Digital Information

INFO/CSE 100, Spring 2005

Fluency in Information Technology

http://www.cs.washington.edu/100

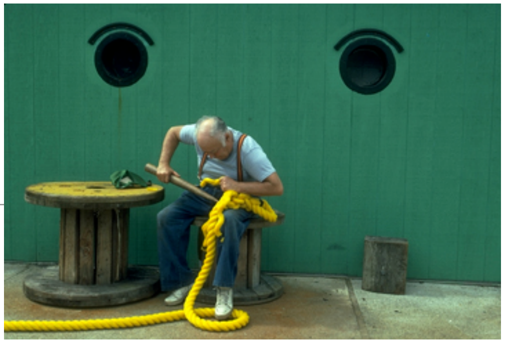
Readings and References

- Reading
  - » *Fluency with Information Technology*
    - Chapters 9, 11 18-21

Overview

- During this quarter, we're looking at the actual workings of computer systems
- Organized as “*layers of abstraction*”
  - » application programs
  - » higher level languages: Javascript, SQL, …
  - » operating system concepts
  - » bits, bytes, assembly language
  - » transistors, electrons, photons

Layers of Abstraction



- At any level of abstraction, there are
  - » elements at that level
  - » the building blocks for those elements
- Abstraction
  - » isolates a layer from changes in the layer below
  - » improves developer productivity by reducing detail needed to accomplish a task
  - » helps define a single architecture that can be implemented with more than one organization

Architecture & Organization

- Architecture (the *logical definition*)
  - » defines elements and interfaces between layers
  - » Instruction Set Architecture
    - instructions, registers, addressing
- Organization (the *physical implementation*)
  - » components and connections
  - » how instructions are implemented in hardware
  - » many different organizations can implement a single architecture

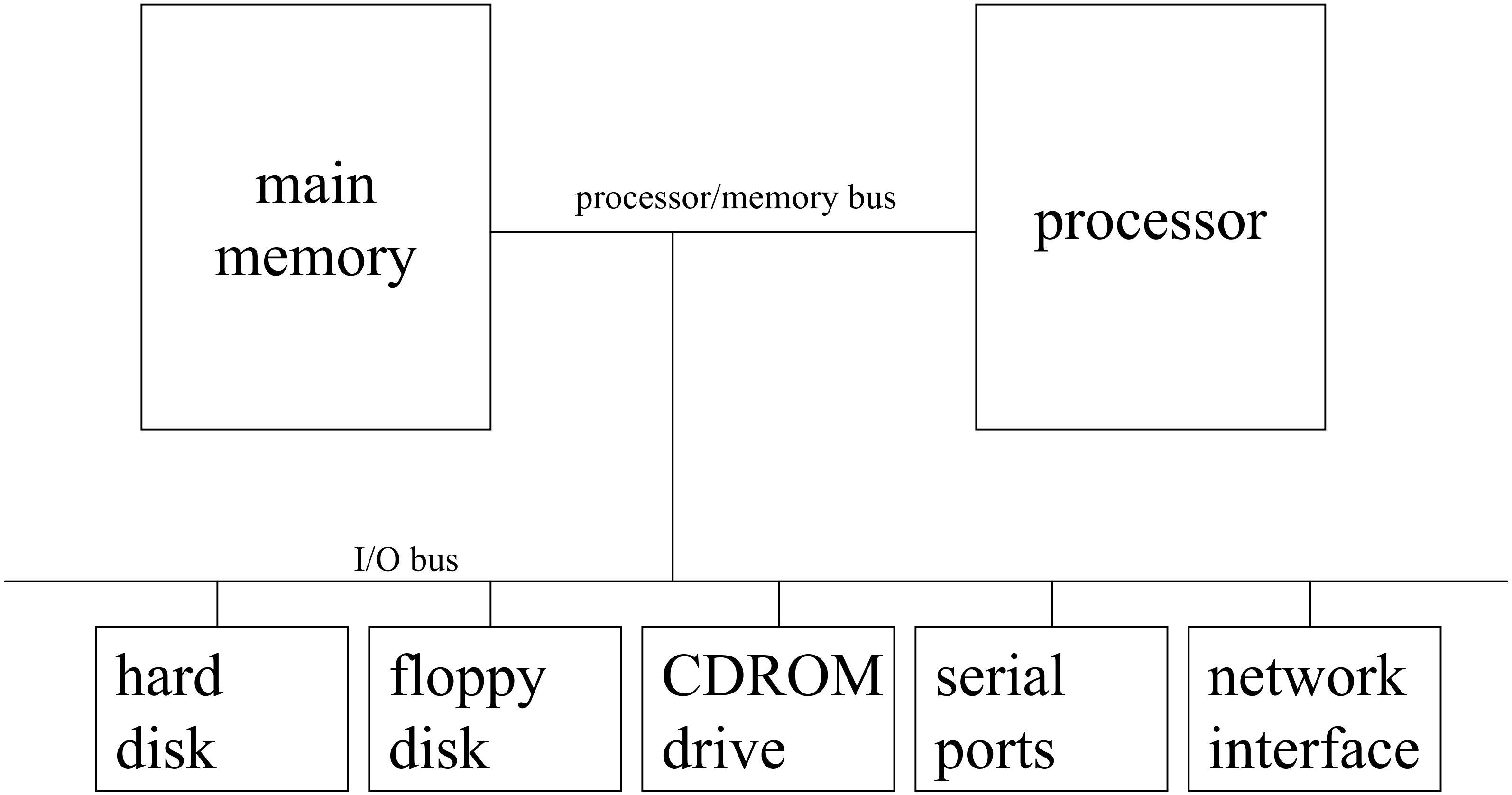
Computer Architecture

- Specification of how to program a specific computer family
  - » what instructions are available?
  - » how are the instructions formatted into bits?
  - » how many registers and what is their function?
  - » how is memory addressed?
- Some examples architectures
  - » IBM 360, 370, …
  - » PowerPC 601, 603, G5, …
  - » Intel x86 286, 386, 486, Pentium, …
  - » MIPS R2000, R3000, R4000, R5000, ...

