Review Session 1

R.J. Aquino, '14

Quiz 1

Quiz 1 Information

- https://cs50.harvard.edu/quizzes/2013/1
- Cumulative, but with an emphasis on material covered since Quiz 0
- Typically more challenging than Quiz 0
- Use CS50 Discuss and take practice quizzes!

Quiz 1 Review Session

- This is NOT an exhaustive list of topics
- This is NOT necessarily everything you need to know about any given topic
- This IS meant to review topics we covered in lecture and section

File I/O

Week 7 Monday, Section 6, Problem Set 5

File I/O

- fopen, fclose, fwrite, fread, fseek
- You should be pretty familiar with these functions after pset5!
- What are common file-related bugs?
  - Forgetting to check if fopen returned NULL or succeeded
  - Forgetting to fclose a file that you fopen'd
  - Forgetting to check if you have reached the end of a file

Structs

Week 7 Monday

Structs

```
// structure representing a student
typedef struct
{
    string name;
    int age;
}
student;
```

Structs, cont.

```
// declare an instanct of struct like any variable
student s;

// set fields of a struct with '.'
s.name = "RJ";
s.age = 21;

// update fields the same way
s.name = "R.J.";

// access fields the same way
printf("%s is %d years old\n.", s.name, s.age);
```

Structs, cont.

```
// you often will have a pointer to a struct
student* ptr = &s;

// to get to the fields, you first need to dereference
(*ptr).age = 22;

// the arrow syntax is a nice shortcut for this!
ptr->age = 22;
```

Data Structures

Data Structures

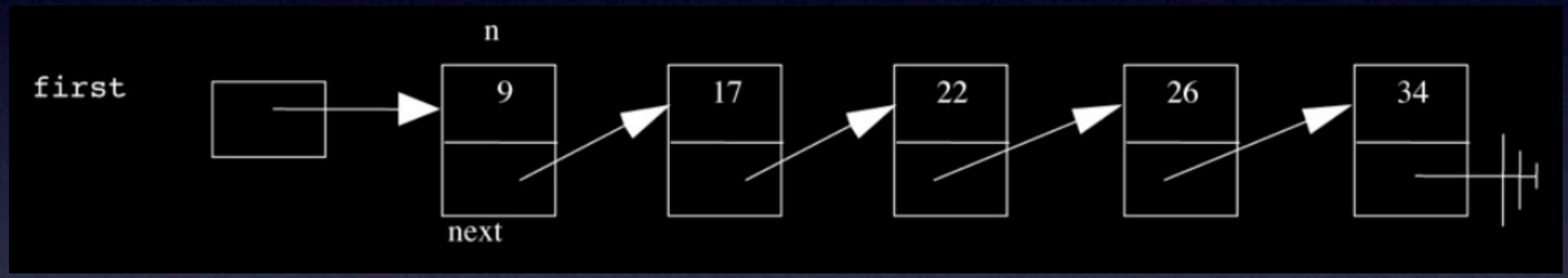
1. Understand each structure at a high level
   - Can you explain how it works in English?
2. Understand the implementation/operations
   - E.g., can you insert into a linked list?
   - Can you write C code related to these structures?
   - Understand pointers and structs
3. Know the runtimes/limitations
   - E.g., how fast is a hash table lookup?
   - Understand “Big-O” notation

Linked Lists

Week 7 Monday and Wednesday, Section 7

Linked Lists

High Level



Linked Lists

High Level

- Easy to insert - *O*(1) for unsorted lists
- Hard to find - *O*(*n*)
- Compare with arrays - when is a linked list better? When is an array better?

Linked Lists

Implementation

```
typedef struct node
{
    int n;
    struct node* next;
}
node;
```

Linked Lists

Implementation

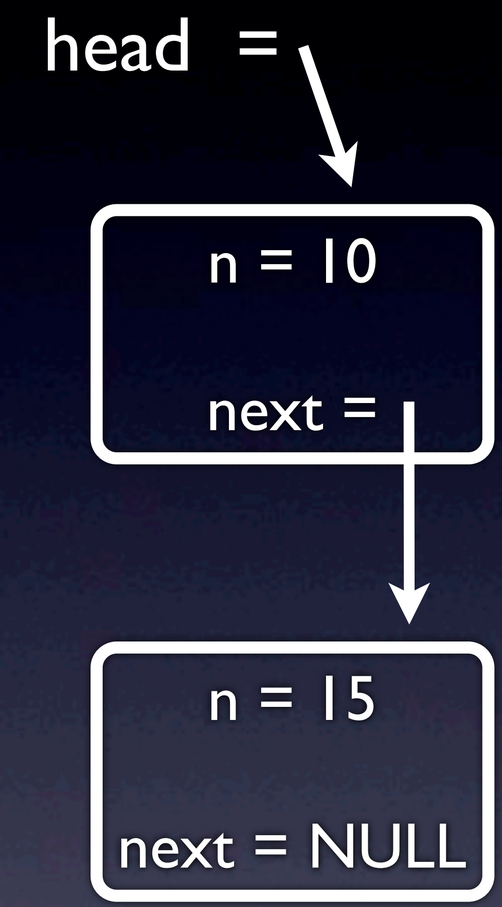
```
typedef struct node
{
    int n;
    struct node* next;
}
node;
```

Could be any type. In pset6, we stored char* or char arrays!

Linked Lists

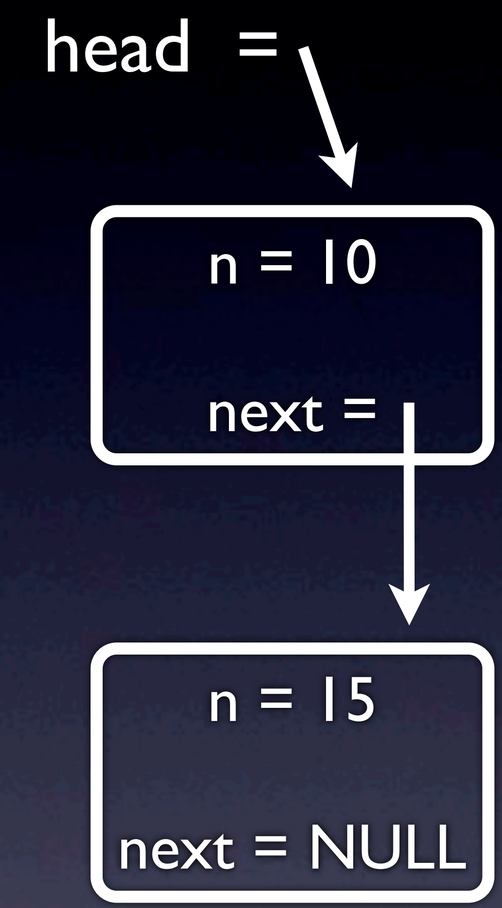
Operations

```
node* head;
bool insert(int new_n)
{

}
```



Linked Lists
Operations

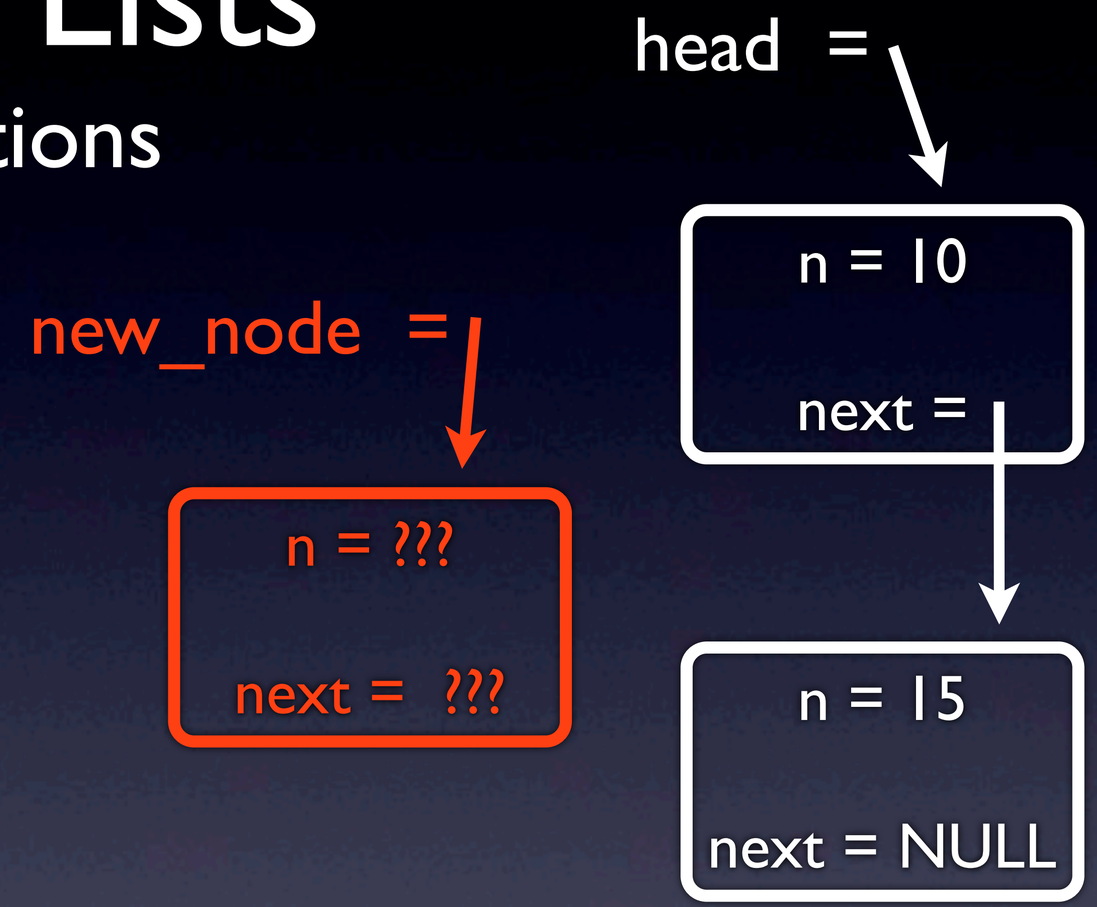
```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
       return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

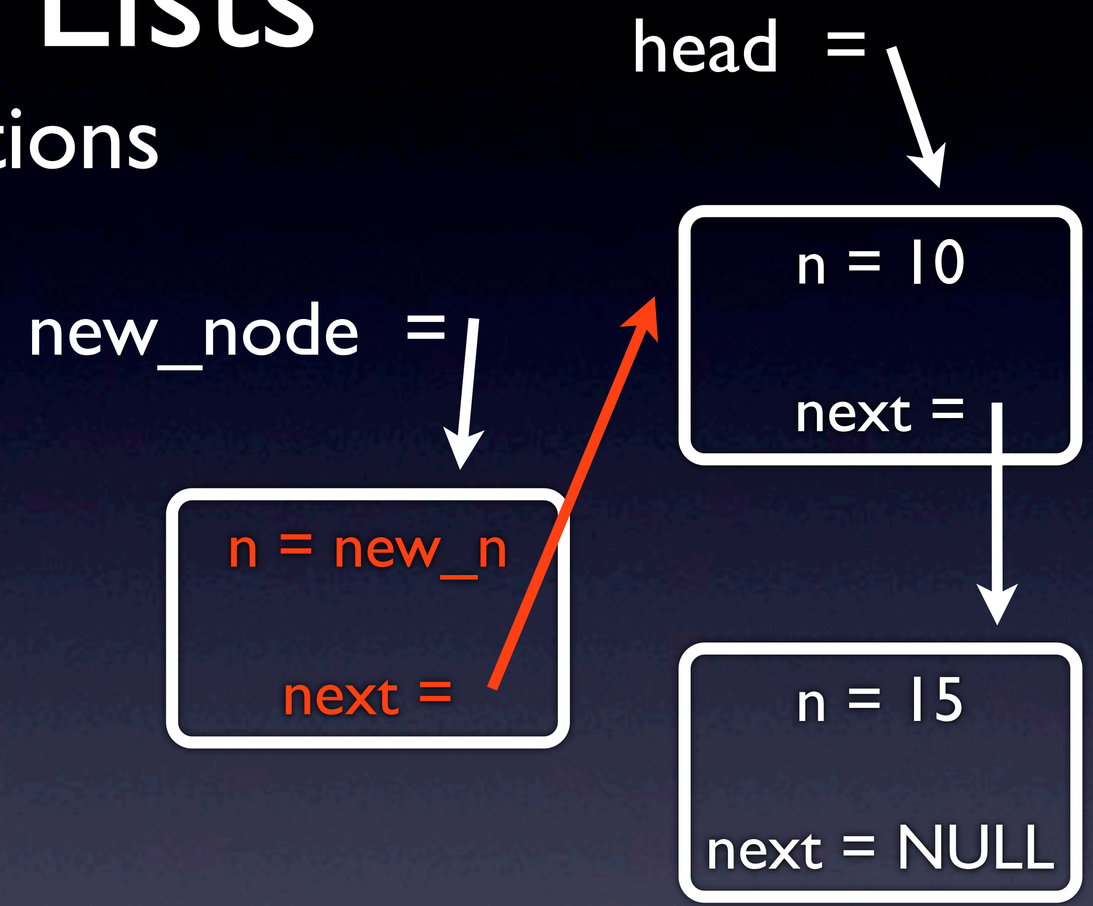
```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

