Final Review

CS 144 Web Application

TA: Jin Wang and Zhehan Li

03/16/2018

Outline

- General Information
- JavaScript
- MEAN stack
- Scalability
- Other topics

General Information

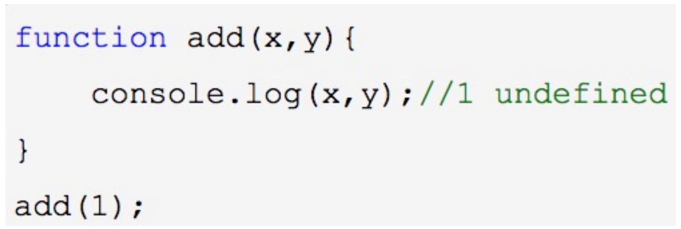
- Final Exam
  - Time: 11:30 a.m. – 2:30 p.m. March 19th
  - Location: BH 3400
- Rules
  - Similar format with sample final
  - Close book, close notes
  - Allow bring two double-sided cheat sheets
- Programming languages (HTML, CSS, JavaScript etc.)
  - Know basic ideas and simple structured program
  - No need to recite the detailed concepts
- Scope: contents through week 1 to 9 in lectures
- Only the **handouts provided by the instructor** are official materials!

JavaScript: basic issues

- Keywords and syntax
  - Case sensitive
- Variables and identifiers
- Primitive Types
  - number, string, boolean
  - undefined and null
- Object Type
  - Properties
  - Object assignment
- Array

JavaScript: Function and Scope

- Functions are objects
  - Can be assigned to a variable/passed as a parameter
  - Can have properties
- Arrow Function
  - Do not have its own this
  - Cannot serve as constructor in OOP
- Parameter
  - Do not check the type and number of parameters
  - When there are fewer number of arguments: initialized as undefined



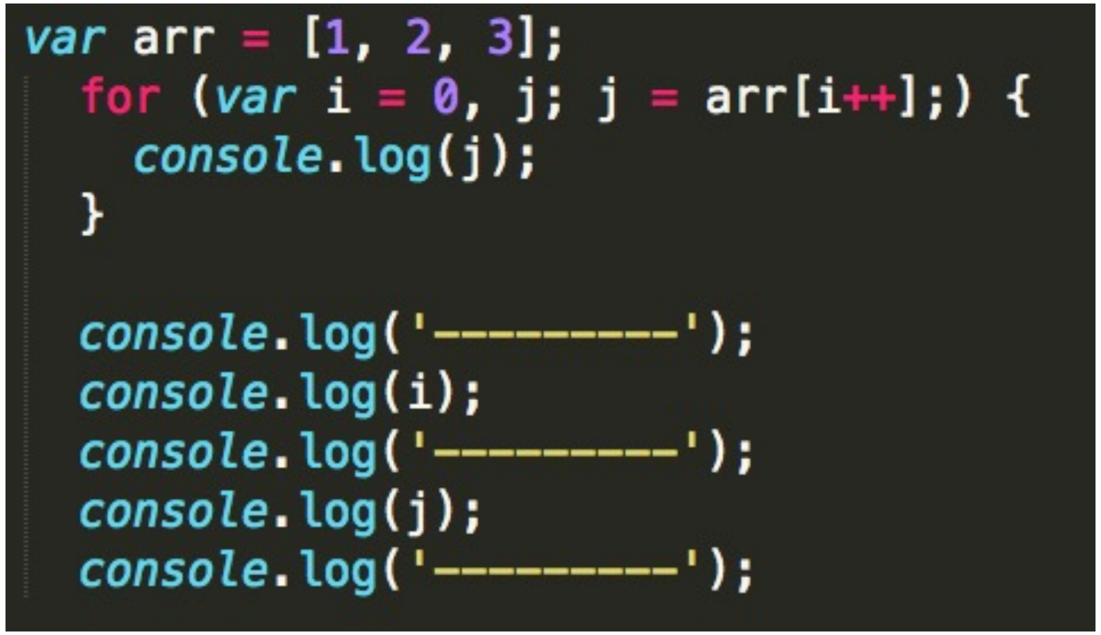
```
function add(x,y){
    console.log(x,y);//1 undefined
}
add(1);
```

JavaScript: Function and Scope

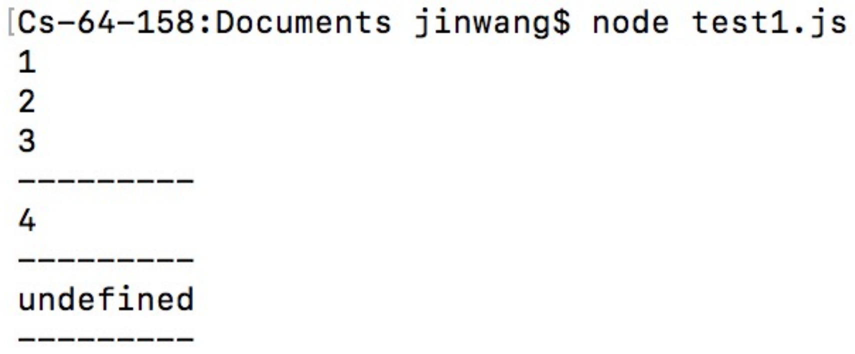
- Nested Function
  - Closure: the nested function will return its context as well.
- Global parameter
  - Declared out of a function, whether use let or not.
  - Declared inside a function, not use let. [not recommended!]
- Local parameter
  - Declared inside a function, use let

JavaScript: Example 1

- What is the output of following program?



