of names (strings) to values
  - E.g., object created with object literal notation:
    - e.g., `const obj = { x: 3, y: “w00t” }`
  - Properties are accessed with dot or bracket notation:
    - e.g., `obj.x` or `obj[“x”]`
  - Methods are function-valued properties
What are JavaScript Objects?

- Objects are maps of names (strings) to values
  - E.g., object created with object literal notation:
    - e.g., `const obj = { x: 3, y: “w00t” }`
  - Properties are accessed with dot or bracket notation:
    - e.g., `obj.x` or `obj[“x”]`
  - Methods are function-valued properties
    - e.g., `obj.f = function (y) { return this.x + y; }`
What is “this”?

• this is called the receiver
  ➤ Comes from Self (Smalltalk dialect)
  ➤ Will see more of this in objects lecture

• Intuitively: this points to the object which has the function as a method
  ➤ Really: this is bound when the function is called
goto: receiver example
I thought JavaScript had classes

- Now it does! But it didn’t always
- How did people program before?
  - Used to use functions as constructors!
What is a function constructor?

• Just a function!

➤ When you call function with new the runtime binds the this keyword to newly created object

➤ You can set properties on the receiver to populate object

➤ One property of the object is special: __proto__

➤ This is automatically set to the constructor prototype field (that’s right! functions treated as objects)
goto: class examples
Why are objects powerful?

• Useful for organizing programs

  ➤ Can hide details about the actual implementation and present clean interface that others can rely on

  ➤ I.e., they provide a way to build reliable software

• Enable reuse

  ➤ E.g., may want to add new kind of vehicle to the pipeline, can reuse lots of code that deals with assembling it

  ➤ E.g., in JavaScript an array is just an object!
Week 1

• A little bit of JavaScript history ✓

• Concepts from JavaScript ✓
  ➤ First-class functions ✓
  ➤ Objects ✓
  ➤ Language flexibility
Language flexibility

• Does not require lines end in ‘;’
  ➤ Automatic ‘;’ insertion not always what you expect

• Casts implicitly to avoid “failures”
  ➤ Useful in some case, usually source of errors (see notes)

• Hoisting
  ➤ Sometimes useful, but, variable declarations (though not definitions) are also hoisted
Language flexibility

• Evaluate string as code with eval
  ➤ Need access to full scope at point of call
  ➤ Scope depends on whether call is direct or not

• Can alter almost every object ("monkey patch")
  ➤ Even built-in objects like window and fs
  ➤ What’s the problem with this?
Language flexibility

See site notes for example of what can go wrong
Takeaways

• First-class functions are extremely powerful
  ➤ We’ll see this over and over

• Language “flexibility” is not free
  ➤ Think about features before shipping them!