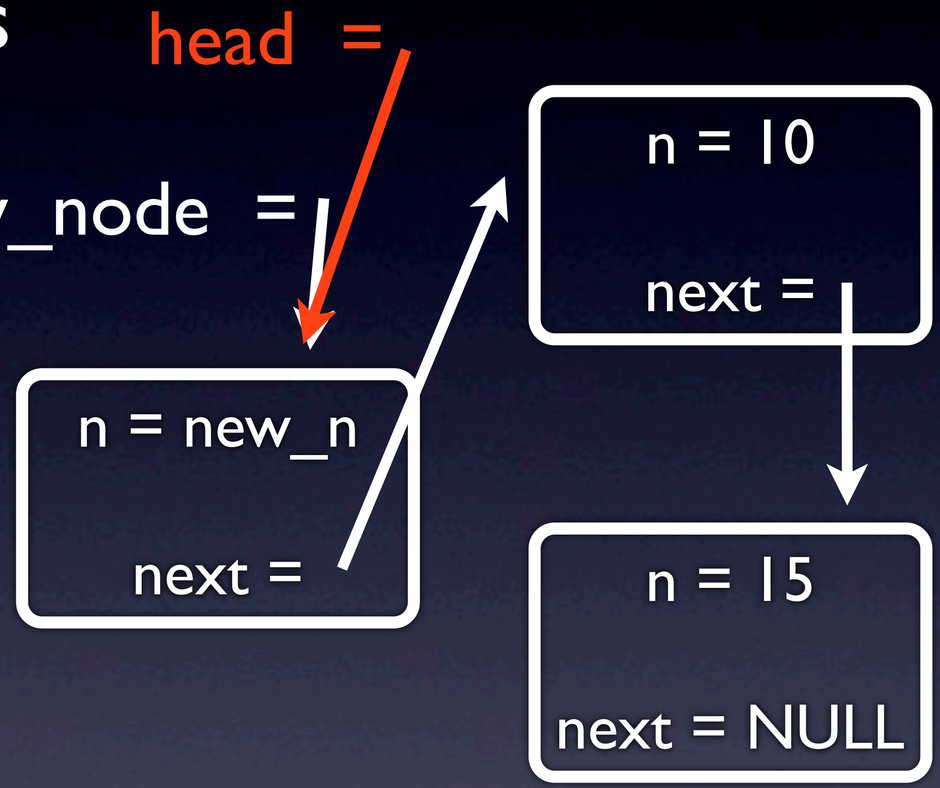
```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
       return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

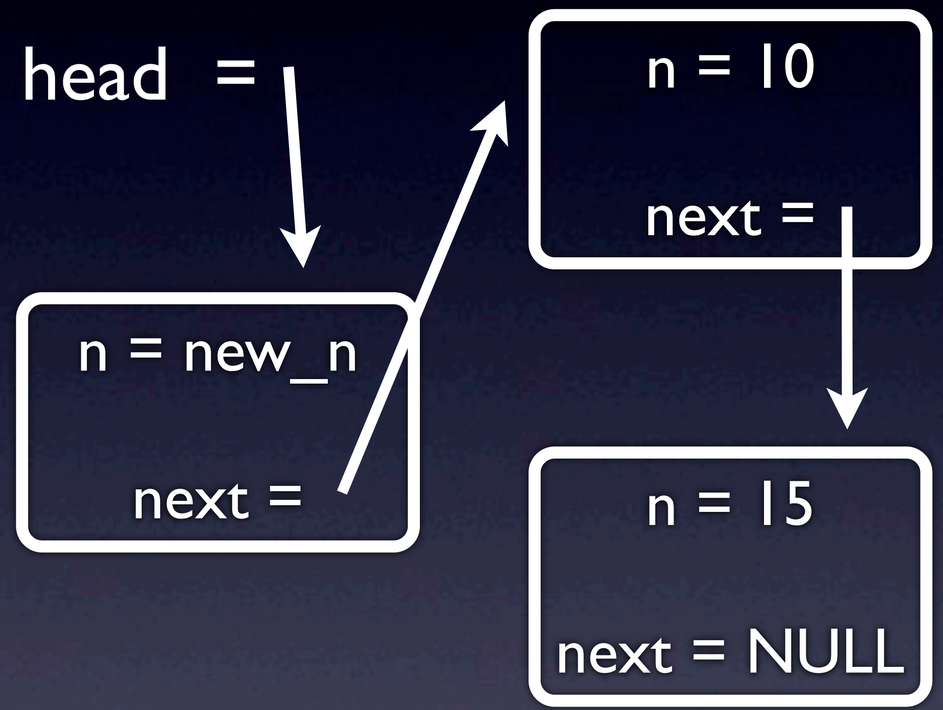
```
node* head;
void insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
       return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

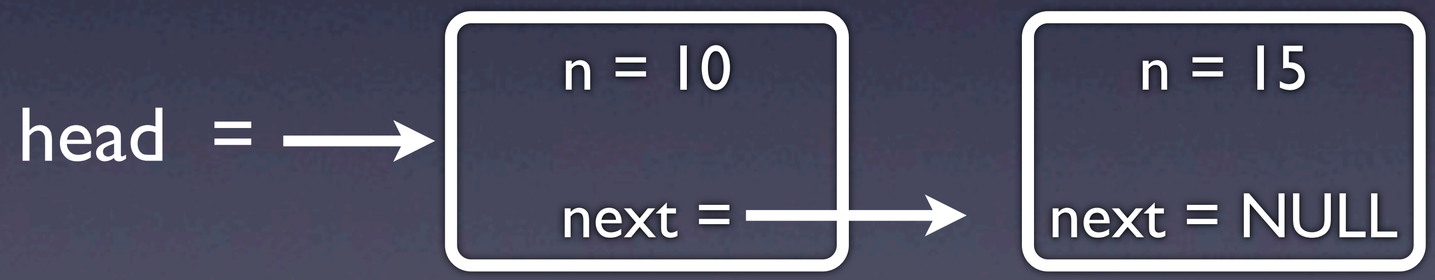
```
node* head;
void insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
       return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

- When in doubt, draw a picture!
- Try to implement delete and find!
- Also note that there are “doubly” linked lists, where each node stores a “prev” pointer too!



Stacks

Week 8 Monday

Stacks

High Level

Stacks
High Level

- "Last in, first out" - LIFO
- Two operations - push and pop
- We can implement these functions using an array.

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

How would we implement push?

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

```
stack s;
bool push(int n)
{
    s.trays[s.size] = n;
    s.size++;
    return true;
}
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Does this work?

