This is a list of topics that we have covered since midterm exam 1. This is not all inclusive of every detail and there may be items on the exam that are not explicitly listed here, but these are the primary topics of interest.

For specific examples of the various JavaScript constructs described here, refer to the lecture notes, the labs, and the homework.

Computer Basics

Layers of abstraction are used to describe the design of computer software and hardware systems. The fundamental idea is that describing the characteristics of the objects in a layer lets you understand the operation of that layer without having to know exactly how it accomplishes the stated functions. This is a very powerful concept since it means that you can do useful work while only understanding a portion of the whole system. Also, it means that a layer can be re-implemented using a different technology without affecting any of the layers above it.

Architecture and Organization are two terms that are used in order to distinguish between the definition of a layer and the actual implementation of the layer. There is only one layer per definition, since the definition is the only thing that actually exists for an abstract layer. However, there may be many different implementations of that definition. Thus, there are numerous implementations of the PowerPC Instruction Set Architecture. The different implementations provided different mixes of characteristics. Some are fast but costly. Some are slow but inexpensive. Some run hot, others are relatively cool. Etc.

At a high level, computers can be thought of as having a Processor, Memory, and a set of Input/Output devices. Most of the electronics in a computer (the processor, the memory, and the control circuitry for I/O) is implemented with transistors in Integrated Circuits. The rapid and continuing increase in the number of transistors that can be packed into an IC is a major reason for the growth of computer capability over the last few decades.

The processor implements a fetch and execute cycle. It fetches an instruction from memory, decodes the meaning of the instruction, gets the data it needs for the instruction, executes the requested instruction, and stores the result back in the destination address. All processors operate this way, although they differ significantly in the details of how they do it.

Memory is the storage space where instructions and data are stored. On most modern systems, memory can be addressed in byte-sized chunks (8-bits) and the size of the memory is fairly large (500 MegaBytes to several GigaBytes).

Input / Output devices move data values from the computer to the outside world and vice-versa. IO devices vary widely in terms of speed, ranging from the slow (keyboard, clicker receiver) to the fast (disk drive, network interface).

Algorithms

An algorithm is a precise, systematic method to produce a desired result. For an algorithm to be well specified it must have:

- Inputs specified — The range of possible inputs is well defined
- Outputs specified — The desired output is well defined
- Definiteness — The steps to take are definite and understandable
- Effectiveness — The steps must be possible to accomplish
- Finiteness — A processor that follows the algorithm will eventually finish

JavaScript Intro

JavaScript is one of many "scripting" languages used to add dynamic capabilities to web pages. A basic HTML page is static, and it is rendered the same way each time it is displayed. Scripting languages let us create and update pages "on the fly" to respond to dynamic conditions and user inputs.

The program code for a script on a web page is identified using `<script>` and `</script>` tags. These tags can appear in the `<head>` or the `<body>` of a web page. Between the tags the program is specified as a series of statements. In general, each statement ends with a semi-colon "`;`". When the browser is reading the page to display, it also reads the embedded code. Depending on how the code is written, it is either executed immediately as it is read in, or it is executed later in response to user commands.

A variable has a name and a value. The name of a variable starts with a letter, followed by letters, numbers, or the underscore character "_". A common convention for naming variables is that they start with a lower case letter and if the name comprises several words they are all run together with the second and following words capitalized. Examples are `index`, `riverCount`, `dotWidth`.

Variables are containers for values. Common values are numbers (0, 1, 3.14159, -7), strings ("hi", "car", "Hello there"), and boolean (`true` or `false`).

Expressions are combinations of variables and literal values using appropriate operators. Note that a single value (like 5 or "texas") is a very simple expression in its own right. Similarly, a single variable name can be the entire definition of an expression, if that is appropriate. Expressions are formulas that say how to manipulate existing values to calculate a new value. The math operators that we have discussed are `+, -, *, /`. The `+` operator is also used to concatenate string values to make a new string. The relational operators are `>, <, >=, <=, ==,` and `!=`. Be sure to recognize the use of the double equals "`==`" for equality checking. The boolean operators are "`&&`" (and), "`||`" (or), and "`!`" (not). The boolean operators are used to combine boolean (`true` or `false`) operands to calculate a new boolean value.

An assignment statement takes the value of the expression on the right-hand side of a single equals sign and assigns that value to the variable named on the left-hand side of the equals sign.

Thus `sum=2+2;` assigns the value `4` to the variable `sum`. Assignment statements can be used with numbers, strings, and boolean operands (among others), as well as functions that return values of these types.

Functions

A function is a way to bundle a set of instructions and give them a name so that you can reuse them easily. Functions have a specific layout within a `<script>` block.

```
function name(parameter list){
  statements
}
```

name ← The function name is an identifier like a variable name.
parameter list ← The parameter list is a list of input variables for the function.
statements ← The statements do the actual work accomplished by the function.

Our practice in this class has been to define functions in the `<head>` block, and use them with calls from code in the `<body>` block. This is not the only way to organize JavaScript programs, but it is clear and readable and avoids some potential problems.

The name of a function must be unique within the program. The names are composed the same way that variable names are.

The parameter names are also identifiers. They are the variable names that your function uses when it is performing its calculations. When the function is called, the calling code supplies a value for each parameter. Within the body of the function, those values are assigned to the parameter names and the statements in the function body can use the parameter names to access the supplied values.

The function body includes whichever statements are required to implement the desired capability. Functions have access to variables defined outside of any function body (global variables) or can define their own variables inside the function body (local variables). Global variables are defined throughout the life of the program, whereas local variables are defined only for the period of time while the function is executing.

