JavaScript

CS565

Purdue University

January 25, 2010

Parsing

But before we get to JavaScript ...

- How is parsing going?
- Everybody on the mailing list?
- Due dates

Runtime

- JavaScript's runtime library is somewhat complicated
- Options:
  - Write from scratch
    Pro: Well-documented
    Con: A lot of functions
  - Use v8's library written in JavaScript
    Pro: Only approx. 160 functions to implement
    Con: Effectively no documentation on those functions
- Also: You can use PCRE (the Perl Compatible Regular Expressions library) instead of implementing regex

Introduction to JavaScript

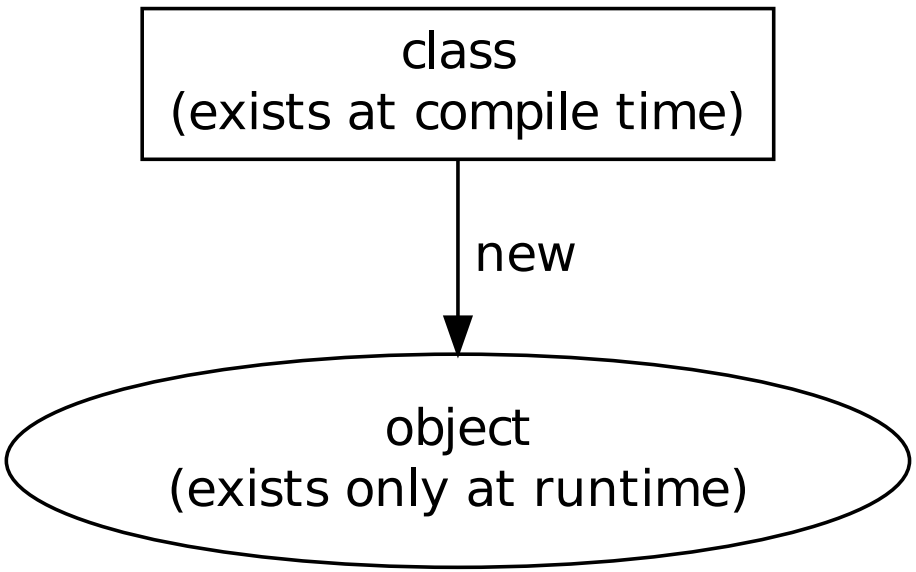
- Imperative
- Object-oriented
- Highly dynamic
- First-class functions

Objects

- Everything is an object
- There are no classes
- Prototype-based object inheritance

Objects — Prototypes

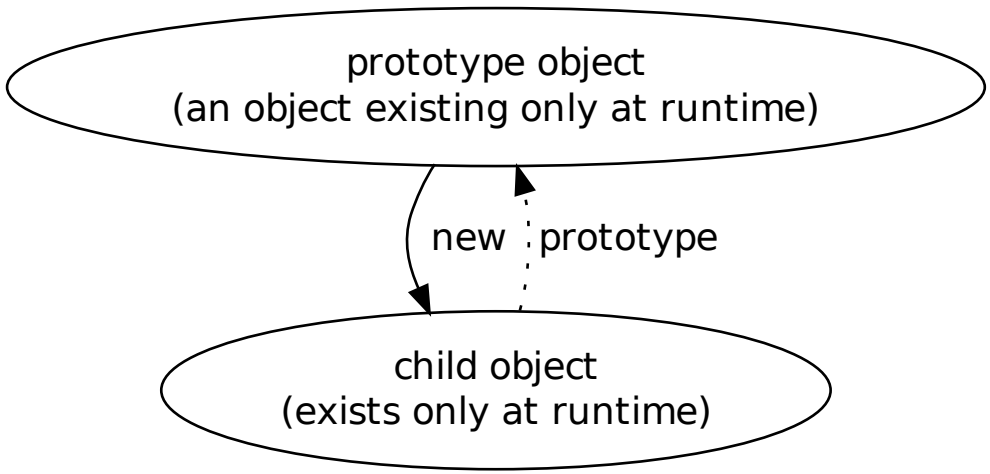
Traditional class-based system:



Since the class exists at compile time, the shape of the object is fixed

Objects — Prototypes

Prototype-based system:



Prototype is an *object*, exists at runtime, and can be changed. The shape of the child is fluid, changes when the prototype does.