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Fails if size == CAPACITY

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

```
stack s;
bool push(int n)
{
    if (s.size == CAPACITY)
    {
        return false;
    }

    s.trays[s.size] = n;
    s.size++;
    return true;
}
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

What else could we ask about?

- implementation of pop
- non-array implementation
- look at past quizzes!!

Queues

Week 8 Monday

Queues

- "First in, first out" - FIFO
- Two operations - enqueue, dequeue
- Again, can be implemented using an array

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

The index of the next element to dequeue (starts at 0)

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

Important things to keep track of:

- Wrapping around if front + size > CAPACITY

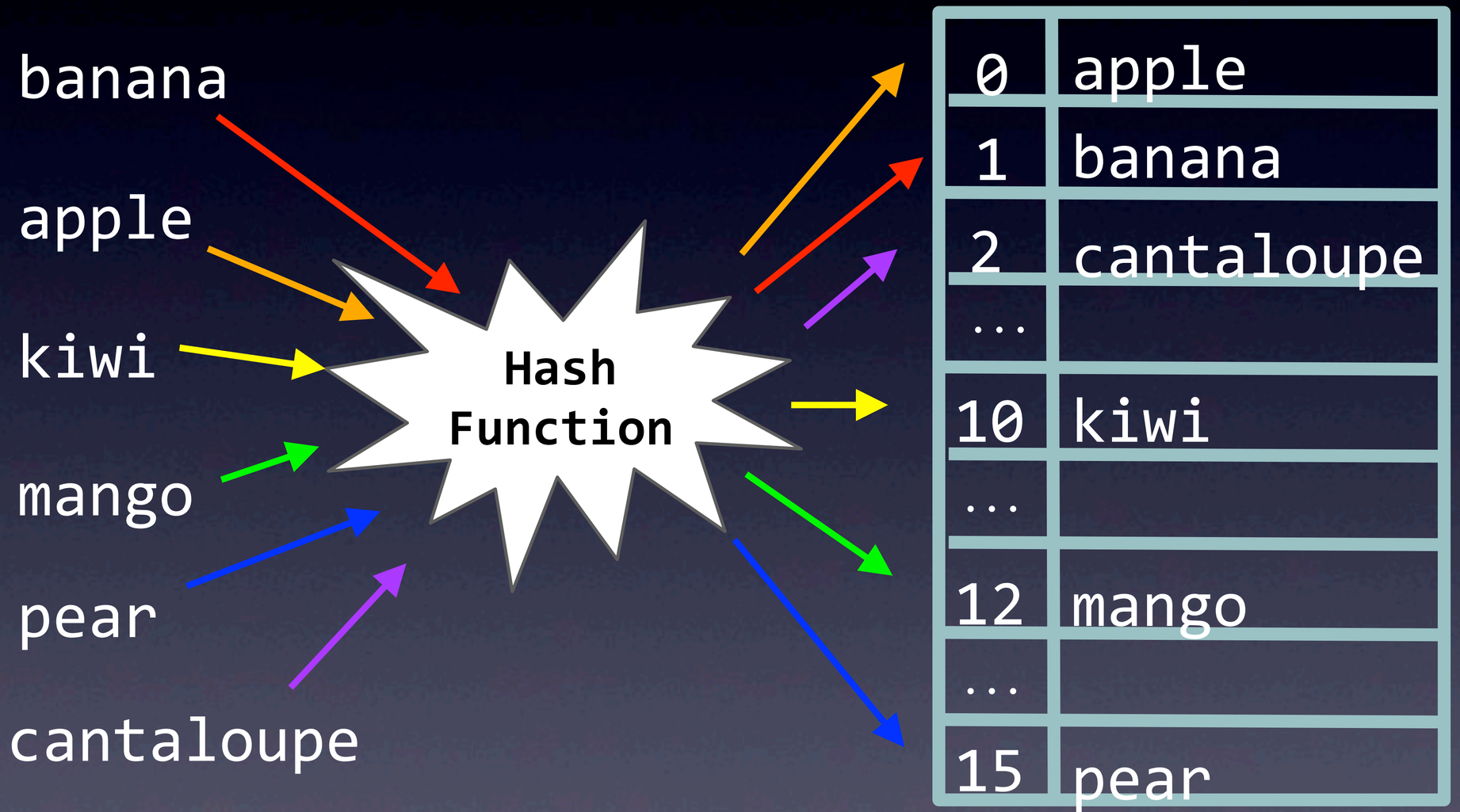
Hash Tables

Week 7 Wednesday, Section 7

Hash Tables

- A structure that aims for *O*(1) insertion and *O*(1) lookup
- In CS50, implemented as an array of linked lists
- Key component - hash function
  - Converts our input (say, a word) into a number
  - Used as an index into our array.

Hash Tables


banana
apple
kiwi
mango
pear
cantaloupe
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

Hash Tables

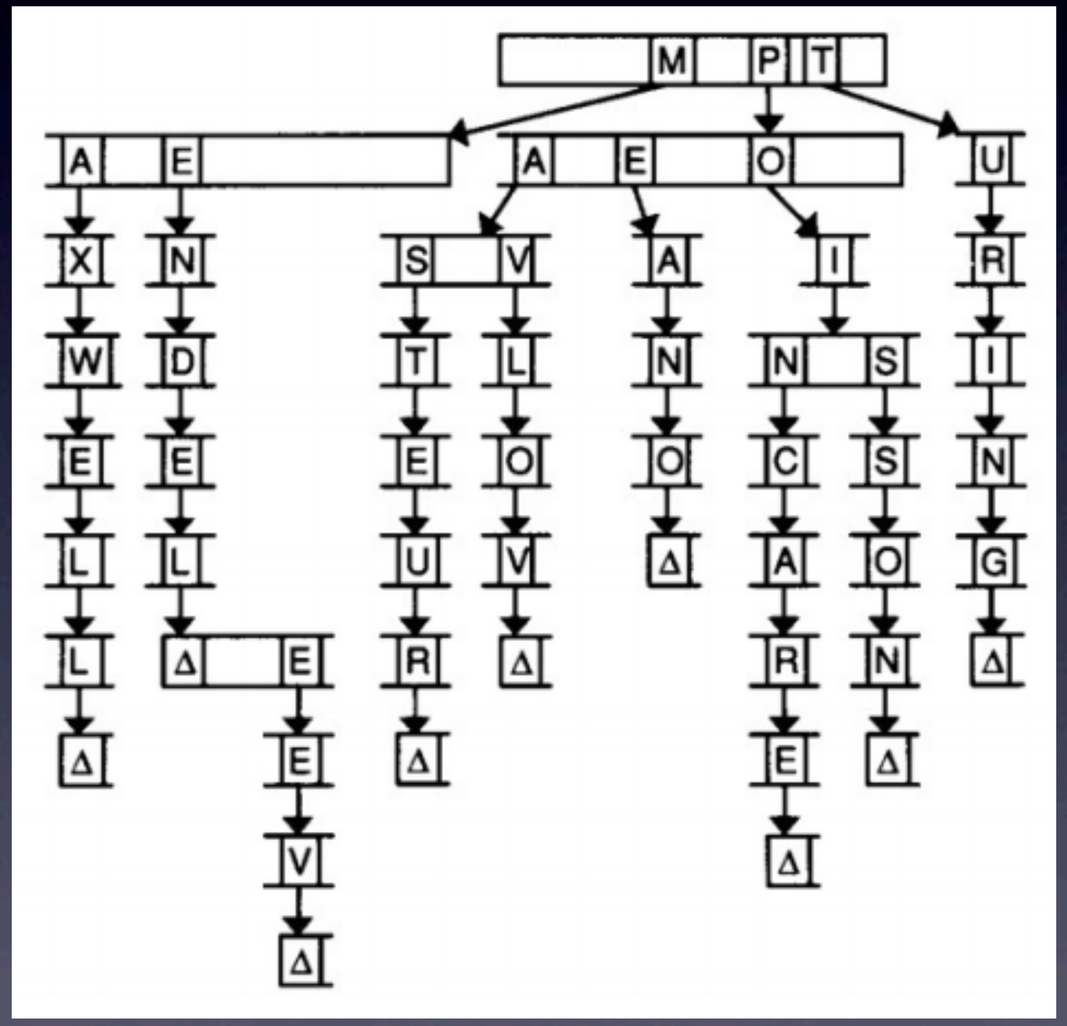
- What happens on collision?
- Instead of storing one value at, say, hashtable[3], store a linked list!
- Most of you implemented this for pset6, but check out Rob's postmortem for more implementation details!

Tries

Week 7 Wednesday

Tries

High Level



Tries

High Level

- Designed to store data alongside a keyword input, like a hash table.
- In the case of pset6, the data is “am I a word”
- Insertion and lookup in *O*(length of word)

Tries

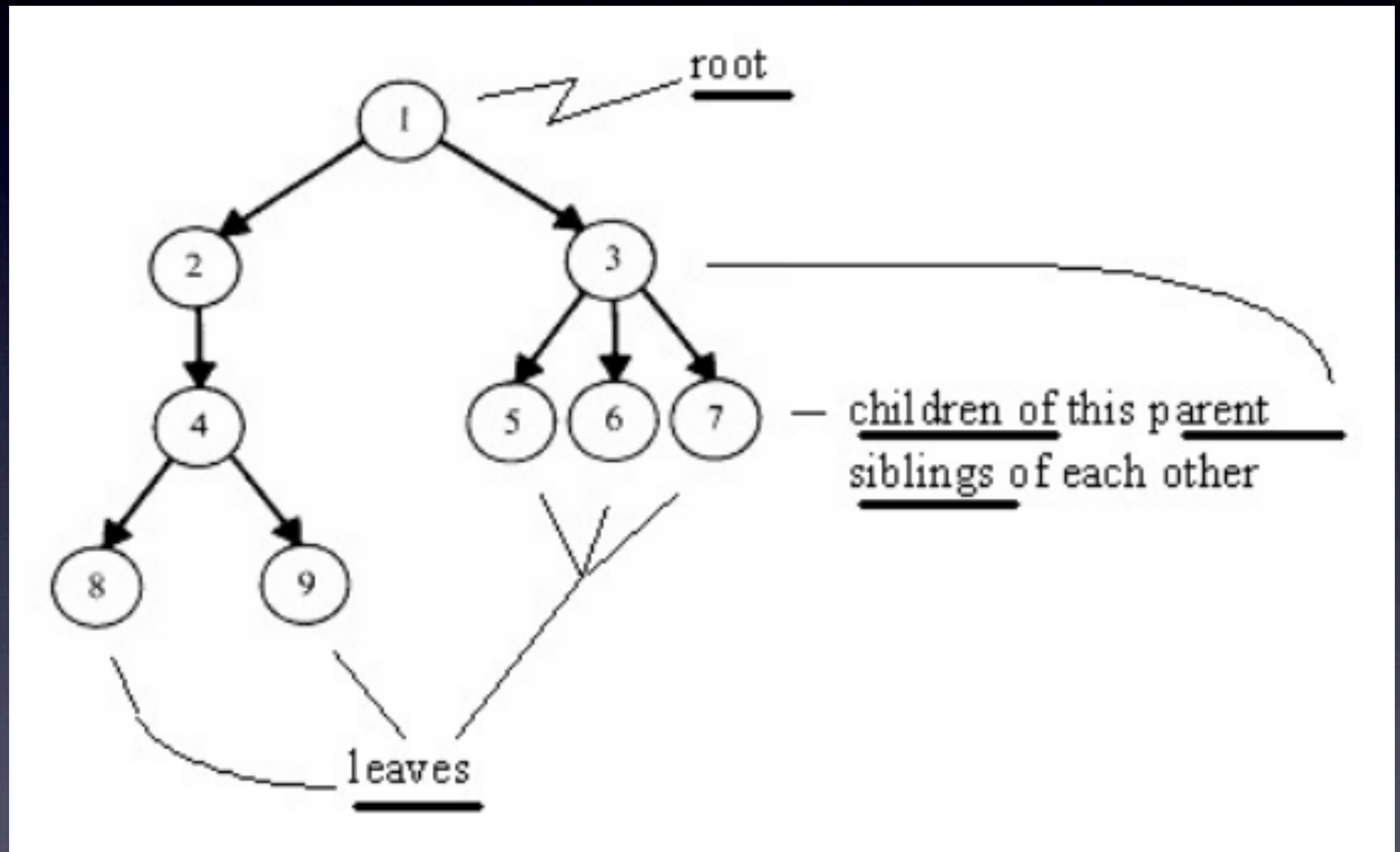
Implementation

```
typedef struct node
{
    bool is_word;
    struct node* children[27];
}
node;
```

Trees/Binary Search Trees

Week 8 Monday

Trees


root
1
2
3
4
5
6
7
8
9
children of this parent
siblings of each other
leaves

Trees

- Like a trie, a tree is a structure of nodes, where each node has 0 or more children. In a trie, we stated that each node had up to 27 children.
- A common type of tree is a “binary tree”, where each node has 0, 1, or 2 children.

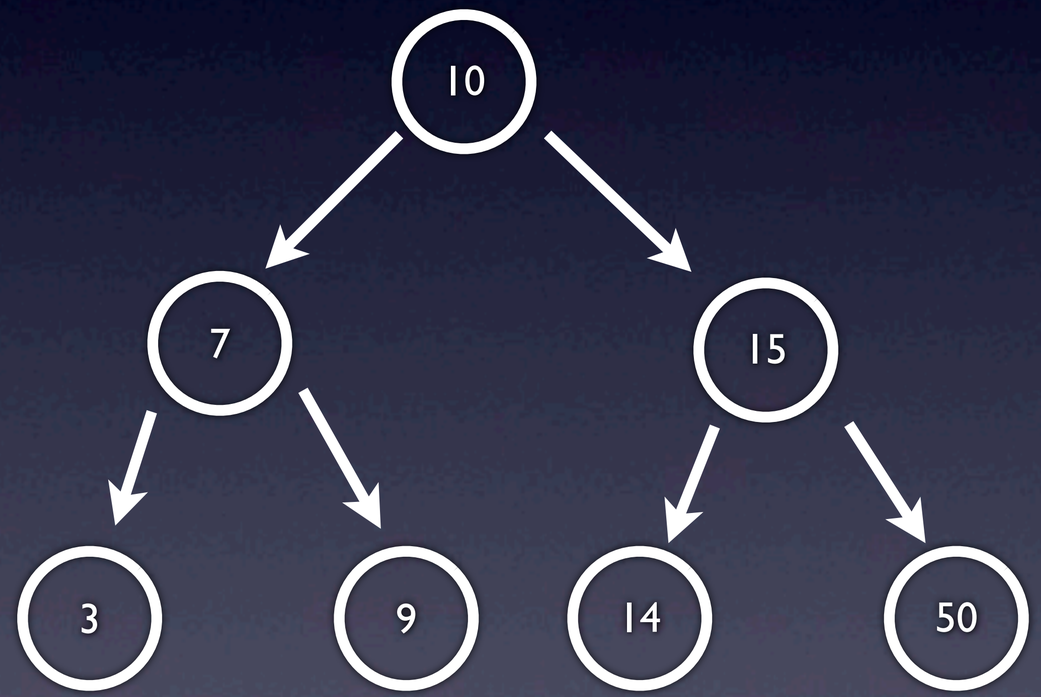
Binary Trees