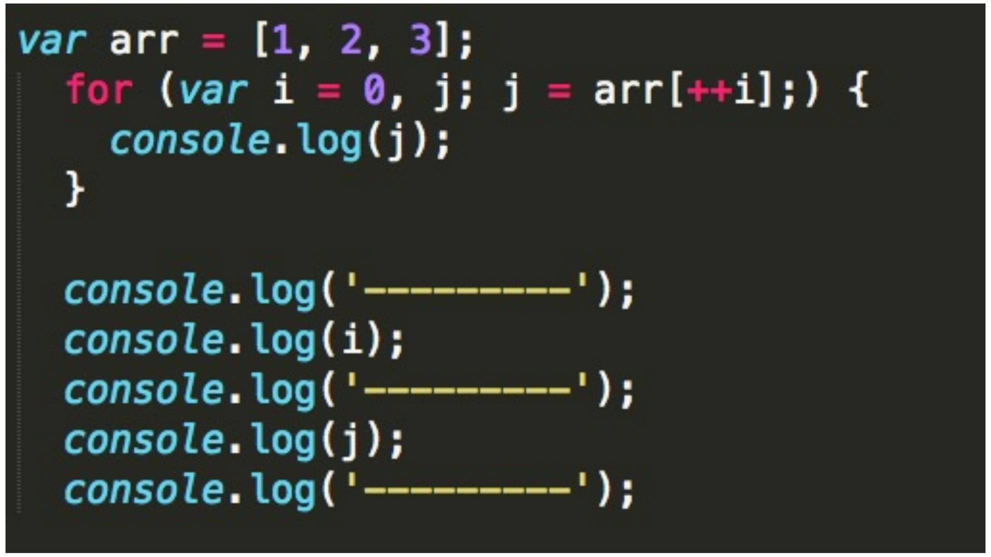
```
var arr = [1, 2, 3];
  for (var i = 0, j; j = arr[i++];) {
    console.log(j);
  }

  console.log('---------');
  console.log(i);
  console.log('---------');
  console.log(j);
  console.log('---------');
```



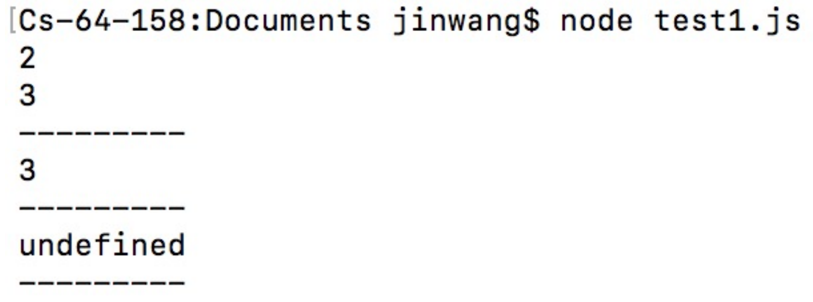
```
[Cs-64-158:Documents jinwang$ node test1.js
1
2
3
---------
4
---------
undefined
---------
```

JavaScript: Example 1 Follow-up

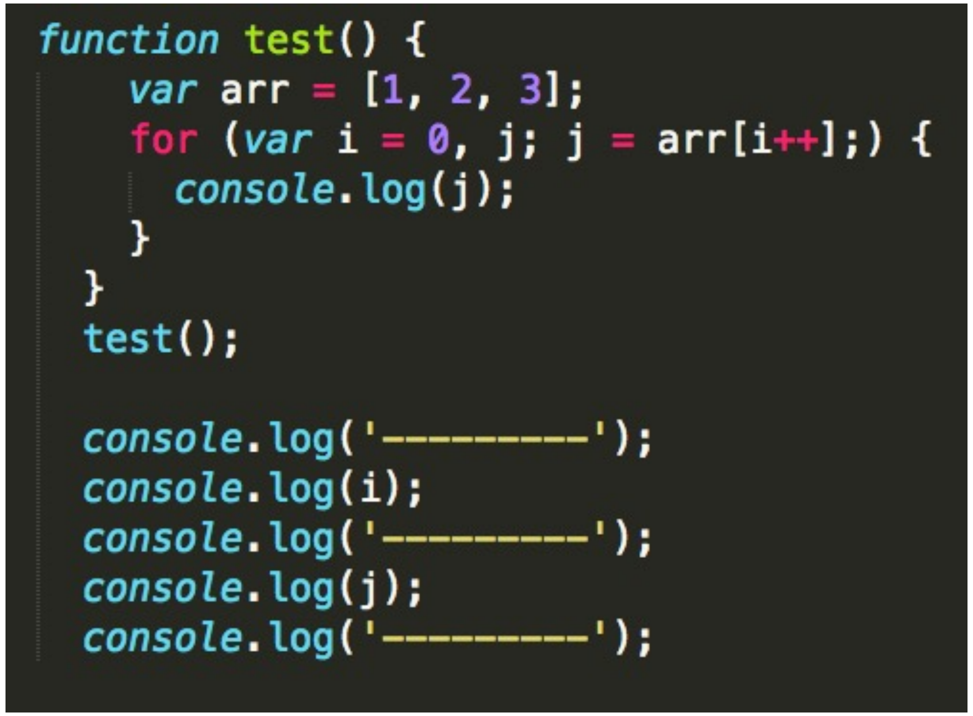
- What about these?