Objects — Prototype demo

Demo

Objects — Constructors

- Objects created by constructors
- Constructor is just a function
- Special poorly-named **prototype** property

Objects — Constructor name madness

- `Foo.prototype` is NOT the prototype of `Foo`!
- `Foo.prototype` is the prototype of objects created by the *constructor* `Foo`
- `Foo`'s prototype is probably `Function.prototype`

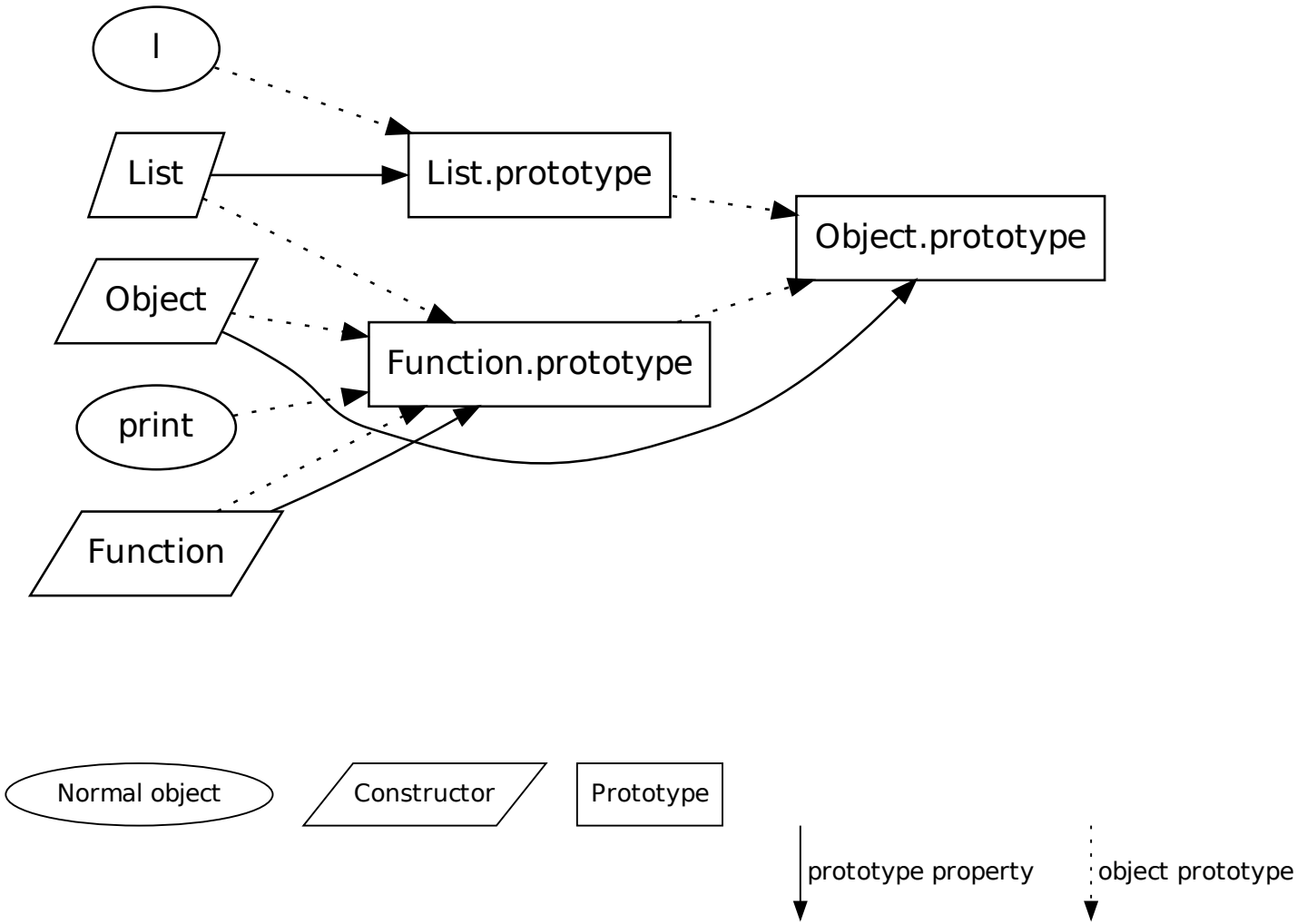
Objects — Constructors

- Objects created by constructors
- Constructor is just a function
- Special poorly-named **prototype** property
- Constructor function called with keyword `this` referring to a new object

```
function List(val, next) {
    this.val = val;
    this.next = next;
}
var l = new List(1, null);
```

Objects — Putting it together

Prepare yourself for a truly insane graph...



Objects — Prototype consequences

Demo

Functions — First glance

```
function sum(a, b) {
    return a + b;
}

var x = sum(1, 2);
assert(x == 3);
```

Functions — Methods (or are they member functions?)

```
function List(val, next) {
    this.val = val;
    this.next = next;
}

List.prototype.map = function (f) {
    return new List(f(this.val),
                    (this.next == null) ?
                    null : next.map(f));
}

function plus1(a) {
    return a + 1;
}

var l = new List(1, new List(2, null));
l.map(plus1);
```

Functions — How many arguments has a function?

```
function sum(a, b, c) {
    return a + b;
    // didn't actually use c ...
}
var x = sum(1, 2); // lol, works
```

Undefined

A brief diversion ...

In most languages, this crashes:

```
var x = Object.notAMemberOfObject;
```

In JavaScript...
(Demo)

Undefined — This section undefined

- In JavaScript, undefined is a *type*!
- And `undefined` is a value![1]

```
assert(typeof(Object.notAMemberOfObject) == "undefined");