```
typedef struct node
{
    int n;
    struct node* left;
    struct node* right;
}
node;
```

Binary Trees

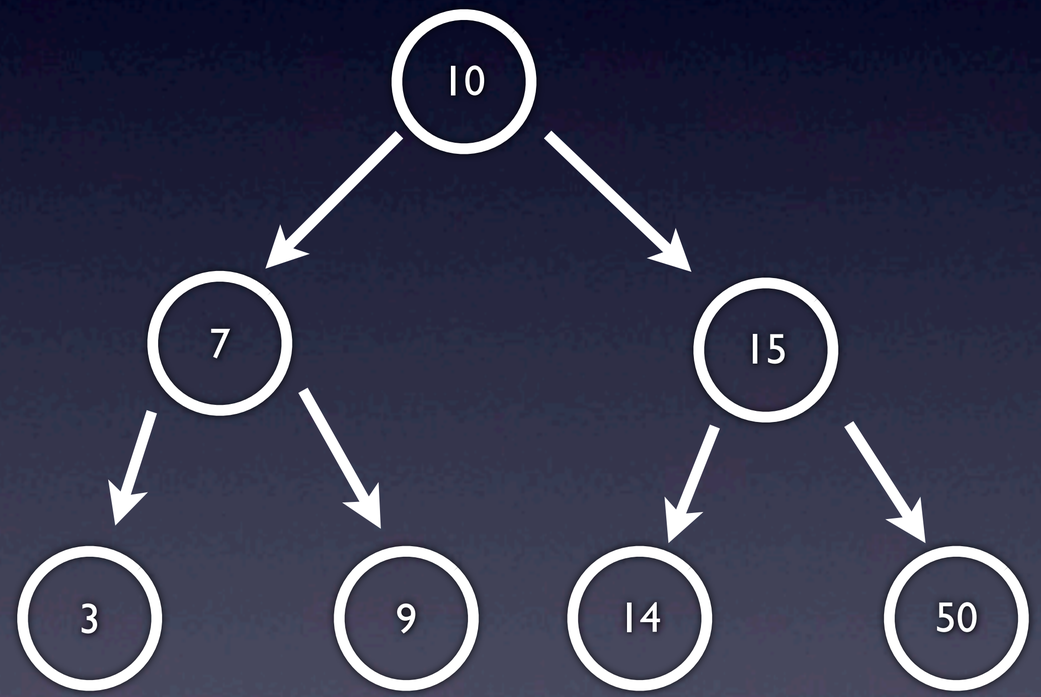
- How is a binary tree useful?
- If we make rules about where we put nodes, we can make search faster.
- In a binary search tree, all nodes on the left subtree of a node have a smaller value than the root node, and all nodes on the right subtree have a greater value than the root node.

"In a binary search tree, all nodes on the left subtree of a node have a smaller value than the root node, and all nodes on the right subtree have a greater value than the root node."



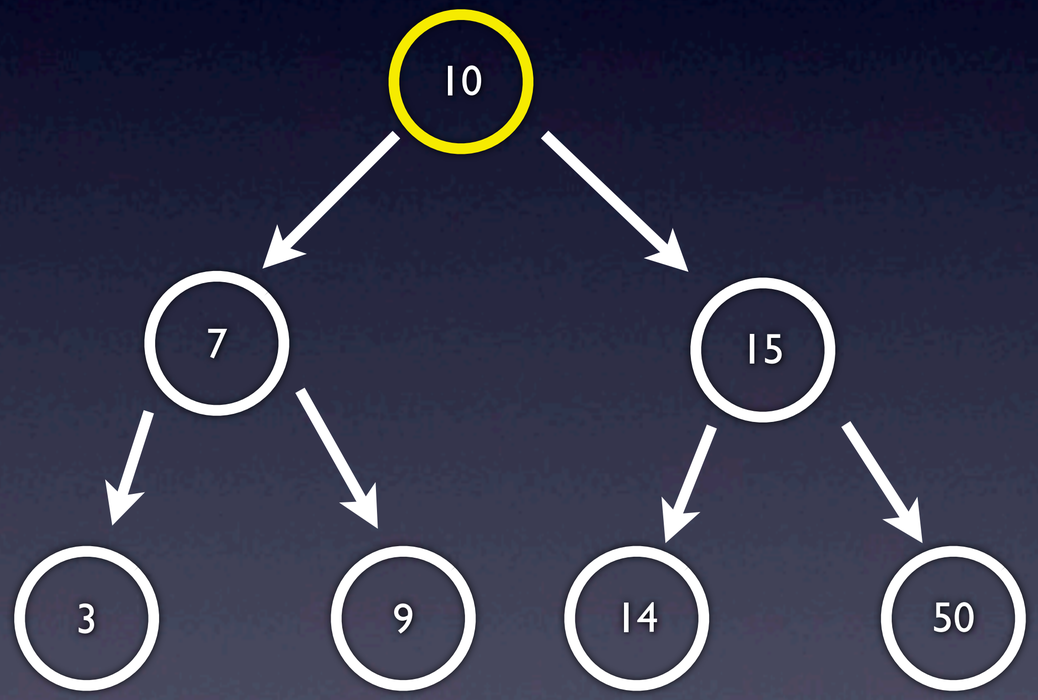
Search:

Find 14



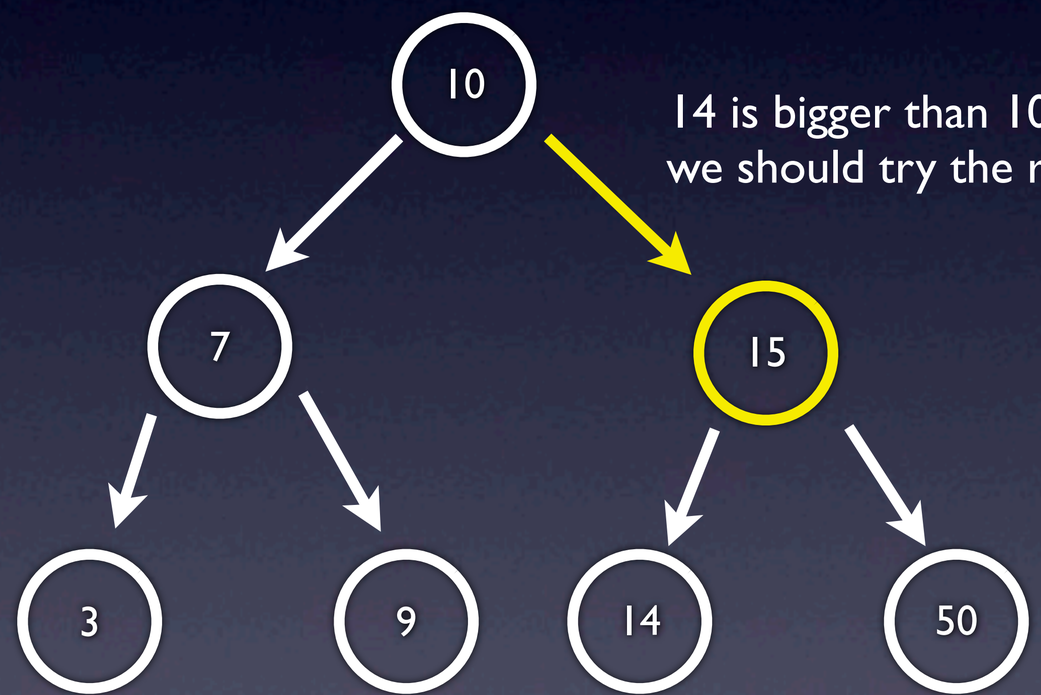
Search:

Find 14



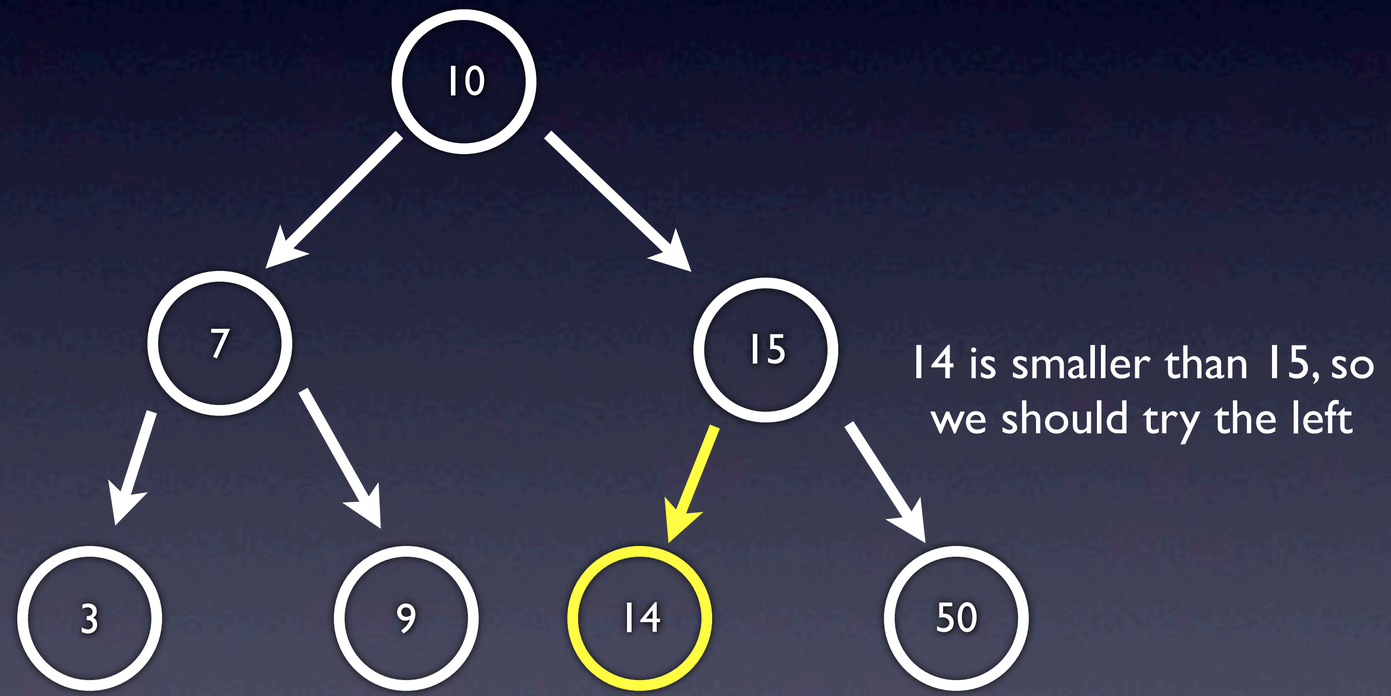
Search:

Find 14



Search:

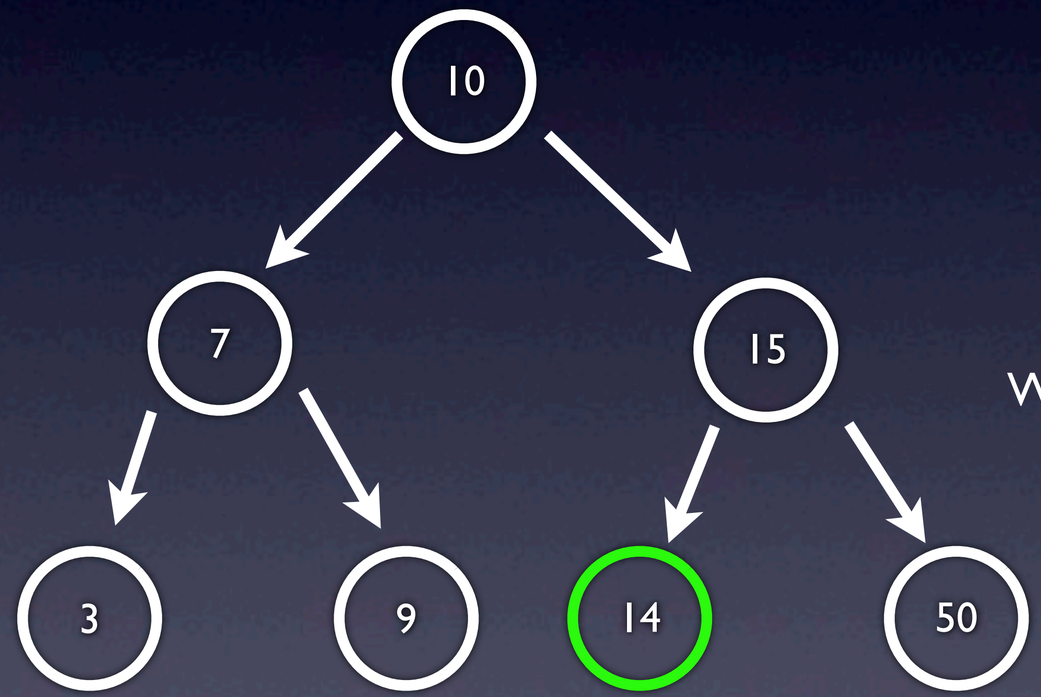
Find 14



14 is smaller than 15, so we should try the left

Search:

Find 14



We found 14!

Search:

Implementation