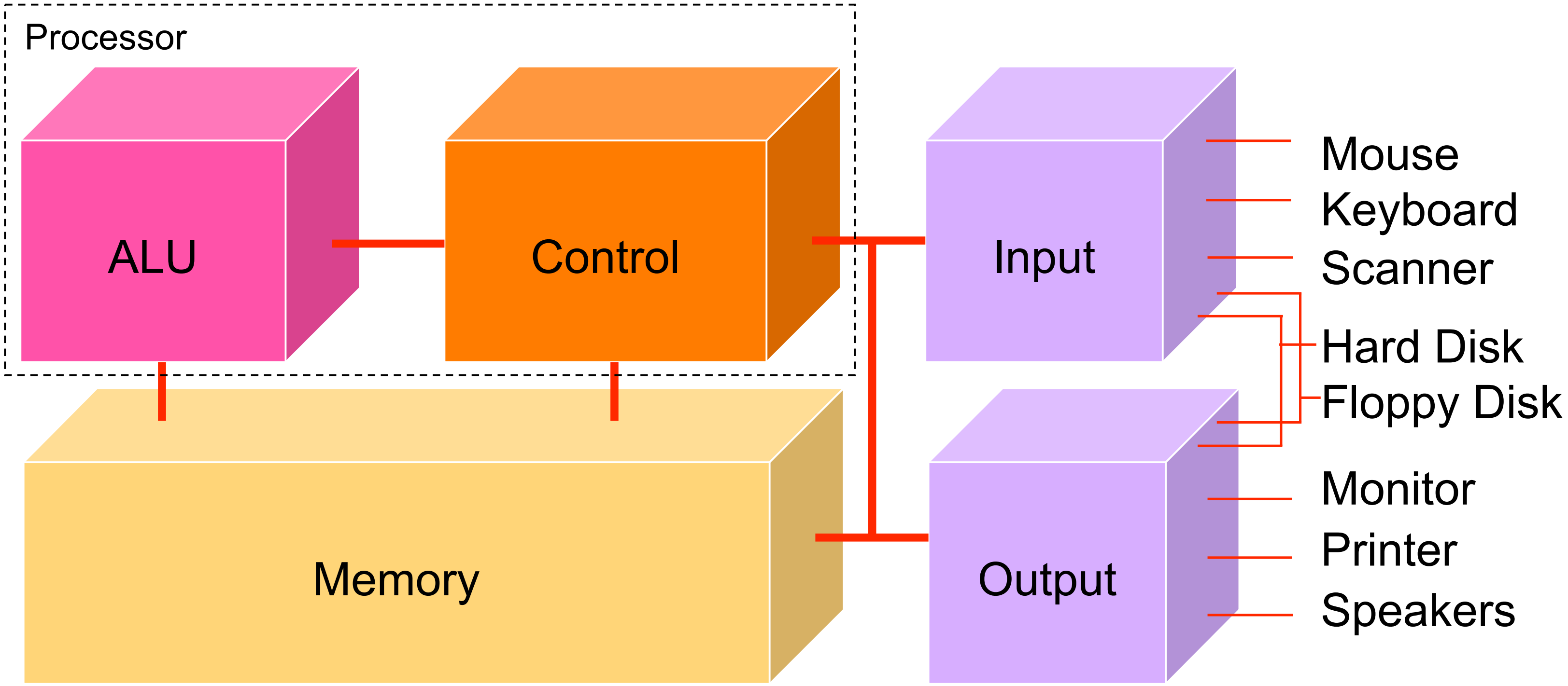
Computer Organization

- Processor
  - » Data path (ALU) manipulate the bits
  - » The control controls the manipulation
- Memory
  - » cache memory - smaller, higher speed
  - » main memory - larger, slower speed
- Input / Output
  - » interface to the rest of the world

A Typical Organization



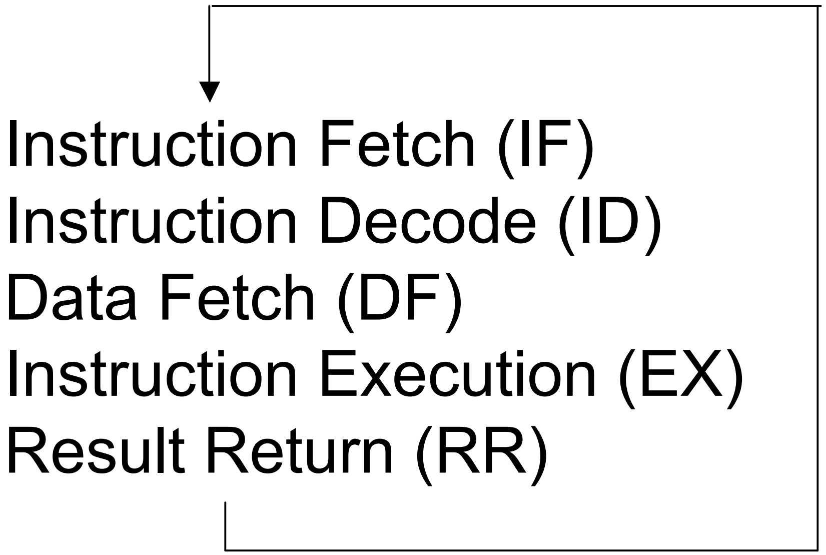
Anatomy of a Computer



Fetch/Execute Cycle

Computer = instruction execution engine

- » The fetch/execute cycle is the process that executes instructions

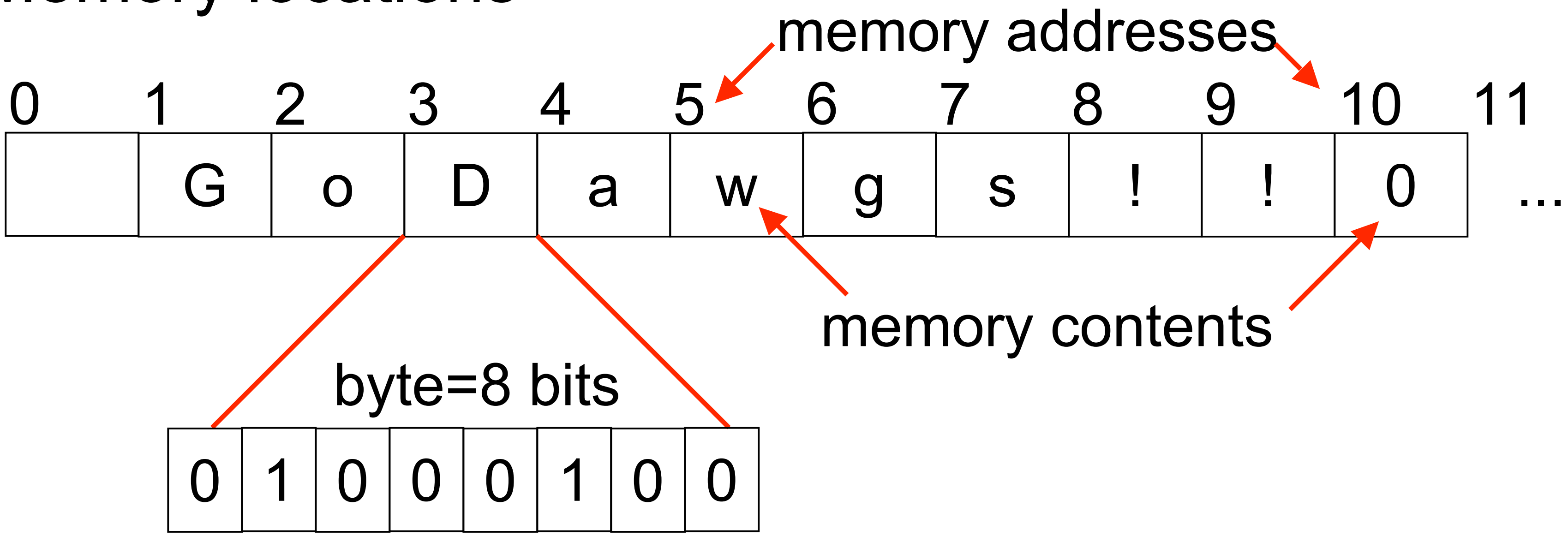


Instruction Fetch (IF)
Instruction Decode (ID)
Data Fetch (DF)
Instruction Execution (EX)
Result Return (RR)

Memory ...