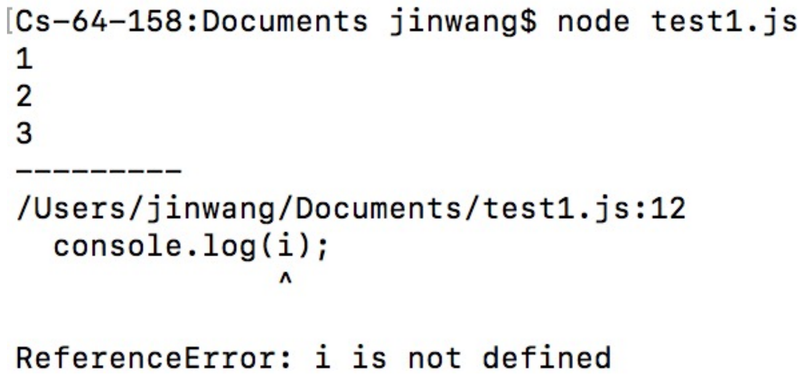
```
var arr = [1, 2, 3];
  for (var i = 0, j; j = arr[++i];) {
    console.log(j);
  }

  console.log('---------');
  console.log(i);
  console.log('---------');
  console.log(j);
  console.log('---------');
```



```
Cs-64-158:Documents jinwang$ node test1.js
2
3
---------
3
---------
undefined
---------
```



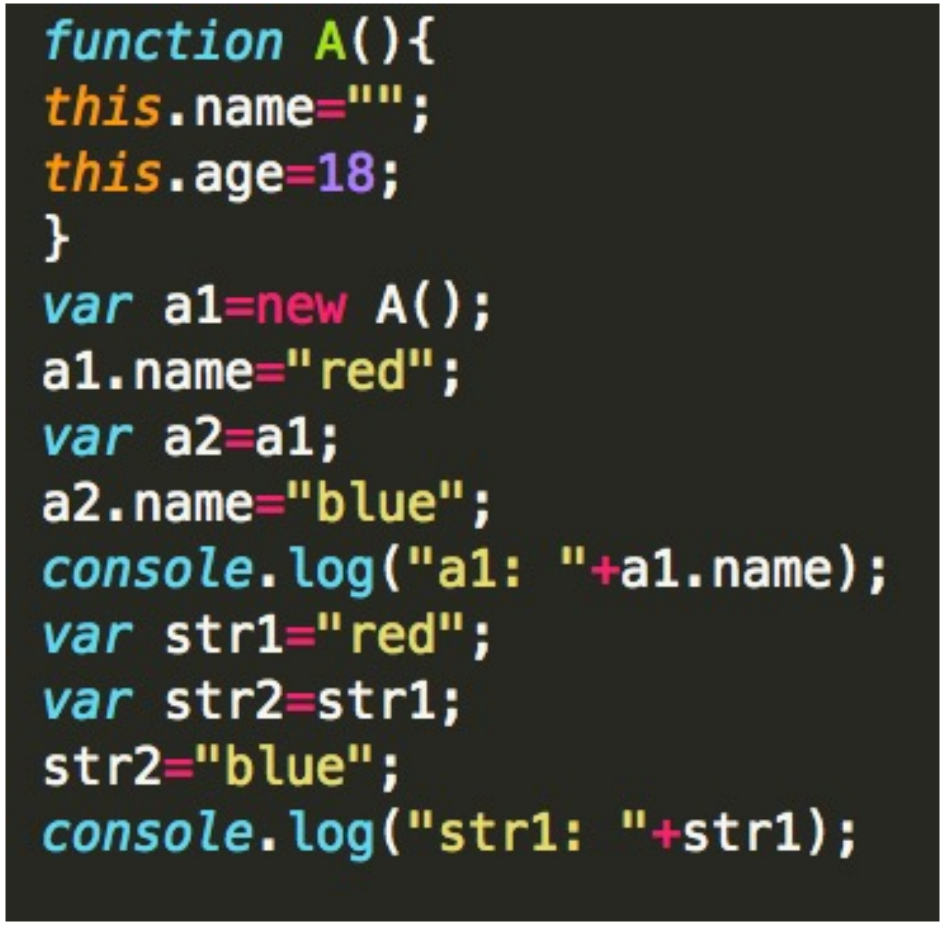
```
function test() {
    var arr = [1, 2, 3];
    for (var i = 0, j; j = arr[i++];) {
      console.log(j);
    }
  }
  test();

  console.log('---------');
  console.log(i);
  console.log('---------');
  console.log(j);
  console.log('---------');
```



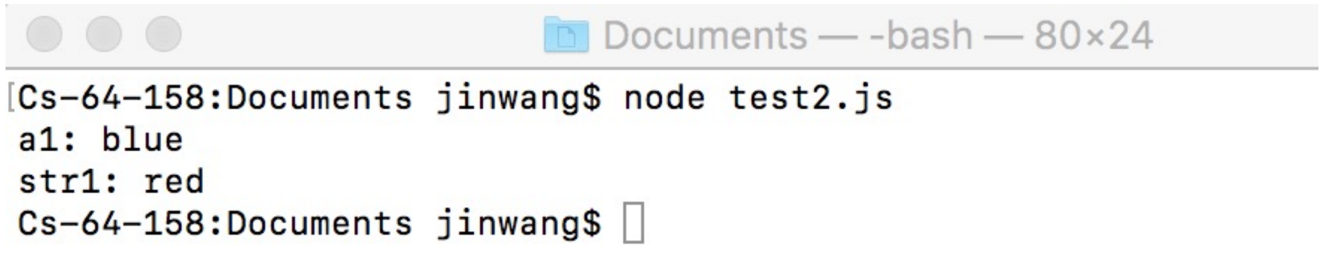
```
Cs-64-158:Documents jinwang$ node test1.js
1
2
3
---------
/Users/jinwang/Documents/test1.js:12
  console.log(i);
              ^

ReferenceError: i is not defined
```

JavaScript: Example 2

- What is the output of following program?