```
bool search(int n, node* tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }
    else
    {
        return true;
    }
}
```

BSTs

- Things we could ask you to do:
  - Write `insert`
  - Write an iterative version
  - Compare runtimes/explain when you would want to use a BST over a hashtable, for instance.

Hi there

(I'm Angela)

Don’t worry, you got this

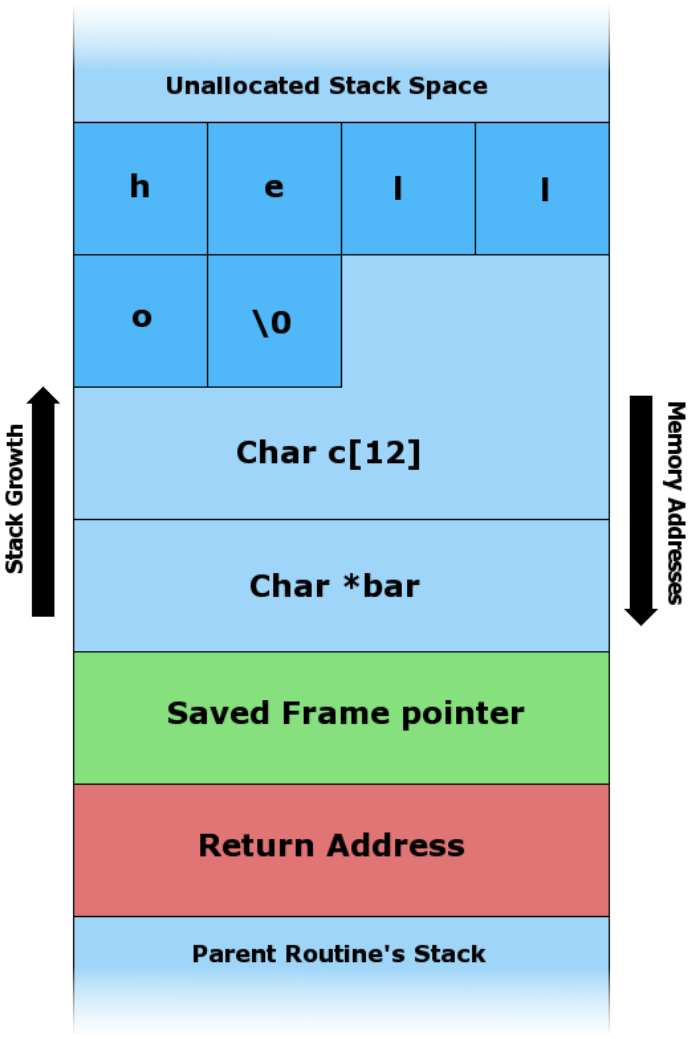
- Buffer overflow attacks
- CS50 library
- HTTP
- HTML
- CSS

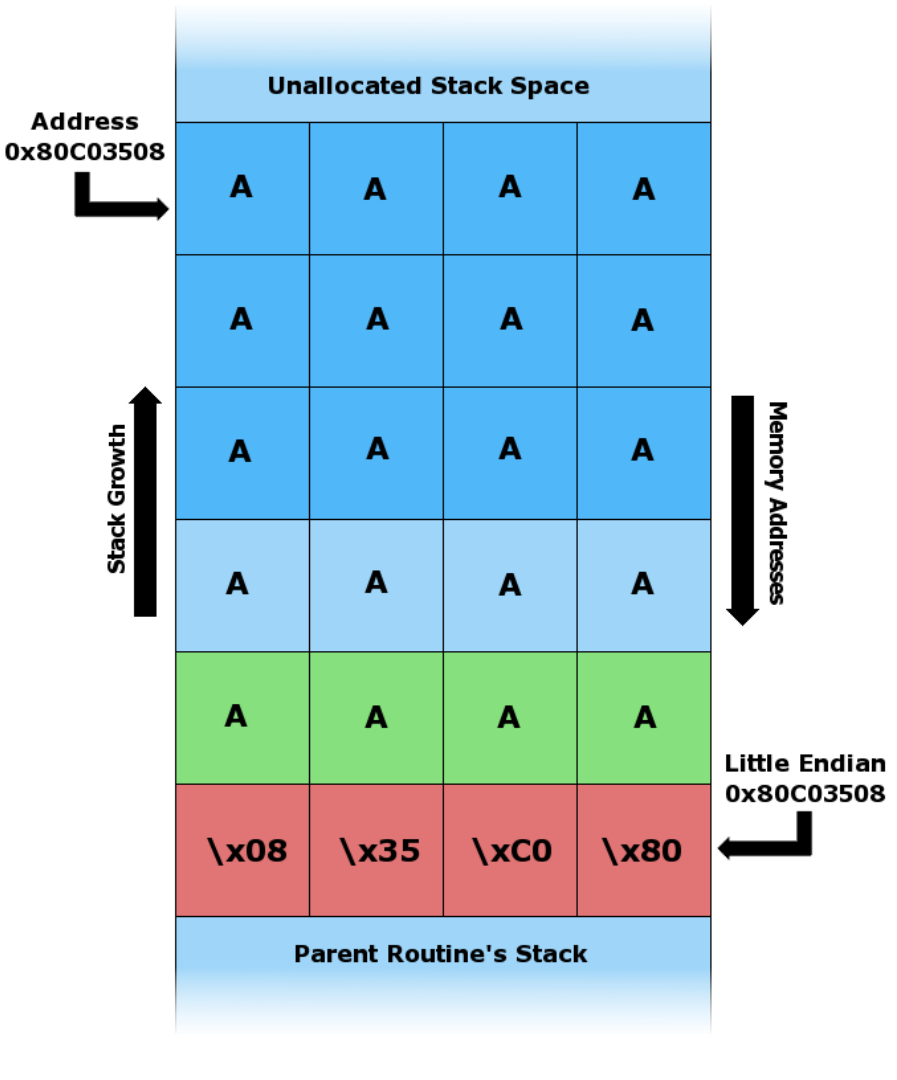
Buffer overflow

Buffer overflow

- Asking for trouble:

```
void foo(char* bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}
```


Unallocated Stack Space
h
e
l
l
o
\0
Char c[12]
Char *bar
Saved Frame pointer
Return Address
Parent Routine's Stack
Stack Growth
Memory Addresses


Unallocated Stack Space
Address
0x80C03508
A A A A
A A A A
A A A A
A A A A
A A A A
\x08 \x35 \xC0 \x80
Parent Routine's Stack
Stack Growth
Memory Addresses
Little Endian
0x80C03508

CS50 library

CS50 library

- `#include <cs50.h>`
- `GetString(), GetInt(), GetFloat(), …`
- `typedef char* string;`

CS50 library

```
string GetString(void)
{
    // …
    while ((c = fgetc(stdin)) != '\n' && c != EOF)
    {
        if (n + 1 > capacity)
            buffer = realloc(buffer, capacity * 2);
        buffer[n++] = c;
    }
    // …
}
```

CS50 library

```
int GetInt(void)
{
    while (true)
    {
        string line = GetString();
        if (line == NULL)
            return INT_MAX;

        int n; char c;
        if (sscanf(line, " %i %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}
```

HTTP

HTTP

- **H**yper**T**ext **T**ransfer **P**rotocol
- Heart and soul of the world wide web
- **Hypertext**: HTML!
- **Transfer**: requests (from clients) and responses (from servers)
- **Protocol**: set of standards or conventions

HTTP requests

- GET /me HTTP/1.1

  Host: www.facebook.com

  …

HTTP responses

- HTTP/1.1 200 OK

  Content-Type: text/html; charset=UTF-8

  Server: Apache

  …

HTTP status codes

- 200 OK
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- *Angela's favorite:* 418 I'm a teapot

HTML

HTML

- **H**yper**T**ext **M**arkup **L**anguage
- Defines **structure** and **semantics** of webpages
  - (Not style—that's CSS!)
- Series of nested tags, or elements

HTML example

```
<!DOCTYPE html>

<html>
    <head>
        <title>Hello, world!</title>
        <link href="style.css" rel="stylesheet" />
        <script src="script.js"></script>
    </head>
    <body>
        <p><img alt="Kittens." src="kittens.jpg" /></p>
    </body>
</html>
```

CSS

CSS

- **C**ascading **S**tyle **S**heets
- Used to **style** webpages
    - (Webpage **structure** first defined with HTML)

Including CSS

- Acceptable method: between <style> tags

```
<html>
    <head>
        <style>
            /* CSS here */
        </style>
    </head>
    <body></body>
</html>
```

Including CSS

- Better method: linked external stylesheet

```
<link href="style.css" rel="stylesheet" />
```

- In **style.css**:

```
body
{
    background-color: #000000;
    /* ... */
}
```

Including CSS

- Bad method: inline styles

```
<div style="background: #000000; color: #FFFFFF"></div>
```

- Avoid inline styling!
  - Why?
  - Muddles **structure** (HTML) and **style** (CSS)

CSS example

```
body
{
    font-family: "Comic Sans";
}

#main
{
    margin: 20px;
    text-align: center;
}

.section
{
    background-color: #61D2D6;
}
```

PHP, MVC, SQL

PHP

- PHP Hypertext Preprocessor
- server-side scripting language
- backend & logical underpinnings of our website

PHP Syntax

- must start with `<?php` and end with `?>`

```
<?php
    // code goes here
?>
```

PHP variables

- start with $
- loosely typed
  - That doesn't mean that there aren't types - you just don't declare them!
- global scope (except for functions)
  - loops don't limit the scope of variables

```
<?php
    $number_int = 1;
    $number_string = "1";
    $number_float = 1.0;
?>
```

loops and conditions: if

```
if ($name == "Harry")
{
    printf("Yer a wizard!");
}
```

loops and conditions: while

```
while ($voldy == false)
{
    printf("All was well");
}
```

loops and conditions: for

```
for ($i = 0; $i <= 8; $i++)
{
    submitPset($i);
}
```

loops and conditions: for

```
$psets = [0, 1, 2, 3, 4, 5, 6, 7, 8];

foreach ($psets as $pset_i)
{
    submitPset($pset_i);
}
```

arrays

indices	0	1	2	…
values	1	2	12	…

```
$my_list = [];
$my_list[0] = 1;
$my_list[1] = 2;
$my_list[2] = 12;
…
```

associative arrays

indices	"cats"	"owls"	"dogs"	…
values	1	2	12	…

```
$my_list = [];
$my_list["cats"] = 1;
$my_list["owls"] = 2;
$my_list["dogs"] = 12;
…
```

string concatenation: . operator

```
<?php
	$first = "Milo";
	$last = "Banana";

	$fullname = $first . " " . $last;

	// prints "Milo Banana"
	echo $fullname;
?>
```

useful PHP functions

`require(filename.php)`

- includes PHP code from the specified file and evaluates it

`echo`

`exit`

`empty`

PHP and HTML

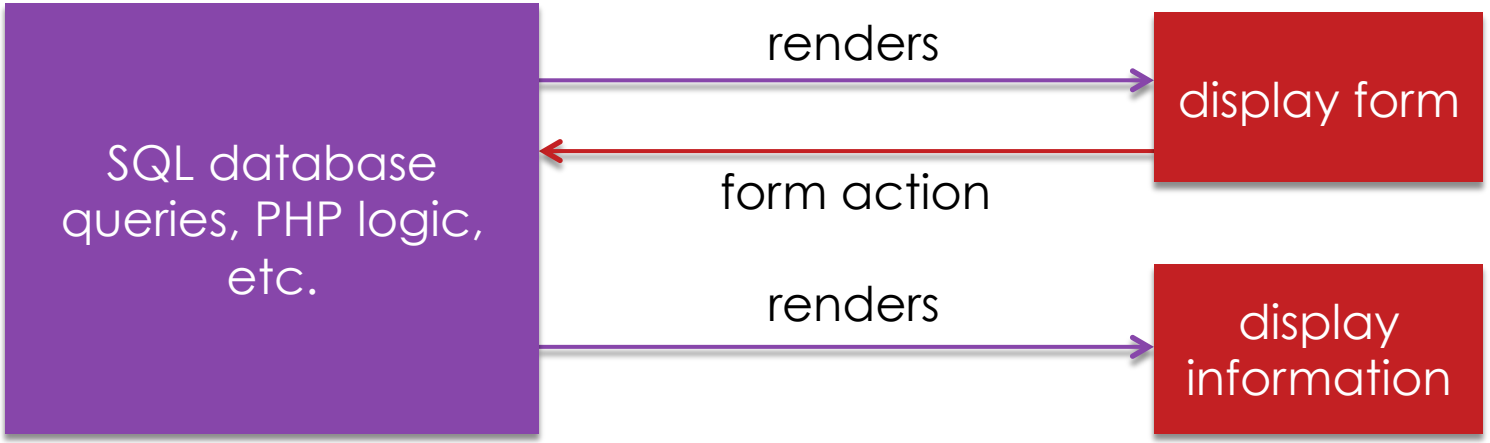
- PHP gives us the power to alter a page's HTML prior to loading, based on different conditions

```
<?= "Hello, " . $name . "!" ?>
```

MVC

- Model
  - interactions with database
- View
  - aesthetics of the website
- Controller
  - handles user requests; passes data
- paradigm for organizing code

MVC


SQL database queries, PHP logic, etc.
renders
display form
form action
renders
display information

render

public/hello.php

```
render("hello.php",
  ["msg" => "Who's a good boy?",
   "name" => "Milo"]);
```

templates/hello.php

```
<p>
  <?= htmlspecialchars($msg) ?>
  <?= htmlspecialchars($name) ?> is!
</p>
```

render

public/hello.php