Programs and the data they operate on must be in the memory while they are running

Memory locations



memory addresses

0	1	2	3	4	5	6	7	8	9	10	11
	G	o	D	a	w	g	s	!	!	0	...

memory contents

byte=8 bits

0	1	0	0	0	1	0	0

Control

- The Fetch/Execute cycle is hardwired into the computer's control, i.e. it is the actual "engine"
- Depending on the Instruction Set Architecture, the instructions say things like
  - » Put in memory location 20 the contents of memory location 10 + contents of memory location 16
  - » The instructions executed have the form ADDB 10, 16, 20
    - Add the bytes from memory address 10 and memory address 16 and store the result in memory address 20

10	11	12	13	14	15	16	17	18	19	20	21
6						12				18	…

ALU

The Arithmetic/Logic Unit does the actual computation

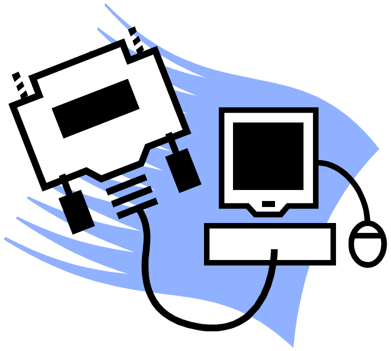
Depending on the Instruction Set Architecture, each type of data has its own separate instructions

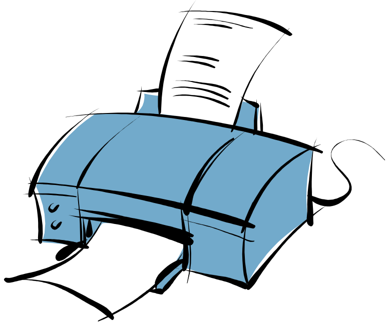
ADDB	: add bytes	ADDBU	: add bytes unsigned
ADDH	: add half words	ADDHU	: add halves unsigned
ADD	: add words	ADDU	: add words unsigned
ADDS	: add short decimal numbers		
ADDD	: add long decimal numbers		

Most computers have only about a 100-150 instructions hard wired

Input/Output

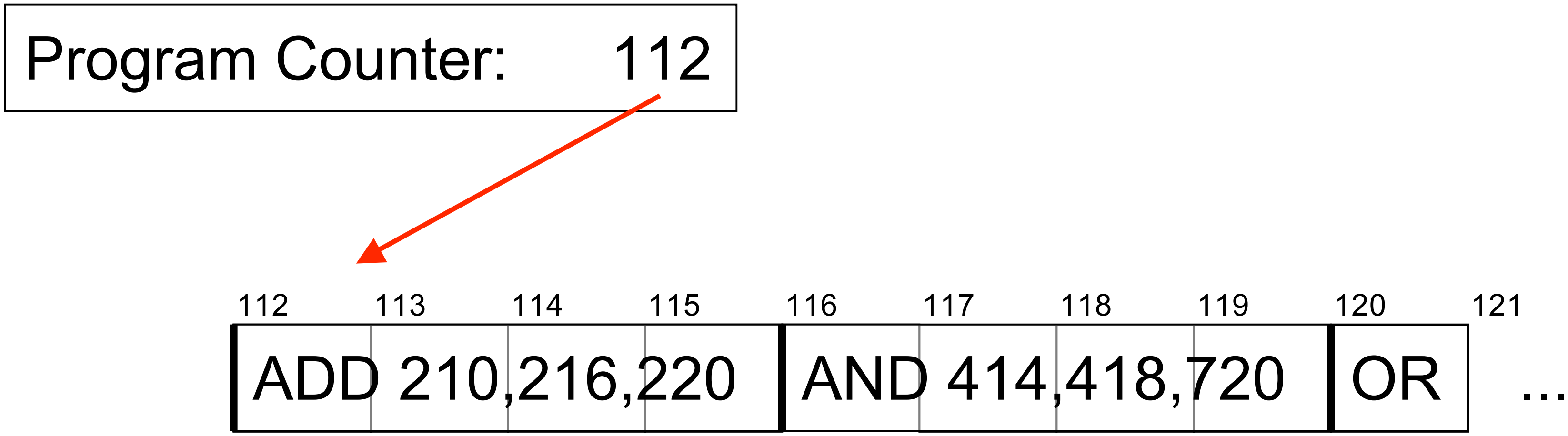
- Input units bring data to memory from outside world; output units send data to outside world from memory
  - » Most peripheral devices are “dumb”, meaning that the processor assists in their operation





The PC's PC

- The program counter (PC) tells where the next instruction comes from
  - » In some architectures, instructions are always 4 bytes long, so add 4 to the PC to find the next instruction



Program Counter: 112

112 113 114 115 116 117 118 119 120 121

ADD 210,216,220 AND 414,418,720 OR ...

Clocks Run The Engine

- The rate that a computer “spins around” the Fetch/Execute cycle is controlled by its clock
  - » Current clocks run 2-3 GHz
  - » The computer tries do at least one instruction per cycle, depending on the instruction and the availability of memory contents
  - » Modern processors often try to do more than one instruction per cycle

Clock rate is not a good indicator of speed anymore, because several things are happening every clock cycle

Algorithm

- Algorithm
  - » a precise, systematic method to produce a desired result
- For example, the placeholder technique for deleting a short string except where it occurs in longer strings is an algorithm with an easy specification:

```
longStringWithShortStringInIt ← placeholder
ShortString ← e
placeholder ← longStringWithShortStringInIt
```

Programs vs Algorithms

- A program is an algorithm specialized to a particular situation
  - » an Algorithm

```
longStringWithShortStringInIt ← placeholder
ShortString ← e
placeholder ← longStringWithShortStringInIt
```

  - » a Program that implements the Algorithm

```
↲↲ ← #        // replace double <newlines> with <#>
↲ ← e         // delete all single < newlines>
# ← ↲↲        // restore all double <newlines>
```

Variables In Real Life

- A variable is a "container" for information you want to store
  - » The name of the variable stays the same, but the value associated with that name can change

That's why it's called a "variable"!

