JavaScript (cont)

Today

- Continue our very basic intro to JavaScript

- Lambda calculus

Last lecture recap

- JavaScript was designed in 10 days
  - Lots of unsatisfactory parts (in retrospect); many due due to the design goals (more on this today)
  - Got some things very right: **first-class functions** and **objects**
- First-class functions: what are the 3 properties?
  - declaring functions in any scope: mimic block scoping
  - funcs taking funcs as args: callbacks, filter, map

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 ∘ g)(x) = f (g(x))`
  - Here `∘` is a function that takes 2 functions: `f` and `g`
  - E.g., instead of `map(map(list, f), g)` we can do `map(list, g ∘ f)`: way faster!

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 ∘ g)(x) = f (g(x))`
  - Here `∘` is a function that takes 2 functions: `f` and `g`
  - E.g., instead of `map(map(list, f), g)` we can do `map(list, g ∘ f)`: way faster!

hof.js

Aren’t these just function pointers?

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

closure.js

What else can functions be used for?

- 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

module*.js

JavaScript intro

- A little bit of 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

receiver.js

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)

class.js

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!

Today

- A little bit of 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?

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!