```
render("hello.php",
  ["msg" => "Who's a good boy?",
   "name" => "Milo"]);
```

templates/hello.php

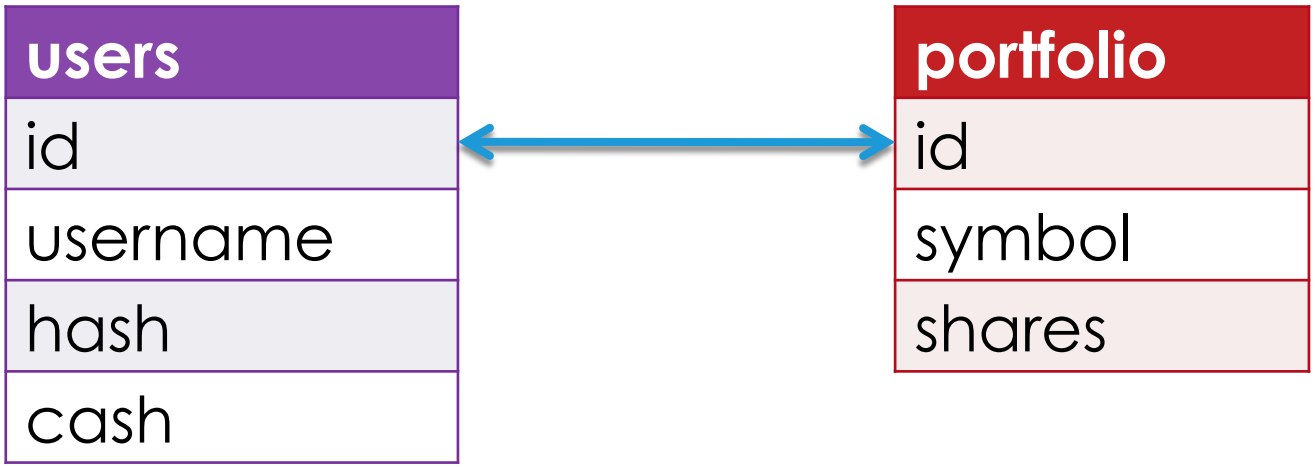
```
<p>
  Who's a good boy?
  Milo is!
</p>
```

SQL

- Structured Query Language
- programming language designed for managing databases

why SQL?

- easy way to track and manage objects



how?

SELECT

INSERT

UPDATE

DELETE

SELECT

```
SELECT * FROM wizards WHERE house = "Ravenclaw";
```

query

- ? is a placeholder

```
$rows = query("SELECT * FROM wizards WHERE house = ?", $curr_house);
if ($rows === false)
{
    apologize("Query failed");
}
```

SELECT

- `query` returns an array of rows

```
foreach ($rows as $row)
{
    print("<tr>");
    print("<td>{$row["Name"]}</td>");
    print("<td>{$row["Year"]}</td>");
    print("<td>{$row["House"]}</td>");
    print("</tr>");
}
```

INSERT

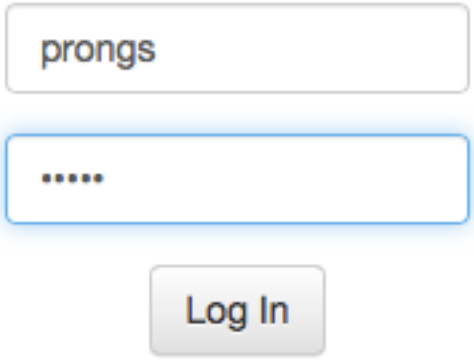
```
INSERT INTO wizards (name, year, house)
    VALUES("zamyla", 7, "Ravenclaw")

INSERT INTO gringotts (id, galleons)
    VALUES(1, 500) ON DUPLICATE KEY UPDATE
    galleons = galleons + VALUES(galleons)
```

UPDATE, DELETE

```
UPDATE gringotts SET galleons = galleons - 5
    WHERE id = 1

DELETE FROM gringotts WHERE id = 1
```

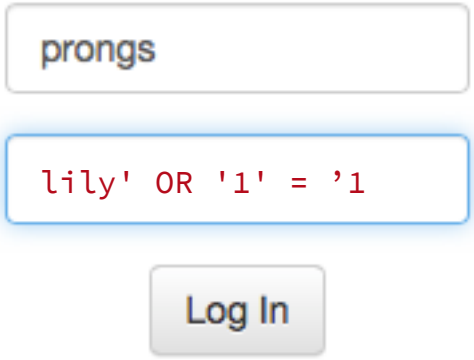
SQL Injection Attack

```
$username = $_POST["username"];
$password = $_POST["password"];
$rows = query("SELECT * FROM users WHERE username='$username'
       AND password='$password'");
```



SQL Injection Attack

```
$username = $_POST["username"];
$password = $_POST["password"];
$rows = query("SELECT * FROM users WHERE username='$username'
       AND password='$password'");
```




```
SELECT * FROM users WHERE username='prongs'
       AND password='lily' OR '1' = '1'
```

CS50 Quiz 1 Review

this is a cat

JavaScript

CS50 Quiz 1 Review

first, recall from zamyla

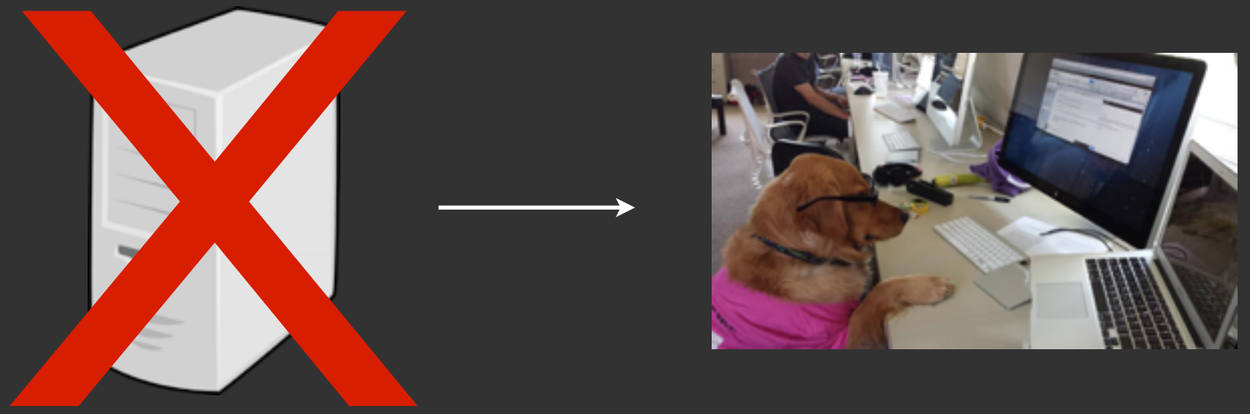
- Remember, PHP is run **server-side**. The HTML output of this PHP code is sent to the user.

Server
(PHP)

Browser

first, recall from zamyla

- Remember, PHP is run **server-side**. The HTML output of this PHP code is sent to the user.
- No more PHP code can be executed after this HTML output is sent, unless you reload!

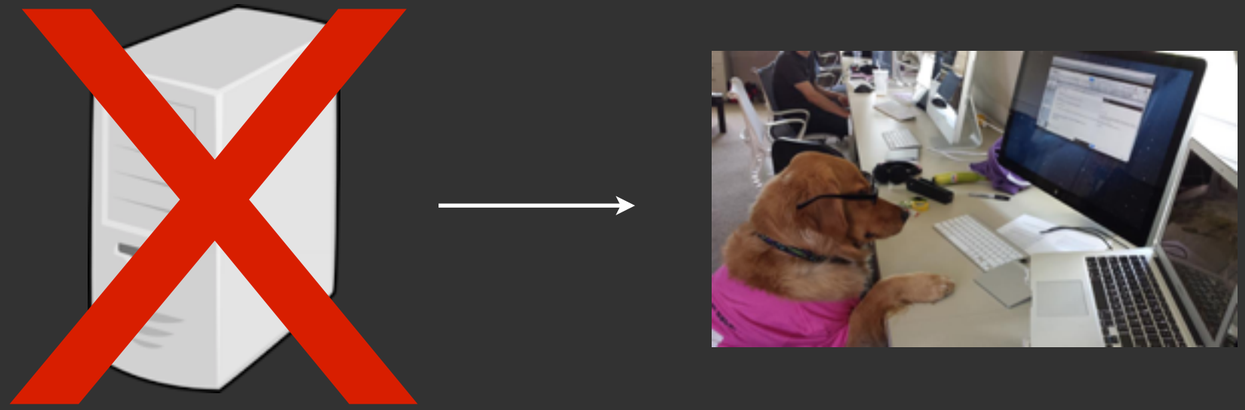


Server
(PHP)

Browser

why javascript?

- What if want something to happen when a user interacts with the page?
  - clicks on a button; presses a key; resizes the page?



Server
(PHP)

Browser
(JavaScript)

- JavaScript fulfills this role. It's a **client-side** programming language,
  - used to control interactions and logic in a web browser.

JavaScript != PHP

some different **syntax**, very different **uses**

syntax differences

```javascript
var i = 0;

var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

milo.owner = "Lauren";
milo["owner"] = "Lauren";

console.log(milo);

console.log(milo.owner + " rocks!");
```

JavaScript

```php
$i = 0;

$milo = [
    "name"  => "Milo Banana",
    "house" => "CS50 House",
    "role"  => "Mascot"
];

$milo["owner"] = "Lauren";


print_r($milo);

print($milo["owner"] . " rocks!")
```

PHP

control flow differences

```
var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

for (var i in milo)
{
    console.log(i);
}
```

```
name
house
role
```

JavaScript, keys

```
$milo = [
    "name"  => "Milo Banana",
    "house" => "CS50 House",
    "role"  => "Mascot"
];

foreach ($milo as $i)
{
    print($i);
}
```

```
Milo Banana
CS50 House
Mascot
```

PHP, values

control flow differences

```
var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

for (var i in milo)
{
    console.log(milo[i]);
}
```

```
Milo Banana
CS50 House
Mascot
```

JavaScript, values

```
$milo = [
    "name"  => "Milo Banana",
    "house" => "CS50 House",
    "role"  => "Mascot"
];

foreach ($milo as $key => $value)
{
    print($key);
}
```

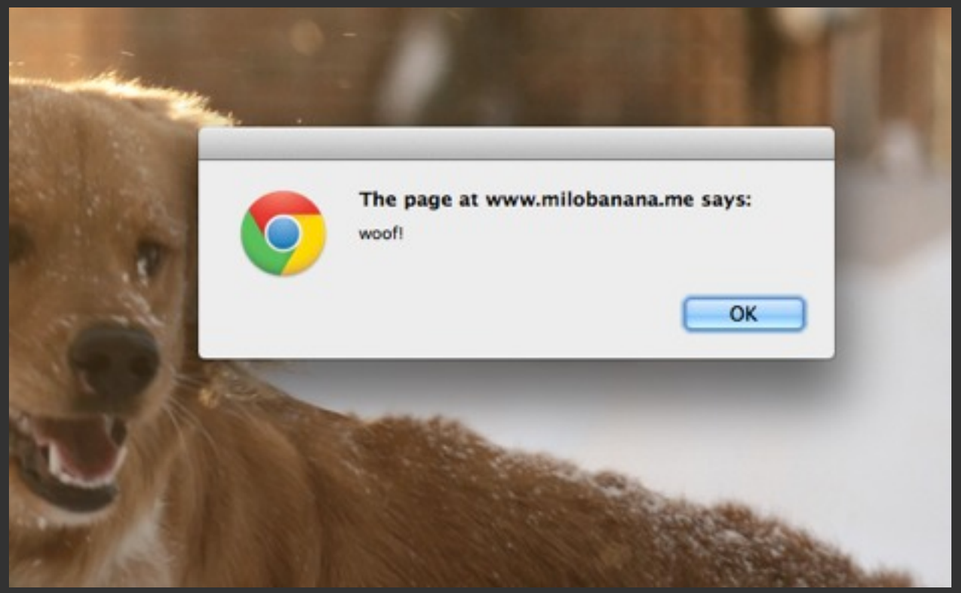
```
name
house
role
```

PHP, keys

JS objects

- You already know it's a mapping of keys to values. An associative array.
- But realize that values can be anything -- even a function!

```
var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot",
    "bark":  function() {
        alert("woof!");
    }
};

milo.bark();
```



"loosely typed"

- JavaScript **does have types**. We just don't explicitly state them in declarations and the like.
- We can freely convert between types.

```
var i = 123;
i = "hello" + i;  // gives us the string "hello123" in i
```

- We can also freely compare between types using ==.