Variable **Name**	Current **Value**	Previous **Value**
#1 Single	My Boo, Usher And Alicia Keys	Goodies, Ciara
AL Champion	Boston Red Sox	New York Yankees
#1 Box Office	Shark Tale	Shark Tale
Day Of The Week	Monday	Sunday
Husky Card Balance	$52	$60

Variables In Programming

- Program variables have names and values
  - » Names (also called identifiers)
    - generally start with a letter and can contain letters, numbers, and underscore characters "_"
    - Names are *case sensitive*
  - » Values
    - can be numbers, strings, boolean, etc
    - change as the program executes

Variable **Name**	Current **Value**	Previous **Value**
No_1_Single	My Boo, Usher And Alicia Keys	Goodies, Ciara
ALChampion	Boston Red Sox	New York Yankees
No_1_Box_Office	Shark Tale	Shark Tale
dayOfTheWeek	Monday	Sunday
huskyCardBalance	$52	$60

Variable Declarations

```
<script type="text/javascript">

var eyeColor;  <<< undefined!

var eyeColor = "green";  <<< initialized

var eyeColor = ""; <<< initilized, empty

var eyeColor = "green", hairColor="blonde";

hairColor = "carmel";
</script>
```

Basic Data Types in Javascript

Numbers:

```
var gasPrice = 2.55;
```

Strings

```
var eyeColor = "hazel green";
```

Boolean

```
var isFriday = true;
var isWeekend = 0;
```

Expressions

- The right-hand side of an assignment statement can be any valid *expression*
- Expressions are “formulas” saying how to manipulate existing values to compute new values

```
balance = balance - transaction;
seconds = 60*minutes;
message = "Status code is " + codeValue;
isFreezing = (temp < 32);
```

Operators

Use operators to build expressions

» Numeric operators

+ - * / *mean* add, subtract, multiply, divide

3 + 3 = 6

» String operator

+ *means* concatenate strings

"3" + "3" = "33"

» Relational operators

< <= == != >= > *mean* less than, less than or equal to, equal to, not equal to, greater than or equal to, greater than

» Boolean operators

&& || ! *mean* and, or, not

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

- » *<name>* ← the function name is an identifier
- » *<parameter list>* ← list of input variables for the function
- » *<statements>* ← the statements do the work

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

Example Function

template

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

Write a simple function to compute the Body Mass Index when the inputs are in English units (ie, US units)

example

```
// Calculate Body Mass Index in English units
// weight in pounds
// height in inches
// returns body mass index

function bmiE(weightLBS, heightIN)  {
  var heightFt = heightIn / 12; // convert to feet
  return 4.89 * weightLBS / (heightFt * heightFt);
}
```

Global or Local?!?

- Scope of a variable describes where and when it can be referenced
  - Local variables are only known inside of a function (curly braces)
  - Global variables are know by all the Javascript inside of <script> </script> pairs

```
// Calculate Percentage of Study Hours/Week
// time in hours
// returns hours
var days = 7;
function calculateStudyHrs(time)  {
  var totalHrs = 24 * days;
  return time/totalHrs;
}
```

Layout of the GUI

- The layout of the page is controlled with HTML in the body of the page

```
<body>
 HTML form layout and specification
</body>
</html>
```

- The layout and controls are provided using new tags
  - » <form id="buttonForm">
  - » <button type="button" ...
  - » <input type="text" …
  - » <input type="radio" …
  - » <button type="reset" …

A simple example

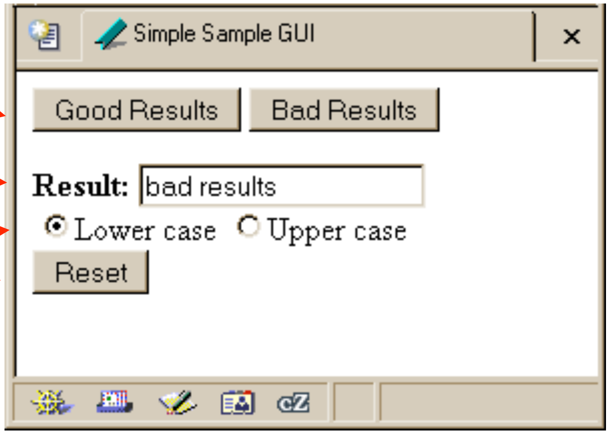
This GUI has several simple controls.

Two buttons to control the results

One text field to display the results

One pair of radio buttons to control the display

One button to reinitialize



Result: bad results

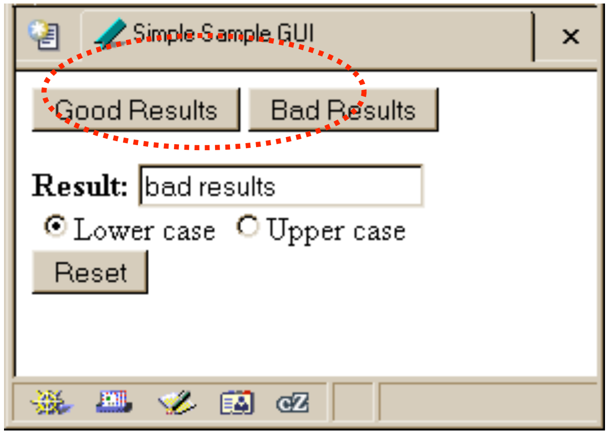
● Lower case ○ Upper case

http://www.cs.washington.edu/education/courses/100/04au/slides/13-gui/gui.html

<button type="button" ...>

```
<form>
<button type="button"
  onclick="setResults('good results')">Good Results</button>
<button type="button"
  onclick="setResults('bad results')">Bad Results</button>
</form>
```

- a <button> can have one of three types
  - type “button” is used locally
  - type “ submit” sends data back to the server
  - type “reset” re-initializes the form
- the value of the “onclick” attribute is some JavaScript code, in this case a call to the function `setResults(string)`