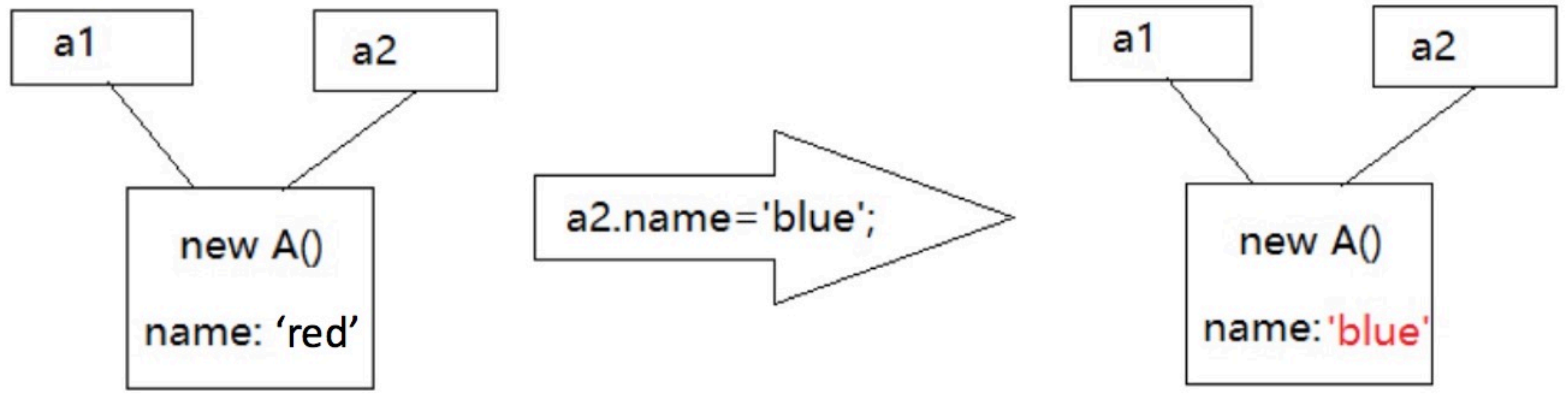


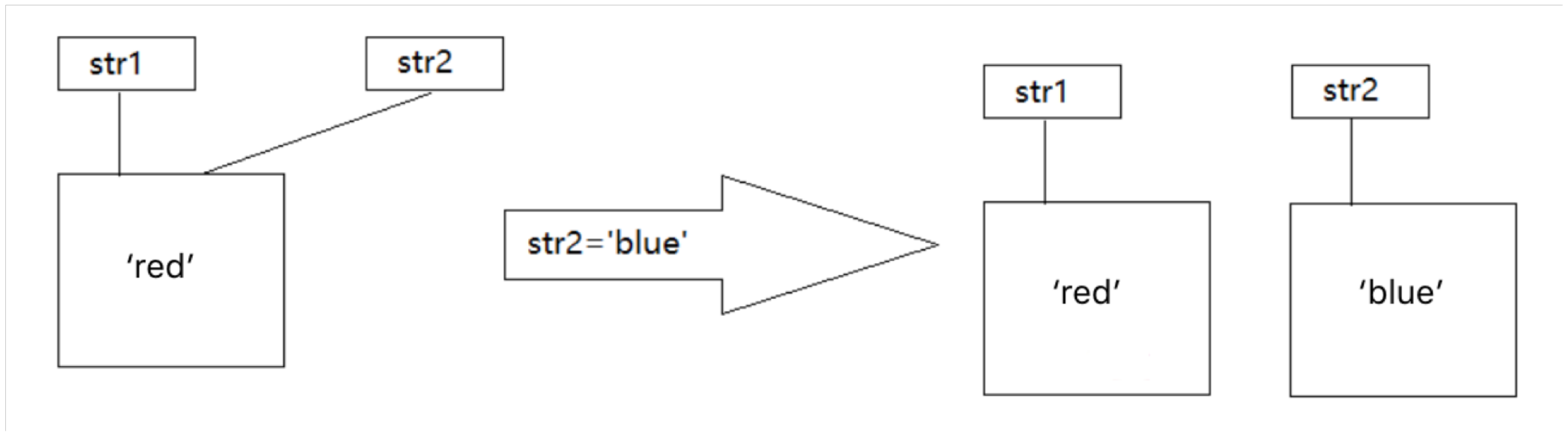
```
function A(){
this.name="";
this.age=18;
}
var a1=new A();
a1.name="red";
var a2=a1;
a2.name="blue";
console.log("a1: "+a1.name);
var str1="red";
var str2=str1;
str2="blue";
console.log("str1: "+str1);
```



```
Cs-64-158:Documents jinwang$ node test2.js
a1: blue
str1: red
Cs-64-158:Documents jinwang$ 
```