```
console.log("123" == 123);
true
```

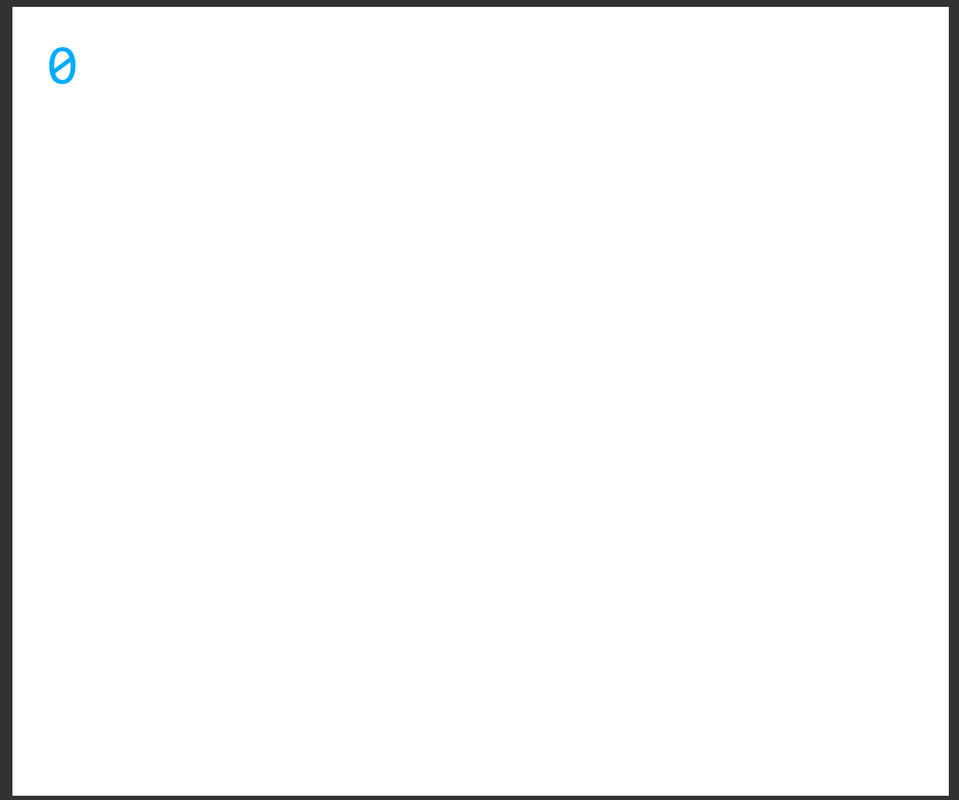
- You can enforce comparison with types by using ===.

```
console.log("123" === 123);
false
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```



global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
3
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
3
4
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

!!! ⟶  console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
3
4
5
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
3
4
5
6
```

global scope, nooooo

except in **functions**! phew!

```
function foo(i)
{
    i++;
    console.log(i);
}

var i = 999;
for (var i = 0; i < 5; i++)
{
    console.log(i);
}

console.log(i);
foo(i);
console.log(i);
```

```
0
1
2
3
4
5
6
5
```

objects are passed by reference

(similar to arrays in C)

```
function cattify(object) {
    object.name = "cat";
}

var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

→ console.log(milo.name);
  cattify(milo);
  console.log(milo.name);
```

```
Milo Banana
```

objects are passed by reference

(similar to arrays in C)

```
function cattify(object) {
    object.name = "cat";
}

var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

console.log(milo.name);
cattify(milo);
console.log(milo.name);
```

```
Milo Banana
cat
```

objects are passed by reference

(similar to arrays in C)

```
function cattify(object) {
    object.name = "cat";
}

var milo = {
    "name":  "Milo Banana",
    "house": "CS50 House",
    "role":  "Mascot"
};

console.log(milo.name);
cattify(milo);
console.log(milo.name);
```

```
Milo Banana
cat
```

D:

using javascript in a webpage

```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
    <script>
      alert("I executed!");
    </script>
  </head>
  <body>
    <div>hello world</div>
  </body>
</html>
```

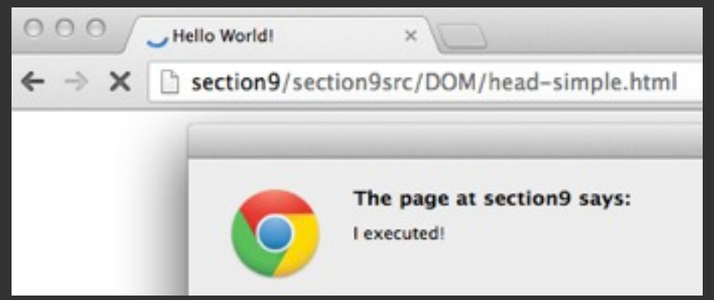
inline JS, works, but messy

```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
    <script src="path/to/file.js"></script>
  </head>
  <body>
    <div>hello world</div>
  </body>
</html>
```

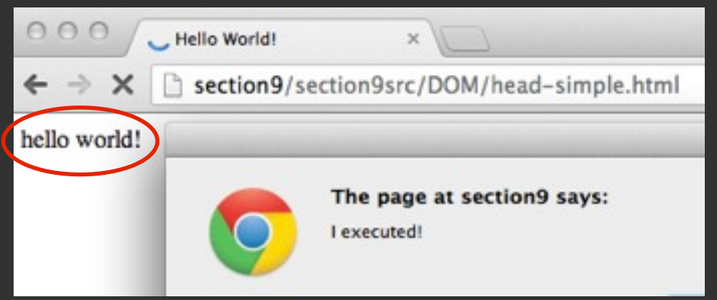
JS in external file, ahhhh … nice.

placement matters

```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
    <script>
      alert("I executed!");
    </script>
  </head>
  <body>
    <div>hello world</div>
  </body>
</html>
```




```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <div>hello world</div>
    <script>
      alert("I executed!");
    </script>
  </body>
</html>
```



how to wait until page loaded?

event handlers, hold the thought, we'll look at those when we get to jQuery

DOM

CS50 Quiz 1 Review

motivation

- So, if you look at HTML source code, it's just a bunch of text.

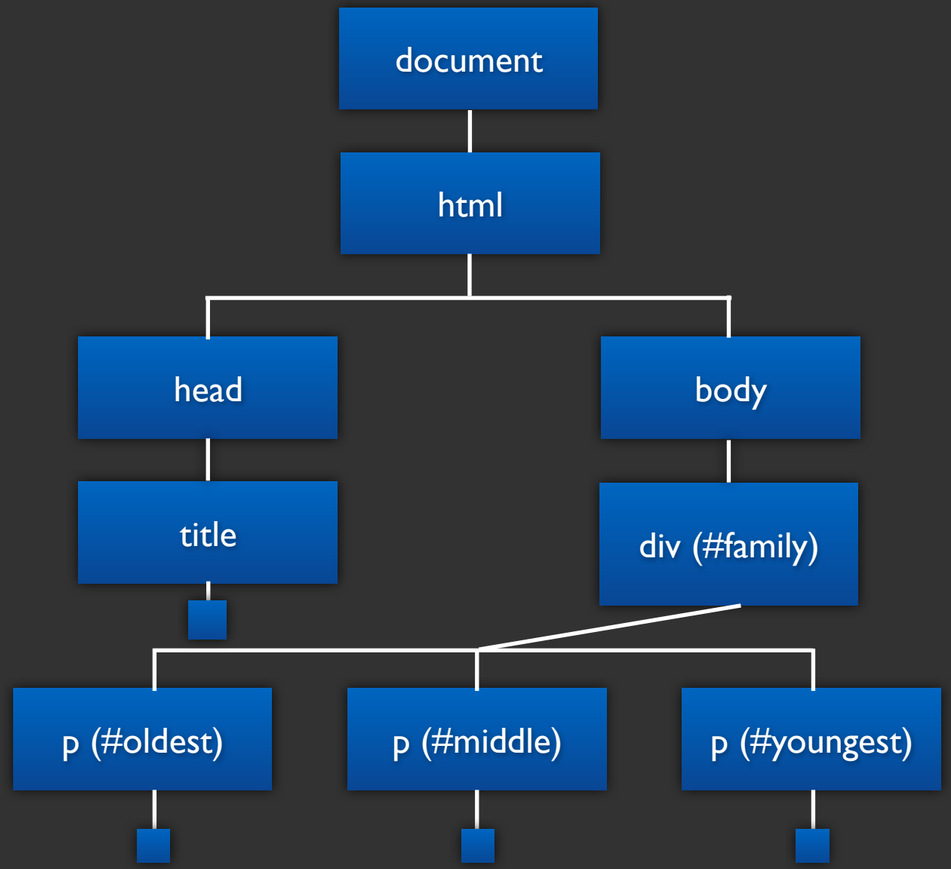
```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <div>hello world</div>
  </body>
</html>
```

- However, JavaScript has to be able to select and modify HTML elements.
  - We need an in-memory representation of these elements.

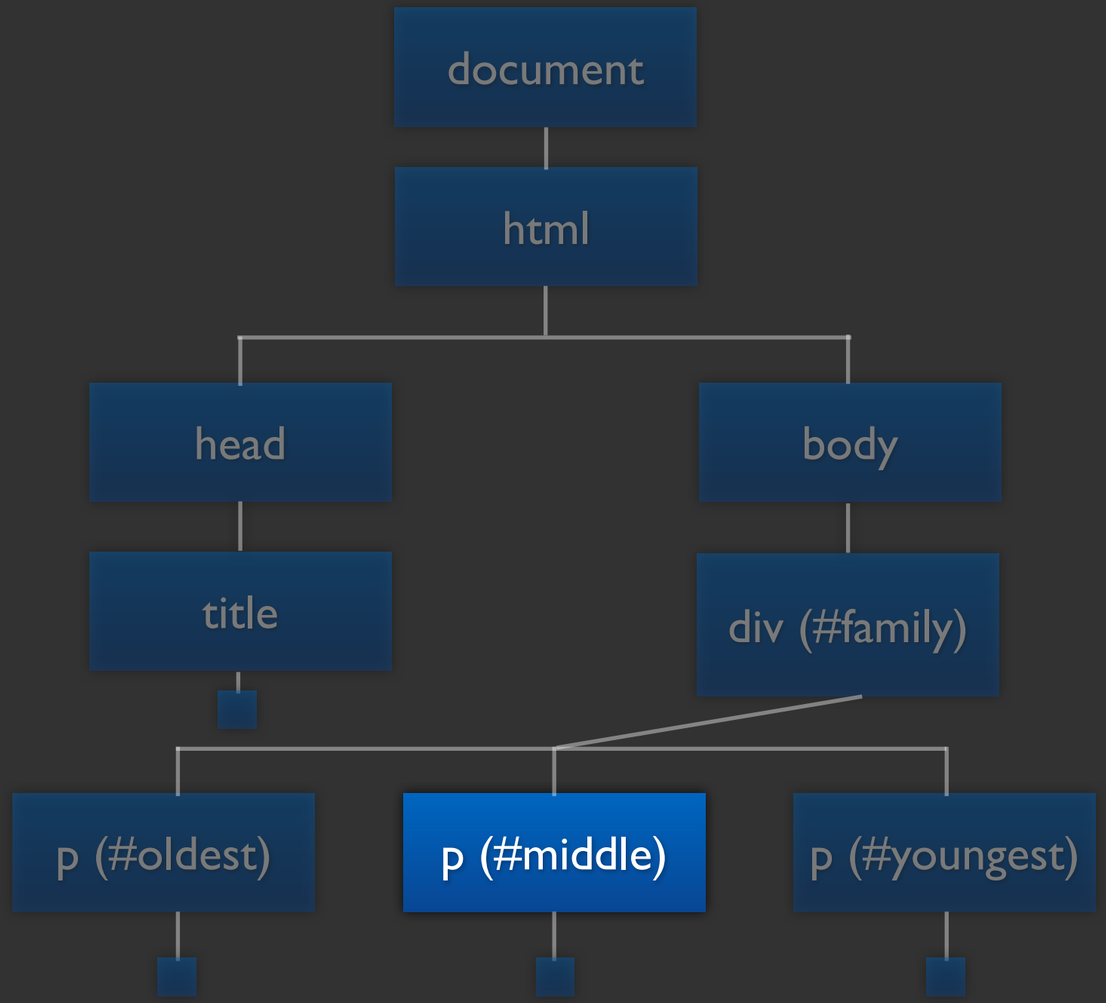
DOM

- This in-memory representation of HTML elements is the DOM. It's a DOM tree!

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```



working with the DOM



Select DOM element

Do stuff to element

- Change html attributes of element
- Change css of element
- Change inner html of element
- Attach an event handler to an element

working with the DOM

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Milo</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Select DOM element

Do stuff to element

- Change html attributes of element
- Change css of element
- Change inner html of element
- Attach an event handler to an element

what does this look like in JavaScript code?

Select DOM element

Do stuff to element

- Change html attributes of element
- Change css of element
- Change inner html of element
- Attach an event handler to an element

jQuery

CS50 Quiz 1 Review

what is jQuery?

- A JavaScript **library** that makes JavaScript easier to write.

- What are some of the things it does?
  - Makes selecting elements easier.
  - Make changing HTML, adding classes to elements, creating elements, easier.
  - Makes writing Ajax requests easier (we'll cover this soon).

- Analogous to how `<string.h>` is a C library.
  - Makes dealing with strings easier (gives you `strlen`, `strcpy`, etc.)

jQuery != JavaScript

jQuery is a **library** written in JavaScript

jQuery != JavaScript

jQuery is **not** a programming language, JavaScript **is**

how you invoke jQuery.

it is not PHP's $.

Selecting DOM Elements

Select DOM element → Do stuff to element

- How do we select HTML elements, once we have the DOM?
- Nasty regular JavaScript way, using the `document` variable.