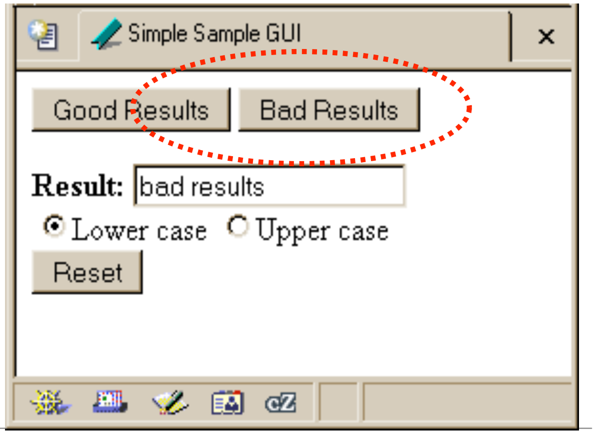
Result: bad results

● Lower case ○ Upper case

<input type="text" ...>

```
<form>
<b>Result:</b>
<input type="text" value="nada" readonly id="resultField">
<br>
<input type="radio" name="case" id="radioLC" checked
  onclick="setResults(document.getElementById('resultField').value)">Lowercase
<input type="radio" name="case" id="radioUC"
  onclick="setResults(document.getElementById('resultField').value)">Uppercase
<br><button type="reset">Reset</button>
</form>
```

- an `<input>` with `type="text"` is used for user input and program output
- `value="nada"` sets the initial (and reset) value
- `readonly` means that the user cannot set the value, only the script can set the value
- `id="resultField"` gives us a way to identify this particular control in our JavaScript



Result: bad results

● Lower case ○ Upper case

Events Cause Processing

- After drawing a page, the browser sits idle waiting for something to happen … when we give input, we cause *events*
- Processing events is the task of a block of code called an event handler
  - » The code to execute is identified in the tag using the appropriate attribute
  - » There are many event types
    - onClick, onChange, onMouseOver ...

setResults(resultString)

```
<script type="text/javascript">
function setResults(resultString) {
  var tempString = resultString;
  if (document.getElementById("radioLC").checked) {
    tempString = tempString.toLowerCase();
  } else if (document.getElementById("radioUC").checked) {
    tempString = tempString.toUpperCase();
  }
  document.getElementById("resultField").value = tempString;
}
</script>
```

parameter variable, local variable, if/else statement, field reference, call to toLowerCase() function

The `if` / `else` statement

The `if` statement is a *conditional statement*

- a conditional expression is evaluated as being `true` or `false`
    - the expression is a *boolean expression* (ie, returns `true` or `false`)
- if the condition is `true`, then one set of statements is executed
- if the statement is `false`, then a different set of statements is executed

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

Examples

```
if (count == 0) {
  ready = false;
} else {
  ready = true;
  count = count-1;
}
```

What is the conditional expression?

What statements are part of the true block?

Which statements are part of the false block?

What happens when count is 21?   0? -1?

```
if (pageCount >= 100) {
  alert("This may take a few minutes.");
}
```

What is the conditional expression?

What statements are part of the true block?

Which statements are part of the false block?

What happens when pageCount is 21?   100? 200?

More if/else Statements

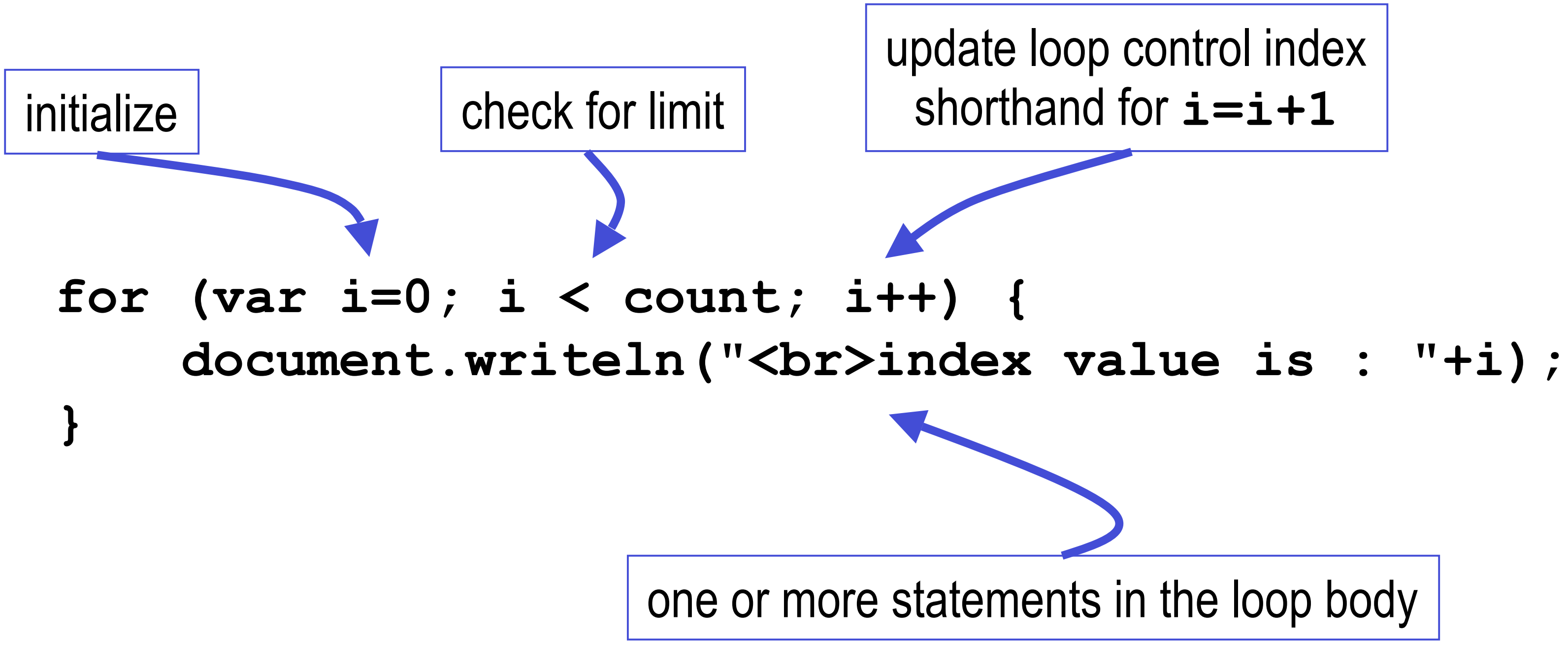
```
if (temp < 32) {
  if (sky == "cloudy) {
    alert("Snow is forecast!");
  }
}

if (temp < 32 && sky == "cloudy") {
  alert("Snow is forecast!");
}
```