assert(Object.notAMemberOfObject == undefined);
```

[1] Not on IE

Functions — How many arguments has a function? (Really now)

```
function sum(a, b, c) {
    print(typeof(c));
    return a + b;
}
var x = sum(1, 2); // x == 3, prints "undefined"
```

Functions — No, seriously, how many arguments?

```
function sum() {
    return arguments[0] + arguments[1];
}
var x = sum(1, 2);
assert(x == 3);
```

Functions — Arguments redux

In short:

- Named arguments are just a convenience
- Arguments will be assigned to names if available
- But the truth is only in the `arguments` array

Runtime library

JavaScript provides a runtime library:

- Global object (provides global variables and functions)
- Object
- Function
- Array
- String
- Boolean
- Number
- Date
- RegExp
- Math
- Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError

Runtime library — Array

- concat (concatenate multiple arrays)
- join (joins elements of an array by a separator)
- push, pop (for use as a stack)
- shift (for use as a queue)
- indexOf, lastIndexOf (searching)
- every, some, forEach, map, filter, reduce, reduceRight (iteration)
- reverse
- slice
- sort
- splice

Runtime library — Undefined strikes again!

```
var x = [0, 1, , 3];
assert(typeof(x[2]) == "undefined");
```

Runtime library — Strings, Booleans, Numbers

Strings, numbers are objects. But kind of goofy:

```
var x = 3;
x.foo = 4; // illegal
Number.prototype.foo = 4; // OK
assert(x.foo == 4);
```

Allows unboxed implementation

Runtime library — RegExp

You all like regular expressions, right?!

```
var x = "Hello, world!";
var y = x.replace(/Hello/, "Goodbye")
         .replace(/, /, ", cruel ");
print(y);
```

Thursday

Next class:

- Prototypes: The nasty, gritty implementation details!
- Objects: Internal cruft!
- Exceptions, dynamic evaluation, and other shocking details