```
→ document.getElementById("family");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
  → <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Selecting DOM Elements

Select DOM element → Do stuff to element

- How do we select HTML elements, once we have the DOM?
- Nasty regular JavaScript way, using the `document` variable.

```
  document.getElementById("family");
→ document.getElementsByTagName("p");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
  → <p>Look at all the children!</p>
    <div id="family">
    → <p id="oldest">Alice</p>
    → <p id="middle">Bob</p>
    → <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Selecting DOM Elements

Select DOM element → Do stuff to element

- How do we select HTML elements, once we have the DOM?
- Nasty regular JavaScript way, using the `document` variable.

```
→ document.getElementById("family");
  document.getElementsByTagName("p");
```

- Nice, simplified way, using a library we call jQuery.

```
→ $("#family");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
→   <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Selecting DOM Elements

Select DOM element → Do stuff to element

- How do we select HTML elements, once we have the DOM?
- Nasty regular JavaScript way, using the `document` variable.

```
  document.getElementById("family");
→ document.getElementsByTagName("p");
```

- Nice, simplified way, using a library we call jQuery.

```
  $("#family");
→ $("p");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
  → <p>Look at all the children!</p>
    <div id="family">
    → <p id="oldest">Alice</p>
    → <p id="middle">Bob</p>
    → <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Selecting DOM Elements

Select DOM element → Do stuff to element

- How do we select HTML elements, once we have the DOM?
- Nasty regular JavaScript way, using the `document` variable.

```
document.getElementById("family");
document.getElementsByTagName("p");
```

- Nice, simplified way, using a library we call jQuery.

```
$("#family");
$("p");
→ $("#family p");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
    → <p id="oldest">Alice</p>
    → <p id="middle">Bob</p>
    → <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

look familiar?

jQuery uses CSS Selectors

```
$("#family");
$("p");
$("#family p");
```

Using DOM Elements

Select DOM element → Do stuff to element

- jQuery also augments DOM elements with extra **methods**, or functions, that make using them easier.
- Let's change an DOM element's HTML with a jQuery method.

```
→ var element = $("#middle");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
    → <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- jQuery also augments DOM elements with extra **methods**, or functions, that make using them easier.

- Let's change an DOM element's HTML with a jQuery method.

```
  var element = $("#middle");
→ element.html("Milo");
```

"method"

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
→     <p id="middle">Milo</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- jQuery also augments DOM elements with extra **methods**, or functions, that make using them easier.

- Let's add a class to a DOM element.

```
  var element = $("#family");
→ element.addClass("shadow");
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
  →<div id="family" class="shadow">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- jQuery also augments DOM elements with extra **methods**, or functions, that make using them easier.

- Let's remove an element from the DOM.

```
  var element = $("#youngest");
→ element.remove();
```

- You can also shorten this to just:

```
$("#youngest").remove();
```

↑ Select DOM element

↑ Do stuff to element

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- jQuery also augments DOM elements with extra **methods**, or functions, that make using them easier.

- Hmmmm, but is it as easy as doing something like this?
  - Remember, execution order, why does it matter?

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
    <script>
       $("#youngest").remove();
    </script>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- JavaScript is **event-driven**. We can go about using events to help solve the problem on the previous slide.

- Let's attach a **ready event handler** to the document, so JS code executes only once the document is ready (HTML has loaded).

```
$(document).ready(foo);
```

↑ select document

↑ when what we selected is ready

↑ run this function

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
    <script>
        $(document).ready(function(event) {
            $("#youngest").remove();
        });
    </script>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Using DOM Elements

Select DOM element → Do stuff to element

- JavaScript is **event-driven**. We can go about using events to **add interactivity to our website**.

- More compellingly, let's attach a **click event handler** this time to do something when the user clicks on Charlie.

```
$("#youngest").click(foo);
```

↑ select document

↑ when what we selected is clicked

↑ run this function

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
    <script>
        $(document).ready(function(event) {
            $("#youngest").click(function(event) {
                alert("I'm Charlie!");
            });
        });
    </script>
  </head>
  <body>
    <p>Look at all the children!</p>
    <div id="family">
      <p id="oldest">Alice</p>
      <p id="middle">Bob</p>
      <p id="youngest">Charlie</p>
    </div>
  </body>
</html>
```

Combining Everything

let's do some **client-side validation** on a form

```
<!DOCTYPE html>

<html>
  <head>
    <title>DOM!</title>
    <script>
       $(document).ready(function(event) {
           $("#registration").submit(function(event) {
               // validate inputted username
               if ($('#registration input[name=username]').val() == '')
               {
                   return false;
               }

               // validate inputted password
               var password = $('#registration input[name=password]').val();
               if (password == '' || password.length < 8)
               {
                   return false;
               }
           });
       });
    </script>
  </head>
  <body>
    <form id="registration" action="register.php" method="post">
      Username: <input name="username" type="text"/><br/>
      Password: <input name="password" type="password"/>
    </form>
  </body>
</html>
```

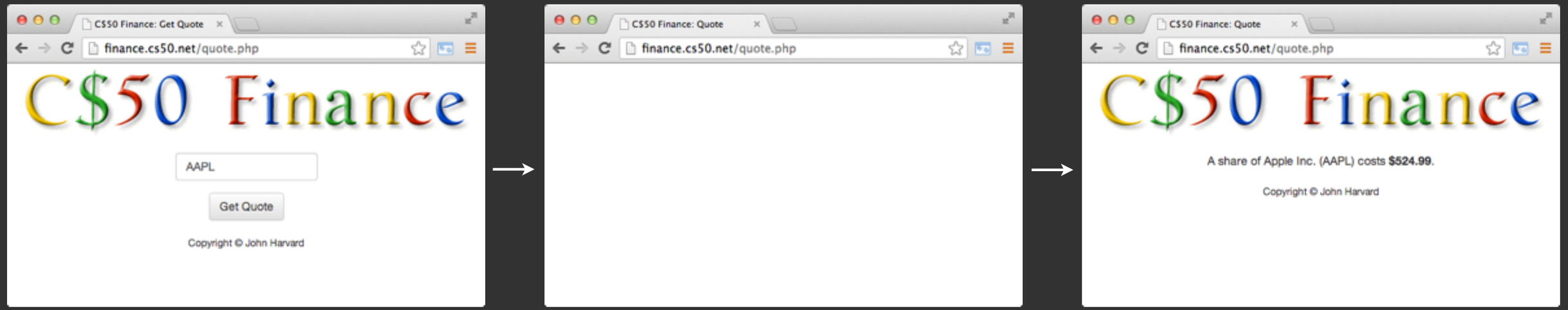
pros, cons

makes JavaScript easier to write, but slower than pure JavaScript

Ajax

CS50 Quiz 1 Review

so far...

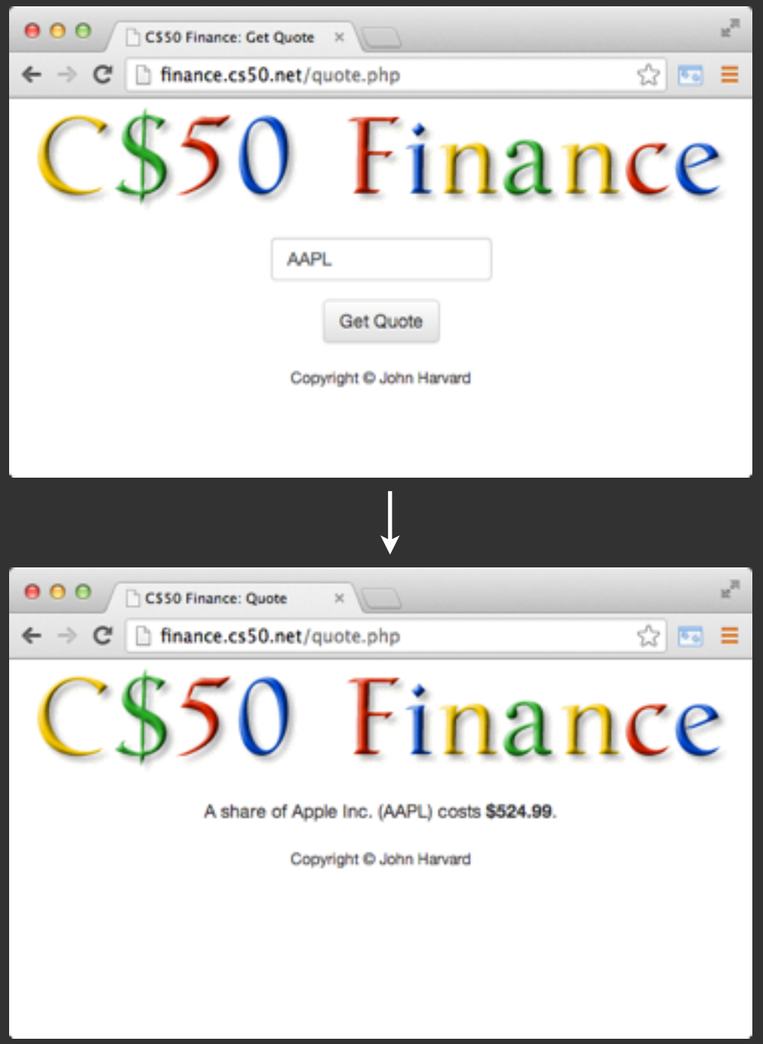


flash when loading new page

:(

something like this would be nice

```
// get quote name from input element
var name = $("#quote_input").val();

// fetch stock price from Yahoo!
var price = stockInfoFromYahoo(name);

// insert message into the DOM, replacing form
var msg = "A share of " + name + " costs " + price;
$("#form_wrapper").html(msg);
```



ahhhh … nice! no refreshing!

problem

"synchronous execution"

a line in javascript only executes after the previous line is done executing

```
var name  = $("#quote_input").val();
var price = stockPriceFromYahoo(name);
var msg   = "A share of " + name + " costs " + price;
$("#form_wrapper").html(msg);
```

What if Yahoo! is having a extremely slow day and this function takes **seconds**?

Your website will completely freeze for those seconds. This is Bad™.

solution

"**asynchronous** javascript and xml (**A**jax)"

keep going for now, but make a promise to execute a function later on when data comes back

```
var name = $("#quote_input").val();
var url = "http://yahoo.com/stock?s=" + name;

// how a GET Ajax request is done with jQuery
$.get(url, function(price) {
    var msg = "A share of " + name + " costs " + price;
    $("#form_wrapper").html(msg);
});
```

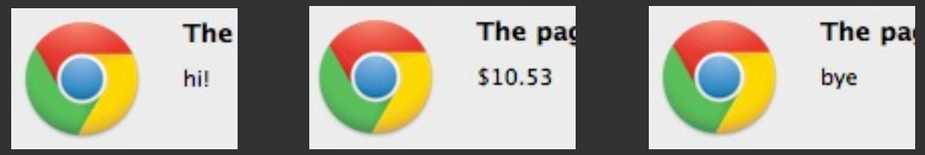
Instead of waiting **synchronously**, we immediately move on.

We'll get notified later when the data is ready for processing.

hopefully, this helps!

```
1 alert("hi!");
2 var price = stockInfoFromYahoo(name);
3 alert(price);
4 alert("bye!");
```

@ 0s 3s 3.001s



synchronous

```
1 alert("hi!");
2 $.get(url, function(price) {
    4 alert(price);
  });
3 alert("bye!");
```

@ 0s 0.001s 3s

asynchronous

Ajax tl;dr

let's us fetch data, without refreshing the current page.
let's us do this in an asynchronous way that doesn't freeze our page.

XSS Attacks

CS50 Quiz 1 Review

email	fullname
jong@college.harvard.edu	<script>postUnflatteringFacebookStatus(); </script>

```
<?php foreach ($friends as $friend): ?>
    <div>
      <?= $friend["fullname"] ?>
    </div>
<?php endforeach; ?>
```

or

```
$("#name_wrapper").html(friend);
```

```
<!DOCTYPE html>

<html>
  <head>
    <title>Facebook!</title>
  </head>
  <body>
    <div>
      Lauren Carvalho
    </div>
    <div>
      Milo Banana
    </div>
    <div>
      <script>postUnflatteringFacebookStatus();</script>
    </div>
  </body>
</html>
```

??????????

simple solution

email	fullname
jong@college.harvard.edu	<script>postUnflatteringFacebookStatus(); </script>

```
<?php foreach ($friends as $friend): ?>
    <div>
      <?= htmlspecialchars($friend["fullname"]) ?>
    </div>
<?php endforeach; ?>
```