The **for** loop

A counting loop is usually implemented with **for**

```
var count = 10;
```



initialize

check for limit

update loop control index
shorthand for **i=i+1**

```
for (var i=0; i < count; i++) {
    document.writeln("<br>index value is : "+i);
}
```

one or more statements in the loop body

`i++` is a shortcut

- `for (i=0; i < count; i++)`
- at the end of every pass through the **`for`** loop body, do the following:
  - » get the value of i
  - » increment i
  - » store the incremented value
- Used as it is here, this is the same as writing
  - » **`i = i + 1`**

body of loop may not execute at all

- Notice that depending on the values of the control variables, it is quite possible that the body of the loop will not execute at all

check for limit condition

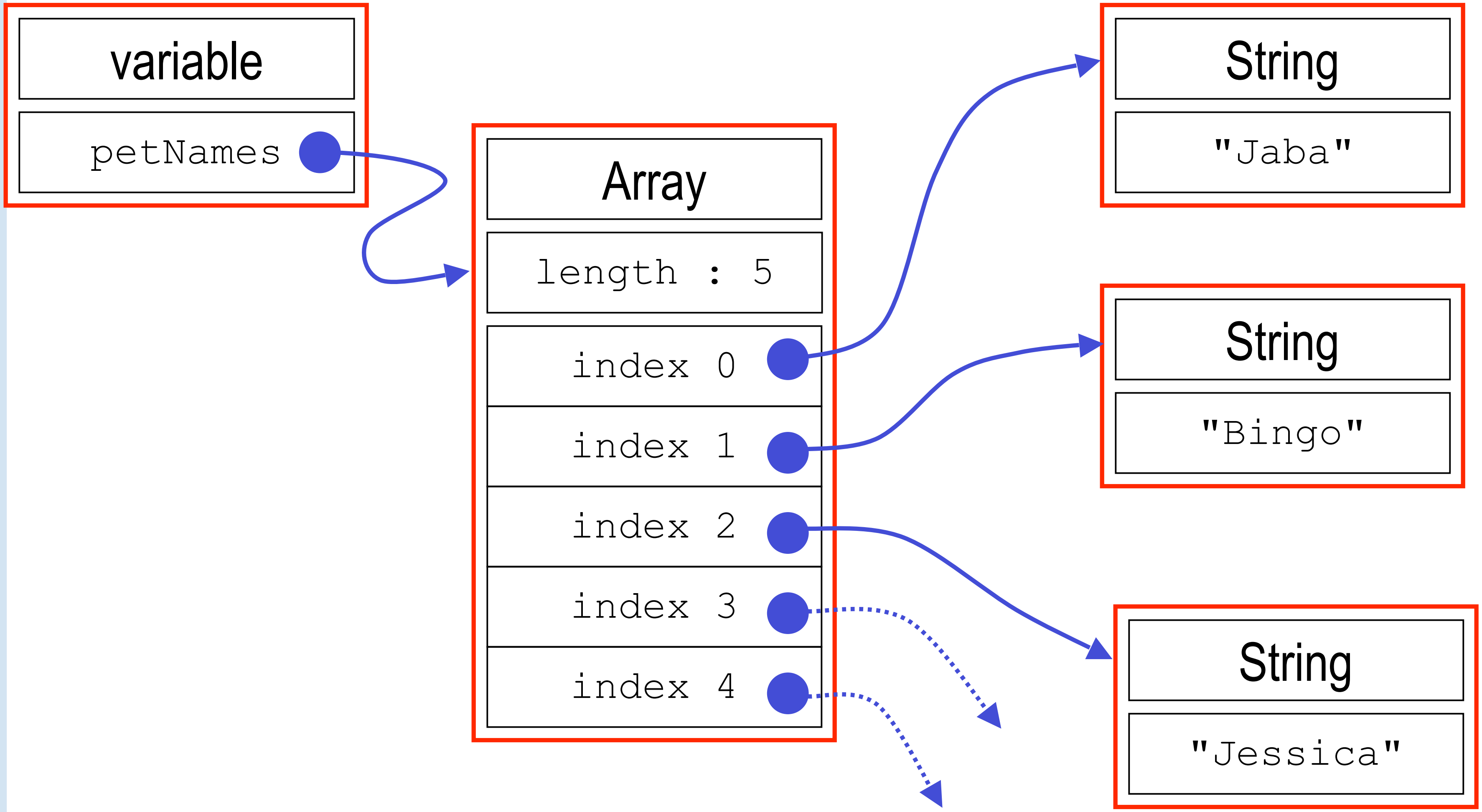
itemCount is 0 when we get here, so **i<itemCount** is immediately false and the loop body is skipped completely

```
var itemCount = 0;
...
for (var i=0; i < itemCount; i++) {
    document.writeln("<br>..processing item "+i);
}
```

Arrays

- JavaScript (and most other languages) includes *arrays* as the most basic kind of collection.
  - » Simple, ordered collections
  - » Special syntax for accessing elements by position
- JavaScript arrays can be created
  - » by the programmer in the script
  - » by the system and provided to the script
    - for example, the elements array in the iCCC program

Array Example



JavaScript Indexed Arrays

- An indexed array is a data type that stores a collection of values, accessible by number
  - » the values in the array are called the *elements* of the array
  - » the elements (or values) are accessed by *index*
    - the index of the first value is 0
  - » the values in the array can be any type
    - usually all the values are the same type
    - but they can be different from one another if necessary

Array Declaration and Creation

- Arrays can be created several different ways
  - » `var petNames = new Array();`
    - 0-length array with no elements in it yet
  - » `var studentNames = new Array(102);`
    - 102-element array, all of which have the value *undefined*
  - » `var myList = ["Sally", "Splat", "Google"];`
    - 3-element array initialized with an *array literal*
- Arrays have a property that stores the length

  <array name>`.length`
  - » you can lengthen or shorten an array by setting the length to a new value

Array Element Access

- Access an array element using the array name and position: *<array name>* [*<position>*]
- Details:
  - » *<position>* is an integer expression.
  - » Positions count from zero
- Update an array element by assigning to it:

  <array name> [*<position>*] = *<new element value>* ;

```
myCurrentCarNo = carList.length-1;
myCurrentCar = carList[myCurrentCarNo];
```

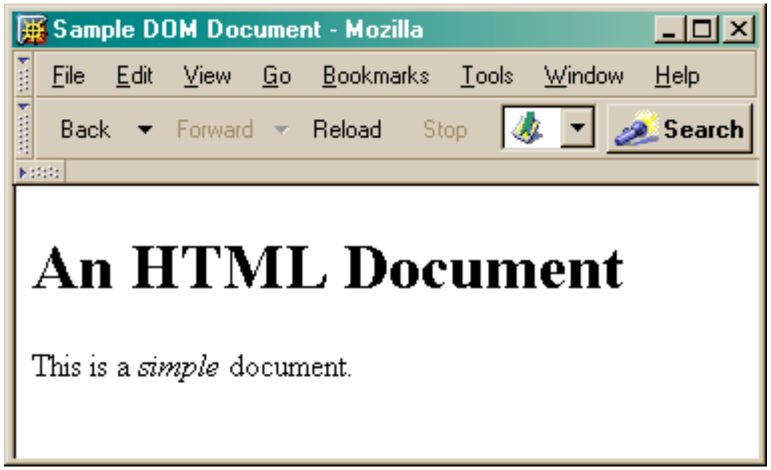
What the heck is the DOM?