If necessary, a function can return a value to its caller using the `return` statement, eg, `return count;`. This is not always necessary because functions are often expected to complete their action internally by setting some visible field in the web page or setting the value of some global variable.

There are many functions provided by JavaScript language. Some functions that we have used include the following. For strings, `charAt(`*position*`)`, `toLowerCase()`, and `toUpperCase()`. For arrays, `join(`*separator string*`)`, `sort()`, and `reverse()`. To write text into an HTML document being created, use `document.write(`*string*`)`. To get a random number, use `Math.random()`.

Control Flow

We studied two primary statements for controlling the flow of execution in a JavaScript program: the if/else statement and the for loop.

The if/else statement has the following format:

```
if (boolean expression){
 statements
} else {
 statements
}
```

The *boolean expression* is evaluated. If the expression is `true`, then the first block of statements is executed and the second block is skipped. If the expression is `false`, then the first block is skipped, and the second block is executed.

The `else` block is optional. If there is no else block, the *boolean expression* is evaluated as before. If the expression is `true`, then the given block of statements is executed. If the expression is `false`, then the given block of statements is skipped.

The `for` loop statement has the following format:

```
for (initialize; limit check; update) {
  <loop body statements>
}
```

The *initialize* statement is executed once before the loop begins operation. The *limit check* statement is executed before each iteration of the loop. If it evaluates to `true`, the statements in the body of the loop are executed. If it evaluates to `false`, then the looping is done and execution continues with the first statement following the loop body. The *update* statement is executed after each iteration of the loop.

A common way to write the update statement is `i++` (assuming that `i` is the loop control variable). This is shorthand for `i=i+1`.

If the *limit check* is `false` the first time it is evaluated, then the loop body is skipped entirely.

`For` loops are used in many different ways. They can be used to sum up a set of values, inspect every item in an array or string, advance a calculation a certain number of steps, and so on.

Graphical User Interfaces (GUI)

One of the powerful capabilities of HTML and JavaScript combined is the ability to provide a graphical user interface for application programs.

Important HTML tags related to GUIs are `<form>`, `<input>`, and `<button>`. The `form` tag defines a block of input/output controls visible to the user.

The `input` and `button` tags are two examples of controls that are placed within forms. There are several categories of input control, including radio buttons, checkboxes, and text.

Important attributes associated with these tags are `id`, `name`, `onclick`, and `value`. The `id` is an identifying string that must be unique within the entire page. The `id` can be used to retrieve information about the specific control using the function `document.getElementById(`*id string*`)`. The `name` attribute is used as a secondary identifier in modern HTML. For example, all radio buttons in one group have the same `name`, although they must have different `id`s. The `onclick` attribute provides a little bit of JavaScript code that is executed when the user clicks on this control. In this class, we have generally put a function call as the `onclick` value, and then defined whatever we want in the function. The `value` attribute can be used to set the initial value for a text control.

Arrays

We use arrays to maintain simple ordered lists of elements. The elements can be numbers, strings, boolean values, or more complex objects like form elements. Arrays have a `length` property that you can get and set. The items in an array are indexed and the index starts at 0 (not at 1!). So an array `myStuff` that has 3 items in it has a length of 3 and the items are accessible using `myStuff[0]`, `myStuff[1]`, and `myStuff[2]`. An array can be created using the Array constructor or using an array literal. The following code creates and initializes two 3-element arrays.

```
var pets = new Array(3);
pets[0] = "Smoky";
pets[1] = "Browny";
pets[2] = "Shadow";
var people = ["Alice", "Bob", "Carol"];
```

An important use of the `for` loop is looking at each item in an array in turn and doing something with one or more of the items like finding the maximum or summing the items.

As mentioned earlier, useful functions for working with arrays include `join(`*separator string*`)`, `sort()`, and `reverse()`.

Document Object Model

Your web browser builds a *model* of the web page (the *document*) that includes all the *objects* in the page (tags, text, etc). This model is called the Document Object Model. All of the properties, methods, and events available to the web developer for manipulating and creating web pages are included in this model. The structure of the model is initially defined by the HTML that defines the page, but it can be changed dynamically by the program if desired.

The elements in the model are arranged in a tree structure. Each HTML tag in the page is a node in the tree. The program code can traverse this tree structure several different ways. The way we have used in our class is to give a tag of interest a specific unique `id` attribute, then use the function `document.getElementById(`*element id*`)` to retrieve the element. Once we have a reference to a particular element we can read and change its various properties as appropriate.

At the top level of the tree, information is available about the web page as a whole. Thus we can find out things like title, referrer, domain, and URL.

The details of the Document Object Model are changing as new standards are defined. However, the basic idea of a model of a well-structured document is very important and will only become more important as time goes on. XML (eXtendible Markup Language) is becoming a very important standard for defining data structure, and in fact the latest revision of HTML is defined in terms of the XML standard.

Context

We have spent a lot of time studying HTML and JavaScript because they are a good example of the use of static data structures and dynamic programming languages.

HTML forms the basic structure for web pages. Modern trends are to separate the structural description of the data the presentation of the data to the user. We process the information according to the content, and display the information as appropriate to the user's needs at the time.

Programming languages like JavaScript make management and display dynamic rather than static. We can select, filter, and combine information using a program. We can modify the display dynamically depending on user input requests. There are numerous other programming languages, varying depending on their intended purpose and place in the overall computer system architecture (client side, server side, dynamically loaded, installed once, etc).