Explanation of Example 2





MEAN Stack

- Traditional Web Development
  - HTML + CSS + JavaScript -> Front End
  - Java -> Back End
  - MySQL -> Database
- Can we just learn JavaScript and get ride of Java?
  - Yes. Node.js
- Can we use JSON as the data model for storage?
  - Yes. MongoDB
- Can we reduce the complexity of Front End/Client side?
  - Yes. Angular

TypeScript

- Superset of JavaScript
  - any JavaScript code is also a TypeScript code!
  - Type/Class(public, private, protected)/Interface/Generics
  - Decorators -> ‘@xxx’
  - TS code (Type mismatch error)
    - function printName(name: string): string {return "You are" + name;}
    - let num : number = 1
    - console.log(printName(num))
  - JS code (Success)
    - function printName(name) { return "You are" + name; }
    - var num = 1;
    - console.log(printName(num));

Angular

- JavaScript/TypeScript framework for client-side development
- Supporting Single-page application (SPA) -> Project 3
- Module = Components + Services
- Intra-component communication: Data binding
  - Interpolation ({{ expression }})
  - Property binding ([property]="expression")
  - Two-way binding ([(ngModel)]="property")
  - Event binding ((event)="statement")
- Inter-component communication
  - Custom event generation + template reference variable
  - Service and Dependency Injection -> Who is responsible for creating service?

Node.js

- JavaScript runtime environment – Server Side!
- Single threaded & Asynchronous programming
- Comparison among different projects

	Project 2	Project 3,4
Backend Language	Java	JavaScript
Runtime Environment	JRE	Node.js
Package/Framework	Tomcat	Express
MVC	MySQL-JSP-Java class	MongoDB-EJS-JavaScript file

Express

- require ('xxxxx') -- VS -- import {xxxxx} from 'xxxxx’
- package.json
- Routing Support
  - E.g. app.get(‘/list/:username’, (req , res) = > {do()});
  - Specify the method (GET), path (‘/list/:username’), and action(do())
- Middleware
  - Every request goes through the sequence of middleware, until one serves the response [e.g. app.use(bodyParser.json ());]
  - Create a separate module to handle requests coming from some URL [e.g. birds = require (‘./ birds’); app.use('/ birds ', birds);]

MongoDB

- NoSQL Database, Document-oriented Database

-

MongoDB	MySQL
Database	Database
Collection	Table
Document	Tuple
Field	Attribute

- Allows data retrieval based on “non-key” fields

Scalability

- Load Balance, Scale out
- Distributed File System
- NoSQL
  - maintaining a shared state
  - CAP Theorem
  - Different kinds of stores
  - Consistent hashing
- Map Reduce and Hadoop (Refer to my slides in week 8)

Other Topics

- Communication: HTTP
  - Cookies, Sessions
  - JSON
- Web Server
  - Architecture
  - App server: MVC
- Encoding
  - MIME
  - Unicode
- AJAX

Principles to Answer Questions

- Understand the requirement
- Locate to the corresponding knowledge
- Solve the problem with such knowledge, add necessary explanation
  - You may get partial credits once your answer is not fully correct.
  - Do highlight the important part in your answer.
- Make a clear statement
  - **Do not be optimistic!**

Example: Problem 7 in Sample Final

Voice-over-IP (VoIP) allows telephone calls to be carried over regular IP networks. Currently, most corporate VoIP installations use VoIP within an office building (or set of buildings) to connect telephones, but do not use the Internet to carry calls outside the company. For example, suppose that VoIP phones are used inside Boelter Hall, but all calls leaving the building are carried on an ordinary phone line. To bridge these two disparate network (Boelter Hall VoIP network and the ordinary phone line), a dedicated server "transforms" all VoIP calls placed from Boelter Hall into regular phone signals before the calls leave Boelter Hall network.

1. In organizations like UCLA, a person making a long distance call has to pay for the call in some way. Explain what requirements this places on the VoIP protocol. Choose from among these terms in writing your answer: confidentiality, authentication, integrity, authorization. Briefly explain how your choice(s) is/are relevant in this context.

General Grading Rubrics

- Problem with specific answer
  - No partial credits will be considered.
    E.g. Problem 5 in sample final.
- Application Development Problem
  - Partial credits will be assigned for **incomplete but correct** answers
- Question Answering
  - Correct conclusion: full credits
  - Partial correct/incorrect conclusion: partial credits for your effort
  - But if you include incorrect common sense in your answers, you will not have corresponding partial credits.

Thank you!

XML

- Structure of XML
- Namespace
- DTD
- XPath

Structure of XML: Tag and Element

- **Tag**: label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly nested
  - Proper nesting
    - <account> … <balance> …. </balance> </account>
  - Formally: every start tag must have a unique matching end tag, in the context of the same parent element.
- Every document must have a single top-level element (root)
- Tree based model: DOM

Structure of XML :Attributes

- Elements can have **attributes**
  - <account acct-type = “checking” >
    <account-number> A-102 </account-number>
    <branch-name> Perryridge </branch-name>
    <balance> 400 </balance>
- Attributes are specified by *name=value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once
  - <account acct-type = “checking” monthly-fee=“5”>

Namespaces

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name
- Avoid using long unique names all over document by using XML Namespaces

```
<bank Xmlns:FB= ‘http://www.FirstBank.com’ >
     …
     <FB:branch>
          <FB:branchname>Downtown</FB:branchname>
           <FB:branchcity> Brooklyn</FB:branchcity>
     </FB:branch>
     …
</bank>
```

Document Type Definition (DTD)

- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
- DTD does not constrain data types
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST   element (attributes) >
- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)

XPath

- XPath is used to address (select) parts of documents using **path expressions**
- A path expression is a sequence of steps separated by “/”
  - Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
- The initial “/” denotes root of the document (above the top-level tag)

XPath (Cont.)

- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in **[ ]**
  - E.g. /bank-2/account[balance > 400]
    - returns account elements with a balance value greater than 400
    - /bank-2/account[balance] returns account elements containing a balance subelement
- Attributes are accessed using "@"
  - E.g. /bank-2/account[balance > 400]/@account-number
    - returns the account numbers of those accounts with balance > 400
  - IDREF attributes are not dereferenced automatically (more on this later)

2017 Final: Problem 1

In the space provided below, draw the HTML DOM tree of the following HTML document according to the W3c standard:

```
<!DOCTYPE html>
<html>
<head><title>CS144 Problem 1</title></head>
<body>This problem is <i>easy!</i></body>
</html>
```

In your answer, make sure to indicate the type, name, and value of each node of your tree like the following attribute node:

Attribute
Name: href
Value: http://cs144.edu

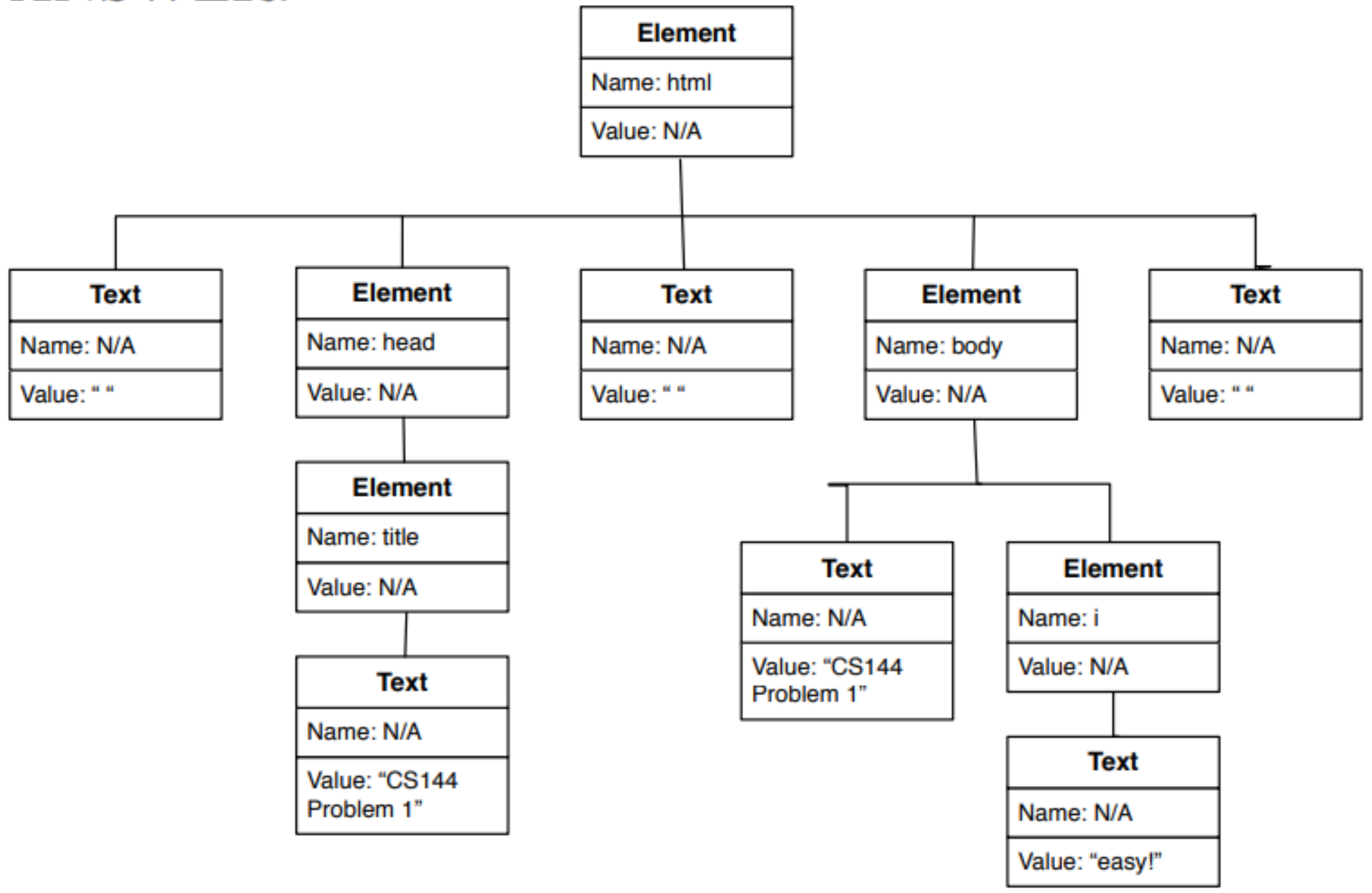
If a particular node does not have a type, name, or value associated with it, write down "N/A" for the relevant field.