- 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)
  - » All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
  - » Those objects are accessible via scripting languages in modern web browsers

This is what the browser reads (sampleDOM.html).

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

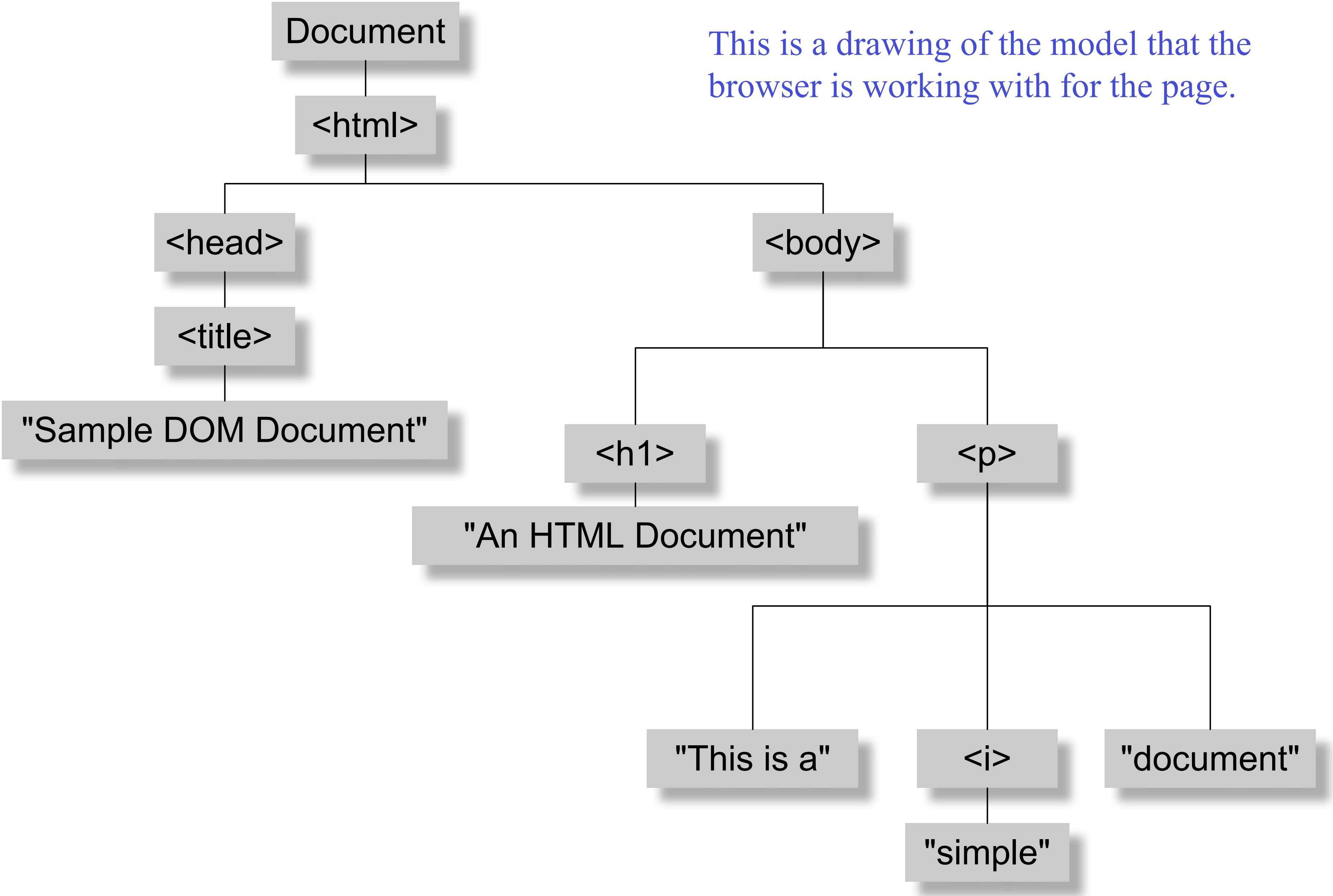
This is what the browser displays on screen.



Sample DOM Document - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
Search

An HTML Document

This is a *simple* document.



This is a drawing of the model that the browser is working with for the page.

Figure 17-1. The tree representation of an HTML document
Copied from JavaScript by Flanagan.

`document.getElementById("radioLC").checked`

- Reference to several nodes in the model of the page that the browser constructed
- **`document`**
  - » The root of the tree is an object of type `HTMLDocument`
  - » Using the global variable `document`, we can access all the nodes in the tree, as well as useful functions and other global information
    - title, referrer, domain, URL, body, images, links, forms, ...
    - open, write, close, getElementById, ...

document.getElementById("radioLC").checked

- getElementById("radioLC")
  - » This is a predefined function that makes use of the `id` that can be defined for any element in the page
  - » An `id` must be unique in the page, so only one element is ever returned by this function
  - » The argument to `getElementById` specifies which element is being requested

`document.getElementById("radioLC").checked`

- `checked`
  - » This is a particular property of the node we are looking at, in this case, a radio button
  - » Each type of node has its own set of properties
    - for radio button: `checked, name, ...`
    - refer to the HTML DOM for specifics for each element type
  - » Some properties can be both read and set

Representing Data as Symbols

- 24 Greek Letters
- And we decide to use 2 symbols, binary, to represent the data.
- How many bits do we need?!?
  - » 24 total possibilities
  - » 2x2x2x2x2 = 2^5 = 32
    - We get 6 extra!

Info Representation

- Adult humans have 32 teeth
  - » sometimes a tooth or two is missing!
- How can we represent a **set** of teeth?
  - » How many different items of information?
    - 2 items - *tooth* or *no tooth*
  - » How many "digits" or positions to use?
    - 32 positions - one per tooth socket
  - » Choose a set of symbols

no tooth: 0   *tooth*: 1

What's your tooth number?

