software
studio

functionals

Daniel Jackson

All rights reserved. Daniel Jackson, 2015
purposes of today’s class

after today’s class, you should
understand closures fully
know what functionals are & why they’re useful
be familiar with standard list functionals
know the 6170 idiom for data abstraction in JS
last time
functions as values
variable scoping & closures

functions as ‘first-class values’
passed in as arguments
passed back as results
functions as arguments
ideas to look out for

roll your own loop!
with recursion & functionals

passing the body
unlike Java-style iterators

looping patterns
a more abstract, structured loop

here, showed examples from JavaScript live slides, section Functions/Functions as Arguments
why looping functionals?

iteration abstractions
capture essential pattern
hide implementation details

look ma, no indexes!
so no indexing errors to make

a more succinct pattern
easier to read & write (for experts, at least)

can write specialized loops
eg, ‘for each neighbor’
comparisons

var sum1 = function (a) {
    var s = 0;
    a.forEach(function (e) {s += e;});
    return s;
}

var sum2 = function (a) {
    var s = 0;
    for (var i = 0; i < a.length; i++)
        s += a[i];
    return s;
}

var sum3 = function (a) {
    [lbl] start:
    var s = 0; var i = 0;
    L1: if (i > a.length) goto end;
    s += a[i]; i++;
    [lbl] end:
    return s;
}

‘I don't condone the use of functions, but if you are stuck with bad code, GOTO will still work’
http://summerofgoto.com/
define a loop abstraction
that takes a function \( f: \rightarrow \text{Bool} \)
and executes it until it returns true

```javascript
var until = function (f) {
    if (!(f())) until(f);
};

var greet = function () {
    var name;
    until (function () {
        name = prompt('Your name?');
        return name;
    })
    alert ('Hello, ' + name);
};
```
the classic list functionals
ideas to look out for

**list functionals**
iteration abstractions for lists (arrays in JS)
common patterns of list processing
widely used in functional languages
basic of MapReduce too

here, showed examples from
JavaScript live slides, section
*Functions/List Functionals*
exercise: sum with list functional

```javascript
sum = function (a) {
    var plus = function (x, y) { return x + y; };
    return reduce (a, plus, 0)
}
sum([1,2,3,4]); // evaluates to 10
```
functions as results
ideas to look out for

real power
from function ‘factories’

here, showed examples from JavaScript live slides, section *Functions/Functions as Results*
data abstraction with closures
ideas to look out for

abstract object as JS object
but the fields are methods!

strange the first time
but easy once you see it

polishing
with prototypes & freezing

here, showed examples from
JavaScript live slides, section
Functions/Data Abstraction
why not new & this?

why new & this?
faking Java in JavaScript
cute idea, but...

if you forget new
bad & strange things happen

this isn’t what you think
_this_ is not bound normally
hence the idiom _this_ = _that_

here, showed examples from
JavaScript live slides, section
*Bad Parts/Prototypes*
quiz redux
Great designers quiz

Who is Apple's chief designer?
- Tim Cook
- Jonathan Ive [Correct]
- Steve Jobs

What company did Dieter Rams design for?
- Alessi
- Braun [Correct]
- Sony
- Sony

What does Frank Gehry design?
- Buildings [Correct]
- Computers
- Typefaces
<!doctype html>
<html>
<head>
  <title>Quiz</title>
  <link rel="stylesheet" href="quiz.css" />
  <script src="jquery-2.1.4.min.js"></script>
  <script src="quiz.js"></script>
  <script src="quiz-designers.js"></script>
  <script src="quiz-display.js"></script>
  <script>
    // add quiz to DOM
    $(function () {displayQuiz(designerQuiz, $('body'))});
  </script>
</head>
<body>
</body>
</html>
var Quiz = function (title) {
    var that = Object.create(Quiz.prototype);
    var questions = [];
    that.add = function (stem, options, answer) {
        questions.push({stem: stem, options: options, answer: answer});
    }
    that.each = function (f) {
        questions.forEach(f);
    }
    that.grade = function (responses) {
        return questions.reduce(
            function (score, question, index) {
                return score + (responses[index] == question.answer ? 1 : 0);
            }, 0)
        }
    that.total = function () {
        return questions.length;
    }
    that.title = function () {return title;}
    Object.freeze(that);
    return that;
};
var displayQuiz = function (quiz, container) {
    var titleElt = $("<h1>", {class: "title", text: quiz.title()});
    container.append(titleElt);
    var responseFuncs = [];
    quiz.each(function (question, qindex) {
        var questionElt = $("<div>", {class: "question"});
        var stemElt = $("<p>", {class: "stem", text: question.stem});
        questionElt.append(stemElt);
        question.options.forEach(function (option, oindex) {
            var radioElt = $("<input>", {type: "radio", name: qindex, val: oindex});
            var optionElt = $("<label>", {text: option});
            optionElt.prepend(radioElt);
            questionElt.append(optionElt);
        });
        responseFuncs.push(function () {
            return questionElt.find(".radio:checked").val();
        });
        container.append(questionElt);
    })
    var button = $("<button>", {text: "submit"});
    container.append(button);
    button.click(function () {
        var responses = responseFuncs.map(function (f) {return f();});
        var score = quiz.grade(responses);
        alert("Congratulations! You scored " + score + " out of " + quiz.total());
    })
}
summary
the power of functions

raising level
eg, no array indexing

capturing patterns
eg, list functionals

controlling namespace
eg, hiding rep in datatype