2017 Final: Problem 1

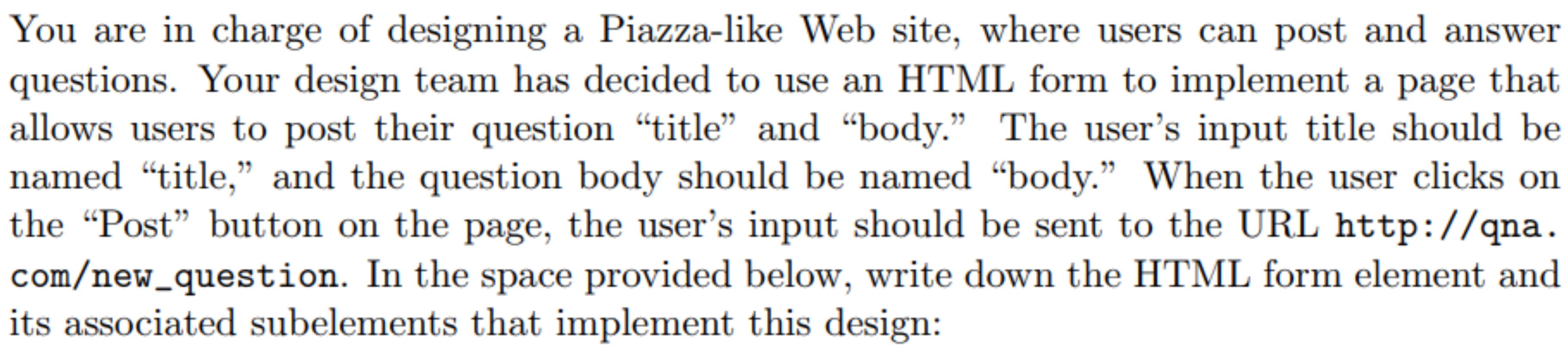
- DOM Tree
  - An HTML element on a page becomes an element node
  - Text inside an HTML element becomes a text node
    - As a child of the element node
  - An attribute of an HTML element becomes an attribute node
    - associated with the element node, but is not a child node
  - nodeType: element, attribute, text, comment . . .
  - nodeName:
    - Element and attribute nodes: tag and attribute names
    - Text nodes: text
  - nodeValue:
    - Text and comment nodes: inside text
    - Attribute nodes: attribute value
    - null otherwise

2017 Final: Problem 1

ANSWER:



2017 Final: Problem 2



You are in charge of designing a Piazza-like Web site, where users can post and answer questions. Your design team has decided to use an HTML form to implement a page that allows users to post their question "title" and "body." The user's input title should be named "title," and the question body should be named "body." When the user clicks on the "Post" button on the page, the user's input should be sent to the URL `http://qna.com/new_question`. In the space provided below, write down the HTML form element and its associated subelements that implement this design:

- HTML Forms
  - Action
  - Method
  - <input type="" value="" name="" />

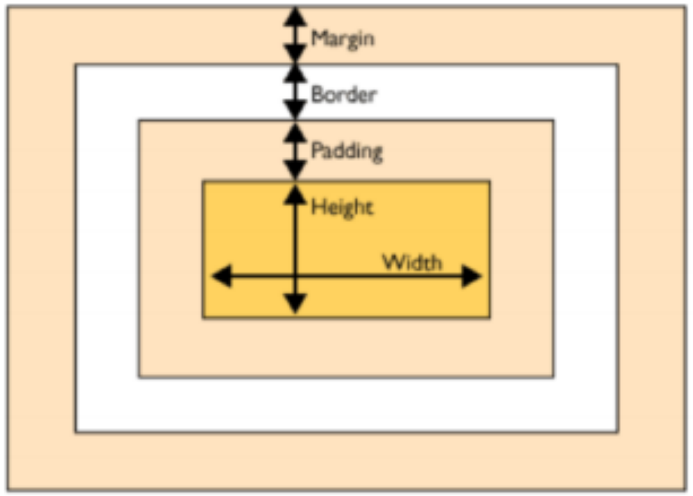
2017 Final: Problem 3

```
<!DOCTYPE html>
<html>
<head>
<style>
body { padding: 30px; height: 100px; }
.t1  { margin: 10px; height: 50px; }
.t2  { margin: 20px; }
#b1  { padding: 40px; }
</style>
<script>
function SetClass() {
   document.getElementById("b1").className = "t2";
}
</script>
</head>
<body class="t1" id="b1" onload="SetClass();">
This is the Problem 3 of CS144 Final Exam. Hopefully this is not too difficult.
</body>
</html>
```

The above HTML page is designed for devices with the screen size 1080px X 1920px. On such devices, what will be the width and height of the body text, once the page is fully loaded? Briefly explain your answer.