incisors canines pre-molars molars

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

no teeth <-> 0000 0000 0000 0000 0000 0000 0000 0000

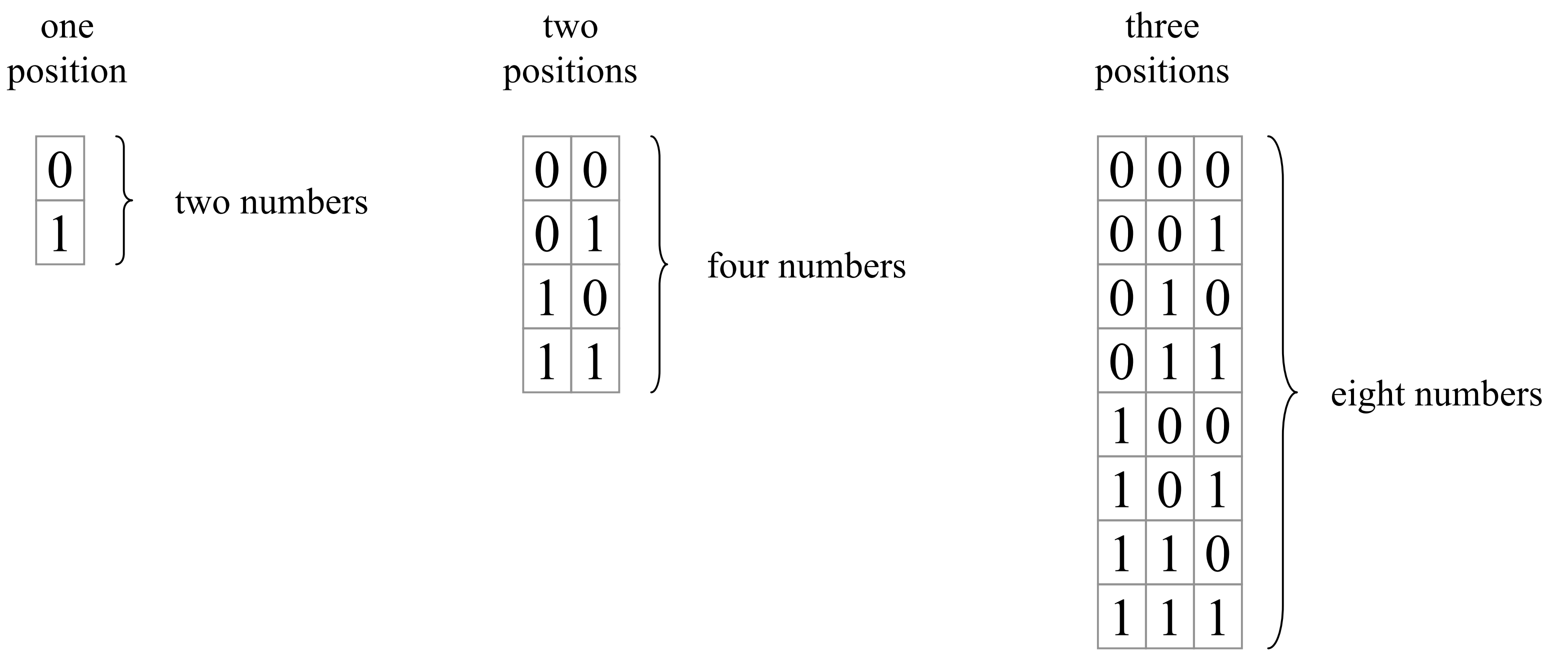
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0

no molars <-> 1111 1111 1111 1111 1111 0000 0000 0000

How many possible **combinations**? $2\times2\times2\times2\times\ldots\times2 = 2^{32} \approx$ 4 Billion

How many positions should we use?

It depends: how many numbers do we need?



one position

0	
1	two numbers

two positions

0	0	
0	1	
1	0	four numbers
1	1	

three positions

0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	eight numbers
1	0	1	
1	1	0	
1	1	1	

Converting from binary to decimal

			2×2×2×2	2×2×2	2×2	2	1	
$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	base 10
1	0	0	0	1	0	1	0	
								base 2

$$1 \cdot 128 + 0 \cdot 64 + 0 \cdot 32 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 138_{10}$$

$$1 \cdot 128 + 1 \cdot 8 + 1 \cdot 2 = 138_{10}$$

Each position represents one more multiplication by the base value.

For binary numbers, the base value is 2, so each new column represents a multiplication by 2.

Base 16 Hexadecimal

- The base value can be **16** - *hexadecimal numbers*
  - » Sixteen symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - » Each column represents a multiplication by sixteen
  - » Hex is easier to use than binary because the numbers are shorter even though *they represent the same value*

16×16×16	**16×16**	**16**	**1**	
$16^3 = 4096$	**$16^2 = 256$**	**$16^1 = 16$**	**$16^0 = 1$**	base 10
0	0	8	A	base 16

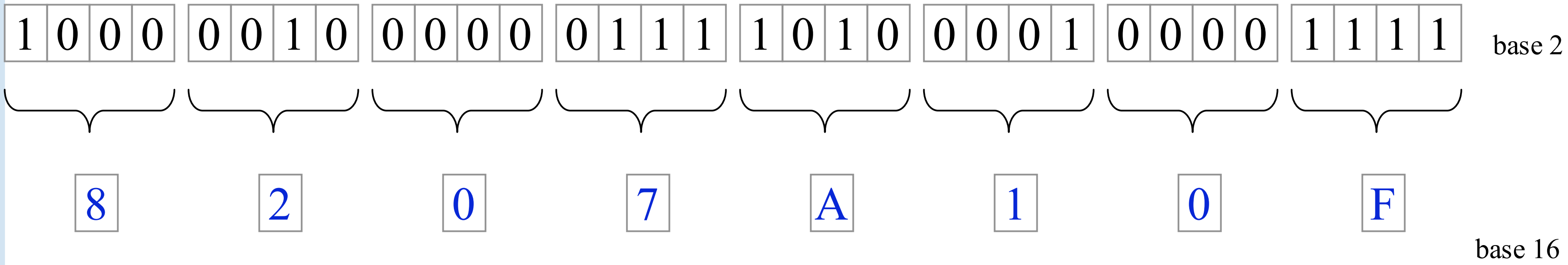
$$8 \cdot 16 + 10 \cdot 1 = 138_{10}$$

Four binary bits ⇔ One hex digit

binary base 2					hexdecimal base 16		decimal base 10
0	0	0	0		0		0
0	0	0	1		1		1
0	0	1	0		2		2
0	0	1	1	⇔	3	⇔	3
0	1	0	0		4		4
0	1	0	1		5		5
0	1	1	0		6		6
0	1	1	1		7		7

binary base 2					hexdecimal base 16		decimal base 10
1	0	0	0		8		8
1	0	0	1		9		9
1	0	1	0		A		10
1	0	1	1	⇔	B	⇔	11
1	1	0	0		C		12
1	1	0	1		D		13
1	1	1	0		E		14
1	1	1	1		F		15

Binary to Hex examples



1 0 0 0	0 0 1 0	0 0 0 0	0 1 1 1	1 0 1 0	0 0 0 1	0 0 0 0	1 1 1 1	base 2
8	2	0	7	A	1	0	F	base 16

$$10000010000001111010000100001111_2 = 8207A10F_{16}$$

1 0 0 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	1 0 0 1	1 0 1 1	1 1 1 0	base 2

$$10000011010001010110100110111110_2 = \underline{\hspace{4cm}}_{16}$$