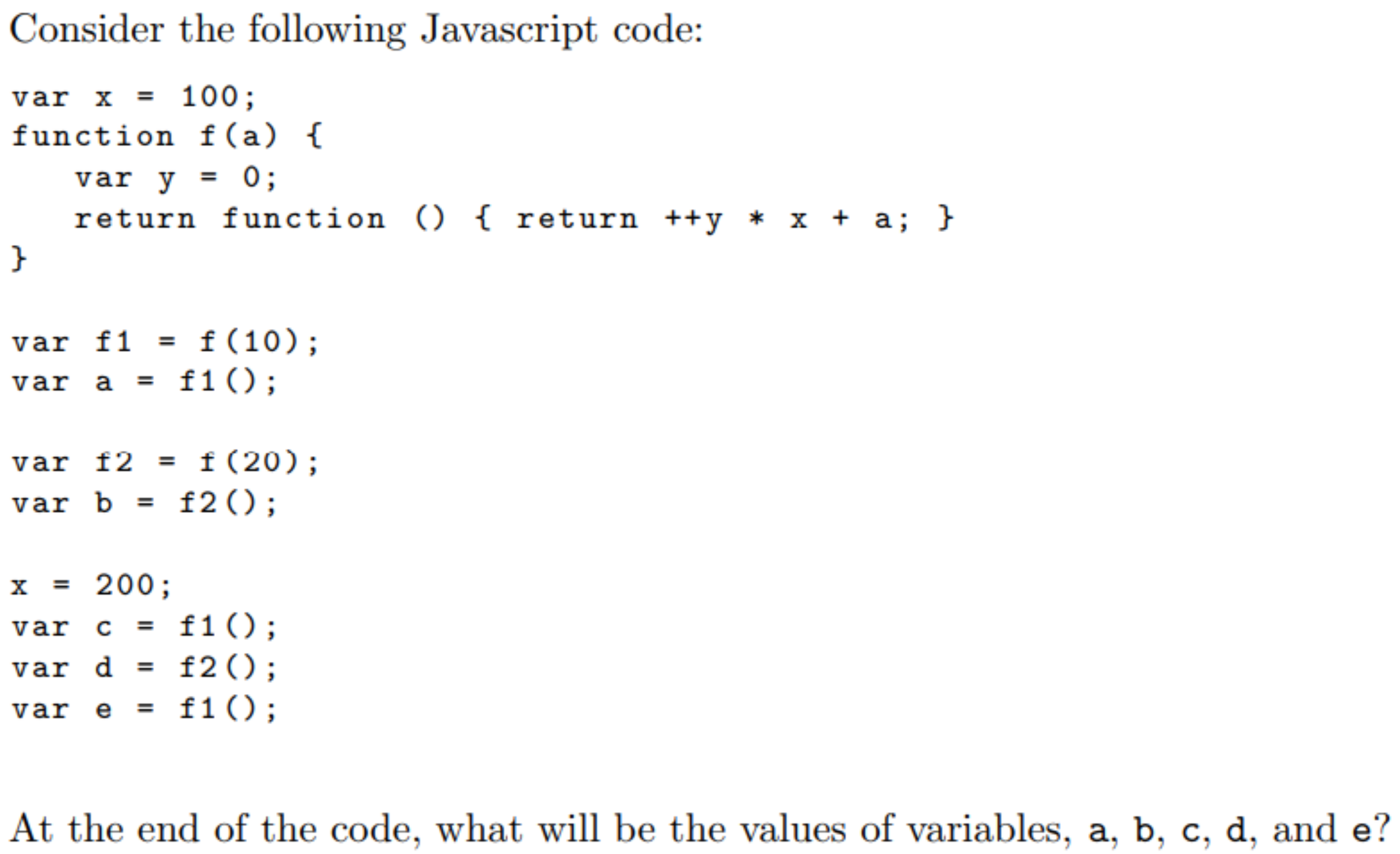
2017 Final: Problem 3

CSS box model



Once the page is fully loaded, the body has class "t2" and id "b1". Therefore, the body's padding is 40px, margin is 20px and height is 100px, which means that its width is 1080 – (40 $\times$ 2 + 20 $\times$ 2) = 960px and its height is 100px.

2017 Final: Problem 5



Consider the following Javascript code:

```
var x = 100;
function f(a) {
   var y = 0;
   return function () { return ++y * x + a; }
}

var f1 = f(10);
var a = f1();

var f2 = f(20);
var b = f2();

x = 200;
var c = f1();
var d = f2();
var e = f1();
```

At the end of the code, what will be the values of variables, **a**, **b**, **c**, **d**, and **e**?

a: 110, b: 120, c: 410, d: 420, e: 610

2017 Final: Problem 6

You are designing a database for online shopping Web site that carries 500,000 different products. The products are classified into one of three product categories, books, movies, and CDs. There are 300,000 books, 150,000 movies and 50,000 CDs. When users browse the Web site, they first select a particular product category that they are interested in and search for and look at products only within the selected category.

Each product is stored as a "tuple" in our database, and the size of each tuple is 1KB on average. A database installation on a single machine can store 1TB worth of tuples and can support up to 200 read (or write) requests per second.

At the peak of our site, 3,000 users concurrently access out Web site. Each user's activity on our Web site generates 1 database request every 10 seconds (the request can be either read or write). Among the 3,000 users at the peak, 60% are browsing the "books" category, 25% are browsing the "movies" category and 15% are browsing the "CDs" category. The database requests generated by the user activities are 60% reads and 40% writes within each category on average.

2017 Final: Problem 6

1. What is the minimum number of machine(s) you will need to support our database workload?

2. What tuples will you store on which the machine(s)? How will you "split" and "replicate" our data among the machine(s)? If needed, you may refer to the "book" tuples as tuples b_1 through $b_{300,000}$, "movie" tuples as tuples m_1 through $m_{150,000}$, and "CD" tuples as c_1 through $c_{50,000}$.